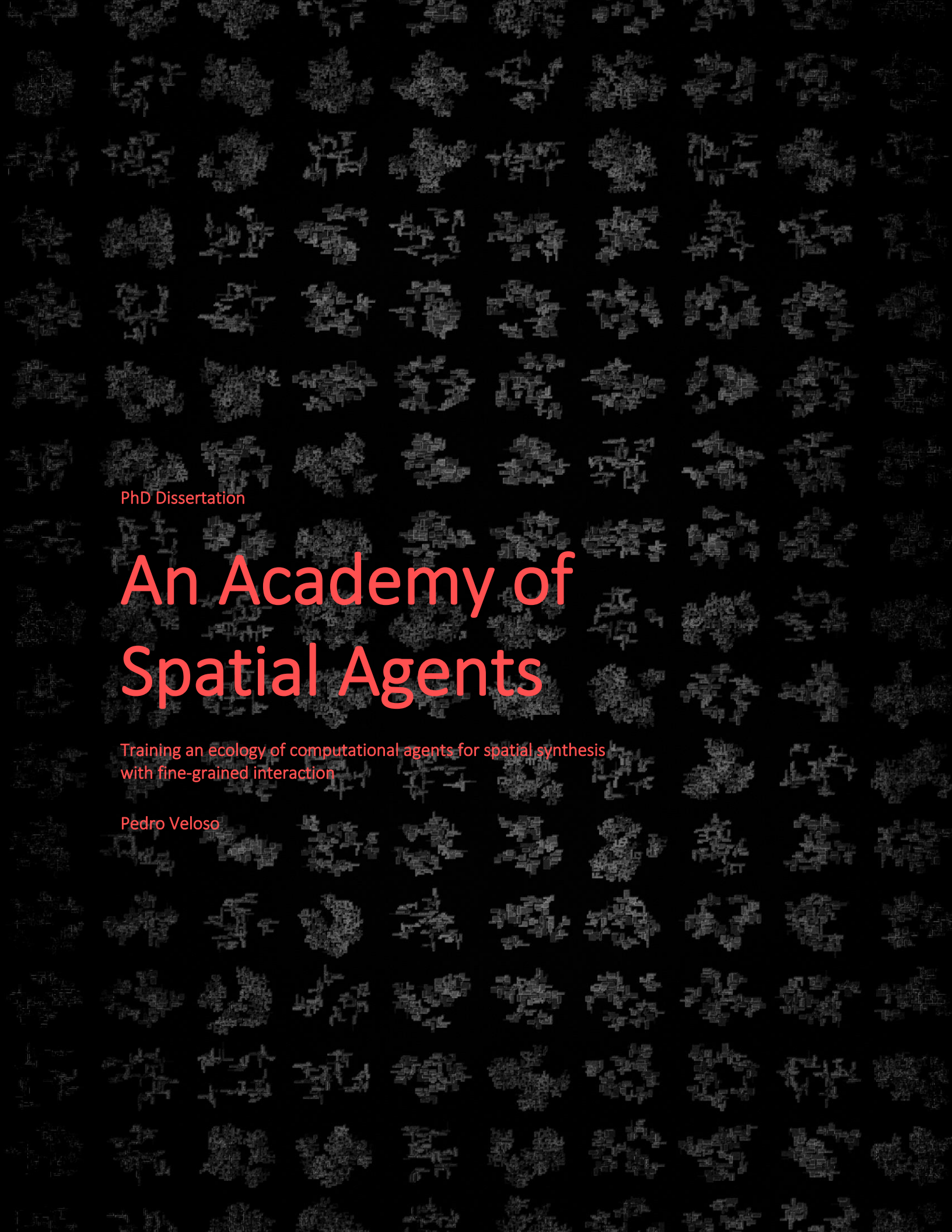


PhD Dissertation

An Academy of Spatial Agents

Training an ecology of computational agents for spatial synthesis with fine-grained interaction

Pedro Veloso

CARNEGIE MELLON UNIVERSITY

College of Fine Arts

School of Architecture

Dissertation

Submitted in Partial Fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

TITLE:

An Academy of Spatial Agents

training an ecology of computational agents for spatial synthesis
with fine-grained interaction

PRESENTED BY: Pedro Luís Alves Veloso

ACCEPTED BY:

Mary Ellen Poole
Mary Ellen Poole (Jan 17, 2023 13:00 EST)

Jan 17, 2023

Mary Ellen Poole Dean DATE

Jan 15, 2023

Omar Khan Head of School DATE

CARNEGIE MELLON UNIVERSITY

College of Fine Arts

School of Architecture

Dissertation

Submitted in Partial Fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

TITLE:

An Academy of Spatial Agents

training an ecology of computational agents for spatial synthesis
with fine-grained interaction

PRESENTED BY: Pedro Luís Alves Veloso

ACCEPTED BY ADVISORY COMMITTEE:

Ramesh Krishnamurti	Principal Advisor	Jan 13, 2023 DATE
Gianni Di Caro	Advisor	Jan 13, 2023 DATE
Reinhard König	Advisor	Jan 13, 2023 DATE

I hereby declare that I am the author of this dissertation

I authorize Carnegie Mellon University to lend this dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Carnegie Mellon University to reproduce this dissertation by photocopy or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

 Jan 13, 2023

Pedro Luis Alves Veloso

Copyright © 2022 Pedro Luis Alves Veloso

All rights reserved

Acknowledgements

First, I would like to thank Prof. Ramesh Krishnamurti for being a sharp, open-minded, and generous advisor. Your support and guidance in a PhD topic of my personal interest was much appreciated. I am also grateful by the fact that this research was supported by the Brazilian National Council for Scientific and Technological Development (CNPq).

I would also like to express my sincere gratitude to all the designers, educators, and researchers who provided inspiration or support on a much longer trajectory that culminates in this PhD dissertation.

Thanks to Prof. Marcelo Malheiros (a grad student in computer engineering at the time) who taught me how to code 25 years ago. As you will notice over the next chapters, many of the topics that I was interested at that time, such as coding, games, and visual arts, have been a constant in my research career. I am also grateful to Prof. Sylvia Ficher for incentivizing my personal interest in thinking beyond conventional architectural ideals and methods. I would like to express my gratitude to the many friends, researchers, and educators that introduced me to the field of computational design more than a decade ago. Among them, I would like to mention Prof. Gabriela Celani and Prof. Gonçalo Henriques for their classes and workshops on design methods, scripting, digital fabrication, parametric design, and complex geometry.

Over the last years, Carnegie Mellon University (CMU) provided a unique intellectual environment and the opportunities for interdisciplinary dialogues that promoted my development as designer and researcher. At CMU, I had the privilege to learn about different aspects of computer science and artificial intelligence with professors such as Ziv Bar-Jospeh, Gianni di Caro, Matt Gormley, and Katerina Fragkiadaki. I also would like to express my gratitude to my CMU friends with whom I had the pleasure not only to discuss ideas about design and computation but also to enjoy great moments with. Among others, thanks Olaitan Awamolo, Varvara Toulkeridou, Eleni Katrini, Manuel Ladron de Guevara, and Jinmo Rhee.

I would like to show appreciation to my family for the support and patience over this long journey. Thanks to Kate for being the best partner and the initiator of many adventures in Pittsburgh, Fayetteville, and many other places in the world. Thanks to Michael and Ellen for the many holiday events, dinners, and enjoyable conversations in the steel city. I would like to thank my dad (Chico) for fostering my early interest in art and culture. Thanks to my mother Debora for her unconditional love, support, and for delicious cooking. Foremost, I would like to express my admiration and gratitude to my late grandad (Vô Leandro), whose unfailingly belief in my capacity to succeed has been the one of my main motivations to fight for my dreams. His memory will be always with me.

Abstract

Motivated by situated cognition in architectural design, this research investigates a computational approach for spatial synthesis that supports designers inside the generative loop with fine-grained interaction. After reviewing and creating a taxonomy of classic paradigms of generative modeling, we present a research guideline for interactive spatial synthesis. This guideline comprehends control of problem parameters, real-time interaction, step-by-step and fine-granular construction of the spatial representations, event-driven computing, reduced assumption about control strategies, and simultaneously addressing multiple spatial goals

To test the feasibility of this research guideline, we investigate multi-agent spatial synthesis (MASS), which comprehends a collection of discrete and autonomous entities referred to as spatial agents that interact in a shared environment. By distributing the computation, MASS benefits from robustness and scalability and can enable designers to engage in interactive simulations to explore spatial patterns in large state spaces under uncertainty. Besides, similarly to agent-based models, MASS can potentially model the dynamics of complex systems and display qualities such as emergence and complexity. Nevertheless, there is no clear methodology to design agents that can manage conflicts with realistic spatial synthesis goals. Existing prototypes for MASS typically rely on trends such as bioinspiration and physics-simulation and often on hybridization with global solutions, such as metaheuristics.

We address this bottleneck of agent design in a MASS research artifact by using multi-agent reinforcement learning as a training methodology for spatial agents. It enabled creating agent policies that address multiple architectural goals and that support fine-grained interaction with reduced dependency on initial design states and order of operations. The resulting artifact is based on training an ecology of spatial agents with parameter sharing and a parameterized reward function to support domain knowledge and enable heterogeneous agent behaviors. The MASS artifact was evaluated in multiple design cases: a house design in different site conditions, a museum on a large interstitial site, and a speculative design of a house complex on a large empty site. Overall, the agent policies satisfied custom spatial goals and produced consistent morphological patterns for the different design cases.

After the evaluation of the MASS artifact, we integrated it to an interactive simulation workflow to support designers in the generative loop with access to fine-grained interaction, event-driven computing, and other properties described in the initial guideline. The success of the design artifact and the implementation of the interactive simulation workflow are proofs by construction of the feasibility of the proposed approach to design spatial agents for interactive spatial synthesis. Besides, they open doors to different lines of research, such as agent-based architectural morphology, spatial synthesis of complex spaces based on ecologies of agents, and mainly, the evaluation of situational and collaborative design behavior with architects.

Keywords: Generative design; Spatial synthesis; Agent-based modeling; Multi-agent reinforcement learning; Fine-grained interaction; Interactive simulation.

Table of Contents

Signature pages i

Acknowledgements vii

Abstract viii

Table of Contents ix

Table of Figures xiii

0. Introduction 1

0.1. Research topic 1

0.2. Research motivation 1

0.3. Research aim 2

0.4. Agent-based computing for research 3

0.5. Research questions 3

0.6. Research method 4

0.7. Research objectives 5

0.8. Impact of the research 5

0.9. Summary of chapters 7

1. In pursuit of agent-based spatial synthesis for situational design 9

1.1. Introduction 9

1.2. Defining generative model and spatial synthesis 9

1.2.1. From ideation to generative models 9

1.2.2. From architectural composition to spatial synthesis 12

1.3. Classic Generative Models 15

1.3.1. Parametric model 16

1.3.2. Rule-based model 18

1.3.3. Optimization 22

1.3.4. Composition 26

1.4. A guideline for spatial synthesis with designers in the generative loop 31

1.5. Agent-based computing in generative design 33

1.5.1. Agent-based computing and simulation 33

1.5.2. Opportunity: fine-grained interaction with spatial agents 38

1.5.3. Challenge: design of agents that can address multiple spatial goals 39

1.6. Training agents for spatial synthesis 40

1.6.1. Reinforcement Learning 42

1.6.2. Deep Reinforcement Learning 44

1.6.3. Multi-agent RL 45

2. Designing and implementing the design artifact 47

2.1. Introduction 47

2.2. Representation 48
2.2.1. Environment as space 48
2.2.2. Agent observation 49
2.2.3. Action space 52

2.3. Simulation 53
2.3.1. The transition function 53
2.3.2. Spatial objectives and reward function 53
2.3.3. Environment parameters 56

2.4. Prototyping 56
2.4.1. P1 58
2.4.2. P2 65
2.4.3. P3 74

3. Evaluating the design artifact 79

3.1. Introduction 79

3.2. Design Case 1: single family housing on small site 79
3.2.1. P1 81
3.2.2. P2 87
3.2.3. P3 93

3.3. Design Case 2: art gallery on a large site 98
3.3.1. P2 99
3.3.2. P3 112

3.4. Design Case 3: site-less house complex 117
3.4.1. P2 with policy Adjacency, Area, and Daylit (75) 118
3.4.2. P2 with policy Adjacency, Area, and Not Folding (3) 121
3.4.3. P3 with policy Adjacency, Area, and Not Folding (5) 124
3.4.4. P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50) 127
3.4.5. Evaluation 130

3.5. Discussion on the design artifact 132

4. An interactive simulation workflow for situational design exploration 133

4.1. General Workflow 133

4.2. Design artifact 134
4.2.1. Parametric model 135

4.3. Game Engine 137
4.3.1. Interface 137
4.3.2. Policy 138
4.3.3. Basic commands 139
4.3.4. Agent body commands 142
4.3.5. Agent connection commands 144
4.3.6. Agent cells edition commands 145

4.3.7. Environment obstacle commands 147
4.3.8. Environment boundary commands 148

4.4. Activating the parametric model 148

4.5. Discussion 150

5. Discussion, future steps, and conclusions 151

5.1. Technical extensions 151

5.2. Agent-based architectural design and morphology 152

5.3. Situational design exploration and collaborative design 154

5.4. Conclusion 155

6. References 159

7. Appendices 177

7.1. Appendix A: Notes on architectural composition 177
7.1.1. Origins of architectural composition 177
7.1.2. Composition and non-composition over the last century 178

7.2. Appendix B: The influence of classic AI methods on generative models 181
7.2.1. Search as a model for architectural design 184
7.2.2. Search as a model for generative models 186
7.2.3. Generative models for spatial synthesis 188

7.3. Appendix C: Beyond objective rationality in generative design? 192
7.3.1. The divide between black-box and glass-box design models 192
7.3.2. From an objective to a pragmatic rationality in design 193
7.3.3. Generative models beyond spatial synthesis 197

7.4. Appendix D: Agent-Based Computing 200
7.4.1. Artificial Intelligence and Multiagent Systems 200
7.4.2. Complex systems and agent-based modeling 201
7.4.3. Bio-Inspired Artificial Intelligence and Engineering 205

7.5. Appendix E: A review of multi-agent spatial synthesis 207
7.5.1. Type 1: Agents as moving spatial units 208
7.5.2. Type 2: Agents that occupy a space 213
7.5.3. Type 3: Agents that partition space 216
7.5.4. Analysis of features 220

7.6. Appendix F: Deep Learning 225
7.6.1. Origins of Deep Learning 225
7.6.2. From symbolic to learning models 226
7.6.3. Neural Networks for inference 228
7.6.4. Training neural networks 230
7.6.5. Types of Machine Learning 231
7.6.6. Generative Learning 232

7.7. Appendix G: Reinforcement learning 236
7.7.1. The goal of a RL agent 236

7.7.2. Challenges in reward functions 237
7.7.3. Challenges in deep reinforcement learning 239
7.7.4. Reward Engineering and Shaping 239
7.7.5. Single-agent RL Algorithms 243
7.7.6. Multi-agent RL algorithms and methods 260
7.7.7. Multi-agent RL environments 261
7.7.8. A brief overview of reinforcement learning in architectural design 263

7.8. Appendix H: Additional data for the design artifact evaluation 265

Table of Figures

FIGURE 1 A: DESIGN IDEATION: THE DIRECT INTERACTION OF DESIGNERS WITH VISUAL MEDIA TO BUILD AN IMAGINARY DESIGN WORLD. B: FORMULATION OF A GENERATIVE MODEL: THE DEFINITION OF INPUT AND GENERATOR IMPLIES THE EXISTENCE OF A DESIGN SPACE TO NAVIGATE AND LOOK FOR OUTPUT DESIGN ALTERNATIVES; (G). C: EXECUTION OF A GENERATIVE MODEL: DESIGNERS CONTROL AN INPUT (I), THE GENERATOR (G) SYNTHESIZES A DESIGN ALTERNATIVE AS AN OUTPUT (O), WHICH IS OBSERVED BY THE DESIGNERS. BY AUTHOR. 11

FIGURE 2 MATRIX OF DIFFERENT COMPOSITIONAL AND NON-COMPOSITIONAL ORDERS. BY THE AUTHOR WITH DIAGRAMS RE-DRAWN FROM (LUCAN 2012, 402, 476; ALEXANDER ET AL. 1977, 518, 883, AND 888; ALLEN 2009, 192, 218; 2012B, 98) 13

FIGURE 3 TAXONOMY FOR BUILDING BLOCKS OF GENERATIVE MODELS. BY AUTHOR. 15

FIGURE 4 A: FUNCTION TO GENERATE THE COORDINATES OF A HYPERBOLIC PARABOLOID. B: REPRESENTATION OF THE SURFACES IN A SPECIFIC DOMAIN OF U AND V, BASED ON THE VARIATION OF THE PARAMETER A. BY AUTHOR. 16

FIGURE 5 A: DIAGRAM OF GENERATIVE MODEL BASED ON THE PARAMETERS A, B AND C AND ON AN INTERNAL NETWORK OF OPERATIONS. B: VARIATIONS OF THE GEOMETRY. C: DIAGRAM OF PARAMETER SPACE OF THE MODEL, ASSOCIATING THE COORDINATES OF THE PARAMETERS WITH THE RESULTING DESIGNS. BY AUTHOR WITH MODEL DEVELOPED BY ADIE ALNOBANI FOR PARAMETRIC MODELING, CMU, SPRING 2016. 17

FIGURE 6 EXERCISE FOR PARAMETRIC MODELING (CMU, SPRING 2016): OUTDOOR THEATER. A: BASE PARAMETRIC MODEL, WITH SPECIFIC GEOMETRIC REQUIREMENTS FOR THE THEATER (SEATS, VOLUME WITH NO COLUMNS AND VOLUME TO SHELTER). B: DIAGRAM OF THREE DIFFERENT PARAMETRIC MODELS TO CREATE A STRUCTURE THAT RESPONDS TO THE VARIATIONS OF THE BASE MODEL. BY AUTHOR WITH MODELS DEVELOPED BY ADIE ALNOBANI, ATEFEH GOLOUJEH, AND CECILIA FERRANDO FOR PARAMETRIC MODELING, CMU, SPRING 2016. 18

FIGURE 7 EXAMPLE BASED ON THE HALF-HEXAGON GRAMMAR (W. MITCHELL 1977, 436–39; 1990, 143–47). A: DEFINITION OF A NON-TERMINATING RULES (X, Y, Z) AND A TERMINATING RULE (W). NOTICE THAT THE DOT REPRESENTS A LABEL, WHICH RESTRICTS THE PRECONDITION FOR FUTURE DERIVATIONS. B: PARTIAL REPRESENTATION OF THE DESIGN SPACE INDUCED BY THE RULE SET. THICKER LINE REPRESENTS EXAMPLE OF A SEQUENCE OF DERIVATIONS. BY AUTHOR. 19

FIGURE 8 A: DIAGRAM OF A RULE-BASED GENERATIVE MODEL USING A VARIATION OF THE HALF-HEXAGON TABLE GRAMMAR. B: EXAMPLE OF RESULTING DESIGN; C: TWO VARIATIONS USING PARAMETERS FOR ANGLE AND WIDTH. BY AUTHOR. 20

FIGURE 9 EXAMPLE OF A GRAMMAR FOR THE CUSTOMIZATION OF AN APARTMENT LAYOUT (VELOSO, CELANI, AND SCHEEREN 2018). A: RULE SET ORGANIZED IN A GRAPH. B: TWO EXAMPLES OF FLOOR PLANS THAT CAN BE GENERATED BY THE RULES. C: ALL THE FLOORPLANS THAT CAN BE GENERATED. BY AUTHOR. 21

FIGURE 10 A: PARAMETRIC MODEL WITH A FITNESS FUNCTION AND AN OPTIMIZER CONTROLLING ITS PARAMETERS. B: VARIATIONS OF THE DESIGN. C: FITNESS LANDSCAPE: DIAGRAM REPRESENTING THE FITNESS VALUE OF EVERY SOLUTION IN THE PARAMETER SPACE (RED IS A LOW VALUE AND BLUE A HIGH VALUE). BY AUTHOR WITH MODEL DEVELOPED BY ADIE ALNOBANI FOR PARAMETRIC MODELING, CMU, SPRING 2016. 22

FIGURE 11 A: OPTIMIZATION OF FLOORPLANS WITH GENETIC ALGORITHMS AND KD-TREE. THE COORDINATES FOR THE POINTS USED AS INPUT FOR A KD-TREE PARTITION ARE THE PARAMETERS. B: EXAMPLES OF RESULTING DESIGNS. BY AUTHOR AND INSPIRED BY (KNECHT AND KÖNIG 2010A). 24

FIGURE 12 USE OF GENETIC PROGRAMMING (EMBRYO) TO GENERATE PARAMETRIC DEFINITIONS FOR SCULPTURES. A-D: VARIATIONS OF PARAMETRIC GRAPHS AND THEIR RESPECTIVE PARAMETRIC SCULPTURE BELOW. WORK PRODUCED FOR GENERATIVE SYSTEMS FOR DESIGN, CMU, FALL 2019. 25

FIGURE 13 A: RULES OF THE QGROWTH GRAMMAR. A-B: GENERATING FORMS WITH STOCHASTIC SELECTION OF RULES (RULE X: 60%, RULE Y: 30%, RULE Z: 10%) AND PARAMETER P (0.2: 50%, 0.8: 50%). B: PRECONDITION SELECTION WITH A QUEUE (QUADRILATERALS ADDED FIRST TO THE FRONTIER ARE SELECTED FIRST). C: PRECONDITION SELECTION WITH A PRIORITY QUEUE (QUADRILATERALS CLOSER TO THE CIRCLE ARE SELECTED FIRST). BY AUTHOR. 27

FIGURE 14 INTERACTIVE AGGREGATIONS. SET OF RULES FOR AGGREGATION OF COMPONENTS OF HOUSING COMPLEX WITH SYMMETRIES: A) CORE MODULE (RED) AND HORIZONTAL CIRCULATION (BLUE); B) HORIZONTAL CIRCULATION (RED) AND HORIZONTAL CIRCULATION (BLUE); C) HORIZONTAL CIRCULATION (RED) AND LIVING/KITCHEN MODULE (BLUE); D) CORE (RED) AND CORE (BLUE); E)

living/kitchen module (red) and double height living (blue); f) living/kitchen module (red) and bedroom (blue); g) living/kitchen module(red) and deck(blue); h) bedroom (red) and deck (blue). a) and b) Two examples of aggregations generated with sequences of random sampling combined with optimization and iterative creation of obstacles as constraints. Reproduced with permission of Hardik Makrubiya and Jesse T. Gonzalez. Work produced for Generative Systems for Design, CMU, Fall 2021. 28
Figure 15 Ocean City Revisited. Elements used as tiles for aggregation (a and b); two examples of aggregations with different ground and boundary constraints. Reproduced with permission of Nicholas Coppula and Davis Dunaway. Work produced for Generative Systems for Design, CMU, Fall 2021. 29
Figure 16 Autoilet. A: parameter space of the input (boundary, corner, and door placement) for rectangular and L-shaped footprints. B: examples of solutions using DFS for placement of toilet, sink and ADA circle. Reproduced with permission of Jichen Wang. Work produced for Generative Systems for Design, Fall 2019. 30
Figure 17 Taxonomy updated with agent-based model and simulation. By Author. 33
Figure 18 A: Moore neighborhood with different radii for single-cell agent and for multicellular agent. B: Von Neumann neighborhood with different radii for single-cell agent and for multicellular agent. C: simulation with multiple agents that use two cells for expansion. Images by Author. Images in C relied on model developed by Pragya Gupta for Design and Computation 2, Fall 2018. 35
Figure 19 Boids: a model that simulates the flocking behavior of bird. A: neighborhood based on radius and visibility angle. B: three basic behaviors (separation, alignment, and cohesion), which add forces to the agent conditioned on the state of the neighbors. C: Boids interacting in a simulation. By Author. A and B were re-drawn from diagrams in (Reynolds 1999). 35
Figure 20 A: Diagram of the agent-based model. The bottom loop represents the time steps, and the top loop represents the updates of the agents for one-time step. B: three distinct steps of the simulation. By Author. 37
Figure 21 Agents generating an underground gallery on campus. A: stochastic program with three rules. B: design of campus gallery based on agents with r2 = 0.59 and r3 = 0.67 at time step 200. C: the resulting spatial configuration for the same initial environment with different parameter: r2 and r3 increase the probability of choosing rules 2 and 3 respectively. Reproduced with permission of Jinmo Rhee. Work produced for Generative Models for Design, CMU, Fall 2019. 38
Figure 22 Updated taxonomy with new level for learning, comprehending the components generative learning and behavioral learning. By Author. 41
Figure 23 A: the formulation of generative model using a data set (D), a neural network (NN), and a learning algorithm (L). B: execution of a generative model: designers control an input (I); the trained neural network works as part or as the whole generator (G) and synthesizes a design alternative as an output (O). By Author. 42
Figure 24 The agent-environment interaction in the case of a MDP. By Author, based on (Sutton and Barto 2018a, 48). 43
Figure 25. Examples of four real floorplans discretized into two levels of grid resolution. By Author. 49
Figure 26 Classification of cells of agent 0: a) visualization of basic configuration; b) body and outside cells of agent 0; c) non-adjacent, adjacent, surface, and structure cells of agent 0. By Author. 50
Figure 27. Left: simplified visualization of agents in the grid with obstacles. Middle: closest neighbours with k = 2: agent 0: [agent 3 and agent 2], agent 1: [agent 4 and agent 0], agent 2: [agent 0 and agent 1], agent 3: [agent 0 and agent 2], agent 4: [agent 1 and agent 0]. Right: example of adjacency goals with l = 2: agent 0: [agent 1, agent 2], agent 1: [agent 0, agent 4], agent 2: [agent 0], agent 3: [], agent 4: [agent 1]. By Author. 51
Figure 28 Variations of perception grid (dotted line) for agent 0, with fixed action grid (5, 5) (continuous line): a) perception grid is not centered to the agent and it is equivalent to the environment grid (20, 20); b) perception grid is centered and has shape (7, 7); c) perception grid is equivalent to the action grid (5, 5). The case c is just displayed for completeness as it overly restricts the agent's perception. By Author. 51
Figure 29: Row 1: visualization of obstacles (agents are added just for visualization); row 2: classification of the cells: structure (dark blue), surface (medium blue), and offset (light blue); row 3: mask with the action space of the agent; row 4: score of the agent with the respect to the area over the action grid: 0: (17/2), 1: (11/ 4), 2: (15/8), 3: (13/ 16), 4: (12/ 25); row 5: soft adjacency values based on Manhattan distance; row 6:indication of folding

cells: L-folds (light blue), U-folds (dark blue); row 7: the current utility of the agent represented in the internal cells of the agent. By Author. 59
Figure 30 Simplified graphical representation of the layers, operations, and arrays of DCNN. Every CNN is composed of 2 convolutional layers with 64 filters of size 3x3, stride 1, padding 1 and a Leaky ReLU activation. The final FC network has 3 linear layers with 104, 52, and 26 neurons. Its two first layers use Leaky ReLU activation, while the last uses a hyperbolic tangent function (Tanh). By Author. 61
Figure 31 96 examples of random boards generated by the engine. By Author. 63
Figure 32 96 examples of agents initialized randomly by the engine. By Author. 63
Figure 33 Cumulative rewards for the agents during training. The blue line is the average cumulative rewards of the six agents on the training episodes. Notice that the oscillations in the blue graph are the result of the custom schedule of reduction of $\boldsymbol{\varepsilon}$. The orange line is average cumulative rewards of the six agents in testing episodes with greedy actions. The vertical dotted line at 43350 represents the best policy, which was adopted for the test in the design scenarios. By Author. 64
Figure 34: Observation for agent 0 with action grid of shape (5, 5), perception grid of shape (15, 15), k=2, and l=2. Row 1: grid with obstacles (agents are added just for visualization). The following rows are based on the g agents that share a relationship of adjacency with agent 0 (1, and 2) and its k closest neighbors that do not share a relationship of adjacency (3 and 4). Row 2: classification of the cells: structure (dark blue), surface (medium blue), and offset (light blue); row 3: score of the agent with the respect to the area over the action grid; row 4: soft adjacency values based on Manhattan distance; row 5:indication of folding cells: L-folds (light blue), U-folds (dark blue); row 6: indication of the daylit cells; row 7: the current utility of the agent represented in the internal cells of the agent. By Author. 67
Figure 35 96 examples of boards generated by the engine. By Author. 70
Figure 36 96 examples of agent initialization by the engine. By Author. 71
Figure 37 Training graphs with training and test rewards for P2. By Author. 72
Figure 38 Training graphs with losses for P2. By Author. 73
Figure 39 96 examples of boards generated by the engine. By Author. 75
Figure 40 96 examples of agent initialization by the engine. By Author. 76
Figure 41 Training graphs with training and test rewards for P3. By Author. 77
Figure 42 Training graphs with losses for P3. By Author. 78
Figure 43 Program of the house in case 1. The leading number is used to identify the spaces in the next images. The numbers in parenthesis indicate target area. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces. By Author. 80
Figure 44 Three different sites: (1) site on the top of the hill, (2) site with central obstacles, (3) site with stream and bridge. By Author. 81
Figure 45 Rows 1-6: simulation of 12 agents from P1 in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author. 82
Figure 46 Rows 1-6: simulation of 12 agents from P1 in a terrain with a large tree and rocks as obstacles at the center. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author. 83
Figure 47 Rows 1-6: simulation of 12 agents from P1 in a terrain with a bridge over a stream. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author. 84
Figure 48 Example of layout generated by P1. Artistic rendering reproduced with permission of Jinmo Rhee. 85
Figure 49 Graphs with the performance of the agents from P1 (y axis) over an episode (x axis) for each of the three environments. The areas between dotted lines indicate the periods of perturbation. The means (m) and standard deviations (σ) consider the three stages with ε-greedy action selection but not the periods with random perturbation. By Author. 86
Figure 50 Rows 1-6: simulation of 12 agents from P2 in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author. 88
Figure 51 Rows 1-6: simulation of 12 agents from P2 in a terrain with a large tree and rocks as obstacles at the center. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author. 89

FIGURE 52 ROWS 1-6: SIMULATION OF 12 AGENTS FROM P2 IN A TERRAIN WITH A BRIDGE OVER A STREAM. EVERY PAIR OF ROWS CONTAIN THE FIRST 4 STEPS IN A STAGE OF THE SIMULATION AND 4 STEPS SELECTED VISUALLY. BY AUTHOR. 90
FIGURE 53 GRAPHS WITH THE PERFORMANCE OF THE AGENTS FROM P2 OVER ONE EPISODE FOR EACH SITE. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS OF PERTURBATION. THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE THREE STAGES WITH THE POLICY BUT NOT THE PERIODS WITH RANDOM PERTURBATION. BY AUTHOR. 91
FIGURE 54 GRAPHS WITH THE AVERAGE PERFORMANCE OF THE AGENTS OVER 50 EPISODES FOR EACH OF THE THREE ENVIRONMENTS. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE THREE STAGES WITH POLICY BUT NOT THE PERIODS WITH RANDOM PERTURBATION. BY AUTHOR. 92
FIGURE 55 ROWS 1-6: SIMULATION OF 12 AGENTS IN A TERRAIN ON THE TOP OF A HILL WITH IRREGULAR BOUNDARIES. EVERY PAIR OF ROWS CONTAIN THE FIRST 4 STEPS IN A STAGE OF THE SIMULATION AND 4 STEPS SELECTED VISUALLY. BY AUTHOR. 93
FIGURE 56 ROWS 1-6: SIMULATION OF 12 AGENTS IN A TERRAIN WITH A LARGE TREE AND ROCKS AS OBSTACLES AT THE CENTER. EVERY PAIR OF ROWS CONTAIN THE FIRST 4 STEPS IN A STAGE OF THE SIMULATION AND 4 STEPS SELECTED VISUALLY. BY AUTHOR. 94
FIGURE 57 ROWS 1-6: SIMULATION OF 12 AGENTS IN A TERRAIN WITH A BRIDGE OVER A STREAM. EVERY PAIR OF ROWS CONTAIN THE FIRST 4 STEPS IN A STAGE OF THE SIMULATION AND 4 STEPS SELECTED VISUALLY. BY AUTHOR. 95
FIGURE 58 GRAPHS WITH THE PERFORMANCE OF THE AGENTS FROM P3 OVER ONE EPISODE FOR EACH SITE. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE THREE STAGES WITH THE POLICY BUT NOT THE PERIODS WITH RANDOM PERTURBATION. BY AUTHOR. 96
FIGURE 59 GRAPHS WITH THE AVERAGE PERFORMANCE OF THE AGENTS FROM P3 OVER 50 EPISODES FOR EACH SITE. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS OF PERTURBATION. THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE THREE STAGES WITH THE POLICY BUT NOT THE PERIODS WITH RANDOM PERTURBATION. BY AUTHOR. 97
FIGURE 60 PROGRAM FOR ART GALLERY. THE LEADING NUMBER IS AN IDENTIFIER OF THE SPACES. THE NUMBERS IN PARENTHESIS INDICATE TARGET AREA IN TERMS OF CELL UNITS. THE CONNECTIONS INDICATE ADJACENCY. THE COLORED GROUPING INDICATES SPATIAL INTEGRATION BETWEEN SPACES. BY AUTHOR. 98
FIGURE 61 SITE FOR THE ART GALLERY. DOTTED RED LINE REPRESENTS THE BOUNDARY AND OBSTACLES FOR THE AGENTS. BY AUTHOR. 99
FIGURE 62 AGENTS TRAINED FOR ADJACENCY AND AREA GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 101
FIGURE 63 PERFORMANCE OF P2 WITH POLICY ADJACENCY AND AREA. ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 102
FIGURE 64 AGENTS FROM P3 TRAINED FOR ADJACENCY, AREA, AND NOT FOLDING (3) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 103
FIGURE 65 PERFORMANCE OF P2 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (3). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 104
FIGURE 66 AGENTS TRAINED FOR ADJACENCY, AREA, AND NOT FOLDING (5) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 106
FIGURE 67 PERFORMANCE OF P2 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (5). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES

INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 107
FIGURE 68 AGENTS TRAINED FOR ADJACENCY, AREA, AND DAYLIT (75%) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 108
FIGURE 69 PERFORMANCE OF P2 WITH POLICY ADJACENCY, AREA, AND DAYLIT (75%). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 109
FIGURE 70 AGENTS TRAINED FOR ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (75%) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 110
FIGURE 71 PERFORMANCE OF P2 WITH POLICY ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (75%). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). BY AUTHOR. 111
FIGURE 72 AGENTS TRAINED FOR ADJACENCY, AREA, AND NOT FOLDING (5) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 113
FIGURE 73 PERFORMANCE OF P3 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (5). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 114
FIGURE 74 AGENTS TRAINED FOR ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50%) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 115
FIGURE 75 PERFORMANCE OF P3 WITH POLICY ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50%). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 116
FIGURE 76 FIRST 25 CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND DAYLIT (75) OVER 2000 STEPS. THE CONFIGURATIONS HIGHLIGHTED WERE SELECTED FOR A MORE DETAILED DESCRIPTION. BY AUTHOR. 118
FIGURE 77 LAST 25 CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND DAYLIT (75) OVER 2000 STEPS. BY AUTHOR. 119
FIGURE 78 SELECTED CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND DAYLIT (75) OVER 2000 STEPS. BY AUTHOR. 120
FIGURE 79 FIRST 25 CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (3) OVER 2000 STEPS. THE CONFIGURATIONS HIGHLIGHTED WERE SELECTED FOR A MORE DETAILED DESCRIPTION. BY AUTHOR. 121
FIGURE 80 LAST 25 CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (3) OVER 2000 STEPS. BY AUTHOR. 122
FIGURE 81 SELECTED CONFIGURATIONS PRODUCED BY P2 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (3) OVER 2000 STEPS. BY AUTHOR. 123
FIGURE 82 FIRST 25 CONFIGURATIONS PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (5) OVER 2000 STEPS. THE CONFIGURATIONS HIGHLIGHTED WERE SELECTED FOR A MORE DETAILED DESCRIPTION. BY AUTHOR. 124
FIGURE 83 LAST 25 CONFIGURATIONS PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (5) OVER 2000 STEPS. BY AUTHOR. 125

FIGURE 84 SELECTED CONFIGURATIONS PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, AND NOT FOLDING (5) OVER 2000 STEPS. BY AUTHOR. 126
FIGURE 85 FIRST 25 CONFIGURATIONS PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50) OVER 2000 STEPS. THE CONFIGURATIONS HIGHLIGHTED WERE SELECTED FOR A MORE DETAILED DESCRIPTION. BY AUTHOR. 128
FIGURE 86 LAST 25 CONFIGURATIONS PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50) OVER 2000 STEPS. THE CONFIGURATIONS HIGHLIGHTED WERE SELECTED FOR A MORE DETAILED DESCRIPTION. BY AUTHOR. 128
FIGURE 87 SELECTED CONFIGURATION PRODUCED BY P3 WITH POLICY ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50) OVER 2000 STEPS. BY AUTHOR. 129
FIGURE 88 AVERAGE PERFORMANCE OF DIFFERENT POLICIES OVER 50 EPISODES AND WITH SPRING LAYOUT INITIALIZATION ROW 1: P3 ADJACENCY, AREA, AND DAYLIT (75). ROW 2: P3 ADJACENCY, AREA, AND NOT FOLDING (3). ROW 4: P3 ADJACENCY, AREA, AND NOT FOLDING (5). ROW 4: P3 ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (50). BY AUTHOR. 131
FIGURE 89 OVERVIEW OF INTERACTIVE SIMULATION WORKFLOW. BY AUTHOR. 134
FIGURE 90 BASIC STEPS FOR THE PARAMETRIC MODEL DERIVED FROM THE ENVIRONMENT INFORMATION USING DESIGN CASE 1 AS AN EXAMPLE. BY AUTHOR. 137
FIGURE 91INTERFACE OF THE GAME ENGINE WITH (A) TITLE, (B) MENU, (C) CONNECTION WITH PARAMETRIC MODEL, AND (D) CANVAS. BY AUTHOR. 138
FIGURE 92 EXAMPLE OF RANDOM POLICY. NOTICE THAT THERE ARE MULTIPLE STEPS BETWEEN SNAPSHOTS. BY AUTHOR. 139
FIGURE 93 EXAMPLE OF SIMULATION STARTING FROM A RANDOM ARRANGEMENT AND SWITCHING TO A RL POLICY. NOTICE THAT THERE ARE MULTIPLE STEPS BETWEEN SNAPSHOTS. BY AUTHOR. 139
FIGURE 94 EXAMPLE OF CHANGING THE AREA OF TWO AGENTS AS THE SIMULATION IS RUNNING WITH A RL POLICY. BY AUTHOR. 140
FIGURE 95 EXAMPLE OF MOVING AN AGENT. BY AUTHOR. 141
FIGURE 96 EXAMPLE OF MOVING AGENTS DURING A SIMULATION WITH A RL POLICY. BY AUTHOR. 142
FIGURE 97 EXAMPLE OF ADDING AGENTS DURING SIMULATION WITH RL POLICY. BY AUTHOR. 143
FIGURE 98 DELETING AGENTS DURING A SIMULATION WITH A RL POLICY. BY AUTHOR. 143
FIGURE 99 ADDING AGENTS AND CONNECTING THEM DURING A SIMULATION WITH A RL POLICY. BY AUTHOR. 144
FIGURE 100 EXAMPLE OF CREATION OF CONNECTIONS BETWEEN AGENTS AS THE SIMULATION RUNS WITH A RL POLICY. BY AUTHOR. 145
FIGURE 101 EXAMPLE OF ADDING CELLS TO AGENTS. BY AUTHOR 146
FIGURE 102 EXAMPLE OF ADDING CELLS TO AGENTS WITH LARGER BRUSH. BY AUTHOR 146
FIGURE 103 EXAMPLE OF DELETING CELLS OF THE AGENTS. BY AUTHOR. 147
FIGURE 104 EXAMPLE OF ADDING OBSTACLES TO THE ENVIRONMENT DURING THE SIMULATION WITH RL POLICY. TRANSPARENT MOUSE CURSORS WERE ADDED TO SHOW MULTIPLE STEPS IN A SINGLE SNAPSHOT. BY AUTHOR. 147
FIGURE 105 EXAMPLE OF CHANGING BOUNDARY DURING SIMULATION WITH RL POLICY. BY AUTHOR. 148
FIGURE 106 EXAMPLE OF ACTIVATION OF THE PARAMETRIC MODEL AND CUSTOMIZATION OF THE PROPERTIES OF THE WALLS, SUCH AS THICKNESS AND HEIGHT. BY AUTHOR. 149
FIGURE 107 MOVING AN AGENT, THEN RUNNING HE SIMULATION WITH RANDOM POLICY WITH THE PARAMETRIC MODEL ON. BY AUTHOR. 149
FIGURE 108 STARTING FROM A RANDOM CONFIGURATION AND RUNNING A SIMULATION WITH A RL POLICY AND THE PARAMETRIC MODEL ACTIVATED. BY AUTHOR. 150
FIGURE 109 BASIC STRUCTURE OF AN AI AGENT (FROM RUSSELL AND NORVIG 2021, 37). REPRODUCED BY PERMISSION OF PEARSON EDUCATION, INC. 183
FIGURE 110 DIAGRAMS WITH THE STRUCTURE AND OF FOUR BASIC AI AGENTS (FROM RUSSELL AND NORVIG 2021, 50, 52, 54, 55): A) REFLEX AGENT; B) MODEL-BASED REFLEX AGENT; C) GOAL-BASED AGENT; D)UTILITY-BASED AGENT. SEE DESCRIPTION IN THE TEXT BELOW. REPRODUCED BY PERMISSION OF PEARSON EDUCATION, INC. 184
FIGURE 111 STATE-ACTION GRAPH. REDRAWN BY AUTHOR, BASED ON (W. MITCHELL 1977, 45). 187
FIGURE 112 LEFT: VENN DIAGRAM SHOWING THE RELATIONS BETWEEN SETS U, G, AND S IN A GENERATIVE SYSTEM; RIGHT: SIX DIFFERENT RELATIONSHIPS BETWEEN SOLUTION GENERATION AND SOLUTION EVALUATION PROCEDURES. THE DARKER SHADED AREAS INDICATE SATISFACTORY SOLUTIONS THAT ARE ACTUALLY PRODUCED. REDRAWN BY AUTHOR, BASED ON (W. MITCHELL 1977, 49, 57) 187

FIGURE 113 A CLASSIFICATION OF SELECTED AUTOMATIC SPACE-PLANNERS (HENRION 1978, 177). REPRODUCED WITH PERMISSION OF ELSEVIER SCIENCE &TECHNOLOGY JOURNALS, FROM AUTOMATIC SPACE-PLANNING: A POSTMORTEM? 1978; PERMISSION CONVEYED THROUGH COPYRIGHT CLEARANCE CENTER,INC. 189
FIGURE 114 ROUTE MAP OF THE DESIGN PROCESS (LAWSON 2005, 38, 40, 49). COPYRIGHT ©2005 FROM HOW DESIGNERS THINK: THE DESIGN PROCESS DEMYSTIFIED BY BRYAN LAWSON. REPRODUCED BY PERMISSION OF TAYLOR AND FRANCIS GROUP, LLC, A DIVISION OF INFORMA PLC. 196
FIGURE 115 CONSTRUCTING CGP'S FROM A SET OF PRIMITIVE MECHANISMS (HOLLAND 1998, 135). FROM EMERGENCE BY JOHN H. HOLLAND, COPYRIGHT © 1998. REPRODUCED WITH PERMISSION OF BASIC BOOKS, AN IMPRINT OF HACHETTE BOOK GROUP, INC. 203
FIGURE 116 BOUNDARIES AND TREE REPRESENTATION OF BOUNDARIES (HOLLAND 1995, 118). FROM HIDDEN ORDER BY JOHN H. HOLLAND, COPYRIGHT © 1995. REPRODUCED WITH PERMISSION OF BASIC BOOKS, AN IMPRINT OF HACHETTE BOOK GROUP, INC. 204
FIGURE 117 FUZZY LAYOUT PLANNER. TOP LEFT: DIMENSIONS OF BUBBLE ADAPT TO PRESERVE AREA; BOTTOM-LEFT: DIVIDING BUBBLES; TOP-RIGHT: REPULSION FORCES; BOTTOM-RIGHT: PACKING BUBBLES (BAYRAKTAR AND ÇAĞDAŞ 2013, 377–78). REPRODUCED WITH PERMISSION OF THE AUTHORS. 209
FIGURE 118 SNIFFING SPACE. TOP ROW: SPATIAL ASSOCIATION GENERATING ZONES (IRELAND 2009A, 255, 256). BOTTOM ROW: EXPANSION OF THE MODEL WITH RECTANGULAR BOUNDARIES (IRELAND 2010, 187–88). REPRODUCED WITH PERMISSION OF THE AUTHOR. 209
FIGURE 119 ACTANTS. TOP-LEFT: AN ACTANT AND ITS COMPONENTS; BOTTOM-LEFT: SCALE OF CONSOLIDATION; TOP-CENTER: ACTANTS' VERTICES REACTING TO PHEROMONE; RIGHT: DIFFERENT ASSOCIATIONS (IRELAND 2015B, 585–88). REPRODUCED WITH PERMISSION OF THE AUTHOR. 210
FIGURE 120 FLOATING BUBBLES. LEFT: THREE STAGES IN THE BASIC FLOATING BUBBLE SYSTEM; RIGHT: ONE STAGE OF THE GENERATIVE PROCESS USING SQUARES (HAO AND TING-LI 2010, 177, 181, 182). REPRODUCED WITH PERMISSION OF THE AUTHORS. 211
FIGURE 121 INTERACTIVE BUBBLES. A-B: SIMULATION OF THE BUBBLE DIAGRAM WITH CHANGING TEMPERATURE; C-E: SIMULATION WITH THE DEFINITION OF A CONTAINMENT AREA (VELOSO 2014). BY AUTHOR. 212
FIGURE 122 NARRATIVE LANDSCAPE. LEFT: SOFT BODIES AND CLASSES FOR SPRINGS CONNECTING PARTICLES; RIGHT: TWO LAYOUTS (CHRISTENSEN 2014, 242, 245). REPRODUCED WITH PERMISSION OF THE AUTHORS. 212
FIGURE 123 MULTI-AGENT + EVOLUTIONARY. TOP-LEFT: SPHERICAL AGENT, CAPSULE-LIKE AGENT AND PUSH OPERATION; BOTTOM-LEFT: SWAP OPERATION; TOP-RIGHT: AGENTS SOLVING ADJACENCIES; BOTTOM-RIGHT: CONVERTING AGENTS TO A GRID REPRESENTATION AND OPTIMIZING THE SOLUTION (GUO AND LI 2017, 54–57). REPRODUCED WITH PERMISSION OF THE AUTHORS. 213
FIGURE 124 SNIFFING SPACE II. LEFT: ROUTING BETWEEN DESTINATION POINTS; RIGHT: CARGO CLUSTERING (IRELAND 2009B, 224–25). REPRODUCED WITH PERMISSION OF THE AUTHOR. 214
FIGURE 125 STIGMERGIC SPACE ADJACENCY SOFTWARE. LEFT: GRID TOPOLOGY AND EXAMPLE OF PHEROMONE VALUES; MIDDLE: TEMPLATES FOR EXTERNAL INFLUENCE AND NODE MASKING; RIGHT: A SOLUTION (MEYBOOM AND REEVES 2013, 201–4). REPRODUCED WITH PERMISSION OF THE AUTHORS. 215
FIGURE 126 COLLECTIVE CONSTRUCTION. RESULTS OF THE EXPERIMENTS (NARAHARA 2017, 347). REPRODUCED WITH PERMISSION OF THE AUTHOR. 216
FIGURE 127 ARTIFICIAL LIFE SPACE PLANNING. LEFT: GRID-PLANNING SYSTEM; CENTER: BUBBLE DIAGRAM; RIGHT; INTEGRATION OF GRID-PLANNING SYSTEM AND BUBBLE DIAGRAM (FERNANDO 2014, 661, 662, 665). REPRODUCED WITH PERMISSION OF THE AUTHOR. 216
FIGURE 128 ASSOCIATIVE SPATIAL NETWORKS. TOP-LEFT: ADJACENCY GRAPH FOR EXHIBITION; BOTTOM-LEFT: GROWING NEURAL GAS MAPPING; MIDDLE: REPULSION ALGORITHM DISTRIBUTES GRAPH NODES OVER THE SPACE; RIGHT: TRANSLATING A GRAPH INTO THE LAYOUT OF THE EXHIBITION WITH A 2D VORONOI DIAGRAM (HARDING AND DERIX 2011, 310, 312, 316, 317). REPRODUCED WITH PERMISSION OF THE AUTHORS. 217
FIGURE 129 DWELLING AGGLOMERATOR. TOP-LEFT: FLOCKING RULES; BOTTOM-LEFT: CREATION OF ACCESS ROUTES; RIGHT: TWO LAYOUTS (PUUSEPP 2014A, 49–50). REPRODUCED WITH PERMISSION OF THE AUTHOR. 218
FIGURE 130 RESPONSIVE ALGORITHMS. TOP-LEFT: CIRCLE PACKING, VORONOI DIAGRAM AND VORONOI DIAGRAM WITH RADII; TOP-RIGHT: FORCE-DIRECTED GRAPH OVER VORONOI DIAGRAM WITH RADII; BOTTOM-LEFT: 3D VORONOI CELLS; BOTTOM-RIGHT: THREE-

DIMENSIONAL PACKING WITH A SKIN (BAZALO AND MOLETA 2015, 212–14, 217). REPRODUCED WITH PERMISSION OF THE AUTHORS. 219
FIGURE 131 A) SPECTRAL LAYOUT; B) PHYSICS SIMULATION; C) OFFSET OF CONCAVE HULL; D) VORONOI TESSELLATION; E) GENERATION OF CIRCULATION USING AN ANT-COLONY ALGORITHM; F) FINAL FLOORPLAN (J. SIMON 2017). REPRODUCED WITH PERMISSION OF THE AUTHOR. 220
FIGURE 132 RADAR CHARTS WITH THE RELATIVE OCCURRENCES OF THE FEATURES FOR EACH TYPE. BY AUTHOR AND JINMO RHEE. 224
FIGURE 133 THE EVOLUTION OF THE ACADEMIC INFLUENCE OF THE CONNECTIONIST AND SYMBOLIC APPROACHES. "THE MAIN PLOT (TOP) REPRESENT[S] THE EVOLUTION OF THE RATIO BETWEEN THE NUMBER OF CITED PUBLICATIONS IN THE "CONNECTIONIST" CORPUS2 (ORANGE) AND THE CORRESPONDING NUMBER IN THE "SYMBOLIC" CORPUS3 (BLUE), BOTH NORMALISED BY THE TOTAL NUMBER OF PUBLICATION INDEXED BY WEB OF SCIENCE. THE PLOTS AT THE BOTTOM REPRESENT FOR EACH CORPUS, THE NUMBER OF CITED PUBLICATIONS CITED DURING THE GIVEN PERIOD" (CARDON, COINTET, AND MAZIÉRES 2018A). IMAGES LICENSED UNDER CC BY-ND 4.0, HTTPS://CREATIVECOMMONS.ORG/LICENSES/BY-ND/4.0/. 225
FIGURE 134 HYPOTHETICAL-DEDUCTIVE MACHINES (1) AND INDUCTIVE MACHINES (2) (CARDON, COINTET, AND MAZIÉRES 2018B, 3). IMAGES LICENSED UNDER CC BY-ND 4.0, HTTPS://CREATIVECOMMONS.ORG/LICENSES/BY-ND/4.0/. 226
FIGURE 135 STRUCTURE OF A GENERAL LEARNING AGENT (RUSSELL AND NORVIG 2021, 56). REPRODUCED BY PERMISSION OF PEARSON EDUCATION, INC. 228
FIGURE 136 A: ARCHITECTURE OF VGG-16; B: EVERY COLUMN CONTAINS EXAMPLES OF INPUT IMAGES THAT ACTIVATE TWO FEATURES OF DIFFERENT LAYERS OF THE NETWORK (INDICATED BY THE DOTTED LINE). BY AUTHOR, WITH CODE BASED ON (OZBULAK 2017). 232
FIGURE 137 A: APPLYING DEEP DREAM TO AN IMAGE OF THE CATHEDRAL OF BRASILIA. IMAGE BY AUTHOR, WITH CODE BASED ON (OZBULAK 2017). B-D: CUSTOM STYLE TRANSFER (BASED ON THE CODE FROM JACQ N.D.). B: STYLE IMAGE USED AND REPRODUCED WITH PERMISSION OF KATE MCLEAN. C: CONTENT IMAGE BY AUTHOR. D: CONTENT IMAGE INCORPORATING STYLE. BY AUTHOR. 233
FIGURE 138 DEEPCLOUD. A: ARCHITECTURE OF AUTOENCODER TRAINED ON A DATASET OF POINT CLOUDS OF CHAIRS. B: MONSTROUS CHAIR SERIES GENERATED BY EXPLORING THE PARAMETERS OF THE LATENT SPACE AS INPUT. BY AUTHOR. 234
FIGURE 139 URBAN STRUCTURE SYNTHESIZER: 3D MODELS (A) AND 2D DIAGRAMS (B) WITH INCREMENTAL VARIATIONS OF A URBAN FABRIC. THE DIAGRAMS WERE GENERATED WITH A MODEL (WGAN-GP) TRAINED ON A DATASET OF IMAGES FROM PITTSBURGH, USA. REPRODUCED WITH PERMISSION OF JINMO RHEE. WORK PRODUCED FOR GENERATIVE MODELS FOR DESIGN (CMU, FALL 2019) AND PUBLISHED (RHEE AND VELOSO 2021). 235
FIGURE 140 TIME-LAPSE SEQUENCES OF PLANAR WALKER, QUADRUPED, AND HUMANOID. SOURCE: FIGURES 4, 5, AND 7 IN (HEESS ET AL. 2017, 6–8). REPRODUCED WITH PERMISSION OF DEEPMIND TECHNOLOGIES LIMITED. 241
FIGURE 141 THE STEP-BY-STEP PROCEDURE OF 3D SHAPE MODELING. THE FIRST ROW OF EACH SUB-FIGURE SHOWS THE PRIM-AGENT APPROXIMATING THE TARGET SHAPE BY OPERATING THE PRIMITIVES (STEPS 5, 10, 20, 40, 60, 80, 100, 200, AND 300). THE SECOND ROW SHOWS THE MESH-AGENT APPROXIMATING THE TARGET BY MESH EDITING (STEPS 10, 20, 30, 40, 50, 60, 70, 80, 90, AND 100). SOURCE: FIGURE 7 IN (LIN ET AL. 2020, 11). REPRODUCED BY PERMISSION FROM SPRINGER NATURE: SPRINGER COMPUTER VISION – ECCV 2020 MODELING 3D SHAPES BY REINFORCEMENT LEARNING, LIN ET AL. © 2020 SPRINGER NATURE SWITZERLAND AG (2020). 242
FIGURE 142 CLASSIFICATION OF RL, DRL, AND ADJACENT METHODS BASED ON THE FUNCTION LEARNED/USED. BY AUTHOR, INSPIRED BY (AI 2018; GRAESSER AND KENG 2019, 11; CAUSEVIC 2020; MORALES 2020, 26). 243
FIGURE 143 TEMPLATE DIAGRAM WITH THE ANATOMY OF RL ALGORITHMS. DEVELOPED BY THE AUTHOR BASED ON (INSPIRED BY (SUTTON AND BARTO 2018B, 162). BY AUTHOR. 244
FIGURE 144 DIAGRAM OF DYNAMIC PROGRAMMING. BY AUTHOR. 244
FIGURE 145 DIAGRAM OF MODEL-FREE AND VALUE-BASED RL. BY AUTHOR. 246
FIGURE 146 DIAGRAM OF VANILLA POLICY GRADIENT. BY AUTHOR. 249
FIGURE 147 DIAGRAM OF ADVANTAGE ACTOR-CRITIC ALGORITHMS. BY AUTHOR. 252
FIGURE 148 DIAGRAM OF BLACK-BOX POLICY OPTIMIZATION. BY AUTHOR. 254
FIGURE 149 DIAGRAM OF DYNA-Q. BY AUTHOR. 257
FIGURE 150 DIAGRAM OF MCTS. BY AUTHOR. 259
FIGURE 151 THE AGENT-ENVIRONMENT INTERACTION IN THE CASE OF A DEC-POMDP. BY AUTHOR. 262
FIGURE 152 THE AGENT-ENVIRONMENT INTERACTION IN REINFORCEMENT LEARNING IN THE CASE OF A POSG. BY AUTHOR. 262

FIGURE 153 AGENTS TRAINED FOR ADJACENCY AND AREA GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 265
FIGURE 154 PERFORMANCE OF P1 WITH POLICY ADJACENCY AND AREA. ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 266
FIGURE 155 AGENTS TRAINED FOR ADJACENCY, AREA, AND DAYLIT (75%) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 267
FIGURE 156 PERFORMANCE OF P1 WITH POLICY FOR ADJACENCY, AREA, AND DAYLIT (75%). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 268
FIGURE 157 AGENTS TRAINED FOR ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (75%) GOALS. COL 1: SIMULATION OF 18 AGENTS WITH RANDOM INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES COL 2: SIMULATION OF 18 AGENTS WITH SPRING LAYOUT INITIALIZATION IN A LARGE TERRAIN WITH OBSTACLES. BY AUTHOR. 269
FIGURE 158 PERFORMANCE OF P1 WITH POLICY FOR ADJACENCY, AREA, NOT FOLDING (5), AND DAYLIT (75%). ROW 1: SINGLE EPISODE AND RANDOM INITIALIZATION. ROW 2: SINGLE EPISODE AND SPRING LAYOUT INITIALIZATION. ROW 3: AVERAGE OF 50 EPISODES WITH RANDOM INITIALIZATION. ROW 4 AVERAGE OF 50 EPISODES WITH SPRING LAYOUT INITIALIZATION. THE AREAS BETWEEN DOTTED LINES INDICATE THE PERIODS WHERE ACTION WAS SELECTED RANDOMLY (PERTURBATION). THE MEANS (M) AND STANDARD DEVIATIONS (Σ) CONSIDER THE PERIODS FOLLOWING THE POLICY. BY AUTHOR. 270

0. Introduction

0.1. Research topic

In the last decade, there has been a resurgent interest in spatial synthesis both in academia and in the architecture, engineering, and construction (AEC) industry (Carta 2021, 6–7). By the term spatial synthesis, I refer to computational methods for the generation of building elements such as partitions, zones, rooms, and circulations, to address programmatic requirements, environmental and site conditions, spatial relationships and order, or any other relevant design factor. Among the many design challenges and opportunities at stake, this research focuses on providing computational support for situated cognition and play with design material. Primarily, the research addresses **the design and development of a multi-agent spatial synthesis (MASS) method that supports designers in the generative loop with fine-grained interaction**. It investigates the practical implications of using interactive simulations with agents to allow designers to study, test, and subvert spatial hypotheses.

0.2. Research motivation

Historically, generative design for spatial synthesis relies on classic techniques of problem-solving, which have been originally developed for areas where an objective rationality is often adopted to interpret and operationalize design, such as engineering and artificial intelligence. In this view of design, the conceptual knowledge about the spatial configurations to be designed, the consequences of the decision making, and often the forms of evaluation of the solutions can be described as collections of facts and rules. Therefore, it becomes possible to decompose design problems into parts and the design synthesis can be solved with stronger computational methods, which can explore the space of alternatives before generating the spatial solutions. This is typically achieved with search algorithms and design optimization, which can look for good solutions or optimal solutions according to pre-existing knowledge.

One of the consequences of this anticipatory approach is that it generally relies on an indirect design method where the model is responsible for the strict generation of design alternatives, while designers can select the preconditions, such as goals or input parameters to guide the generation. Then, designers have access to one or more fully formed design alternatives in the output in a pre-defined design space, which can potentially be visualized and selected using catalogues and dashboards. If the alternatives are not satisfactory, the input can be changed, establishing an interaction based on turn-taking, which is a common approach in configurators and design tools for AEC. In this meta-design, designers can potentially learn about the relationships between design parameters, configuration, and performance to restructure the problem and extend design exploration.

This form of design exploration is appropriate for cases where the knowledge to support design analysis, evaluation, and synthesis is not only available but is also robust with the respect to the designers' intentions. However, anticipating the problem structure, the design rationale, the decision process, and the consequences of contemplated action can hamper design exploration

and creative tasks. The concept of **situated cognition** captures this problem by challenging the idea that intelligent behavior can be modelled as a collection of facts and rules. Instead, it suggests that intelligent behavior is an activity where feedback and emergence are paramount, because thinking, perception, and action are coupled in the circumstance of the task (Clancey 1997). This view captures the contingencies of material and social circumstances that enable adaptation and *ad hoc* responses to the action of others involved in a certain task (Suchman 2006).

Situated cognition also supports a critique of the model of problem solving and its emphasis on anticipating and explicitly describing facts and rules in architectural design. Over the last decades the field of design theory has emphasized the importance of design feedback and of coupling thinking, perception, and action in design practices. For example, in the conversational model of design, design problem and solution co-evolve during a cyclic conversation between architects and the design media, such as drawings or physical models (Anderson 1966; Schön 1983; 1992; Lawson 2005; Gänshirt 2007).

This research intends to bridge the gap between these two views of design in spatial synthesis: problem solving, which relies on the development of rational design methods that are amenable for computation, and conversational or situated design, which understands design as an open-ended and intuitive activity based on coupling and intense feedback with the design situation. This research is motivated by a desired scenario where designers can use generative design as a platform for creative conversations. In this scenario, generative models would enable designers to interact, collaborate, and adapt dynamically to changing circumstances to investigate design hypotheses and emerging perspective.

0.3. Research aim

Motivated by ideas of situated cognition, this research aims at investigating generative models that can subvert the indirectness and rigidity of the decision structure of classic models, and benefits both from direct human collaboration and from the state of art in artificial intelligence. It particularly aims at **developing methods that allow the designers to operate in the generative loop with fine-grained interaction**.

We use the term **designers in the generative loop** to refer to a form of generative process where designers share the same simulation with computational agents, which allow them to access the design information with low cognitive loads, enabling human sensibilities and values to provide guidance to the generation. In this interactive simulation, the design material is not a set of solutions generated by the algorithm or the information about their performance but the design in formation itself. This contrast with the idea of turn taking in classic models, where designers have indirect control of a generative process that happens in a separate turn.

The research also relies on the concept of **fine-grained interaction**, which not only comprehends increasing the speed of the feedback between human and generative model but also providing more granular control to enable humans and agents to execute a shared task jointly (Tran Luciani, Löwgren, and Lundberg 2019, 685–86). The idea is that human-computer interaction should

happen at small steps of formation of the design alternatives with intense percept and control feedback to blur the boundaries between analysis and synthesis.

0.4. Agent-based computing for research

This research focuses on the development of a spatial synthesis framework and artifact that integrate situated elements of design creativity into more open-ended and collaborative generative methods. For that, this research relies on two computational paradigms based on agents: agent-based modeling and multi-agent deep reinforcement learning.

Agent-based modeling relies on the behavior of autonomous computational entities – agents – that interact in a shared environment, over time, producing a certain collective behavior or organizational pattern. Agent-based modeling is a computational paradigm that supports interactive simulations with human in the loop and fine-grained control. Besides, it also addresses phenomena that are related to more open-ended aspects of design, such as pattern formation, self-organization, and emergence.

Typically, agent-based models are developed to represent the behavior of real-world phenomena, such as fluid flow, pattern formation, reaction diffusion of chemicals, physics, urban dynamics, and social interaction. Therefore, agents-based models can be adapted from other areas, such as complex systems, to architectural problems. There are interesting examples of the application of agent-based models to speculative design (C. M. Herr and Ford 2015), design inspiration (Gausa et al. 2003; Allen 2009; Gausa 2010), building occupancy, or urban design (Koenig 2011). However, while agent-based models from other fields have been successfully adopted in these formal, architectural, and urban explorations, it is not trivial to import and adapt them to address the specific constraints in spatial synthesis. Due to the lack of an approach to design custom agents, generative methods based on agent-based models have historically struggled in multi-objective and constrained problems, such as spatial synthesis in the scale of building design.

The second paradigm, multi agent deep reinforcement learning, which has been one of the main areas of deep learning to spearhead the latest wave in artificial intelligence (Russell and Norvig 2021, 24–27). In deep reinforcement learning, agents are trained to maximize their performance with the respect to a feedback signal (rewards) in a sequential decision process. In this approach, instead of adapting and fine-tuning pre-existing agent-based models, designers can create spatial goal based on reward signals that support the training of proper agent behaviors as part of an ecology of agents. Then, the resulting agents can be deployed in an interactive simulation with designers.

0.5. Research questions

This research uses the ill-structured and qualitative nature of situated design to investigate future interactive generative design practices. It addresses the problem of multi-agent spatial synthesis (MASS) and situated cognition in design exploration by envisioning new forms of generative models. The main research question is: can agent-based modeling and current deep reinforcement learning techniques support the design of custom agents for a spatial synthesis

based on fine-grained interaction? A secondary question that is partially addressed by this research is: can the proposed forms of interaction be integrated to a generative workflow centered on human experience?

0.6. Research method

This dissertation addresses the dual goal of providing methodological insights on the problem of spatial synthesis with designers in the loop and on designing, developing, and evaluating an innovative design artifact that contributes to its solution. It establishes an arc that begins with the motivation to explore spatial synthesis with designers in the loop and culminates with an innovative design artifact that extends and enhances this reality.

Due to the design-theoretical motivation, the exploratory part of the research evaluates the rationale and technology of current generative systems available using taxonomy building and integrative literature review. Taxonomy building as adopted in the dissertation is a process of organizing the knowledge on generative models for architectural design and understanding their underlying ideas and paradigms. The taxonomy is built based on a review of existing categories and taxonomies. Instead of focusing on specific instances, applications, and algorithms, it provides a consistent terminology and a deeper understanding about the relationship between the different methods and components of a generative model and their implications for design interaction. Then, an integrative literature review assesses the current state of the specific area of research: multi-agent spatial synthesis.

To explore new perspectives for the research problem of interactive spatial synthesis with agent-based models, the constructive part of the research comprehends designing, implementing, and evaluating an original design artifact.

First, we build a spatial synthesis artifact from the ground up, which intends to prove the feasibility of using agent-based modeling and multi-agent deep reinforcement learning to design custom spatial agents that have good performance in a spatial synthesis problem and enable finer-granular control in the generative loop. Different iterations of the artifact are evaluated quantitatively and qualitatively with the respect to the training process and to the performance of the agents in design cases.

After the evaluation of the artifact, the research incorporates its interactive simulation to event-driven computing using a game engine. The idea here is to create a design tool that enables designers to restructure the design problem and guide the design exploration to preferable or unexpected alternatives in the generative loop with situated actions and fine-granular control.

It is important to clarify that while our method shares qualities that are common both in computer science, engineering, and architectural design research, it is customized to the specific inquiry described in this introduction. By proposing a spatial synthesis formulation and method from the ground up, this research has a larger set of tasks to address, including the creative re-formulation of spatial synthesis, so it does not benefit from existing task environments, which are common in

computer science and engineering research. In terms of architecture, the construction and evaluation of the design artifact incorporate questions related both to its efficiency and to the quality of the resulting spatial types. However, the formulation is still constrained by certain abstractions that are related to the development of a research artifact and not to specific design practices. Over the time, as the method becomes more developed and stable, I expect that its properties can be tailored to specific design pedagogies and creative practices based on the preferences of the architects and stakeholders.

0.7. Research objectives

I pursued the following objectives during my research:

- Reviewing the existing generative models for architectural design, including classic, agent-based methods, and deep learning methods, including reinforcement learning
- Identifying the dominant approaches to interaction in classic generative models
- Reviewing the existing research prototypes that use agent-based and similar models for spatial synthesis and identify the specific gaps
- Incorporating learning into a multi-agent spatial synthesis
- Identifying the available deep reinforcement learning methods to train spatial agents
- Formalizing the multi-agent spatial synthesis problem with simultaneous spatial goals, such as adjacency, shape, area, and daylight access
- Training and evaluating custom agents for spatial synthesis using reinforcement learning techniques
- Implementing a design artifact that integrates custom spatial agents and a parametric model in an interactive environment
- Exploring different interaction mechanisms that are compatible with the multi-agent model
- Discussing the exploration of possible spatial configurations based on underlying agent-based constraints and policies
- Discussing how the interactive simulation workflow instantiates a specific view of spatial synthesis that can support collaborative workflows with artificial intelligence

0.8. Impact of the research

The first contribution of this research is a conceptual framework for multi-agent spatial synthesis with designer in the loop based on fine-grained interactions. This framework is substantiated by a design artifact and an interactive simulation workflow that integrate parametric modeling, agent-based modeling, and multi-agent deep reinforcement learning to a game engine. This artifact and

workflow enable designers to interact with AI agents using fine-grained control mechanisms to build a design configuration incrementally. To my knowledge, both the initial version of the prototype (Veloso and Krishnamurti 2020; 2021b) and the final ones are an original contribution to the problem of agent design for spatial synthesis with designers in the loop. The main innovations in the prototype are related to the use of multi-agent reinforcement learning to

- to generate heterogeneous agent behaviors based on parameter sharing
- address multiple spatial synthesis goals and support domain knowledge using a parameterized reward function
- use training with randomized environments and problem specifications to incentivize better policy generalization and reduce the dependence on pre-defined settings
- support event-driven computing, real-time and fine-grained interaction
- the use of granular actions to learn higher-level behaviors and spatial compositions
- the integration of the proposed model with parametric modeling and a game engine to support a fully interactive environment

By providing a deep learning approach that solves the problem of custom agent design, this research not only makes multi-agent spatial synthesis a viable design method but also

- promotes the importance of fine-grained interactions in generative design workflows for creative research practices
- situates designers inside an interactive simulation with agents as one possible answer for problems of design ethics, transparency, and authorship in automated methods
- incentivizes the incorporation of principles from agent-based computing to architectural morphology and design practice

There are also secondary contributions to design education and architectural design.

The proposed framework subsumes some areas of rule-based modeling in design. The policy of an agent is in essence a function that decides what action to take, given a certain observation of the current state. Therefore, combining policy learning with granular actions is a form to infer sequences of actions that result in high-level state transformations that leads to large cumulative rewards (in reinforcement learning) or to other forms of objectives, such as coherence with recorded behaviors (in imitation learning). In other words, the proposed framework can be interpreted as a method to create rule sets and grammars for high dimensional design states.

In terms of generative design education, the lack of consistency and rigor in the treatment of underlying ideas and terms is a barrier for architectural education. The current knowledge on generative models is fragmentary and the techniques to develop custom generative models are not systematized, so its incorporation in design education is still challenging. Therefore, the

taxonomy of generative modeling developed over the dissertation contributes to the systematization of existing generative models and to development of new ones. I introduced this taxonomy in computational design courses at Carnegie Mellon University and it resulted in insightful feedback and interesting projects – some of these projects are displayed in this dissertation as examples.

0.9. Summary of chapters

The structure of the dissertation follows the pattern of the hourglass.

Chapter 1 starts with a broad and theoretical interpretation of ideas and configurations of generative models and provides the theoretical elements to narrow it down to the scope of the research problem. First, it introduces basic definitions of generative model and spatial synthesis and contextualize them in the scope of this research. Then, it starts building a taxonomy of generative models with paradigms consolidated in architecture, such as parametric modeling, rule-based modeling, search/composition, and design optimization for architectural design. Based on these categories, the chapter problematizes the "indirectness" of the design generation with optimization and composition and identifies the opportunity for novel approaches.

To address the indirectness of design interaction and set the scope of the research, the chapter establishes an underlying structure in the form of a research guideline for situated and interactive spatial synthesis. This guideline for interactive spatial synthesis is based on a combination of elements such as a parameter control, event-driven computing, real-time and fine-grained control, and reduced assumptions about the problem structure and resolution.

The first response to the guideline is the investigation of agent-based computing in the context of generative modeling, which results in the identification of research opportunities and challenges. While agent-based computing satisfies most of the elements of the guideline, it lacks a general methodology for agent design that can address spatial synthesis problems. The second computational paradigm, multi-agent reinforcement learning, provides computational methods to solve the problem of agent design and to enable the development of a design artifact. At the end of the chapter, the taxonomy is updated with additional categories, such as agent-based model, simulation, generative learning, and behavioral learning, which works as a map to situate the approach proposed in this research.

The next chapters are the core of the applied research, and they cover the development of the design artifact for spatial synthesis with fine-granular interaction.

Chapter 2 describes the formulation and implementation of the design artifact, which is based on designing, training, and deploying an ecology of heterogeneous spatial agents to build spatial layouts. The chapter introduces the design decisions in terms of agent and environment representation, which are based on the discretization of the information into grids. It presents the parameters and the functions used to model the environment, such as a transition function, and a parameterized reward function based on multiple spatial objectives. Then, the chapter

introduces a series of three prototypes, with specification of their reinforcement learning algorithms, training settings, and results.

Chapter 3 is a proof by construction of the feasibility of research framework. It evaluates the capacity of the design artifact to support step-by-step formation of valid designs in face of adversity, such as site conditions and design perturbations. It evaluates the different versions of the design artifact using three design cases: a single-family house in three different sites, a single-story complex of art galleries located in a site between urban lots with small buildings and an existing park, and a complex of eight houses to be placed on a large and empty site. Besides presenting the performance of the artifact under different conditions, the chapter also addresses the different configurations that are generated based on variations of the reward function.

Chapter 4 introduces the integration of the design artifact to an interactive simulation workflow comprehending parametric modeling and a game engine. The chapter describes the general workflow, the geometric post-processing of the agent states in a parametric editor, and the proposed interface, tools, and commands in the game engine. This workflow is a proof of the viability of the design artifact to support spatial synthesis with designers in an interactive simulation with agents.

Chapter 5 contains a general discussion about three main aspects of the research and its implication for future research. First, it addresses technical aspects of the design artifact that will be developed in the future to tackle more complex and larger problems. This includes extending the forms and parameters of agent representation, enabling variations of spatial objectives to increase specialization and clustering of agents, and improving the training workflow. Secondly, it comprehends the capacity of the artifact to explore design alternatives with coherent design constraints and order. This leads to two future research challenges: inferring agent behaviors directly from design examples and developing an approach to capture top-down compositional properties with an agent-based model.

Then, the chapter emphasizes the capacity of the proposed model to support designers inside the loop sharing a fine-grained and local control with computational agents. This is compared with other interactive models of mainstream generative models, such as design optimization and generative learning, which typically rely on some form of turn taking. Considering that the proposed interactive simulation workflow succeeded in satisfying the properties of the research framework, it is considered a starting point for future research into situational design behavior with designers and computational agents.

The chapter concludes by revisiting the trajectory of the research and the future lines of research.

1. In pursuit of agent-based spatial synthesis for situational design

1.1. Introduction

This research explores a novel computational approach to spatial synthesis that allow the designers to be in the generative loop with decentralized design control and real-time and fine-grained interaction. This chapter is divided in the following parts:

- Defining generative model and spatial synthesis
- Classic Generative Models
- A guideline for spatial synthesis with designers in the generative loop
- Agent-based computing in generative design
- Training agents for spatial synthesis

The first two parts identify fundamental definitions and relationships underlying the concepts of artificial intelligence, generative model, and spatial synthesis. Then, the chapter discusses classic generative models for spatial synthesis – i.e., models that rely on computational paradigms from Artificial Intelligence (AI) and related fields. It introduces a taxonomy that describes the underlying computational ideas and the current modes of human-computer interaction in classic generative models.

The third introduces a computational guideline to address situated design methods that allows designers to use the affordances of the media to iteratively test design hypotheses, to interpret, frame and reframe the problem itself according to frictions and emerging perspectives. The following two parts address two paradigms adopted in this dissertation to substantiate this guideline into a design artifact. The first extends the taxonomy of generative model with agent-based computing and discusses the benefits of using agent-based generative models in our design artifact and exposes the current challenges, such as agent design. The second extends the taxonomy with the notion of learning, then introduces multi-agent deep reinforcement learning as a method to train the agents in our design artifact.

1.2. Defining generative model and spatial synthesis

1.2.1. From ideation to generative models

Natural sciences are concerned with how things are; design, on the other hand, is concerned with how things ought to be (H. A. Simon 1996, 114). Likewise, their respective models resonate this distinction between analysis and synthesis. Models in natural sciences are instruments to investigate hypothetical or existing phenomena and are generally based on symbolic and mathematical concepts. In contrast, architects traditionally rely on different types of models to produce building configurations to respond not only to the required goals and constraints of the project but also to spatialize their design principles and personal values. These models encompass a vast range visual media, such as in sketches or physical prototypes (which can be considered as

models as well) to express relevant characteristics of an as-yet-unknown spatial construct.

Under the actions of designers, the conventions and structure of the design model are used both to represent their ideas about the desired building artifacts and to enable imaginative exploration of novel alternatives and updates. They modify and refine models that partially represent a spatial construct that is under construction with the models themselves (Figure 1: A), i.e., an architectural representation precedes and actualizes the referent (Corona-Martínez 2003, 25–40; Picon 2004, 115). The progressive actions over the model provide feedbacks for the designers, which can update the internal representations in their minds. This relationship between the ideas of the designers and the architectural models mediated by action and perception is so crucial for ideation that design theorists refer to it as a form of cyclic reasoning or design conversation (Anderson 1966; Schön 1983; Corona-Martínez 2003; Lawson 2005; Gänshirt 2007; Cross 2006). Particularly in the early stages of design, the "conversation" with partial representations is important for acquiring knowledge, creating a frame of reference, and reformulating the problem and solution in parallel (Cross 2006). In this process, "(…) the designer constructs the design world within which he/she sets the dimensions of his/her problem space, and invents the moves by which he/she attempts to find solutions" (Schön 1992, 11).

Notwithstanding being a well-established practice, the direct, creative, and intuitive engagement with visual media as the architectural model is not the only alternative for design conception with models. Instead of using static models for visual and tactile feedback, designers employ models that embed some form of decision-making, such as instructions or behaviors, to generate design alternatives, which is the essence of generative models[1]. Designing with generative models – i.e., generative design – is typically an indirect methodology where designer externalize specific design rationales.

[1] The terms generative system and generative model are common in the literature. In fields such as systems engineering and cybernetics (Ashby 1956), system is used to comprehend the studied phenomenon as a cohesive organization composed of interacting parts that together produces results not obtainable by the parts in isolation. The term model refers to the representation of a system. I opted for the term generative model for three reasons: (1) it describes representations of concrete generative systems in the taxonomy presented in this chapter; (2) it can be considered an instance of architectural models used for design conception; (3) it also unifies the nomenclature of the diverse areas addressed in the chapter, such as mathematical models, geometric modeling, parametric modeling, agent-based models, rule-based models, etc.

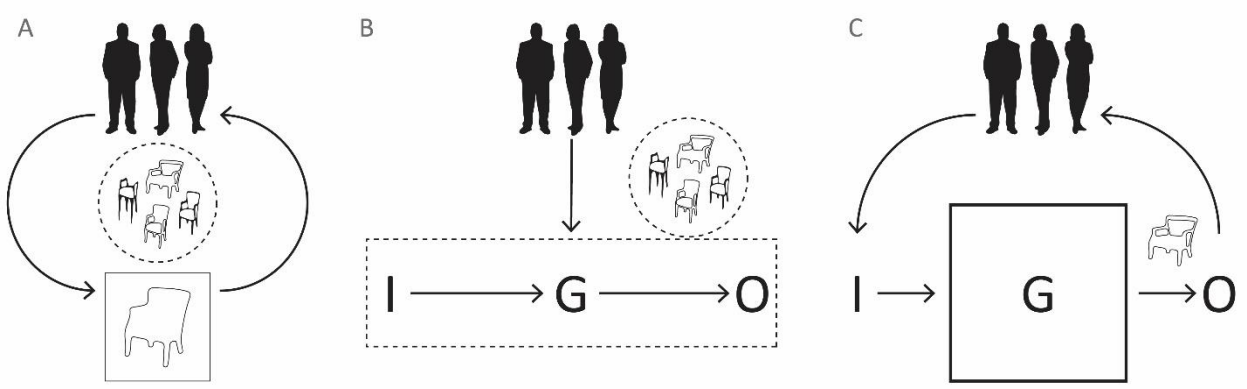


Figure 1 A: design ideation: the direct interaction of designers with visual media to build an imaginary design world. B: formulation of a generative model: the definition of input and generator implies the existence of a design space to navigate and look for output design alternatives; (G). C: execution of a generative model: designers control an input (I), the generator (G) synthesizes a design alternative as an output (O), which is observed by the designers. By Author.

Our basic representation[2] of generative design has two stages – formulation and execution –, and the generative model has three main components: input (I), generator (G), and output (O).

In the formulation stage (Figure 1: B), the designer develops the representations for the input (I), for the output (O), and the instructions to be encoded in the generator (G). This generator is usually an algorithm that specifies how to synthesize the output design alternatives, conditioned on the input information. In contrast to conventional design ideation, where designers imagine design worlds by interacting with design media, the configuration of the generative model induces the space of possible designs (design space) and the ways that the designer can use the input operations and data to navigate between solutions (design navigation). Thus, in the execution stage (Figure 1: C), the design space embedded in the model and the mechanisms for navigation are fundamental for the designer to explore and learn with the design alternatives. Designers interact with the generator by controlling the input information and observing design alternatives as feedback in the output of the model. They can either exploit a subspace of solutions that satisfy certain goals or explore unexpected alternatives in the design space.

Generative models are not restricted to the use of computers. Currently, parametric modeler, algorithmic editors, plug-ins, and scripting are available to design education and can support the description and automation of generative tasks using symbolic operations, algorithms, and data structures. However, while computational generative models are the main topic of this dissertation, it is important to clarify that creating a generative model does not require using computational logic and the use of computer does not imply the use of generative models. Generative models can (retrospectively) include existing methods, such as form-finding models, like pneumatic, soap bubbles, soap-film, sand, and suspended chain models (Otto 2003; Vrachliotis

[2] While our representation is unique and represents our approach to the topic, it certainly has precedents in the literature, such as (Fischer and Herr 2001; Cagan et al. 2005, 172; Gänshirt 2007, 78–79; Bohnacker et al. 2012, 461; Veloso and Krishnamurti 2019).

et al. 2017), or the procedures to generate building elements and configurations from architectural treatises. In the case of form-finding models, the step of formulation requires constraining existing natural phenomena to make their behavior work as an algorithm for form exploration. In the case of a written procedure to generate a building element, the execution stage depends on the designer.

Nevertheless, the abstraction and generality of numerical and symbolic computation enable the control over all the elements and stages of a generative model and can subsume most existing generative models. With computers, it is not only possible to explicitly model geometry and building information – as in CAD and BIM software –, but also to simulate form-finding models, to automate the generation of a building types, and to create custom generative models. Generative models are constructs situated between design and computational logic. Their logic can be inspired by different sources, such as human cognition, expert knowledge, brain architecture, social organisms, natural processes, and dynamics. They are a powerful and compelling alternative for design that benefits both from the rigor of scientific modeling and the open-endedness of the design artifacts.

1.2.2. From architectural composition to spatial synthesis

Historically, architecture composition comprehends both to method and result of organizing spatial elements such as partitions, zones, rooms, and circulation, and relations, according to some ordering logic. This spatial order used in a composition relies both on the satisfaction of factors that are external and internal to the designers and to the discipline of architecture[3]. The former are the pressures over an architectural configuration derived from outside of the designers' will, such as the structural, environmental, and programmatic requirements or constraints of a given project.

While these external factors are necessary, they are not sufficient for determining good architectural configurations. In contrast to fields of engineering where empirical facts and objective factors tend to be dominant, the reliance of designers on principles shared by architects in publications (Lucan 2012) or on personal and collective disciplinary constructs is a common place in design (Colquhoun 1969). Architects work with intuition, personal preferences, and local paradigms that are internal to the discipline to create custom spatial configurations, which can include shared precedents, aesthetic codes, visual language, design principles, methods, and rules.

In a composition each part of the design is determined by its own spatial configuration, by connections with other spaces, and by the functions they have in relation to the whole. In the classical sense, the practice of composition relies on a repertoire of clearly spatial figures and elements, and uses academic principles of regularity, axiality, symmetry and hierarchy, to ensure a closed and organic unit of the whole. However, due to changes both in the external pressures

[3] For more details on spatial orders derived from the field of architectural design and theory, see **7.1 Appendix A: Notes on architectural composition**.

and internal changes in architectural paradigms, the design of architectural configuration has been under constant redefinitions in the last century. Many design strategies, such as free-plan, transparencies, collages, montages, formal operations, and different forms of diagramming have been adopted by designers to disrupt the classical discipline of composition in the design of contemporary buildings (Figure 2).

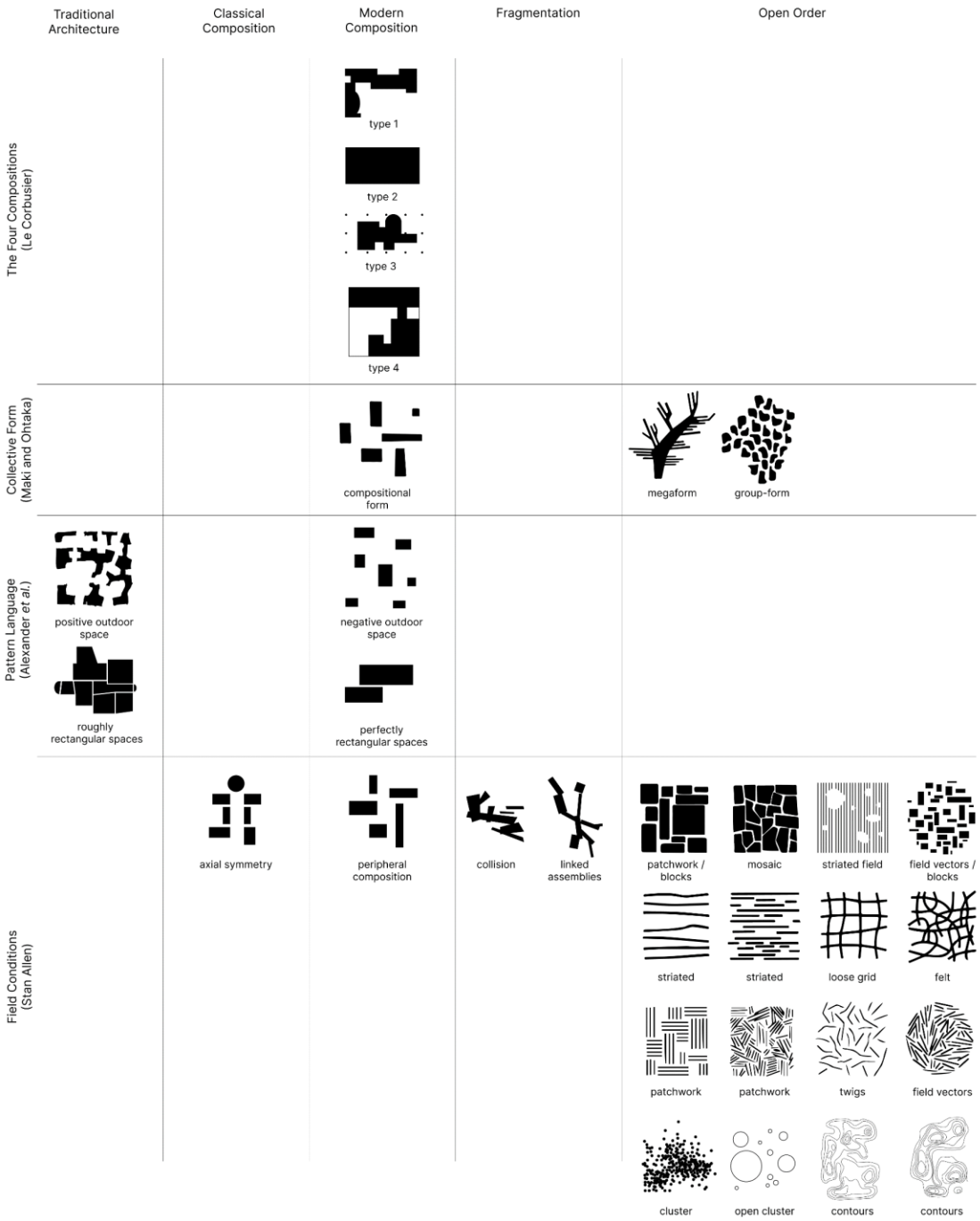


Figure 2 Matrix of different compositional and non-compositional orders. By the Author with diagrams re-drawn from (Lucan 2012, 402, 476; Alexander et al. 1977, 518, 883, and 888; Allen 2009, 192, 218; 2012b, 98)

In parallel to the development of these design strategies, the automatic generation of architectural layouts was one of the main areas of early computational design research (W. Mitchell 1977; C. N. Eastman 1975). This research area was largely informed by ideas from artificial intelligence and operations research, as described in 7.2 Appendix B: The influence of classic AI methods on generative models. Overall, early research in the field emphasized a systematic approach to design based on problem-solving, which could benefit from the use of computers. Algorithms and data-structures were used to perform different design tasks, requiring the externalization of the instructions for design. Due to the multi-disciplinary nature of the problem, its proponents adopted not only different approaches but also a variety of names. It comprehended a diversity of methods and a vast terminology, including terms such as automated spatial synthesis, space planning, facility layout, autonomous layout design, automated layout, automated floor plan generation, space allocation, etc. In this research I will adopt the term spatial synthesis. These are some of the definitions of the problem of spatial synthesis (Table 1).

Table 1 Definitions of spatial synthesis problem

Author	Definition
C. M. Eastman 1970, 242	"Problems involving the arrangement of objects in two- or three-space where the objective function primarily consists of derivatives of the distance between objects or their arrangement are called space planning problems."
Mitchell 1977, 425	"Given a data structure capable of representing a range of building designs, find a state of the data structure (i.e., a particular design solution) such that specified objectives and/or constraints are complied with."
Henrion 1978, 176	"A space-planning problem consists of three parts: A set of elements to be located, a receiving space in which to locate them and a criterion function to evaluate the arrangements."
Rowe 1987, 40	"In architecture and urban design the class [of well-defined problems] would include the *space-planning problem*, in which a set of building spaces is prescribed, together with a site in which they are to be assembled and some expression of adjacency requirements among the spaces. Here, the problem can be formally expressed by letting P represent all possible combinations of building spaces, $s_1, \ldots, s_n$, with a subset S of combinations that satisfy the adjacency requirements, $A(s_i, s_j)$. The objective is to find a satisfactory combination."
Liggett 2000, 197	"Facility layout is concerned with the allocation of activities to space such that a set of criteria (for example, area requirements) are met and/or some objective optimizes (usually some measure of communication cost)."

The essence of spatial synthesis can be reduced to three basic elements:

(1) a form of representation of design alternatives

(2) a set of design requirements with objectives and/or constraints

(3) a solution procedure to generate good solutions in the format of 1 and according to the design criteria of 2.

In other words, given the hypothesis about the spatial problem to be solved, such as its representation, requirements and constraints, researchers could systematically develop a planned sequence of analysis, synthesis, and evaluation steps to generate one or more spatial solutions to support design decisions. In this scenario, the designer provides the specifications for the design and the generative model executes its program and generate one or more solutions, or even try

to enumerate all possible solutions to support future design decisions.

1.3. Classic Generative Models

Despite the large volume of research on generative models, the knowledge is still fragmentary and there is also a lack of consistency and rigor in the treatment of underlying ideas and terms. In this section I organize the ideas about classic generative models using part of a larger taxonomy that was developed both for educational and a research purpose [4]. The goal of introducing this initial taxonomy is to identify how the components of classic generative models result in different types of design spaces and strategies for design navigation, and, therefore, are strongly associated with the possible forms of interaction with designers. The taxonomy was developed from research on existing classifications of classic spatial synthesis techniques and generative design methods (see 7.2 and 7.3.2).

This initial taxonomy (Figure 3) focuses on the essential components of classic generative models and its components are organized in two levels, according to their function: expression and synthesis. Each generative model is represented in the taxonomy as a combination of the building blocks[5]. The vertical and diagonal connections between building blocks in different levels represent the combination of components for models with multiple functions. The horizontal connections represent hybridization of components for the same function. The thickness of the lines indicate affinity. Thus, rule-based models have relatively stronger affinity with composition, even more so than parametric models with optimization.

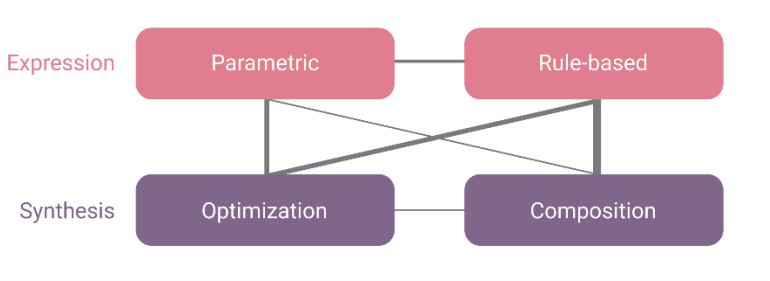


Figure 3 Taxonomy for building blocks of generative models. By Author.

The blocks in the first row are the three common expressions of a generative model: parametric- and rule-based model. These blocks are essential because they define specific formulations for the

[4] The taxonomy was introduced in computational design courses at Carnegie Mellon University, and it resulted in insightful feedback and interesting projects – some of these projects are displayed in this chapter as examples. While the initial version of the taxonomy was published in (Veloso and Krishnamurti 2019), the content on the final taxonomy was organized and published in (Veloso and Krishnamurti 2021a).

[5] While the taxonomy shown here is not exhaustive, it is broad enough to subsume earlier classifications that were based, perhaps, on finer considerations, for example, AI techniques such as knowledge representation, case-based reasoning, and expert systems. Some of the techniques might be a special case of the existing blocks or might fit in between the blocks.

models that result in different types of design spaces and forms of design navigation.

Once the expression of the model has been formalized, it is possible to explore design alternatives with some form of user control. The design representation can be modified, the parameter values of parametric models can be tweaked, rule-based models can be computed by selecting shapes and rules. However, generative models usually incorporate a strategy to navigate in the design space and automatically synthesize design alternatives. The second row contains computational methods for automatic design synthesis, such as optimization[6], and composition. These blocks support the execution stage, where the designers can explore design alternatives by interacting with the design input and outputs.

1.3.1. Parametric model

One of the more popular ways of expressing design synthesis is through parametric modeling, which has precedents in the history of rationalist architecture and in computational modeling (Davis 2013a). Roughly, a parametric model can be defined as an explicit representation of the design logic that can be controlled by independent variables (or input). Mathematically, these variables are named parameters and are associated with equations that map them to a certain quantity in the output (Davis 2013b, 21–22). For example, the explicit representation of the coordinates of a hyperbolic paraboloid parameterized on the straight lines maps the input parameters (u, v, a) to the coordinates of the surface $(u,\ v, auv)$ (Figure 4). The parameter a is of special interest because it can be used to change the height of the hyperbolic paraboloid.

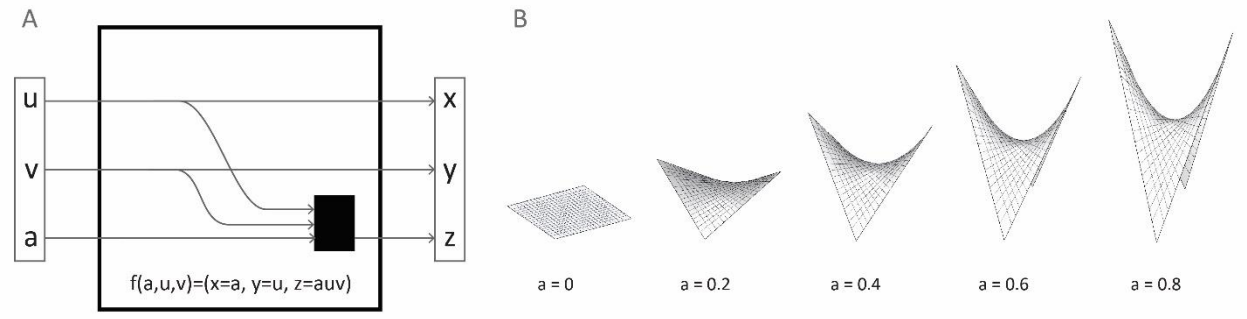


Figure 4 A: function to generate the coordinates of a hyperbolic paraboloid. B: representation of the surfaces in a specific domain of u and v, based on the variation of the parameter a. By Author.

In design, the notion of a parametric function is more general than a mathematical function, because it can also comprehend any type of modeling operation available in design software. Thus, parametric modeling is employed as a method to explore and visualize variations of architectural solutions, based on an algorithm that receives an input, executes finite sequence of well-defined, geometric, symbolic, or even numerical operations and generate the design alternatives as output.

[6] These building blocks are related to a specific aspect of generative modeling, so it is important to distinguish them from the algorithms that they comprehend. For example, the category optimization is mostly based on optimization algorithms for the design synthesis. However, optimization algorithms are very general and can be applied for other ends, such as training neural networks (see 7.6 Appendix F: Deep Learning).

In other words, the generator is typically a fixed structure where the changes in the parameters propagate through all the dependencies, updating the representation.

In the early programming courses for design (W. Mitchell, Liggett, and Kvan 1987; Coates and Thum 1995), this explicit parametric function was developed procedurally by scripting in CAD software to generate a certain shape. Furthermore, the designer could change some of the parameters of this script, re-execute it, and watch the change propagate through the function to update the shape. Today, most current parametric editors for design are propagation-based in which the parametric function is explicitly modeled as an acyclic graph with dependencies, modeling operations and entities (R. Woodbury 2010, 12; Janssen and Stouffs 2015, 158–62). Any change in the input information or in some component of this graph will trigger the re-execution of the function, updating the output.

Parametric modeling enables fast change in a pre-defined range of variation, which relies on the domain of the parameters and on the parametric function. If the parameters are all numerical or each parameter can be depicted on a separate axis, the design space can be represented by a multi-dimensional parameter space. The designer can explore combination of parameters, which are mapped to a geometric solution by the parametric function (if such a solution is possible) (Figure 4). In the cases where the input is geometrical, the variations can also be achieved by conventional modeling operations that affect the input.

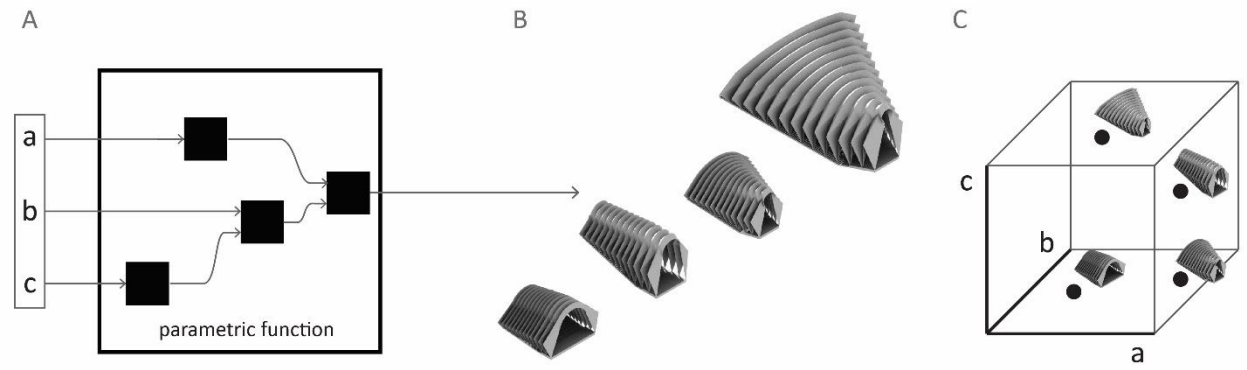


Figure 5 A: diagram of generative model based on the parameters a, b and c and on an internal network of operations. B: variations of the geometry. C: diagram of parameter space of the model, associating the coordinates of the parameters with the resulting designs. By Author with model developed by Adie Alnobani for Parametric Modeling, CMU, Spring 2016.

Different parametric models can capture different design intentions (Figure 6). Owing to a commitment to a parametric structure, the design of a good model requires significant frontloading. The designer should anticipate and accommodate the intended interdependencies, changes, and future explorations. Otherwise, design decisions and changes in the design space will require reconstructing part or the whole parametric function (Davis 2013b, 37–47).

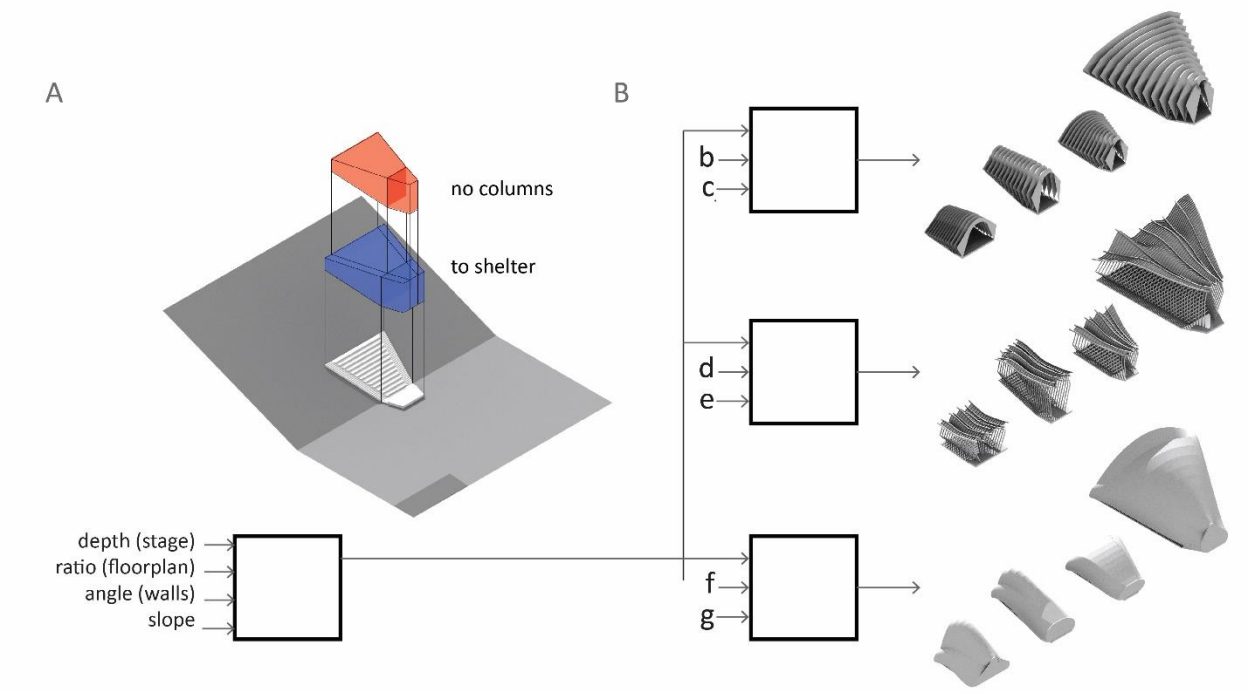


Figure 6 Exercise for Parametric Modeling (CMU, Spring 2016): outdoor theater. A: base parametric model, with specific geometric requirements for the theater (seats, volume with no columns and volume to shelter). B: diagram of three different parametric models to create a structure that responds to the variations of the base model. By author with models developed by Adie Alnobani, Atefeh Goloujeh, and Cecilia Ferrando for Parametric Modeling, CMU, Spring 2016.

1.3.2. Rule-based model

Rule-based modeling encapsulates a kit of part approach. In the example below, it is composed of blocks with design transformations of the design representation. With access to this kit of parts, the generation consists of executing a sequence of rules, which will build a design representation incrementally. These blocks can be represented as a set of instructions or be formalized into productions or if-then rules (see rules x, y, z and w, in Figure 7: A and in Figure 8: A: item 1). In the case of if-then rules, the design transformation maps two elements:

the **if** or left-hand side (lhs) represents the preconditions for the application of the rule.

The **then** or the right-hand side (rhs) depicts the result of the transformation.

Rules are either terminating or non-terminating. Non-terminating rules contain elements that represent preconditions, which enables further derivation of the design. Terminating rules do not have preconditions on the right side, that is, they prevent further changes in the representation. During the execution of a rule-based model, (Figure 8) an initial design representation (usually, the input of our generative model) will be incrementally changed by the rules available in the rule set.

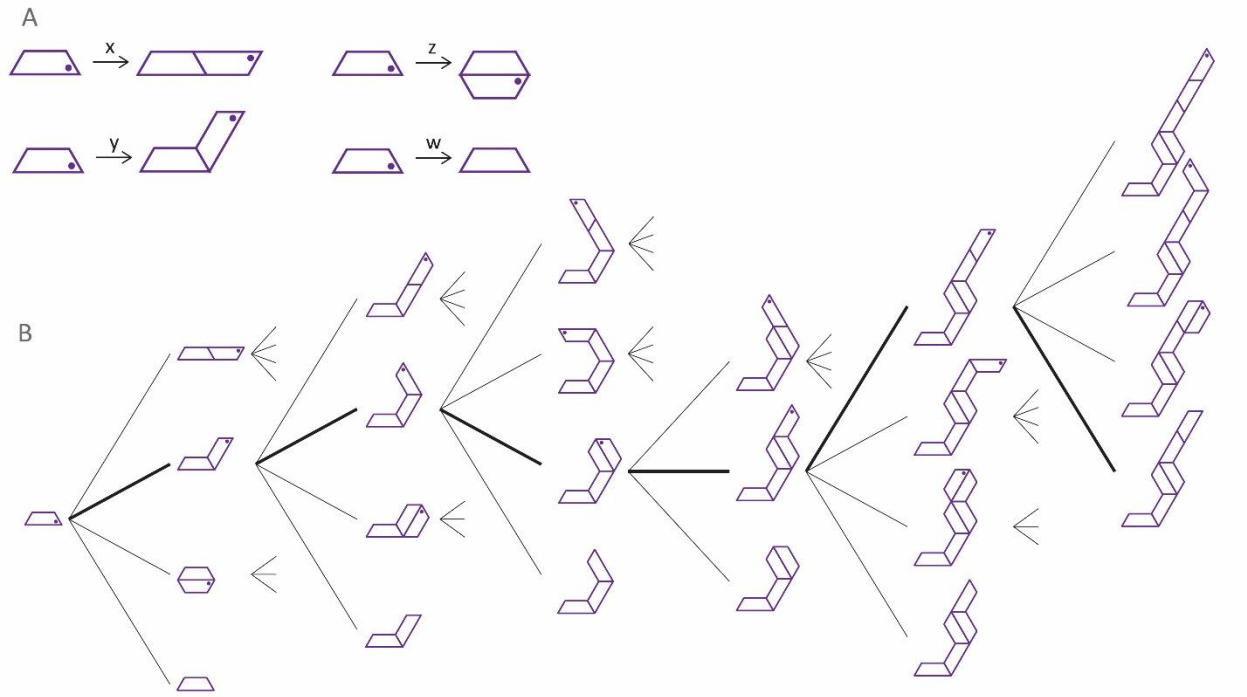


Figure 7 Example based on the half-hexagon grammar (W. Mitchell 1977, 436–39; 1990, 143–47). A: definition of a non-terminating rules (x, y, z) and a terminating rule (w). Notice that the dot represents a label, which restricts the precondition for future derivations. B: partial representation of the design space induced by the rule set. Thicker line represents example of a sequence of derivations. By Author.

Designing with rule-based models is equivalent to selecting the preconditions and rules incrementally – that is to say, the decision process is represented by sequences of rule matching. The designer or the generator interprets the current design state to detect existing preconditions (Figure 8: A). Detecting the preconditions is simple in the case where it is an atomic representation, such as a character, a string, or a line. Moreover, geometric preconditions can be subject to geometric transformations, such as rotation or scaling. However, when the representation is ambiguous, forms and preconditions that are unanticipated during the development of the ruleset can emerge with the derivations. In these cases, interpreting preconditions in the representation can be hard. After discovering preconditions, it is necessary to pick a rule compatible to one of the available preconditions. If there is only one rule application available for every state of the representation, the model is deterministic, and it will generate a single trajectory of changes, as in standard fractals or simple L-systems. However, if multiple preconditions and/or multiple rules can be selected, there is a tree of possible design alternatives formed by the distinct paths of transformations starting from an initial representation (Figure 7: B). The latter is the common case for generative design.

Besides, rule-based modeling must address the tradeoff between control and emergence. If the goal is to explore emergent solutions, the rules are designed to favor an unbounded design space and allow a more unstructured exploration. In the case where the emphasis is on control, the designer tends to adopt rules that guarantee that the design space contains good and fully formed design alternatives are reachable in a reasonable number of derivations.

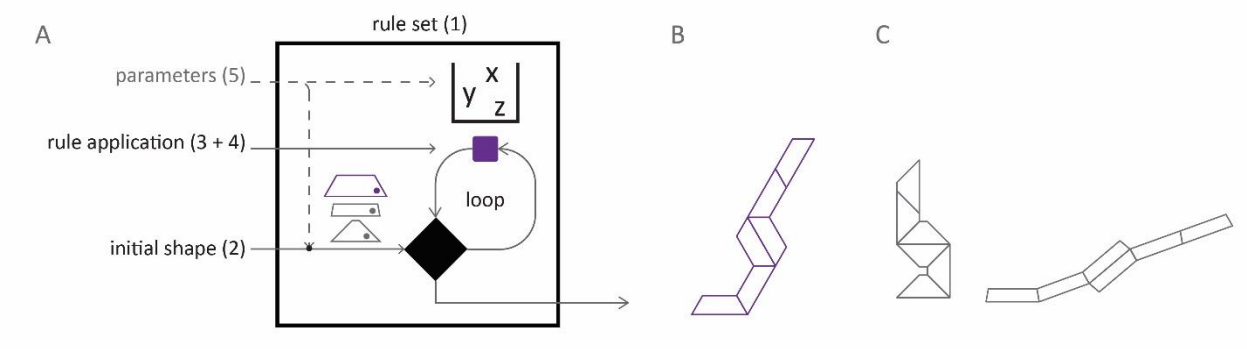


Figure 8 A: diagram of a rule-based generative model using a variation of the half-hexagon table grammar. B: example of resulting design; C: two variations using parameters for angle and width. By Author.

Models with if-then rules have been widely adopted and formalized in different fields, such as logic (Bratko 2011), shape grammars (Stiny 2006), L-systems (Prusinkiewicz and Lindenmayer 2004), and fractals (Mandelbrot 1982). Overall, rule-based modeling covers a vast variety of application areas and as such results in different nomenclature and specificities. The table below characterizes three different formalisms:

Table 2 Different formalisms based on rules (inspired by Gips and Stiny 1980).

	L-Systems	**Shape Grammars**	**Fractals**
Representation	String with constants and variables from a fixed alphabet	Shapes composed of maximal lines	Atomic geometric entities (lines, polygons, etc.)
Original area of application	Developmental biology	Spatial design and visual arts	Mathematics of self-similarity
Rule set	Set of production rules based on a fixed alphabet	Set of rules in a grammar	Typically, one rule defines a fractal
Left-hand side (lhs)	Variable from the alphabet	Shape	Initiator geometry
Right-hand side (rhs)	String with variables and/or constants	Shape	Generator geometry (a set of transformed initiators)
Interpreter	Detect substrings	Detect subshapes	Detect initiators (self-similarity)
Mode of application	Parallel	Serial	Parallel
Types of operations	String rewriting	Shape substitution	Shape substitution
Post-processing	A mechanism converts the string into geometry.	-	-

Among these, shape grammars are historically the main formalism for rule-based architectural

design (March 2011). Shapes are based on sets of maximal lines (Stiny 1980; Krishnamurti 1980) with rewriting rules that enable the substitution of shapes and, potentially, the emergence of unanticipated shapes (Knight and Stiny 2001, 364–72). Starting from an initial shape and following the rules defined by a certain grammar, it is possible to explore designs with a certain characteristic (a design language). The grammar produces shapes, which can resemble floorplans, sections or even 3D models.

For example, in a grammar for apartment layout customization (see Figure 9), the generation might start with a pre-defined footprint followed by operations to divide it into sectors. Then, each sector can be divided into rooms. The navigation in the space of possible layouts can be executed by a human or algorithm that selects the precedent floorplan shapes and rules.

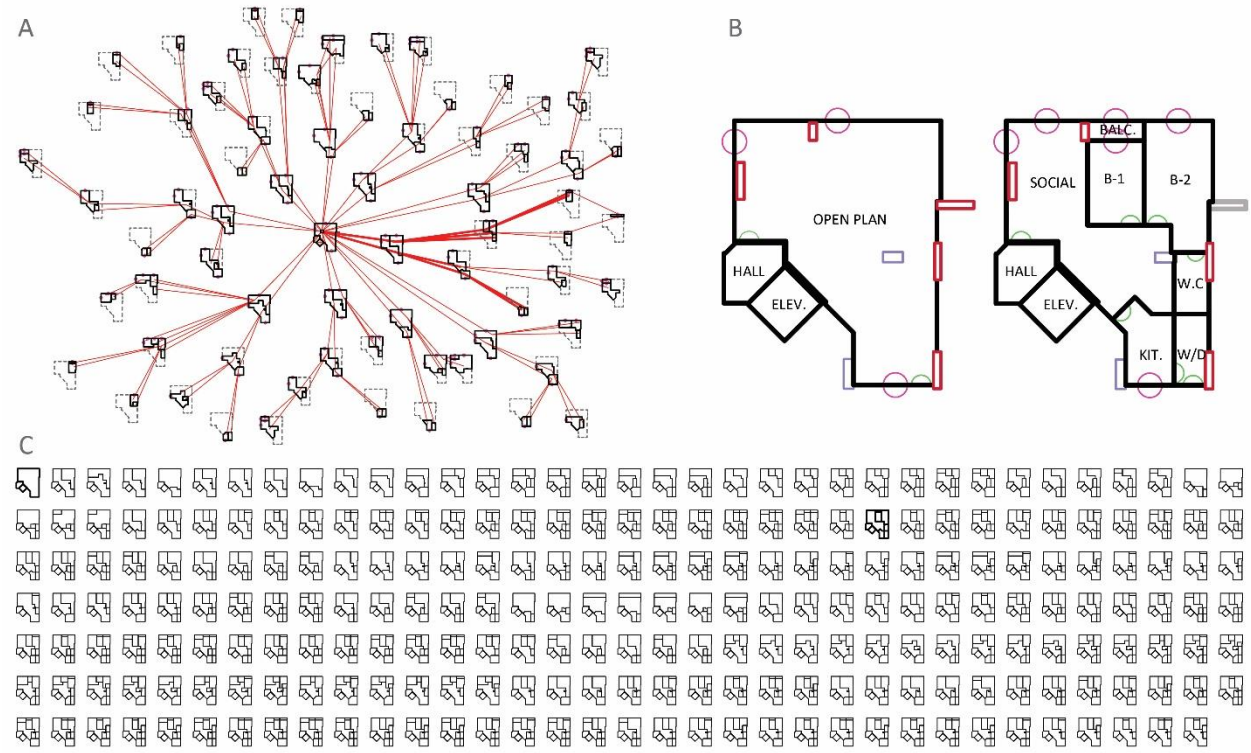


Figure 9 Example of a grammar for the customization of an apartment layout (Veloso, Celani, and Scheeren 2018). A: rule set organized in a graph. B: two examples of floor plans that can be generated by the rules. C: all the floorplans that can be generated. By Author.

Parameter- and rule-based models capture complementary aspects of design logic. In parametric modeling, the parameters typically define and bound the design space, while in rule-based models it is a network that, with proper rules and preconditions, can be indefinitely expanded. Despite these differences, they can be combined in hybrid models. A parametric function can represent a finite trajectory of rule derivations explicitly, with the preconditions and rules selection controlled by the input. On the other hand, rule-based models can be extended with a parametric layer, such as in parametric shape grammars (see Figure 8: A: item 5, and C) or in parametric L-Systems.

1.3.3. Optimization

Synthesis by optimization involves searching for the best solutions in a certain design space, with regard to certain quantifiable criteria of performance[7]. It is often associated with a parametric model, which expresses the design outcome as a function of specific input parameters. It adds two components to the generative model: a fitness function and an optimization algorithm.

The fitness function typically evaluates the performance of a design mathematically. It receives the design output from our generator and returns its performance values, subject to constraints. The fitness function can be as simple as a function that measures the surface area of a solution or as complex as a structural or energy simulation. Constraints can be values, combinations of or relations between parameters that are either unacceptable or that result in an unacceptable solution. These constrained solutions can be either avoided or eliminated from the model in the generation process or be represented in the fitness function as penalization.

By outputting a fitness value, this function creates a new space for the generative model: the fitness space. In this context, exploring design alternatives involves understanding the relation between the space of possible choices (usually, the parameter space of the model) and the space of the performance. If the fitness function is based on a single objective or performance value, each design alternative tested can be ranked on a one-dimensional space, which makes selection easy – i.e., by just picking solutions with higher value – for example, in Figure 10, the ones closer to the blue area.

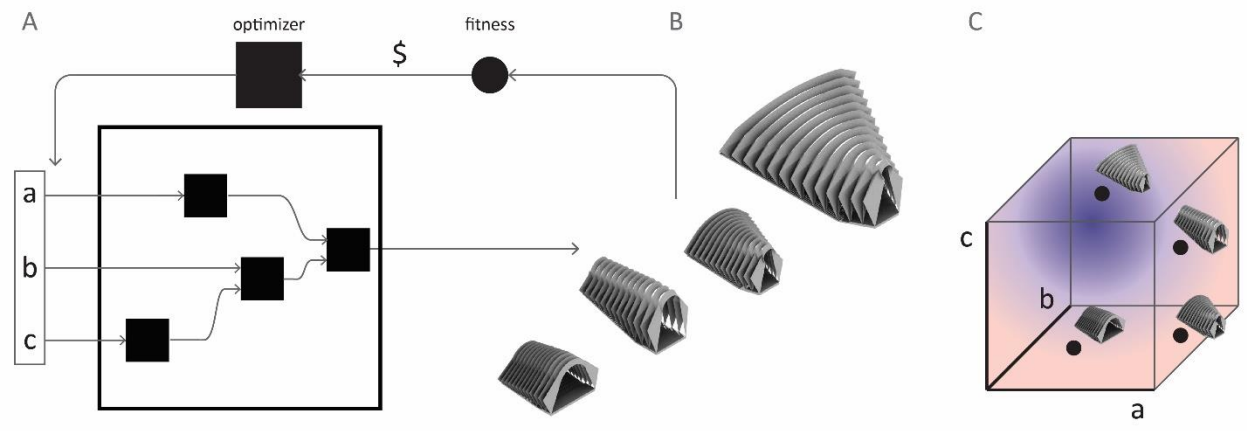


Figure 10 A: parametric model with a fitness function and an optimizer controlling its parameters. B: variations of the design. C: fitness landscape: diagram representing the fitness value of every solution in the parameter space (red is a low value and blue a high value). By author with model developed by Adie Alnobani for Parametric Modeling, CMU, Spring 2016.

If there are multiple performance criteria to be represented in a fitness function, it is necessary to

[7] For a more detailed discussion on the relevance of optimization to generative design, see **7.2 Appendix B: The influence of classic AI methods on generative models.**

define some strategy to address trade-offs and rank solutions[8].

The second additional component is an optimization method that can automatically navigate in the decision space of our generator (usually the parameters) to incrementally find solutions with better fitness values – i.e., by finding the parameters, subject to constraints, that maximizes this fitness function.

Generative models originally inherited optimization techniques used in operations research for decision making in military operations and industrial organization (Churchman, Ackoff, and Arnoff 1957, 3). These mathematical techniques for decision making relied on specific analytical forms of the problem description[9] (W. Mitchell 1977, 468–72; Radford and Gero 1988). Nevertheless, the elements of generative models, such as the parameters, generator, the constraints or even the fitness function, extrapolate the forms required by mathematical optimization. They require techniques with less mathematical restrictions and that enable designers to explore partial solutions or to learn about the correlation between decisions and performance.

Over the last decades, with the development of novel black box methods – namely, algorithms that do not require knowledge of the fitness function – optimization became predominant in generative modeling. There are three categories of black-box algorithms: metaheuristics, direct search and model-based (Wortmann 2018, 100–114). Recently, genetic algorithms, simulated annealing, particle-swarm optimization[10], and many other black-box techniques have been incorporated to parametric modeling as plug-ins (Rutten 2013; Wortmann 2017a). These techniques contain different strategies to navigate in the parameter space of the model based on the fitness signal of the objective function.

1.3.3.1. Optimization of parameters

Optimization is a very general method and allows design synthesis for different formulations of generative models. Typically, optimization is applied to the parametric model to choose the parameters that increment the fitness values of the resulting design alternatives. However, it allows for many other applications. For example, optimization of parameters can also be applied

---

[8] For example, it is possible to rank sets of solutions based on their hypervolume in the objective space, to combine the different objectives in a singular fitness dimension, or to look for the non-dominated elements in the set of solutions. Dominance is characterized between two solutions. A solution a dominates a solution b if it is no worse than b in all objectives and it is better than b in at least one objective. A solution is nondominated if no other solution in the set dominates it.

[9] For example, calculus requires access to a continuous and differentiable function and classic optimization techniques usually restrict of the problem formulation to a specific analytical form, such as a convex space (Radford and Gero 1988, 48, 90; Boyd and Vandenberghe 2004, 1–2)

[10] These examples are metaheuristics that use higher-level strategies inspired by Nature to explore good solutions to an optimization problem (Brownlee 2012).

to rule-based models. In this case, the parameters can represent a way to select the initial state (see Figure 11), precedents, rules, and the number of time steps. The fitness function can evaluate the characteristics of the designs achieved by these parameters.

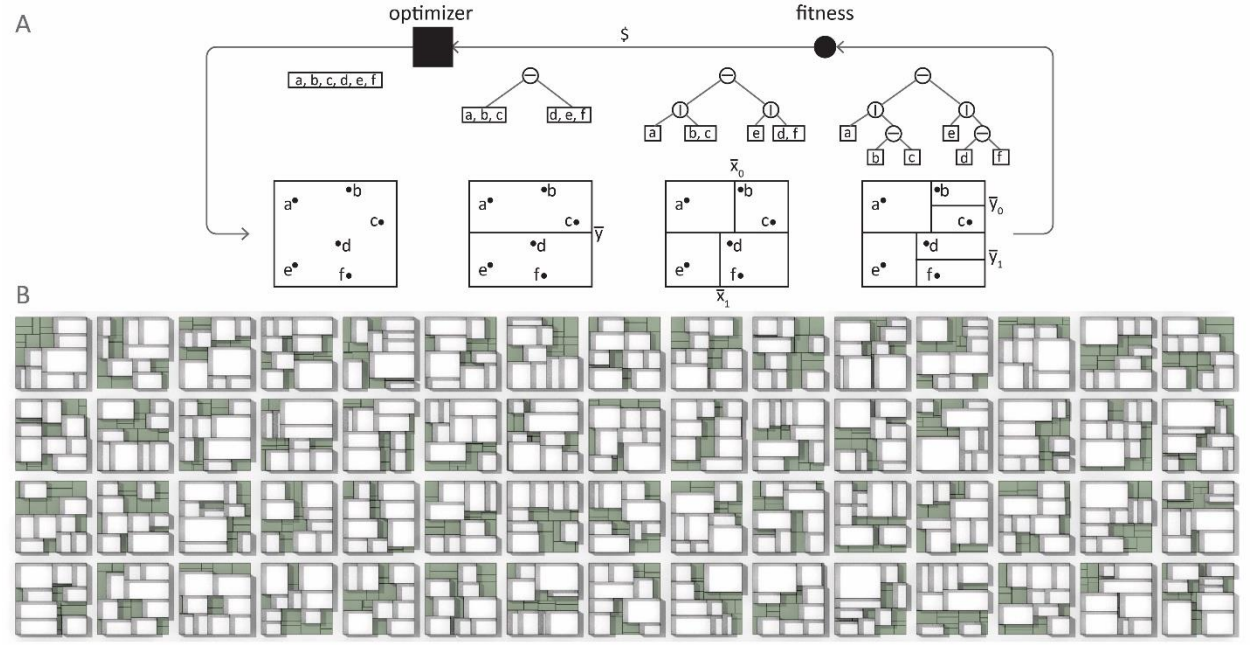


Figure 11 A: optimization of floorplans with genetic algorithms and KD-Tree. The coordinates for the points used as input for a KD-Tree partition are the parameters. B: examples of resulting designs. By Author and inspired by (Knecht and König 2010a).

1.3.3.2. Optimization of programs

Optimization explores a bounded and pre-defined space of decision variables, such as the parameter space of a parametric model. In the previous optimization examples, the topological structure of the parametric model or the logic of the algorithm is pre-defined, so the optimization can focus on the parameter space. However, if the design space is not satisfactory, it is necessary to re-design the parametric model itself to enable further exploration of alternatives.

In this sense, it is possible to use a meta-optimization to induce generative programs. The challenge is that the algorithms should be able to operate on data structures more complex than a list of parameters, such as trees or directed graphs with variables and operations. Conventionally, the algorithm used for this task is genetic programming (Koza 1992), an extension of genetic algorithm (GA) that deals with hierarchical computer programs of dynamically varying size and shape. It uses variations of the same genetic operators as the genetic algorithm, namely, selection, crossover, and mutation, to evolve a program (Figure 12). This approach can also be employed for generating programs such as parametric functions (Harding 2016), where the algorithm generates parametric functions and a uses a fitness function evaluates the performance of its output. Overall, inferring the program to achieve certain goal is a task that is close to the two categories of learning in the taxonomy.

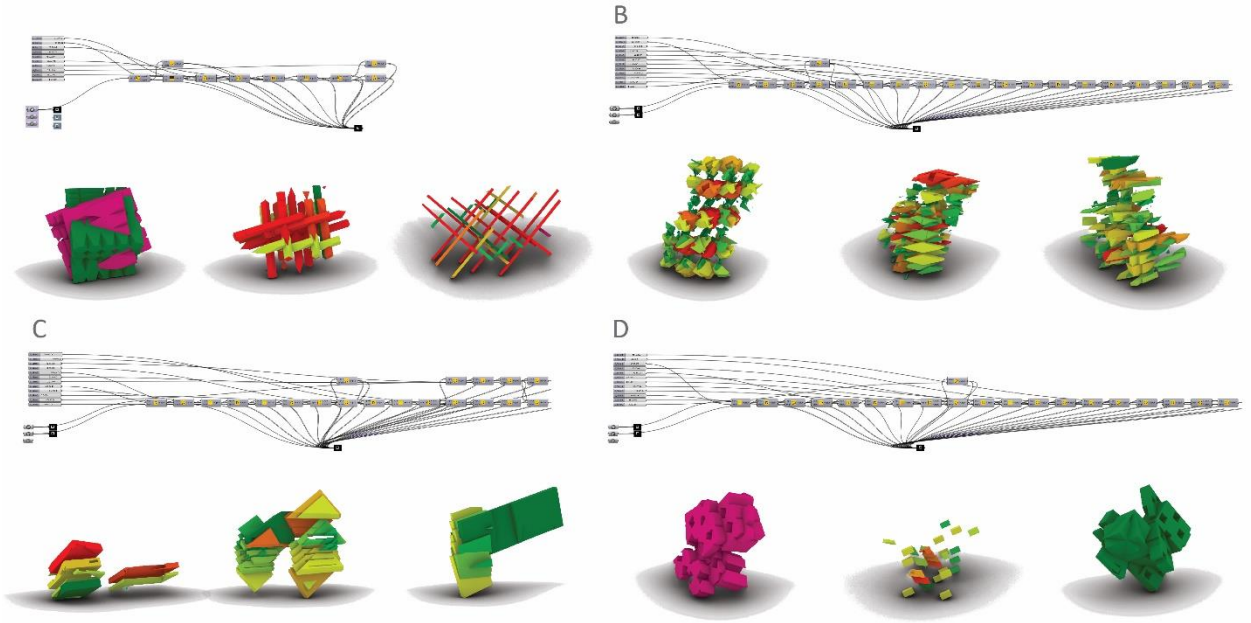


Figure 12 Use of genetic programming (Embryo) to generate parametric definitions for sculptures. A-D: variations of parametric graphs and their respective parametric sculpture below. Work produced for Generative Systems for Design, CMU, Fall 2019.

1.3.3.3. Challenges in interactive optimization

Typically, a configurator based on optimization relies on designers guiding the process by changing input parameters related to the problem and to the algorithm, then visualizing catalogues and dashboards with representations of the correlation between design alternatives and their respective performance (Tomasetti 2019). Many new techniques to map performance have been developed beyond the conventional parallel and cartesian coordinate representation of the fitness of the solutions. These include radial/start coordinates (Wortmann 2017b), and custom dimensionality reduction and clustering techniques (Koenig, Standfest, and Schmitt 2014; Sjoberg, Beorkrem, and Ellinger 2017). Despite being powerful tools for design exploration based on the performance space, these maps do not provide access to the generative loop.

As an alternative, research on interactive optimization extends human-computer interaction with control mechanisms inside the loop of optimization algorithms. The integration of interaction with optimizations aims at allowing "the user to bring their additional knowledge into the solution process and means that the tool is less brittle and can more flexibly adjust to unforeseen situations" (Liu et al. 2020, 1). It is a promising area of research in CAAD, with practical results already available for designers in the form of plug-ins. This comprehends the adaptation of populations of solutions based on user interaction with the spatial representation (Knecht and König 2010b), the manual selection and ranking of solution candidates, or dynamic control of the decision variables and objective functions inside the optimization loop (Von Buelow 2012; Danhaive and Mueller 2015; Harding and Brandt-Olsen 2018; Newton 2021).

While Interactive optimization is a promising field, combining efficient and non-destructive optimization methods to support local decisions, sequential construction of the spatial

representation, and user control in the loop of optimization algorithms is still an open problem.

1.3.4. Composition

In the decision-theoretic paradigm of symbolic AI, design synthesis can be modeled as a sequential decision process based on the search of "satisficing" solutions rather than optimizing performance (H. A. Simon 1975; 1996, 114–22)[11]. In this case, the generative process starts with an initial representation of the design (the root of the tree). The designer uses some form of heuristics to select incremental transformations, which can be interpreted as a sequential composition[12]. While the interaction of the designer with any type of model can be interpreted as a sequential decision process, this type of synthesis is built-in in rule-based models, where the design space is defined by a tree with implicit design alternatives.

1.3.4.1. Sampling

To explore the design space of a rule-based model without being committed to specific goals it is possible to sample solutions using a stochastic or deterministic strategy to select preconditions and rules. This weaker method is widely adopted in architectural design to explore solutions with different formal qualities conditioned.

The preconditions and rule selection that can lead to certain types of formal configurations. For example, in Figure 13 I display a grammar that contains three rules that can be applied to any convex quadrilateral: x: expansion, y: parametric division, and z: elimination. All the quadrilaterals in the representation are preconditions. They can be ordered for the next derivations based on a fixed logic, such as a queue (first in, first out), a stack (last in, first out) or a priority queue (order according to a property). The rules and parameters for the derivation are sampled from a probability distribution defined by the designer. A collision test is used to prevent new quadrilaterals to collide with pre-existing elements in the environment (other quadrilaterals and obstacles).

[11] For a more detailed discussion on the relevance of heuristic search to generative design, see 7.2 Appendix B: The influence of classic AI methods on generative models.

[12] The term composition is related both to function composition and to architectural composition. In mathematics this is an operation that sequentially applies two or more functions – ex: $g(f(x))$. In architecture, it is related to the process of creating an architectural configuration (see **1.2.2 From architectural composition to spatial synthesis**).

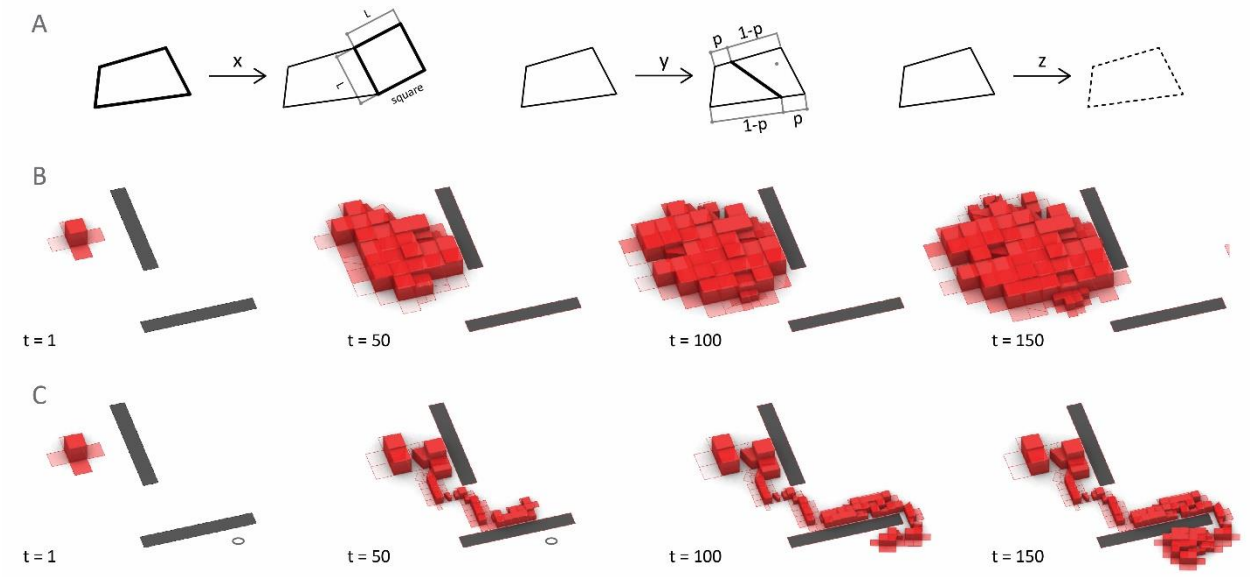


Figure 13 A: Rules of the qGrowth grammar. A-B: Generating forms with stochastic selection of rules (rule x: 60%, rule y: 30%, rule z: 10%) and parameter p (0.2: 50%, 0.8: 50%). B: precondition selection with a queue (quadrilaterals added first to the frontier are selected first). C: precondition selection with a priority queue (quadrilaterals closer to the circle are selected first). By Author.

Using weak methods such as sampling from the implicit design space of a rule-based model is not only part of the original research on spatial synthesis and shape grammars but also a trend that has recently became widespread in parametric modeling. For example, plug-ins such as Wasp ("Wasp" 2017) enables the user to define additive rules for spatial design, spatial constraints, and to derive aggregations automatically. It supports stochastic, graph-grammar based, and field-driven aggregations. See example in Figure 14.

Information about constraint can also be incorporated as a heuristic for the sampling. For example, the wave-function collapse (WFC) algorithm (Gumin 2016) was created to for the procedural generation of bitmaps that are locally similar to an input bitmap. WFC is a constraint solver for a tiling problem with additional requirements for local similarity – i.e., the patterns in the input should occur in the output and should approximate the original distribution in the input. It stores the information about the input tiles that are still legal for each location of the target grid. At every iteration, the algorithm uses a heuristic that select the empty location with fewest allowed assignments. Then, it uses the information from the tile distribution in the input to select among the legal tiles for that location. After placing the tile in the location, it propagates the new constraints imposed over the unoccupied cells of the grid, so in the next iterations it can repeat the same procedure with updated information about the constraints.

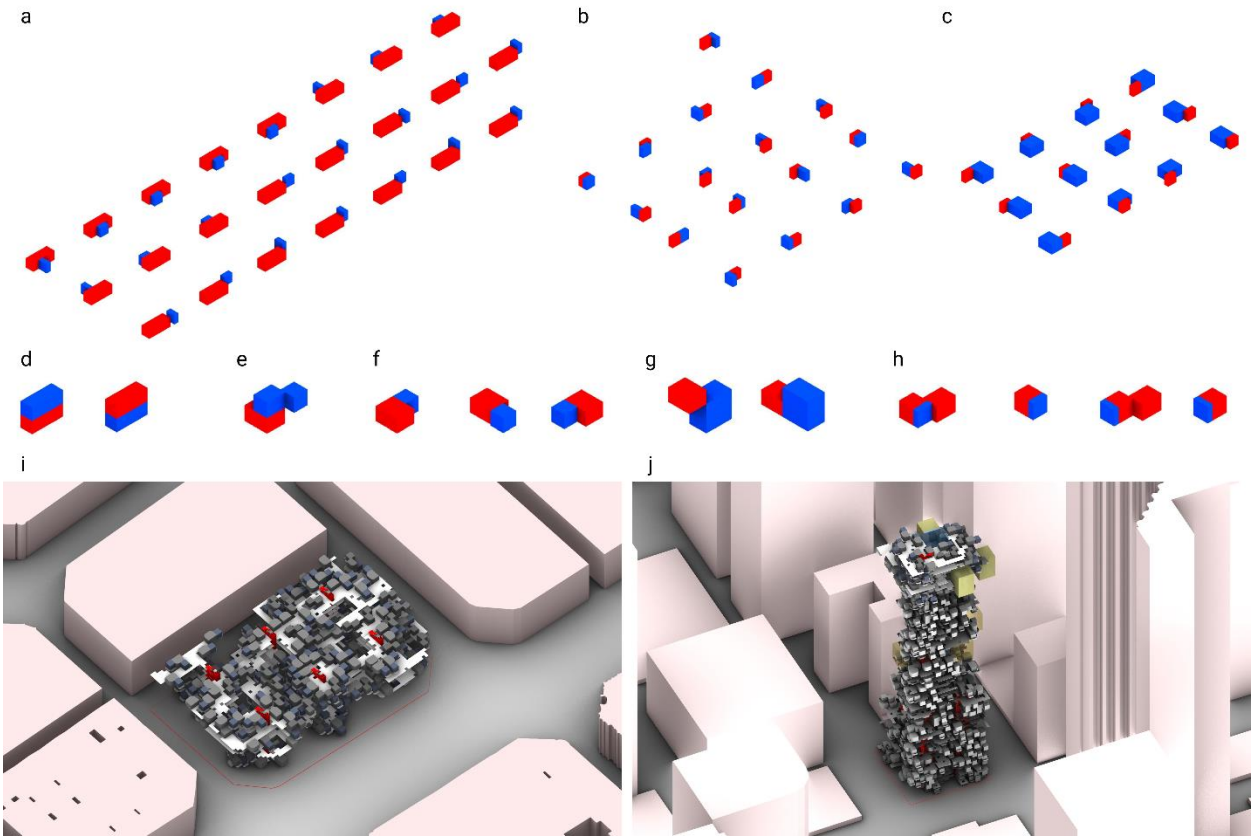


Figure 14 Interactive Aggregations. Set of rules for aggregation of components of housing complex with symmetries: a) core module (red) and horizontal circulation (blue); b) horizontal circulation (red) and horizontal circulation (blue); c) horizontal circulation (red) and living/kitchen module (blue); d) core (red) and core (blue); e) living/kitchen module (red) and double height living (blue); f) living/kitchen module (red) and bedroom (blue); g) living/kitchen module(red) and deck(blue); h) bedroom (red) and deck (blue). a) and b) Two examples of aggregations generated with sequences of random sampling combined with optimization and iterative creation of obstacles as constraints. Reproduced with permission of Hardik Makrubiya and Jesse T. Gonzalez. Work produced for Generative Systems for Design, CMU, Fall 2021.

WFC became widespread in generative design for procedural generation of games scenarios (Møller and Billeskov 2019) speculative design of modular structures in architecture (Chasioti 2020). It is also part of a plug-in called Monoceros ("Monoceros" 2021), which allows the users to define the adjacency allowances in the parametric modeling environment and use WFC to sample compositions. See example in Figure 15.

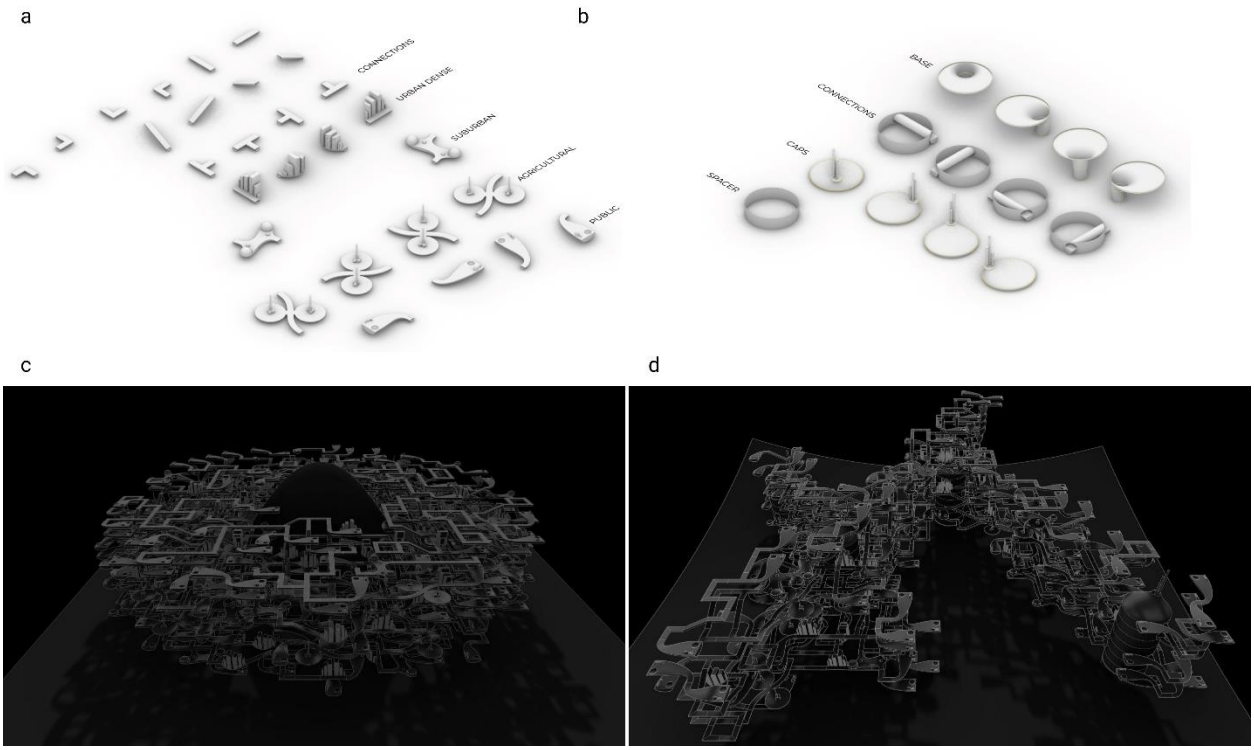


Figure 15 Ocean City Revisited. Elements used as tiles for aggregation (a and b); two examples of aggregations with different ground and boundary constraints. Reproduced with permission of Nicholas Coppula and Davis Dunaway. Work produced for Generative Systems for Design, CMU, Fall 2021.

1.3.4.2. Search

For a more systematic composition, it is necessary (1) a goal function to evaluate the qualities of the design alternatives in the design space and (2) a systematic search strategy to build solutions. In contrast to the fitness function used for optimization, which returns a numerical value indicating performance, the goal function typically indicates if a design is satisficing or not, based on experience, preferences, or any other criteria. A search algorithm starts at the root of the tree and explores multiple paths incrementally until it eventually synthesizes "satisficing" designs based on designer aspirations and criteria (H. A. Simon 1975, 297–301; 1996, 119–21). The search can rely on personal criteria or on strategies from AI (Russell and Norvig 2021), such as the ones described on Table 3. It is also possible to eliminate large portions of the search tree at once by identifying combinations that violate certain constraints, such as collision, area, etc.

Table 3 Examples of search algorithms.

	Type of knowledge	Order of exploration	Data structure
Breadth-first search (BFS)	Order of branches	Level-by-level	Frontier: Queue (first in, first out)
Uniform-cost search (UCS)	Path cost	Best path cost first	Frontier: Priority queue (highest priority first out)
Depth-first search (DFS)	Order of branches	The deepest nodes first	Frontier: Stack (last in, first out)
Greedy best-first search	Estimate of value of next nodes	Best estimate to reach goal first	Frontier: Priority queue (highest priority first out)
A*	Path cost and estimate of value of next nodes	Best path cost and estimate to reach goal first	Frontier: Priority queue (highest priority first out)
Monte Carlo Tree Search (MCTS)	Estimate of node values based on sampling and backpropagation	Policy that balances exploration and exploitation of current estimates. Random search for unexplored nodes.	Estimates are stored at the nodes of the tree

One of the classical examples for applications of search in design is in layout automation. For example, Autoilet (Figure 16) combines a parametric and rule-based model with a search to automate the design of single-user toilet rooms according to the standards of the Americans with Disabilities Act (ADA). The inputs for the algorithm are the dimensions of the room, whether the room is L-shape or not, the dimensions of the corner, the position of the door. For each element of the toilet, there is a geometry for the element as well as a clearance. The goal function requires that the geometry and clearance of each element should be within the boundary of the room and there should be no conflicts between objects and clearances. A depth first search (DFS) is used to sequentially place all the equipment elements and return satisficing solution.



Figure 16 Autoilet. A: parameter space of the input (boundary, corner, and door placement) for rectangular and L-shaped footprints. B: examples of solutions using DFS for placement of toilet, sink and ADA circle. Reproduced with permission of Jichen Wang. Work produced for Generative Systems for Design, Fall 2019.

1.3.4.3. Traversal

In some cases, the designer enumerates the possible solutions under a certain criterion – such as all possible rectangular and non-dimensional floorplans up to a certain number of rooms (Steadman 1973; Krishnamurti and Roe 1978; Steadman 1983). While this type of enumeration can be done by sampling multiple solutions and organizing the unique results in a set, it is more efficient to use a search algorithm to explore the solutions systematically. However, instead of stopping the search when a certain goal function is satisfied, the algorithm should keep exploring and accumulating all the satisficing solutions until the tree is exhausted or a certain stopping condition is triggered.

1.3.4.4. Challenges in interactive composition

Rule-based and related modeling techniques create an implicit design space in the form of a network of design states connected by transformations. This design space provides a structure of navigation in which one or more solutions can be built step-by-step, which enables the incorporation of the designers' decisions and preferences at any point of the generation

However, there are some challenges in automatic composition with sampling and search algorithms. First, algorithms for composition typically anticipate a sequence of operations and a clear criterion to end the process, which can be restrictive for design space exploration centered on designers. It is not trivial to create rules to form a large design space with varied and potentially emergent design alternatives. Secondly, the design space grows exponentially, with the number of transformations available at each state (branching factor) as the base, and the depth of the search as the exponent. So, there can be a combinatorial explosion of alternatives or depending on the model, the implicit design space can be unbounded with infinite alternatives.

Therefore, automated spatial synthesis with composition typically requires constraining the state space and the search tree to reduce the computation. Over the last decades the use of heuristic strategies related to the adjacency of the rooms or of its internal components to define an order of operation to reach good configurations has been a common practice (W. Mitchell 1977, 459–62; Egor et al. 2020). In other cases, an efficient sampling algorithm for aggregation is defined, so the user can control the output indirectly by manipulating a seed or parameters of the problem, such as the definition of the spatial parts, connections, obstacles, and catalogues of parts ("Wasp" 2017; "Monoceros" 2021).

1.4. A guideline for spatial synthesis with designers in the generative loop

In generative design, the designers customize the model information during the formulation stage, and, during the execution, they customize the input information (Figure 1), which can contain changes in the state of the design representation or even parameters of the synthesis algorithms. In essence, classic generative models emphasize an indirect form of interaction where the designers concentrate on the input information while the generator uses it to search for

alternatives in the design space and display them in the output[13]. Current computational technology comprehends sophisticated forms of cloud, multiprocessing, and parallel computing, and by accessible programming, algorithm editors and plug-ins connected to or embedded in design software applications. While the "indirectness" of the design generation with optimization and composition can be alleviated by faster or better algorithms, modes of visualization, or new forms of design control, addressing fine-grained interaction is still an open problem.

Instead of proposing modifications and extensions to the mechanisms of classic generative models, this research will investigate an alternative generative model based on the premise that the designer is inside an interactive simulation during the generation of design alternatives. This approach relies on a form of rationality based on facts and values related to the practitioner, who can guide the design process instead of looking for an objective model for design. This form of pragmatic rationality benefits from existing practices and emphasizes situated action and cognition. In 7.3 Appendix C: Beyond objective rationality in generative design?, we discuss the limitations of problem-solving and review the importance of the conversational and situated aspects of design creativity in the literature of design theory.

Motivated by these aspects of design, we suggest a guideline for interactive simulation motivated by the importance of designers as orchestrators of a conversational, situated, and fluid design exploration. The basic assumption of this guideline is that an interactive simulation can increase the opportunities for architects to use of personal and collective disciplinary constructs to guide the design exploration. This can not only support the exploration of partial, preferable, or unexpected spatial configurations based on *ad hoc* design decisions and contingencies but also other elements of design, such as learning and problem restructuring. Overall, this strategy intends to provide a spatial synthesis experience that uses situated knowledge and action to align spatial computation and design values.

The initial characteristics of this guideline for spatial synthesis with designers inside the generative loop are

1. Real-time response to prevent the disruption of the thought process
2. Step-by-step constructive procedure to access partial representations and to engage designers in the building logic of the design configuration
3. Indirect level of control of the problem parameters to enable exploration of different versions of the problems and design spaces
4. Fine-grained control adjustment by human and computers that enables direct manipulation of small parts of the spatial representation with immediate feedback between designers and the generator

[13] For more information, check 7.1 Appendix A: Notes on architectural composition.

5. **Reduced assumptions about initial setup, order of operations, and control strategies** to support guidance without explicit objectives to spaces of personal preference and the emergence of spatial types based on situational aspects of the generation
6. **Simultaneously address different goals** to allow designers to visualize and learn with the spatial consequences of the multiple and entwined goals embedded in the system
7. **Event-driven computing** to respond to identifiable actions of the designer during the generation

The next sections address two computational paradigms that support the construction of the interactive simulation for spatial synthesis from the ground up: agent-based computing and reinforcement learning.

1.5. Agent-based computing in generative design

Over the last decades, many paradigms have been incorporated in generative modeling, extrapolating the classic setup of spatial synthesis (see **7.3.3**). Among these, agent-based computing and many of its techniques became essential components of generative models. Agent-based computing supports the use of distributed computation of reactive agents to address general problems, or the creation of novel algorithms based on bio-inspired models, extending some categories already present in our taxonomy, such as optimization and composition (Figure 17). Given their distributed nature, agent-based computing and simulation provide many opportunities for generative design, such as the development of robust, scalable, and flexible strategies of problem solving, and the exploration of patterns at the macro-level from the interactions between the parts at the micro-level without outside control (emergence and self-organization).

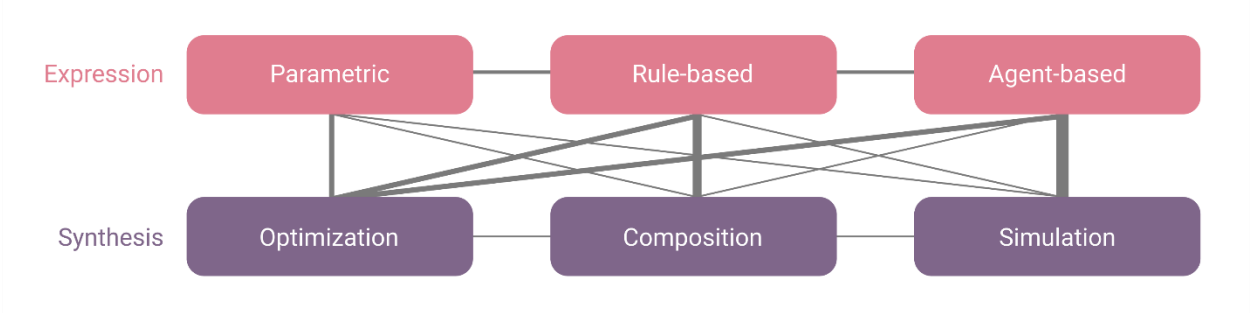


Figure 17 Taxonomy updated with agent-based model and simulation. By Author.

1.5.1. Agent-based computing and simulation

The domain of agent-based computing is vast, as it comprehends many areas of study that employ decentralized computational methods based on a collection of discrete and autonomous entities referred to as agents that are situated in a shared environment. Agent-based computing is not restricted to computer science as it is also present in scientific domains such as life sciences, ecological sciences and social sciences, which can lead to confusion with regard to semantics (Niazi and Hussain 2011, 479–80). In **7.4 Appendix D: Agent-Based Computing** I describe some of the

interpretations of distributed systems based on agents, with a focus on the following areas: multi-agent systems (MAS), agent-based modeling (ABM), and bio-inspired artificial intelligence and engineering.

In AI and MAS, multiple agents are used to support algorithms for solving problems such as distributed constraint-satisfaction problems, distributed optimization, multiagent learning, communication, and social choice (Shoham and Leyton-Brown 2009). In contrast, in ABM multiple agents are used to model the dynamics of real-world or hypothetical systems, such as fluid flow, pattern formation, reaction diffusion of chemicals, physics, markets, and even social interaction in spaces. In the latter, agents are algorithmic descriptions of entities of the system that can take local decisions based on the local perception and interact in a shared environment.

In contrast, agent-based models used in generative design do not intend to capture the dynamics of existing systems but to emphasize the exploratory aspect of modeling. They allow designers to explore different design scenarios and to discover association between parameters, constraints, and goals that are encoded in the characteristics of the agents and environments. With that, designers can visualize and evaluate the behavior of different scenarios and emerging spatial pattern over time and under uncertainty.

Despite this distinction, the basic definitions of agent-based models still hold. Agents and environment are differentiated by the notion of control where everything that is directly controlled by the agent is part of the agent, while everything else is part of the environment. Autonomy refers to the capacity of the agent to perceive the environment and other agents, then use its internal program to take actions. Each agent is an entity with local control, interweaving individual perception and action to satisfy certain spatial objectives. It observes its internal state, other agents, and the information from the environment. The local interactions of the agents are restricted by a notion of neighborhood, which is paramount to agent-based computing. When the agent is represented in a grid, its neighborhood can be defined in various ways, for example, Moore neighborhood, von Neumann neighborhood (Figure 18: A and B) or by an arbitrary collection of cells. Agents in a Euclidean space can receive the information constrained by a certain radius (Figure 19: A), by signals from radial pulses, or any other form of perception.

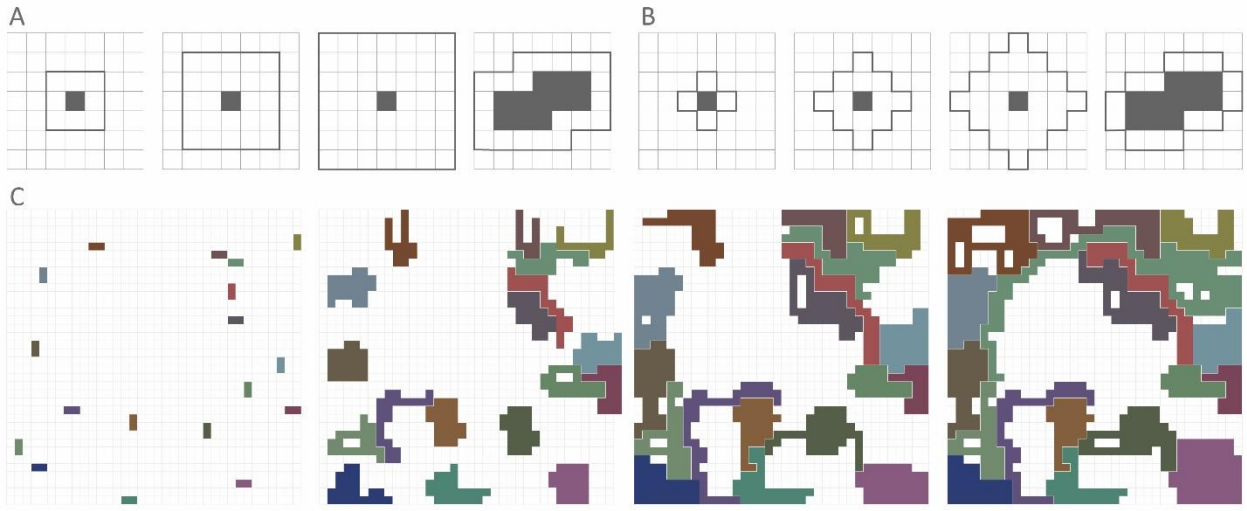


Figure 18 A: Moore neighborhood with different radii for single-cell agent and for multicellular agent. B: Von Neumann neighborhood with different radii for single-cell agent and for multicellular agent. C: simulation with multiple agents that use two cells for expansion. Images by Author. Images in C relied on model developed by Pragya Gupta for Design and Computation 2, Fall 2018.

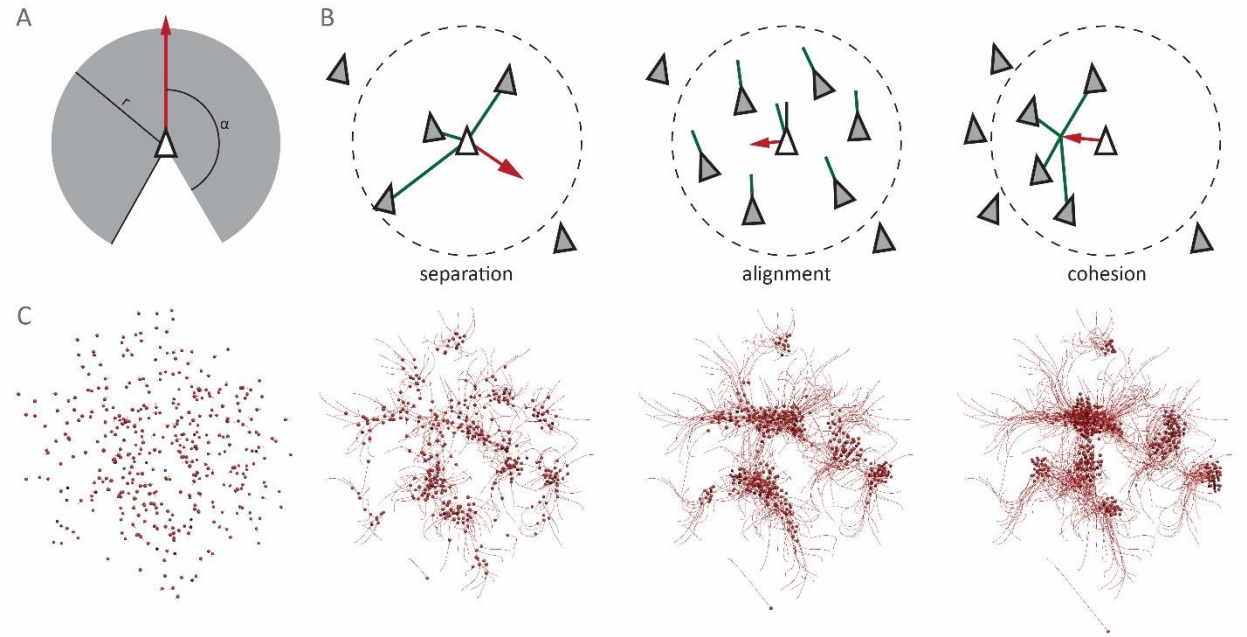


Figure 19 Boids: a model that simulates the flocking behavior of bird. A: neighborhood based on radius and visibility angle. B: three basic behaviors (separation, alignment, and cohesion), which add forces to the agent conditioned on the state of the neighbors. C: Boids interacting in a simulation. By Author. A and B were re-drawn from diagrams in (Reynolds 1999).

Table 4 contains models that are applicable to agent-based form exploration, and architectural and urban design.

Table 4 Examples of agent-based models

Models	***Discrete models***		***Swarm models***		***Growth models***	
Example	*Cellular Automata (Wolfram 2002)*	*Reaction-diffusion (Turing 1952; Pearson 1993)*	*Ant foraging (Bonebau, Dorigo, and Theraulaz 1999)*	*Boids (Reynolds 1987; 1999)*	*Diffusion-limited aggregation (Witten Jr and Sander 1981)*	*Differential Growth*
Standard Environment	*Grid*	*Grid*	*Grid or graph with pheromone markers, food, and nest*	*Euclidean*	*Euclidean*	*Euclidean*
Agent	*Cell with binary states*	*Cell with concentration of two chemicals*	*Ant that occupies a cell*	*Agent with position, velocity, and acceleration*	*Particles that move; aggregated particles*	*Network of connected discs*
Standard neighborhood	*Moore*	*Moore*	*Adjacent cells/nodes*	*Agents within distance and visual field*	*-*	*Connected discs*
Action	*Update state based on neighborhood*	*Update state based on reaction of chemicals*	*Move to adjacent cell based on pheromone; deposit pheromone*	*Behaviors update trajectory*	*Random movement; attach particles close to aggregation*	*Disc repulsion; extended link creates new agent*
Result	*Dynamic patterns*	*Dynamic patterns with chemicals*	*Pheromone network between food and nest*	*Flocking patterns*	*Brownian trees*	*Self-folding curve or mesh*

The internal program of an agent or its policy defines what action to take conditioned on the information received by the agent over time. This policy can be a simple set of if-then-rules that looks for preconditions in the input to select the action or that follow the gradient of a certain function (e.g., pheromone or distance to neighbor), a set of cumulative behaviors (Figure 19: B), or a more sophisticated function that chooses action based on an estimate of the future performance. Overall, due to the distributed and real-time computation, these tend to be reflex agents.

While the structure and the program of the agents are typically designed manually, it is also possible to employ some form of optimization for generating the internal programs of agents, such as in the examples of the cas framework (Holland 1995) described in **7.4**. In this case, the evaluation function can be associated with the performance of the agent during the interaction with the other elements in the model.

In terms of actions, in a grid, an agent might be able to switch the states of its current cell, such as in cellular automata. In other cases, it might be able to move to another cell, such as in ant algorithms, or even expand or retract to change its shape if its representation allows multiple cells

(Figure 18). In the Euclidean space, an agent can move using acceleration (Figure 19: C) or change aspects of its shape, such as the position of vertices of polygons.

The behavior of agents in a shared environment results in interaction and changes in the design representation that can lead to the emergence of patterns – i.e., more comes out than what was put in. Agent-based models enable this type of surprise because the mechanism for decision making is embedded in the program of the agents and conditioned by their neighborhood over time.

For deterministic agents and environment, the design space is a single trajectory of states, parameterized by the initial configuration of the agents, as in standard cellular automata. However, if the agents can take multiple actions for the same circumstance, the space of design alternatives is a large tree involving the joint action of the agents in the environment. Each of the possible paths of this tree display a certain evolution of patterns over time. The space of possible configurations tends to be astronomic and it is not possible to predict the behavior based on the average agent (Holland 1995, 118). As a result, the algorithm to form solutions relies on executing the interaction of the agents in the environment over time—that is, a simulation. In a simulation, the model will update the states of the agents and of the environment over multiple time steps (Figure 20: B). To explore different alternatives in a simulation, the designer must push the simulation toward certain directions by customizing the environment and the agents. This customization can involve tasks such as, parameter tuning, the design of new behaviors, or simply adding randomness (Figure 21).

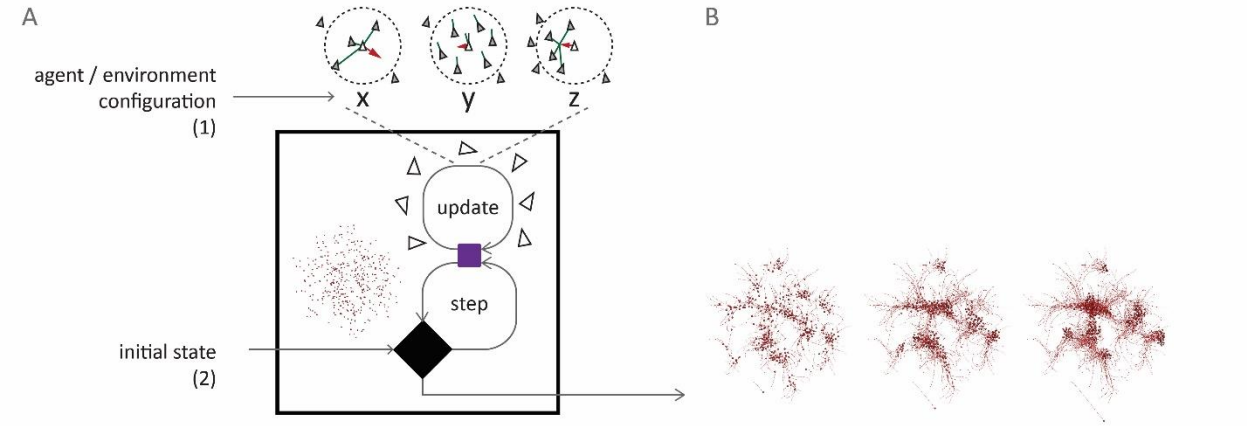


Figure 20 A: Diagram of the agent-based model. The bottom loop represents the time steps, and the top loop represents the updates of the agents for one-time step. B: three distinct steps of the simulation. By Author.

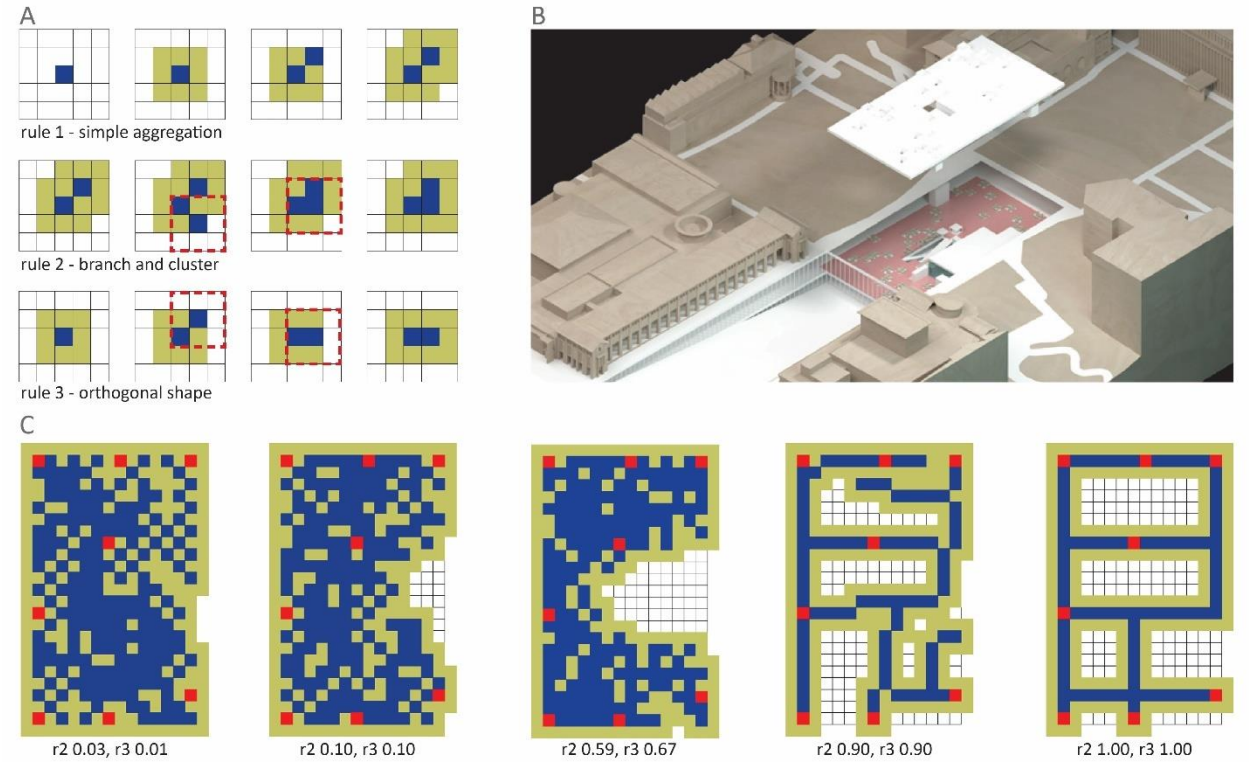


Figure 21 Agents generating an underground gallery on campus. A: stochastic program with three rules. B: design of campus gallery based on agents with r2 = 0.59 and r3 = 0.67 at time step 200. C: the resulting spatial configuration for the same initial environment with different parameter: r2 and r3 increase the probability of choosing rules 2 and 3 respectively. Reproduced with permission of Jinmo Rhee. Work produced for Generative Models for Design, CMU, Fall 2019.

1.5.2. Opportunity: fine-grained interaction with spatial agents

The standard definition of agent-based computation comprehends agents taking decisions and action based on their perception of its neighborhood, in a shared environment, and over time. This situated, distributed, and multi-step decision processes provides not only opportunities for generative design with the designer in the loop but also presents challenges inherited from the paradigm itself and from the area of application in consideration: interactive spatial synthesis.

Agent-based computing satisfies many of the conditions of the framework for fine-grained interaction with the designer in the loop. It relies on a simulation based on the interaction of the agents with the environment. The computation happens in the local representation of the agents, so it attenuates the problem of the exponential joint action spaces and enables the creation of reactive agents in settings with variable numbers of agents. Besides, the agents are typically reflex agents, so the computation for decision making does not require expensive planning procedures and it can run in real-time. The interaction between computational model and designers can happen at every step of the generation of the design alternatives, which supports sequential construction, fine-granular control, and event-based computing.

Fine-grained interaction with agents enables designers to participate in the simulation without dealing with destructive transformations or large state space. This is important to provide access to information about design generation to support learning and situated actions. In contrast to

generative models that focus on centralized algorithms for problem-solving, agent-based models provide a structure of interaction over which designers can improvise during the simulation with contextual actions. This type of local navigation allows designers to learn with the rich information exchange between the agents and the environment. For example, a designer can locally tweak the interactive simulation towards design alternatives that embed unstated qualities or explore patterns that emerging during the simulation.

1.5.3. Challenge: design of agents that can address multiple spatial goals

Spatial synthesis relies both on the satisfaction of factors that are external and internal to the designers and to the discipline of architecture. The former are requirements or constraints of a given project, such as the structural, environmental, and programmatic. The later includes the intuition and paradigms adopted by the architects, which can include shared precedents, aesthetic codes, visual language, design principles, methods, and rules.

There is an emerging niche of the field where agent-based techniques are employed in the generative models to augment exploration of spatial configurations in real-time, which I refer to as multi-agent spatial synthesis (MASS). In MASS, each of these agents is an autonomous entity that can use its local perception and its program to reason and act over the spatial representation. Its program embeds an implicit or explicit goals that involves changing the spatial representation towards a certain direction with local control. The agents can have different relationships with the spatial representations during the simulation, such as

1. Type 1: agents that embed the representation of a spatial unit, such as a room or a zone, so it can move and reshape it over time.
2. Type 2: agents that move and occupy a pre-existing spatial partition in the environment
3. Type 3: agents that are used as a reference for an algorithm that partition a space

As multiple agents interweaving individual perception and action to satisfy certain spatial objectives they interact in the shared environment and can potentially cooperate or compete. MASS can have either the goal-oriented approach of classic spatial synthesis in order to create agents that satisfy certain requirements, or it can be closer to the exploratory branch of ABM in the development of models that investigate emerging spatial patterns resulting from the interaction of custom agents.

The literature review on MASS prototypes in **7.5 Appendix E: A review of multi-agent spatial synthesis** and published in (Veloso, Rhee, and Krishnamurti 2019) shows that the use of a rich repertoire of agent-based techniques for interactive spatial synthesis exploration has been successful in conceptual form exploration. However, it also shows that it not trivial to embed them with domain knowledge and the capacity to handle multiple spatial objectives and conflicts. Most examples fell short in generating spaces that satisfy multiple goals and constraints of architectural layouts.

Most of existing agent-based models for spatial synthesis rely on expert knowledge in the form of

heuristics or on the adaption of existing models from other fields. In the literature review the prototypes relied on three general trends – bioinspired agents, physics-based agents, and hybridization with destructive methods – that have some limitations.

1. Bioinspired models are generally resilient and adaptable. Still, as architectural problems are not always analogous to biological problems, it is usually hard to produce valid architectural patterns and satisfy architectural objectives.

2. Physics simulation is a more general model that conciliates simple control with intuitive interaction and can be adapted for different spatial problems and objectives. It has recently been incorporated into real world practices (Derix and Izaki 2013). However, physics-based agents are reactive agents approximately following laws of physics, so it is not clear how to manage spatial conflicts with actions that address long-term goals.

3. Some of the implementations address this by combining agents with metaheuristics that globally improve the solution, which often makes the granularity of the interaction coarser with isolated algorithms that radically change the representation.

Another aspect of this challenge is that agent-based modeling supports many characteristics of complex systems, such as emergence and self-organization. However, while complex systems can be used as metaphors or analogies for architecture, ultimately an architectural model must deal with the requirements of architecture. So, overall, there is a design tradeoff between models that enable controlled navigation in a design space with alternatives that satisfy architectural requirements and models that support the emergence of unexpected patterns and a self-organized increase in order.

1.6. Training agents for spatial synthesis

While agent-based modeling supports many important characteristics for spatial synthesis with designers in the loop with fine-grained interaction, it also poses challenges in terms of agent design. Overall,

- it is unlikely that a human can design custom agent-based models that comprehend relevant behaviors for all possible fine-grained interactions

- It is not clear how to design agents that can address multiple spatial goals while preserving the finer granularity of the design interaction.

- it is not trivial to adapt agent-based models from other fields, such as social sciences, artificial life, chemistry, or biology, to create agents that address architectural goals

To address the challenge of agent design for spatial synthesis, I investigate the second computational paradigm of our research framework: learning. Particularly, I address the field of deep learning and deep reinforcement learning.

The previous categories of our taxonomy rely on the designer explicitly encoding the generative

knowledge with different formulations (parameter-based, rule-based, or agent-based) and strategies for synthesis (optimization, composition, or simulation). However, one alternative to incorporate implicit design knowledge is to use design data or experience to infer the generative logic of a model, similarly to the examples in deep learning. In our taxonomy of generative modeling this results in a new level referred to as learning (Figure 22).

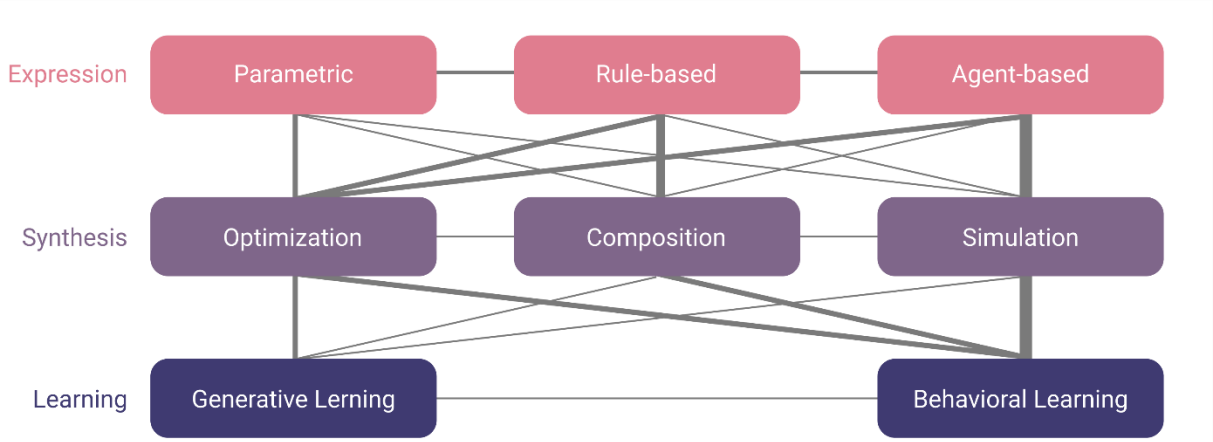


Figure 22 Updated taxonomy with new level for learning, comprehending the components generative learning and behavioral learning. By Author.

An adaptive model such as a neural network enables designers to build components of a generative model without explicitly describing its logic, which is an important step to bridge computation and intuitive design. The recent advancements of deep learning had strong impact in the digital design movements in architecture and in computational design research. In 7.6 we provide a brief review of basic concepts of deep learning.

Deep learning has the potential to support the training of different components of the generative model, such as a fitness function, a parametric model, the program of agents, or a search mechanism. This brings learning algorithms to the formulation stage (Figure 23: A). The logic of the component can be inferred from different sources, such as a dataset curated by the designer or simulations. Then, in the execution stage the component will support the generation of design alternatives and can even be refined with further training (Figure 23: B). For example, the category generative learning (see 7.6.6 Generative Learning in the appendix) emphasizes models that learn from data how to map abstractions, such as categorical images, graphs, textual description, into architectural forms.

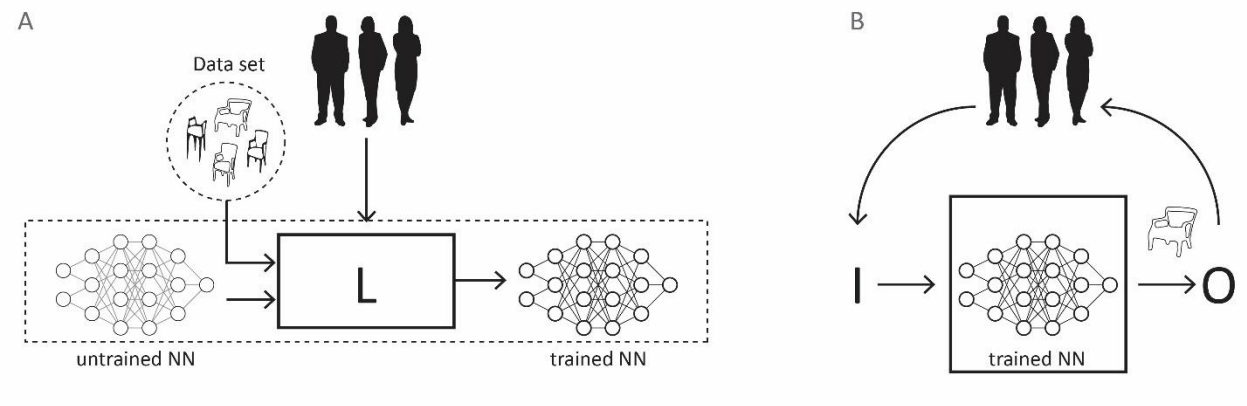


Figure 23 A: the formulation of generative model using a data set (D), a neural network (NN), and a learning algorithm (L). B: execution of a generative model: designers control an input (I); the trained neural network works as part or as the whole generator (G) and synthesizes a design alternative as an output (O). By Author.

Learning is particularly relevant when the generative rationale to formulate a generative model might not be available. For example, in conventional design ideation, designers rely on their cumulative experience to interact with the design media and respond to immediate problems based on unconscious mental processes, with no need for an explicit rationale. In contrast to the glass box view of design, the complexity of the designers' brain itself is the justification for the black box model of design (Jones 1992, 46–47). In this model, the designer is depicted as a magician that controls her actions unconsciously and respond to problems based on pattern recognition and experience. Architects often rely on professional experience with trial and error, with personal experiences of space, on the visual study of design precedents, or any other form of design knowledge that does not require formalization.

1.6.1. Reinforcement Learning

There are different computational methods for agent design based on a custom task environment. These methods rely on different ways of representing the agent, evaluating their performance in the task environment, and in different algorithms for improving them. This section addresses methods to train computational agents, which are included in the taxonomy of this research under the category behavioral learning (Figure 22). Particularly, it focuses on the branch of machine learning called reinforcement learning (RL).

In this section, we will describe elements of reinforcement learning that are directly associated with the research artifact. For a broader understanding of the field and its algorithms, see 7.7 Appendix G: Reinforcement learning. Reinforcement learning is the branch of machine learning that addresses sequential decision problems. In contrast to other branches of machine learning, which use a fixed dataset as the source for learning, RL relies on data produced by the interaction of two decoupled entities – an agent and an environment – over a sequence of timesteps. Typically, the RL agent learns how to take sequential decisions to achieve a goal based on the data generated by the feedback control loop with the environment (Figure 24).

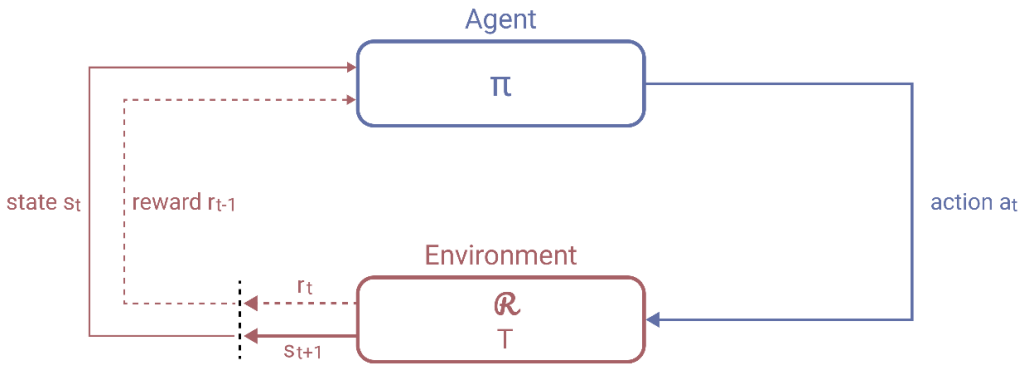


Figure 24 The agent-environment interaction in the case of a MDP. By Author, based on (Sutton and Barto 2018a, 48)[14].

A RL agent is the learner and decision-making entity encoded into a behavioral function commonly known as a policy $\pi(a_t|s_t)$. At every timestep, it uses sensors to receive two signals from the environment: an observation of the environment with complete or partial state information, and a reward signal, which are described next. The policy function maps the observation of the environment to an action in the action space ($a \in A$) to be devised by its actuators to affect the environment. This policy can be deterministic and output the best action, or stochastic, returning a distribution with the probability of selecting each of the available actions.

In RL, the environment is typically a simulation or even the real world, which provides feedback for the agent for the control task and for learning. It is an entity comprehending everything that is external to the agent and therefore cannot be directly controlled by the agent. The environment contains an internal configuration defined by a state in the state space ($s \in S$), which can be influenced by the actions of the agents and by internal dynamics over time.

The sequential decision process represented by the environment is idealized and modeled as a Markov Decision Process (MDP), from which precise theoretical statements can be made. The state of the environment at a given time step has all the necessary information to define the next transition – i.e., it has the Markov property (Sutton and Barto 2018a, 49). Thus, the discrete probability distributions that define the interactions at a given timestep are dependent only on the preceding state and action and not on any information from preceding timesteps. So, the changes in the environment can be represented by the dynamics function $P(s_{t+1}, r_t|s_t, a_t)$, which maps a pair with a given state and action to the probability of reaching ending up in the following state and receiving the respective reward.

From this dynamics function, many other functions of the environment can be decoupled, such as the transition function and the reward function.

---

[14] Notice that I opted for aligning the time index of the reward with the action that generated it and not with the state that was produced in the same environment processing. While this change is not intuitive, it simplifies the notation for the trajectory and for the description of the equations and algorithms.

The transition function $T(s_{t+1}|s_t, a_t)$ represents how the environment works, independently of the reward signal. It maps the action taken by the agent at a certain time step and its current state, to the probability if reaching a new state.

With the new state defined, the reward function $\mathcal{R}(r_t|s_t, a_t, s_{t+1})$ maps both states and the action of the agent to a reward signal, in the form of a scalar. This scalar reward signal sent by the environment to the agent is the essence of reinforcement learning as the term reinforcement refers precisely to the use of positive or negative reinforcements to condition behaviors. It is a very general instructional feedback that can potentially enable the agent "(…) to specialise to the problems it is currently facing, to continually verify and correct the most pressing holes in its knowledge, and to find new behaviours (…)" (Silver et al. 2021, 11).

In the ideal case, the MDP of the environment can be defined by the 4-tuple $(S, A, T, \mathcal{R})$. If the MDP is discrete it can also be represented by a discrete graph. Each state is a node with outer edges connected to a set of nodes representing the legal or available actions. Each action node can have one or more edges connecting to different state nodes. If there is only one edge, the environment is deterministic, so taking an action a at a state s will always lead to the next state s'. If there are multiple edges, is stochastic, so the same action at a certain state can take to different states. However, the environment can be a partially observable MDP (POMDP) – i.e., the agent does not observe the environment state s but an observation o with partial information about s. So, the POMDP adds an observation function O and observation space Ω to the MDP, resulting in the 6-tuple $(S, A, T, \Omega, O, \mathcal{R})$.

1.6.2. Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a subfield of machine learning that incorporates deep learning to solve problems that are too complex for traditional reinforcement learning algorithms. The recent success of deep reinforcement learning (DRL) in games, robotics and multi-agent navigation suggests that it is a promising field for training spatial agents that can generate spatial patterns and satisfy goals.

In RL, the elements of the MDP, such as the state and the state-action pairs (actions available at every state state) or agent-related functions, such as the $\pi(s)$, are represented as tables. This approach is feasible for fully observed and low-dimensional domains. In contrast, DRL relies on neural networks as a function approximator to represent the agent policy or other function used during learning. So, instead of storing every state or state-action values of the function, like in tabular methods, the algorithm only stores the description of the neural network and manipulates a vector $\theta \in \mathbb{R}^d$ with the learnable parameters. So, even if the size of the state and action spaces increase or even become infinite, the RL algorithm will only change a fixed-size vector θ.

Neural networks are also good for generalization. When the parameters of the neural network are updated to match a specific data, the change also affect how the neural network will behave with different data. So even if the space of the MDP is too large to be explored by the algorithm or if the agent does not have access to part of the representation, with a good distribution of samples,

a neural network can generalize the knowledge acquired in training to unknown situations (Sutton and Barto 2018a, 197). This is possible because neural networks have an internal representation that can learn a hierarchy of features, which can be fine-tuned during training by optimization algorithms using the gradient extracted with the backpropagation algorithm.

1.6.3. Multi-agent RL

There are cases where the hypotheses about the problem, such as in agent-based modeling, or requirements from the real world, such as limited computation or limited sensor input, require distributing the agent computation and constraining the structure of interactions. This multi-agent setting is part of the fields of multi-agent reinforcement learning (MARL) and the recent deep learning variant multi-agent deep reinforcement learning (MADRL). There are multiple dimensions to define decision models for MARL, which include aspects such as the observability, the heterogeneity of the agent, the cooperative or competitive nature of the agents, the simultaneity of the actions, and the sharing of rewards. In a decentralized setting the agents typically have access to a partial observation of the state environment. If the multiple agents are heterogeneous, they can potentially have different observation spaces Ω_i and action spaces A_i. Otherwise, they can share the same spaces.

In terms of multi-agent training, this research focuses in the use of parameter sharing (Gupta, Egorov, and Kochenderfer 2017; Justin K. Terry et al. 2021). In the parameter sharing approach, all the agents share the same individual policy, so the other agents are not interpreted merely as part of the environment. To ensure that a single policy can express a diversity of actions and agent types, the information in the observation is customized for the agent, which can include part of its own internal state, its id, or even global information. Parameter sharing is a simple approach that enables the use of single RL algorithms, in the observation and action spaces of the individual agents, with minimal modifications. The use of the shared policy enables training it with the data from all the agents, provides some level of centralization in the learning that can potentially mitigate non-stationarity, accelerate training (Justin K. Terry et al. 2021), and lead to emergent cooperative behavior (Gupta, Egorov, and Kochenderfer 2017).

This approach is distinct from the trend of design generation with RL that emerged in the computational design community, in parallel to the development of this research[15]. This trend relies on a single RL agent to build structures and assemblies for speculative (Wang and Snooks 2021b; 2021a; Chien-hua Huang 2021), building (Akizuki et al. 2020; Wibranek et al. 2021), architectural (Hosmer et al. 2020; Mintrone and Erioli 2021), or urban forms (Song and Yuan 2021; Chenyu Huang, Zhang, and Yao 2022) that satisfy performance criteria. Besides focusing on a single RL agent, most of these examples use RL as a black box mechanism to search for a design solution, so they do not address the problem of an interactive simulation either.

[15] For a more detailed description of this trend, see **7.7.8 A brief overview of reinforcement learning in architectural design** in the appendix.

2. Designing and implementing the design artifact

2.1. Introduction

While the proposed framework can support different approaches to spatial synthesis, this chapter presents the design and implementation of a design artifact based on ABM and MADRL that supports real-time, sequential, and granular construction of the design alternative with reduced assumptions about setup and order of the solution procedure. The problem is defined as a sequential decision process of an ecosystem of N agents sharing the same environment. Formally, this decision model is a variation of a partially observable stochastic game defined by the tuple $(S, N, A, T, \Omega, O_i, \mathcal{R}_i)$, where:

- S is the set of possible states for the environment
- N is the number of agents
- A is the action space for the agents
- T is the transition function
- Ω is the observation space for the agents
- O_i is the observation function that is customized to the different agents
- $\mathcal{R}_i$ is the reward function that is customized to the different agents

The multiple agents observe, act in the environment together, and receive individual observation and reward signals. Notice that instead of separate reward functions, there is a single function that is parameterized to support heterogeneous goals and behavior. This approach assumes that a single policy and a single reward function are enough to model the range of behaviors in the system.

This chapter is organized in the following sections

- Representation discusses the use of a stack of grids as the base for the discrete representation of the environment, for the configuration of spatial agents as polyominoes, and its consequence for the agent's observations of and actions in the world.
- Simulation defines the basic elements of the environment as a simulation engine, such as its parameters, the transition function T, the different spatial objectives, and how these objectives can be combined to form a parameterized reward function $\mathcal{R}_i$ for each agent.
- Implementation describes three implementations of the ideas in this chapter into research prototypes. This comprehends aspects such as how the observation function O_i manages observations o_i for N agents dynamically during the simulation, the reinforcement learning algorithms, the learning models, and the training process.

2.2. Representation

The representation of architectural space (environment and agents) in the design artifact is based on overlapping grids of information. The cartesian grid is a representation that has been widely adopted in the early days of space planning (W. Mitchell 1977, 161–68, 458–68) and in the development of recent agent-based models (Meyboom and Reeves 2013; Fernando 2014; Guo and Li 2017; Narahara 2017). The grids provide a very flexible spatial representation that support an architecture based on an ensemble of discrete entities that encapsulate contextual design information (Carpo 2017; Retsin 2019). Besides, they can be extended to 3D version (voxel grid). However, the general disadvantage of using multi-dimensional arrays of multi-layered spatial grids is that they bias the representation and operations by the horizontal and vertical relationship between cells, so there is information loss for shapes that are not aligned with the grid axes.

From the perspective of a computational data structure, a multi-layered grid is a discrete, homogeneous, and structured spatial representation that can be interpreted as multi-dimensional arrays. Multi-dimensional arrays support

- efficient memory allocation because they occupy a contiguous one-dimensional segment of computer memory
- efficient computation with techniques such as vectorization, multiprocessing, and multithreading
- spatial queries such as convolution/cross-correlation, neighborhood detection, adjacency, distance, etc.
- efficient computer vision, image processing, and deep learning algorithms and models, such as of convolutional neural networks
- the integration and augmentation of the information into the same structure, which increases the opportunities for identifying recurrent pattern in a problem

2.2.1. Environment as space

In terms of space, the environment is a grid with cells in two states: empty or obstacle. That allow the representation of existing buildings and elements of the landscape as constraints. Besides, it enables the designer to artificially impose spatial constraints to the spatial synthesis. The environment grid S has shape $(w_{env}, h_{env}, d_{env})$. While $w_{env} \times h_{env}$ defines the size of the grid, d_{env} defines the number of layers of information. The idea is that any information layer can be discretized and stored in an array of shape (w_{env}, h_{env}).

The basic layer of information of the environment contains two set of cells: $empty_{env}$ and $obstacle_{env}$. Empty cells can be occupied by agents while obstacles cannot. More information can be added to additional layers, increasing the depth of the environment.

The environment contains the internal information of each agent, which is defined by parameters related to goals and by a multi-dimensional array with three dimensions $(w_{env}, h_{env}, d_{agent})$. The

spatial layers of this array can be customized according to the implementation of the prototype. In general, it contains the representation of the agent cells, the different utility representations used in the problem (check **2.3.2**), and potentially auxiliary layers for training.

2.2.2. Agent observation

The spatial partitions of the design are defined by polyominoes with no holes (PnH). A polyomino is a set of grid cells that share edges, resulting in a connected set of cells, based on the von Neumann neighborhood. The PnH presents some interesting properties. It forms a polygon with orthogonal edge boundaries and avoids the containment of one agent by another. The PnH discretizes different geometric entities in a homogeneous representation, which can approximate any polygon, such as orthogonal and non-orthogonal rectangles, conventional room shapes (rectangles, L-shapes, U-shapes, T-shapes, etc.), unconventional and irregular shapes, or free forms such as amoeba-like shapes (see Figure 25). Therefore, it can form custom diagrams of floorplans, comprehending organizations such as tight packing and loose packing.

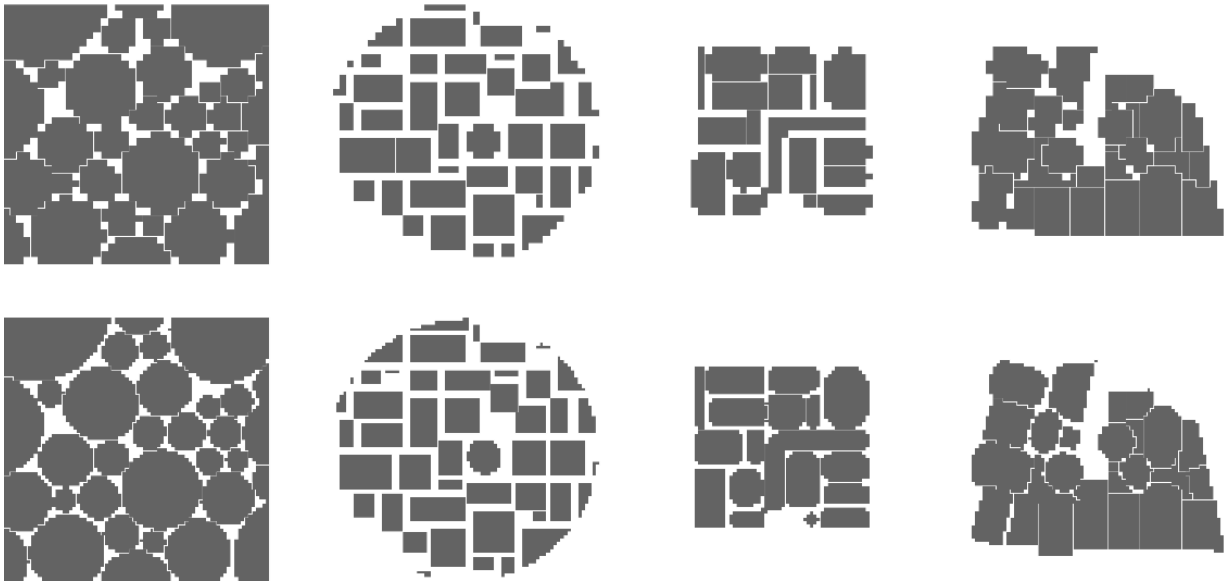


Figure 25. Examples of four real floorplans discretized into two levels of grid resolution. By Author.

The quality of the approximations depends on the resolution of the grid, which incurs to the tradeoff between shape information and the design space of the agent. The adoption of high-resolution enables complex forms but results in a very large space for design space exploration. Low-resolution simplifies the design space but imposes a considerable loss of shape information.

Each agent i is a specific PnH_i, which can represent some form of spatial boundary, such as a room, and induce different types of cells (see Figure 26):

- $nonadjacent_i$: cells outside of the von Neumann neighborhood of the PnH
- $adjacent_i$: cells in the von Neumann neighborhood of the PnH.

- $bridge_i$: subset of $adjacent_i$ containing cells that bridge the PnH_i – i.e., that if expanded, create a hole in it.
- $surface_i$: internal cells that if eliminated will not affect the integrity of the PnH.
- $structure_i$: internal cells that if eliminated will create a hole or divide the PnH_i.
- $body_i$: $surface_i \cup structure_i$.
- $outside_i$: $adjacent_i \cup nonadjacent_i$.

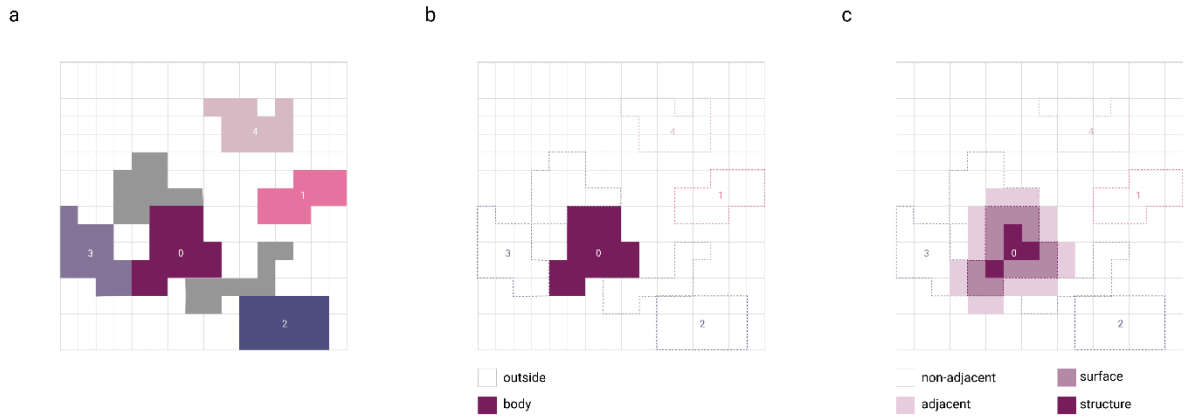


Figure 26 Classification of cells of agent 0: a) visualization of basic configuration; b) body and outside cells of agent 0; c) non-adjacent, adjacent, surface, and structure cells of agent 0. By Author.

For the characterization of these cells, I opted for using custom convolutions to take advantage of vectorization and parallelization. The representation of the body cells of the agent i is used as the input for the algorithms that detect the structure, surface, and adjacent cells of the agent. The algorithms apply Boolean operators and convolutions with custom kernels to detect the structure and offset cells. The surface cell can be derived from the subtraction of the structure from the body representation.

Inspired by agent-based modeling, the information available for the agent i is managed by the observation function $O(s_{t+1}, i)$, which maps the updated environment and agents' layers into a partial observation o_i in the space $\Omega \subset \mathbb{R}^{w_{per} \times h_{per} \times d_{per}}$. More specifically, this observation o_i is represented by an array of shape $(w_{per}, h_{per}, d_{per})$. It is defined by the agent's perception grid of shape (w_{per}, h_{per}) and by a series of d_{per} layers related to the environment, to the agent i, and to other inter-related agents, such as the l agents that share a relationship of adjacency with i and the k-closest and non-adjacent agents.

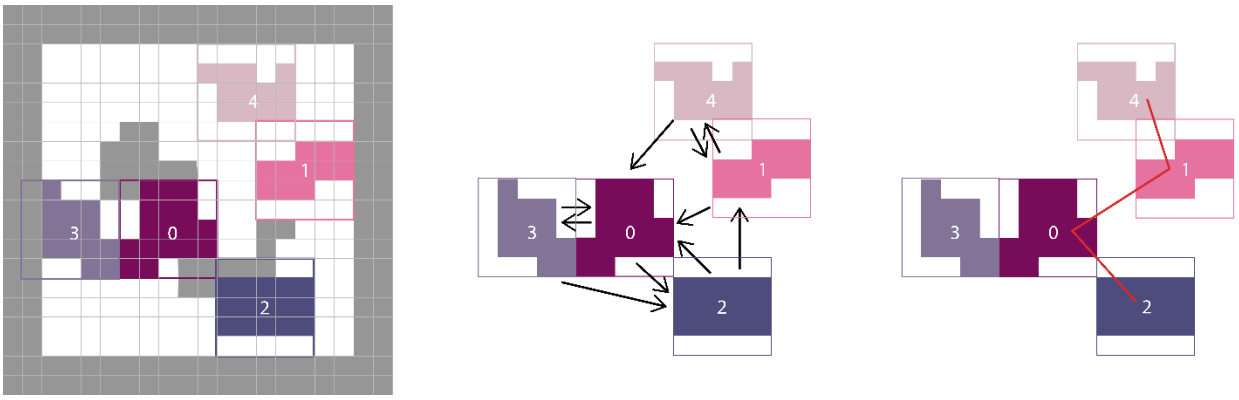


Figure 27. Left: simplified visualization of agents in the grid with obstacles. Middle: closest neighbours with k = 2: agent 0: [agent 3 and agent 2], agent 1: [agent 4 and agent 0], agent 2: [agent 0 and agent 1], agent 3: [agent 0 and agent 2], agent 4: [agent 1 and agent 0]. Right: example of adjacency goals with l = 2: agent 0: [agent 1, agent 2], agent 1: [agent 0, agent 4], agent 2: [agent 0], agent 3: [], agent 4: [agent 1]. By Author.

The perception grid of agent i is a grid restricted by the inequality ($w_{per} \leq w_{env}$ and $h_{per} \leq h_{env}$). If the 2-D dimension of the perception grid is smaller than the dimensions of the environment grid, it will be located on the environment based on the current centroid of the agents' PnH (see examples in Figure 28). This agent-centric observation only depends on the current configuration of the neighborhood of agent i, so it can support a computational method that is adaptable to problems with different environment sizes and with variable number of agents in a simulation. In this case, the environment needs enough padding with obstacles to prevent access to cells outside of the environment grid. The resulting observation o_i is a multi-dimensional array that will described based on the specific implementation in the next sections.

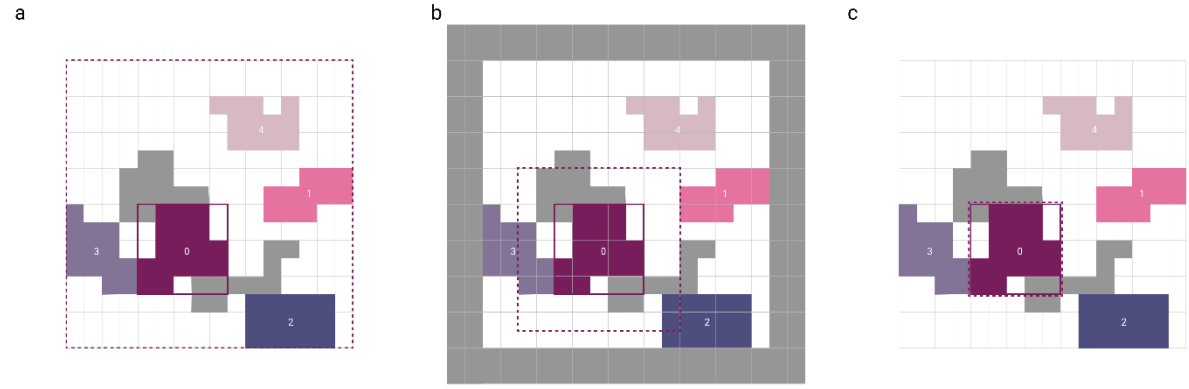


Figure 28 Variations of perception grid (dotted line) for agent 0, with fixed action grid (5, 5) (continuous line): a) perception grid is not centered to the agent and it is equivalent to the environment grid (20, 20); b) perception grid is centered and has shape (7, 7); c) perception grid is equivalent to the action grid (5, 5). The case c is just displayed for completeness as it overly restricts the agent's perception. By Author.

This definition of an agent enables the implementation common properties in complex systems, such as building blocks and hierarchies. For example, each agent can be interpreted as a microenvironment, by converting the $body_i$ to $empty_{env}$ and $adjacent_i$ to $obstacle_{env}$. This enables hierarchical relationships and the representation of different aspects of design, such as

zoning, building layout, and floorplan layout.

2.2.3. Action space

The basic actions available for the agents are (1) single-cell expansion, (2) single-cell retraction and (3) no-action. The legal set of cells the agent's expansion and retraction is defined by the actions that are not blocked by an obstacle of the environment, preserve the PnHs of all the agents and are inside the action grid (see Figure 29 - row 3).

The action grid is a subset of the perception grid, and it defines the cells that the agent can potentially act on. As the perception grid is itself a subset of the environmental grid, the relationship between the 2-D shapes of these grids is restricted by the inequalities $w_{act} \leq w_{per} \leq w_{env}$ and $h_{act} \leq h_{per} \leq h_{env}$. If the dimension of the action grid is smaller than the environment grid, it will be located on the environment based on the current centroid of the agents' PnH (see examples in Figure 28). Therefore, as the agent expand the PnH into a certain direction, it also moves the centroid and, consequently, the action grid. Notice that in this case, the environment needs enough padding with obstacles to prevent access to cells outside of the environment grid.

The action grid constraints the action space of a single agent, which is a vector of size $w_{act} \times h_{act} + 1$. The action that extrapolates the action grid dimension is the no-action, which is spatially represented as the cell to the bottom left of the action grid. The reason for the structured action space is both to limit the number of actions to a feasible size and to enable its representation as the output of a neural network to be trained.

Besides, the cells of the action grid are not always available for expansion or retraction. For each possible polyomino in this grid, there will be a set of cells that will preserve the consistency of the PnH (see Figure 29 - row 4). In this sense, the legal set of cells the agent's expansion and retraction is defined by the current classification of the cells (see Figure 29). More specifically, the set of cells that can be modified at each step is defined by:

- $expansion_i = (adjacent_i - bridge_i - structure_{neighbors} - obstacle_{env}) \cap action\_grid_i$
- $retraction_i = surface_i \cap action\_grid_i$

The basic actions are building blocks that can be combined to form complex interplays such as blocking, pushing, pulling, or attraction. For example, if the agent eliminates all the cells of its PnH using retraction, it then can jump to any empty cell inside the current action grid, using a single expansion. The agent also has the option of not executing any action this turn.

The number of available actions c for each agent ranges from one to $w_{act} \times h_{act} + 1$. The former indicates the situation where all the cells on the board are unavailable – i.e., when they are either obstacles or agent structure –, so the agent only has the option of a no-action. The latter indicates the situation where agent can select the non-action or occupy any cell in the action space, which is the case of the empty agent or any agent with size one or two over an action space with available cells. In this case, the joint state is susceptible to a combinatorial explosion based the n of agents

and on the constraints on the action space. This means that for centralized control, the number of actions at a certain step can range from 1 to $(w_{act} \times h_{act} + 1)^n$. For example, in the case of 16 agents with a 5×5 action space, the upper bound for this number is 26^{16}.

2.3. Simulation

The environment is not only a spatial representation of a site but also a simulation engine that contains and manages information about obstacles, agent parameters, agent state, and agent observation. During training, it is a source of a diverse and large number of experiences that allow the agents to acquire general knowledge about the possible future situations. After training, the environment should run custom design cases that replicate real-world conditions and should enable the designer to specify and update the specifications of the problem on the fly.

Besides containing a series of functions to manage agent observations and actions, the environment is composed by two basic functions: a transition function and a reward function. With access to these functions, an environment can receive an input vector with agents' actions to update its internal configurations and then return the resulting observations and the reward signals to the agents.

2.3.1. The transition function

The changes in the environment rely on the transition function $T(s_{t+1}|s_t, \boldsymbol{a_t})$ that maps a pair with a given state of the agents and their actions to the probability of reaching a following state. The decision process is Markovian, so, the discrete probability distributions that define the interactions at a given timestep are dependent only on the current state and actions and not on any information from preceding timesteps.

The vector $\boldsymbol{a_t}$ is present in the input because T is synchronous. That means that the transition considers all the actions at once to take one step in the environment. The use of synchronous/simultaneous plays simplifies the training with variable number of agents and reduces the pre-defined assumptions about the generative procedure from the perspective of the user. However, considering that the agents dispute cells in the environment, an ordering strategy of the agents for the internal environmental update is necessary. As that the number of agents in the training environment is variable, the agents are sorted based on their current performance (best agent moves first) instead of using ids. Therefore, for the same input of environment state and set of agent actions, T is deterministic, so it will always map to the same next state.

2.3.2. Spatial objectives and reward function

The goals of the agents are modeled by functions f^j that return a scalar value in the unit interval representing the performance of the agent. These goal functions are parameterized by the agents so they can be dynamically modified during the generation of a solution to reframe the design problem on the fly. For each goal function f^j, there is a complementary spatial function g^j that provides a perceptual map related to the agent's parameterization and performance w.r.t a goal j in the form of an array of size (w_{env}, h_{env}) – Table 5. These maps augment S to facilitate training a single policy that can address multiple goals. For the design artifact, there are goal functions to

indicate area, smooth adjacency, not-folding (stimulates shapes with few concave corners, such as rectangle, L or U shapes), and daylight access (stimulates agent cells adjacent to open spaces).

To provide a dense signal related to adjacency, the function g^{adj} labels the cells in the environment based on their distance to the PnH_i. The values are 1 for the cells that are inside or adjacent the PnH_i, then they decay over the Manhattan distance until they reach 0 at the maximum distance ($dist_{max}$) beyond the adjacent cells and on – see Figure 34: row 4. The function f_{min}^{adj} evaluates the maximum values in g^{adj} that intersect with PnH_i for all the agents that share a relationship of adjacency with it (the set D_i). Then, it returns the minimum value in that set, which represents its worst performance in terms of adjacency. f_{mean}^{adj} returns the mean value in that set.

The function f^{area} is a piecewise linear function based on the relationship between the agent area and its target area. It returns 1 when the agent reaches the target area. This value linearly decreases when the area of the agent moves away from the target. It becomes 0 when its area is 0 or when it is twice or more than the target area. The spatial representation g^{area} labels the cells of the action grid with the ratio between the agent and target area shifted by one and bounded by 1. It labels the action grid with -1 when the agent has area 0, 0 when it reaches the target, and 1 when it has twice the target area or more.

To prevent irregular shapes, the spatial function g^{fold} assigns the value of $1/\,fold_{max}$ to the concave corners of the agent's PnH_i that are surrounded by two adjacent edges (a L-fold) and assigns $2/\,fold_{max}$to the ones with three adjacent cell edges (U-fold) – see Figure 34: Row 5. The parameter $fold_{max}$ is controlled by the designer to define how regular the shapes should be. The function f^{fold} returns a value of one minus the sum of all the values in g^{fold}, bounded by zero to prevent negative scores. The functions fold_L and fold_U are indicators that return a zero array of shape (w_{env}, h_{env}) with ones in the position of the respective folds.

Finally, for daylight, we assume that cells in the PnH_i that are adjacent to an empty cell in the environment have daylight access. This is based on a common daylighting rule of thumb adopted in architectural manuals that state that the depth of daylit area in a room is defined by a factor that is proportional to the size of the windows head and depends on orientation or on the use of blinds, light shelves, etc. For example, O'Connor et al. (1997, 13) state that for a South facing façade, this factor is 1.5 for a standard window and between 1.5 and 2.0 for a window with a light shelf. Reinhart uses daylight simulation for a variety of climates, orientations, facade geometries, and usage patterns to test the effectiveness of this rule-of-thumb (2005; 2014). He claims that the daylit factor lies in the range of 1 to 2 times the size of the window head height, which is largely influenced by the glazing type and the target illuminance level in the space. Without shading devices, the ratio of the factor increases to 1.5 to 2.5 (C. F. Reinhart 2005, 7; C. Reinhart 2014, 103–4).

Thus, the daylight function f^{lit} divides the percentage of agent cells that have daylight access by the target percentage and bounds the result by 1. $litcells(i)$ is an indicator function that returns

a zero array of shape (w_{env}, h_{env}) with ones in the position of the daylit cells. g^{lit} returns a map with the daylit cells of an agent tagged with the value of f^{lit}.

Table 5 goal and spatial functions

	Function	Specifications
Smooth adjacency	$f^{adj}_{mean}(i, dist_{max}, c) = \frac{1}{\lvert D_i \rvert} \sum_{d \in D_i} max(g^{adj}(d, dist_{max}, c) \cap body_i)$ $f^{adj}_{min}(i, dist_{max}, c) = \min_{d \in D_i} \left(max\left(g^{adj}(d, dist_{max}, c) \cap PnH_i\right)\right)$	D_i is the set of l agents that share adjacency to agent i clip constraints the values in the map to a minimum and a maximum c is a parameter for the exponential decay of the distance values $dist_{max}$ is the L1 distance considered for adjacency L1distancemap is a distance transform function based on L1 distance
	$g^{adj}(i, dist_{max}, c) = \left(\frac{dist_{max} - \text{clip}(\text{L1distancemap}(PnH_i) - 1, 0, dist_{max})}{dist_{max}} \right)^c$	
Area	$f^{area}(i) = \begin{cases} \frac{area_i}{target_i} & if\ area_i \leq target_i \\ 2 - \frac{area_i}{target_i} & else\ if\ area_i \leq 2\ target_i \\ 0 & else \end{cases}$	$area_i$ is the number of cells in the PnH of an agent $target_i$ is the target number of cells for the PnH of an agent $actiongrid_i$ is a grid with ones in the action grid of the agent
	$g^{area}(i) = \min\left(actiongrid_i \frac{area_i}{target_i} - 1, 1\right)$	
Not-folding	$f^{fold}(i) = \max(1 - sum(g^{fold}(i)), 0)$	fold_L is a function that returns a grid with ones in the place of the L-folds fold_U is a function that returns a grid with ones in the place of the U-folds $fold_{max}$ is a parameter that restricts the number of acceptable folds
	$g^{fold}(i, fold_{max}) = \frac{\text{fold}_L(i)}{fold_{max}} + \frac{2\text{fold}_U(i)}{fold_{max}}$	
Daylight	$f^{lit}(i, lit_{\%}) = \min\left(\frac{sum(lit\_cells(i))}{area_i}, lit_{\%}\right) / lit_{\%}$	$litcells$ is a function that returns a grid with ones in the place of daylit cells.
	$g^{lit}(i, lit_{\%}) = litcells(i) f^{lit}(i, lit_{\%})$	

The utility $f(s, i)$ is a function that combines a subset of these goal functions with a weighted average or with a multiplication of the terms to incentivize simultaneous exploration of goals. For each agent i, there is a parameterized reward function $\mathcal{R}(s_t, s_{t+1}, i)$ that maps the consecutive environment states and the agent information to dense scalar reward signals, to condition heterogeneous behaviors and accelerate training. $\mathcal{R}$ is defined as the difference between the utility values of two consecutive states according to the objectives and the agent parameters: $f(s_{t+1}, i) - f(s_t, i)$.

Also, the objective functions have parameters. For example, the adjacency goal relies on the

specification of a parameter of exponential decay c that changes non-adjacent agents and of the set of neighbor agents that defines an adjacency graph for the agent i. These and other parameters can be potentially modified by the designer during the generation of a solution to support reframing the design problem during the exploration of solutions.

2.3.3. Environment parameters

A new environment is initialized with the following parameters

- utility function: function f that defines the relationship between the different spatial goals (f^{adj}, f^{area}, f^{fold}, and f^{lit}) to map the state of the agent to a utility value and derive $\mathcal{R}^{\theta}$.
- N agents: integer that defines the number of agents in the simulation
- action shape: tuple (w_{act}, h_{act}) that defines the shape of the action grid for each agent (see 2.2.3). It is also the base to define the perception grid (h_{per}, w_{per}).
- board grid is an array with custom size (w_{env}, h_{env}) indicating cells that are either empty or obstacles
- k is the maximum number of the closest neighbor agents detected by the agent.
- l is the maximum number of agents that share a relationship of adjacency.
- goal area is a list of positive integers with the target area for all the agents
- adjacency is list containing the ids of the agents that share a relationship of adjacency for each agent in the environment
- max step is the maximum number of steps used for training episodes
- limit folding is the parameter for maximum allowed folds used for f^{fold}
- p daylit is the parameter of daylight percentage used for f^{lit}

2.4. Prototyping

This section introduces three implementation of the research prototype – P1, P2, and P3 –, which were developed between 2020 and 2022. This section for each prototype contains the specifications for the agent observation, training settings, and variations in terms of parameter sharing and multi-agent RL techniques. These iterations were developed sequentially, so the training process of one iteration was used not only for testing its feasibility but also for identifying opportunities for the following implementations. The development of the different versions of the prototype followed the specifications synthesized in Table 6 and Table 7. Besides, P2 and P3 were trained with multiple spatial goals, both to check the technical feasibility of the training and to study the architectural morphology. The latter refers to notion of morphology proposed by (Steadman 1983), which considers the study of possible spatial configurations given an certain design constraints and order.

Table 6 RL, environment, and training specification of the different iterations of the research prototype.

		P1	P2	P3
RL	RL algorithm	**DDQN**	**PPO**	**PPO**
	Core Python Libraries	PyTorch	PyTorch and Tianshou 0.3.0	PyTorch and Tianshou 0.3.0
	γ	0.92	0.99	0.99
	buffer size	10000	10000	10000
	initial ε	0.9	-	-
	minimum ε	0.1	-	-
	τ	0.02	-	-
	value function weight	-	0.5	0.5
	entropy weight	-	0.01	0.01
	epsilon clip (ϵ in L_{clip})	-	0.3	0.3
	max grad. norm. (clipping)	-	0.5	0.5
	GAE λ	-	0.85	0.85
	reward normalization	-	1	1
	value clip	-	1	1
Environment	n agents in training	6	16	16
	n agents in test	6	4, 8, 12, 16, and 20	4, 8, 12, 16, and 20
	max l adjacencies	3	4	4
	max k-closest neighbors	3	3	3
	action shape	(5, 5)	(5, 5)	(5, 5)
	perception shape	(20, 20) env. with pads	(15, 15) independent	(15, 15) independent
	action space	Discrete (26,)	Discrete (26,)	Discrete (26,)
	observation space	(20, 20, 7)*	(15, 15, 49)	(15, 15, 49)
	board initialization	(17, 17)	(8-31, 8-31)	(15-45, 15-45)
	board configuration in training	random obstacles	random obstacles	random carving
	board padding	(3, 3, 3, 3)	(7, 7, 7, 7)	(7, 7, 7, 7) or (9, 9, 9, 9)
	steps per episode	64	128	256
Training	optimization algorithm	RMSProp	Adam	Adam
	learning model	DCNN	CNN (shared) + FC (heads)	CNN (shared) + FC (heads)
	learning rate	0.0001	0.0003	0.0003
	Loss function	Smooth L1 Loss DDQN TD	PPO policy and value losses	PPO policy and value losses
	batch size	64	64	64
	n of training env.	1	64	32
	n of test env.	50	32	8
	epochs	1000	300	200 (one with 300)
	episodes per epoch	50	512	512
	augmentation	8 (symmetry)	-	-
	repeat per collect	-	2	2
	agent sample per epoch	19,200	262,144	262,144
	agent samples produced	19,200,000	78,643,200	52,428,800 (78,643,200)
	passes over agent samples	153,600,000	157,286,400	104,857,600 (157,286,400)
	* the agent has access to the feature maps of the other agents during the forward pass of DCNN			

Table 7 Goals used in the different iterations of the research prototype.

		P1	P2	P3
Fixed	Max distance for distance decay ($dist_{max}$)	16	45	45
	Exponential decay for distance (c)	10	30	30
Goals	$f_{min}^{adj} f^{area}$		X	X
	$f_{min}^{adj} \frac{(f^{area} + f^{fold})}{2}$ with $fold_{max} = 3$		X	
	$f_{mean}^{adj} \frac{(f^{area} + f^{fold})}{2}$ with $fold_{max} = 5$	X		
	$f_{min}^{adj} \frac{(f^{area} + f^{fold})}{2}$ with $fold_{max} = 5$		X	X
	$f_{min}^{adj} \frac{(f^{area} + f^{lit})}{2}$ with $lit_{75\%}$		X	
	$f_{min}^{adj} \frac{(f^{area} + f^{fold} + f^{lit})}{3}$ with $lit_{50\%}$			X
	$f_{min}^{adj} \frac{(f^{area} + f^{fold} + f^{lit})}{3}$ with $lit_{75\%}$		X	X

2.4.1. P1

The primary goal of P1 is to verify the feasibility of applying parameter sharing as a RL method for an agent-based model. More specifically, it intends to test the capacity of a RL-based system to generalize the knowledge acquired in a training setting to different design scenarios. Variations of the training algorithms, aspects of neural network architecture and training (activations, initialization, resources, batch normalization, etc.), and different resources to shape the agent representation were empirically tested before selecting the initial configuration of P1.

2.4.1.1. Agent observation

In P1, the perception grid has the same size of the environment grid, which enables the possibility of accessing the information from any part of the representation. Each agent is represented by an individual input with its individual information, which results in the observation o_i that contains four arrays: o_i^{state}, o_i^{adj} , o_i^{close}, and o_i^{mask}. The observation of each individual agents is passed to the model, which dynamically builds and propagates the neighborhood information (see **2.4.1.3**). As in the example in Figure 29, the multi-dimensional array o_i^{state} of shape $(w_{env}, h_{env}, 7)$ with state information about the agent i consists of:

- The environment grid with obstacles (w_{env}, h_{env})
- Internal state of agent i with cell classification with structure, surface, and offset (w_{env}, h_{env})
- the action grid (w_{env}, h_{env})
- g^{area} (w_{env}, h_{env})
- g^{adj} (w_{env}, h_{env})
- g^{fold} (w_{env}, h_{env})
- The agent cells scaled by the utility f (w_{env}, h_{env})

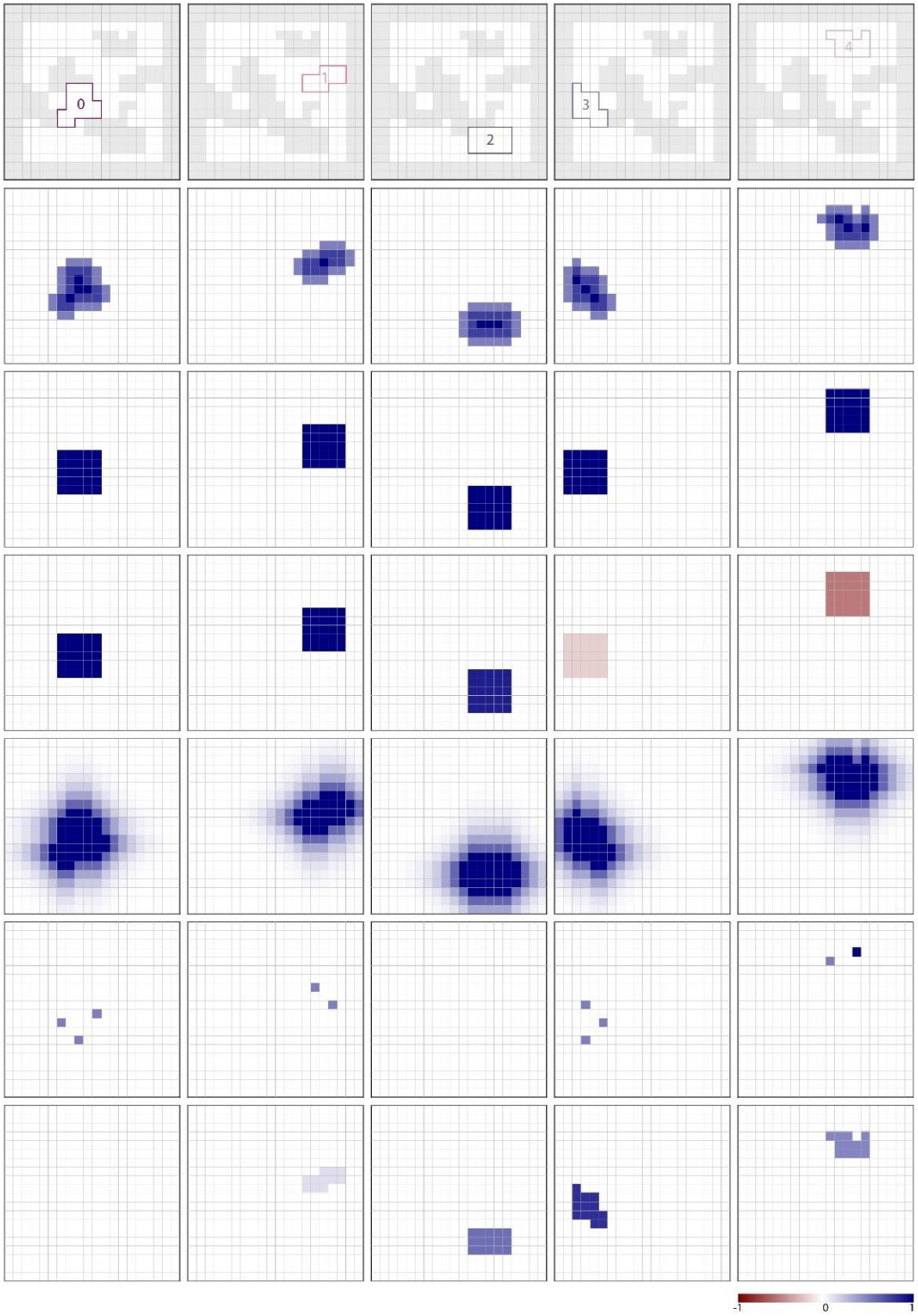


Figure 29: Row 1: visualization of obstacles (agents are added just for visualization); row 2: classification of the cells: structure (dark blue), surface (medium blue), and offset (light blue); row 3: mask with the action space of the agent; row 4: score of the agent with the respect to the area over the action grid: 0: (17/2), 1: (11/ 4), 2: (15/8), 3: (13/ 16), 4: (12/ 25); row 5: soft adjacency values based on Manhattan distance; row 6:indication of folding cells: L-folds (light blue), U-folds (dark blue); row 7: the current utility of the agent represented in the internal cells of the agent. By Author.

The vector o_i^{adj} indicates the agents that share a relationship of adjacency with the agent i. Its shape is (l) but it is latter processed to the shape of the convolutional blocks to work as a filter.

The vector o_i^{close} indicates the k closest neighbor agents detected by agent i. Its shape is (k) but it is latter processed based on the shape of the convolutional blocks to work as a filter.

Finally, the vector o_i^{mask} contains a mask with the legal action for agent i with shape $(w_{act} \times h_{act} + 1)$ It is used to filter the valid state-action estimates to be considered in the action selection algorithm.

2.4.1.2. RL algorithm: DDQN

Due to the discrete nature of the problem representation, I opted for using an off-policy, model-free algorithm called Double Deep Q-Network (DDQN) (van Hassel, Guez, and Silver 2016). DDQN is in the family of value-based methods, which relies on the alternate execution of two steps for learning: the computation of the value function for a given policy (policy evaluation) and the computation of an improved policy given the value function of that policy (policy improvement). DDQN uses two deep neural networks to estimate the Q-values of each agent. Hence, a policy derived from the current estimates is used to interact with the environment and collect experience, which is then used to update these estimates.

Algorithm 1 DDQN

1: Initialize two Q-networks with random weights
2: Initialize replay buffer D, and other parameters, such as γ, τ, α, and ε
3: For each episode do
4: Initialize environment
5: For each step do
6: Define Q^{θ} and Q^{θ^-} randomly
7: Choose a from an ε-greedy policy derived from $Q^{\theta}(z)$ using a mask for legal actions
8: Move to z′ and receive r
9: Augment (z, a, r, z') and store in D
10: If number of steps % training mod == 0:
11: Sample a batch $(Z, A, R, Z') \sim D$
12: Compute target $Q^{\theta}(z, a) = r + \gamma Q^{\theta^-}\left(z', argmax_a, Q^{\theta}(z', a')\right)$
13: Perform mini-batch gradient descent using L1 loss
14: Update network parameters: $\theta^- \leftarrow \tau * \theta + (1 - \tau)\tau * \theta^-$
15: Reduce ε based on schedule

In the steps 6-8, the agent interacts with the environment using an ε-greedy policy to generate observations – i.e., it selects a random action with probability ε or the argmax of action of the online Q-network (Q^{θ}) with a mask for legal actions otherwise. At the beginning of a training session, ε is a large value, so the policy explores random trajectories in the environment. As ε is reduced over training, the agent switches to the exploitation of the best estimates. The observations are augmented with rotation and reflection, then added to a large memory buffer D, from which they are later retrieved for training. The buffer randomizes the data, removing the

correlation between observation samples and reducing the variance for the updates. In step 9, the observations are augmented by rotation and mirroring before adding them to the buffer. This process increases the data by a factor of eight. In the steps 11-14, the algorithm uses the batch of observations (S, A, R, S') retrieved from D to define the target $r + \gamma Q^{\theta^-}\left(s', argmax_a, Q^{\theta}(s', a')\right)$. The target is used to updates the current estimate of Q^{θ} using a gradient descent variation called RMSProp. Then, Q^{θ^-} is slowly updated to approximate Q^{θ} using a step-size τ, which reduces the correlation between Q estimates and target values.

2.4.1.3. Learning Model: Dynamic convolutional neural network (DCNN)

The following neural network architecture was designed specifically for P1 (Figure 30). It was named dynamic convolutional neural network (DCNN), which supports training and simulations with flexible and variable number of agents. It uses the same model for all agents in the environment and captures the variable neighborhood and goal information by a composite architecture. More specifically, each agent is represented as part of a single neural network as a strand. For each of these strands, there are five main network blocks. To capture neighborhood information, the strands are dynamically connected between the blocks based on the topology of the closest neighbors and adjacency relationships provided by o_i^{adj} and o_i^{close}. In other words, DCNN consists mainly of CNN blocks that dynamically change its connections according to the topology of the adjacency and of the closest neighbors at each environment step.

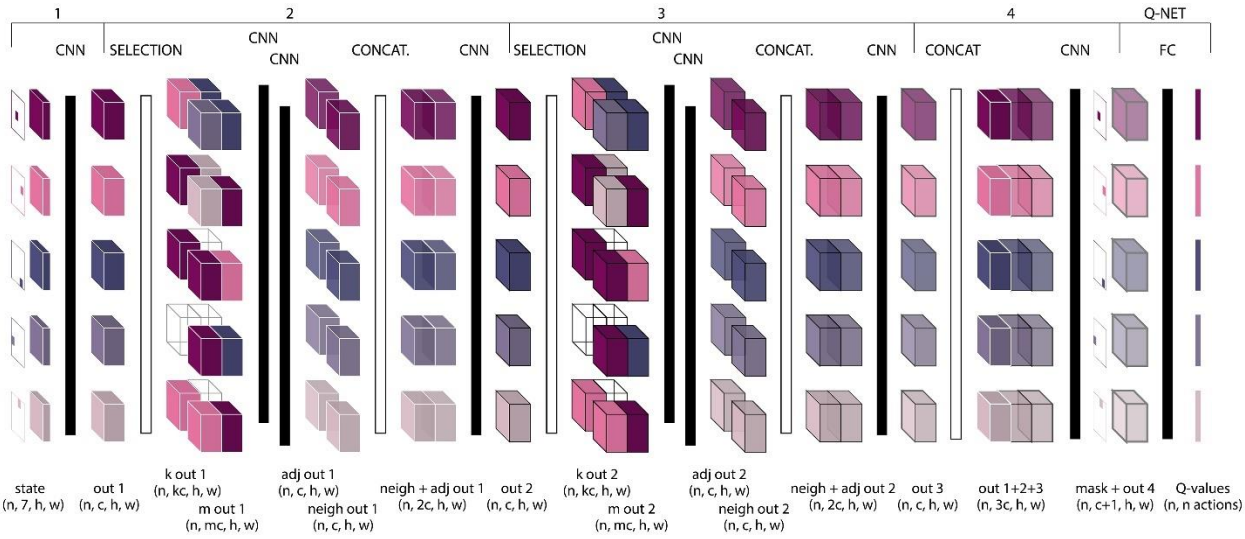


Figure 30 Simplified graphical representation of the layers, operations, and arrays of DCNN. Every CNN is composed of 2 convolutional layers with 64 filters of size 3x3, stride 1, padding 1 and a Leaky ReLU activation. The final FC network has 3 linear layers with 104, 52, and 26 neurons. Its two first layers use Leaky ReLU activation, while the last uses a hyperbolic tangent function (Tanh). By Author.

Over the forward propagation, the network captures a larger and larger receptive field with information of neighbors, the neighbors of neighbors, and so on. The strands of the network for each agent not only share weights but also gradients. Therefore, the backpropagation step of DDQN enables cooperative adjustment of weights between agents. As a result, while the number of closest neighbors (k) and of maximum connected agents (l) should be fixed, the number of

agents in the model can be changed during execution, by adding or removing strands.

DCNN estimates the Q-values after receiving three arrays as the input:

- The first block receives the array z_i^{state} with the state information for each agent, which is processed by a convolutional neural network to create a richer representation.
- The second and third blocks receive the output of the previous block for the respective agent combined with agents indicated by z_i^{adj} and z_i^{close} as the input.
- The fourth block uses a convolutional neural network on the combination of the resulting arrays from all the previous blocks.
- The fifth block is a Q-network that receives the resulting arrays from the previous block concatenated with the z_i^{mask} for each agent, then it computes the respective Q-values using fully connected layers.

2.4.1.4. Training

In the training setting, the environment is the problem generator. It creates six agents are randomly initialized and interact in randomly generated environments during multiple episodes of 64 steps. The reward signals are derived from the following spatial goal $f = f^{adj} \times 0.5(f^{area} + f^{fold})$.

After training, the prototype should enable designers to solve a large variety of design problems, so the RL agents must be able to adapt to changes by the designer without a significant drop in the performance. To prevent over-fitting the agents to a specific environment, the simulation engine randomly initializes the environment at every episode with the following steps:

- randomly initialize a new board grid of $(12, 12)$ with padding of 2, which results in the shape $(16, 16)$
- Populate the board with a random number of obstacles that are either a x-aligned series of cell, a y-aligned series of cell, or squares based of between one and three.

For the agents, the engine randomly

- initializes the area and adjacency targets
- Initializes the layers for the n agents and its initial random positions
- runs several steps with random actions so the agents settle in a random initial configuration

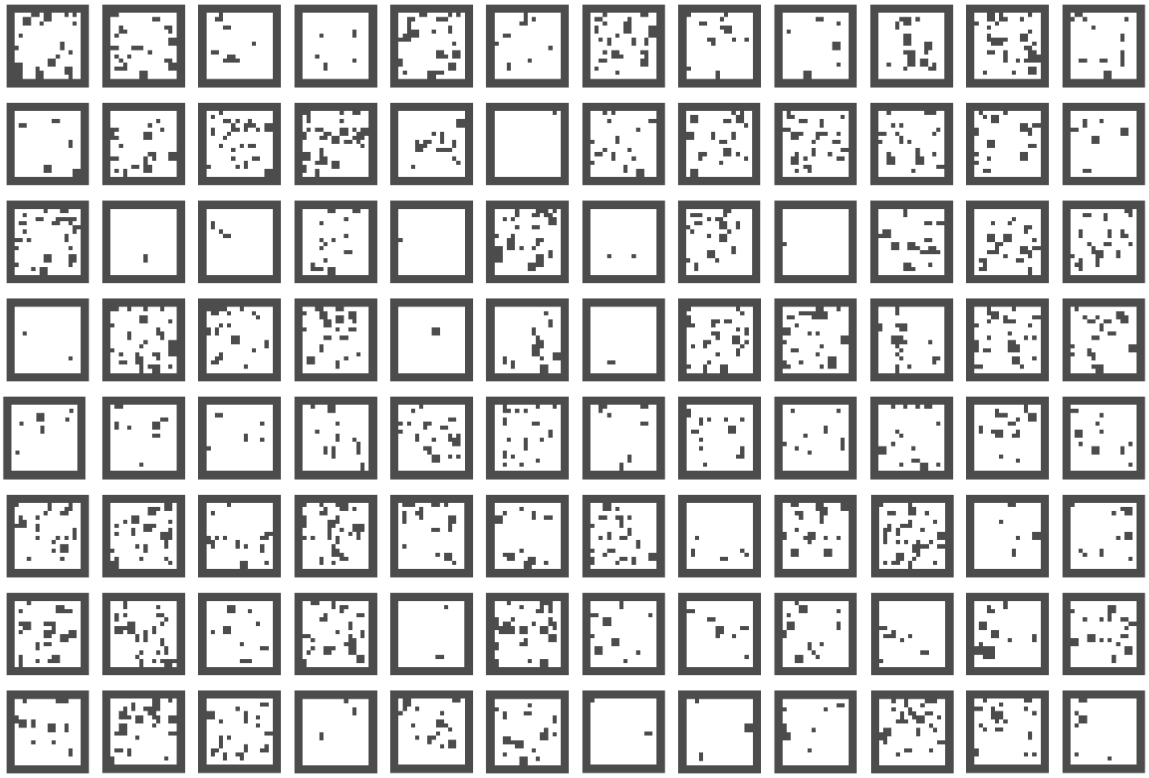


Figure 31 96 examples of random boards generated by the engine. By Author.

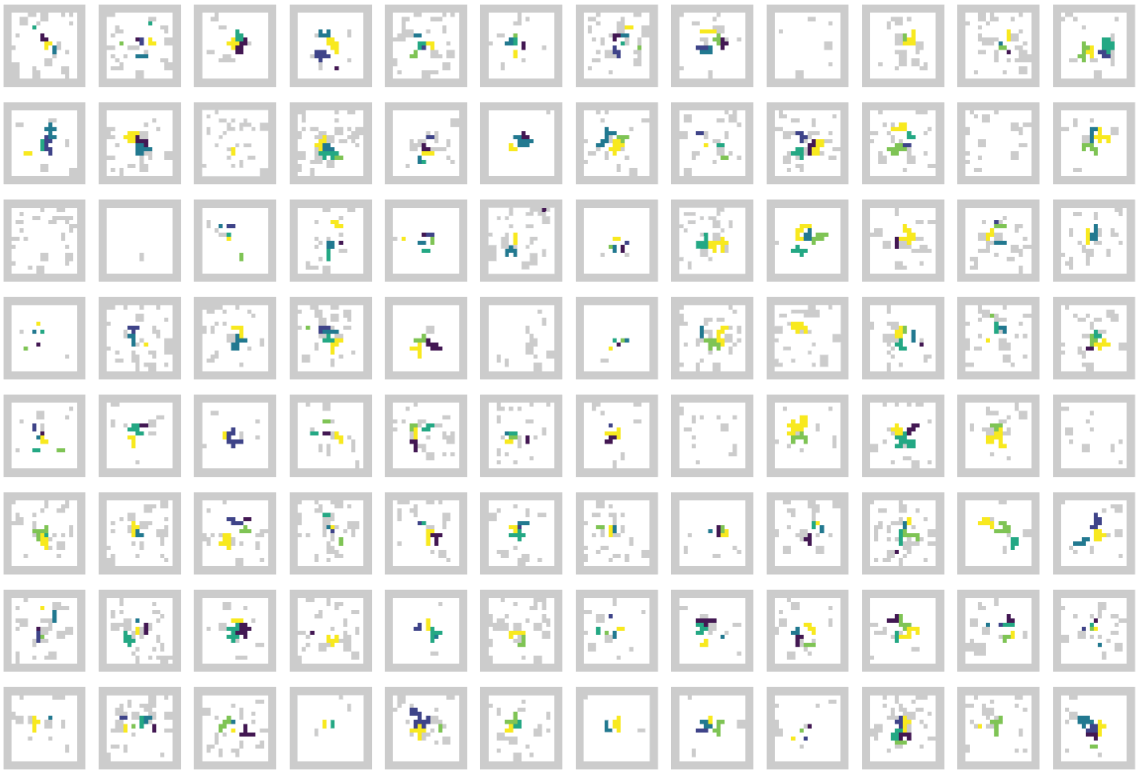


Figure 32 96 examples of agents initialized randomly by the engine. By Author.

This initial prototype relies on a very simple training setup to provide an initial proof that the method is feasible. For example, it limits the grid size to 12×12, it only uses 6 agents during training, and it only relies on one environment for training. The complete description of the training setup is included in Table 6.

The training graph for P1 (Figure 33) shows that the agents successfully improved their policy over time. In the first 10k episodes, this process is noisy, as the agents learn short-term policies, such as moving to the same side of the board and adopting the shape of parallel corridors to minimize distance. After that, the training becomes more stable, and the agents can converge to more successful policies incrementally.

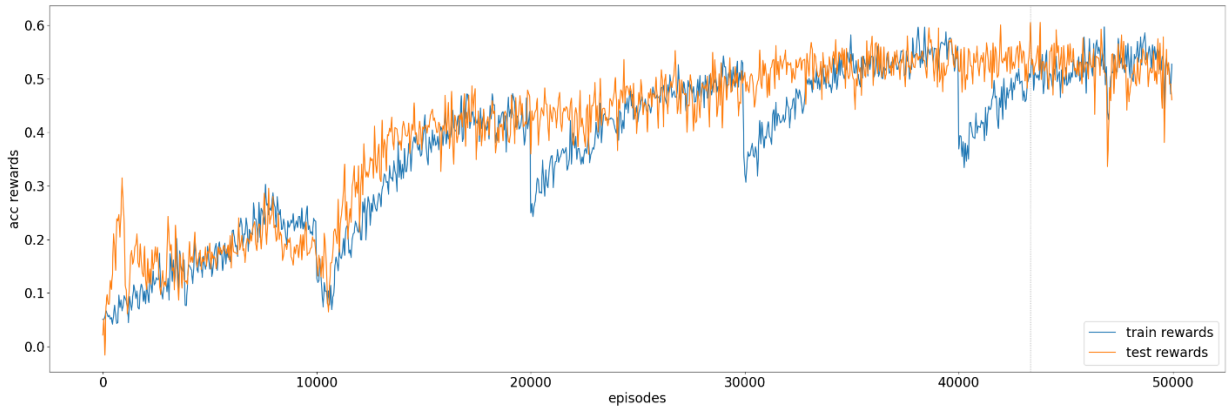


Figure 33 Cumulative rewards for the agents during training. The blue line is the average cumulative rewards of the six agents on the training episodes. Notice that the oscillations in the blue graph are the result of the custom schedule of reduction of $\boldsymbol{\varepsilon}$*. The orange line is average cumulative rewards of the six agents in testing episodes with greedy actions. The vertical dotted line at 43350 represents the best policy, which was adopted for the test in the design scenarios. By Author.*

2.4.1.5. Lessons learned

The most important finding is that agents successfully learned a spatial policy that handles multiple spatial goals in a fine-granular setting. However, there are some drawbacks in P1 related to model architecture and the RL algorithm.

On one hand DCNN enables the dynamic computation of neighborhood information both in the forward and back propagation of the neural network, which is an original and elegant solution for agent-based models with function approximation. On the other hand, DCNN imposes some obstacles for training. While most of the elements in the forward and backward passes can be parallelized, the dynamic reconfiguration of the neural network happens in runtime with serial computation, which slows down the training. The training was executed in a laptop with Intel® Core™ i9-9880H CPU @2.30 GHz, 2304 MHz, 8 cores, 16 logical processors, 32 GB (RAM), and NVIDIA GeForce RTX 2080 with Max-Q Design. The time was not recorded, but it took more than two weeks to run, considering the series of training sessions. Another limitation of DCNN is that it requires a fixed environment size, so it requires using the largest possible environment size as the grid size to comprehend all possible applications.

Model-free RL methods are data hungry. Considering the obstacles in terms of the dynamic

configuration and fixed environment size in DCNN, training can potentially benefit from an accelerated training with simpler architecture and logic of parameter sharing.

Finally, the RL algorithm depends on random action selection to incentivize exploration, which results in constant noise in the configuration of the agents and, consequentially, in the architectural layout. While the randomness DDQN could be potentially controlled manually or be associated with the performance to address exploration and exploitation, ideally the policy itself would take care of this process.

2.4.2. P2

After confirming the feasibility of RL as the method to design the agents, P2 was developed between 2020 and 2021. It overcomes the main bottlenecks of P1. The main strategy for P2 is to

- Use a different RL algorithm that makes training and execution more stable and efficient and ensures that the agents explicitly manage a stochastic policy to reduce the noise in the inference (see 3.2.1).
- disentangle the agents' frame of observation from the environment observation to enable the use of larger environments with flexible shape
- use a simpler parameter sharing approach and neural networks to allow faster training and inference

Besides, P2 also tests if the method can train agents that successfully generate architecture based on different spatial objectives. In the context of architectural design, this intends to investigate if learned policy conditioned by a specific goal and parameters can lead to the emergence of spatial types.

2.4.2.1. RL algorithm: PPO with clipped surrogate objective and Generalized Advantage estimation

P2 uses Proximal Policy Optimization (PPO, Schulman et al. 2017), an algorithm that modifies the original objective of an advantage actor-critic algorithm J to overcome some problems of stability, such as variance and performance collapse, and makes the training more sample-efficient. PPO uses the relative performance identity – i.e., the difference in performance between two policies – to ensure non-negative improvement in a conservative policy iteration (Graesser and Keng 2019, 165–94). In practice, this identity is approximated by using trajectories from the old policy adjusted with weights based on the ratio of action probabilities between the successive policies, which upweights the action that are more likely under the new policy. This creates a surrogate objective function based on importance sampling that has the same gradient as the original loss and guarantees monotonic improvement within an error bound.

Particularly, we opted for an PPO algorithm[16] with a surrogate objective J_{π}^{CLIP}, which uses clipping inside the objective to limit the ratio to an ε-neighborhood to guarantee monotonic improvement. The algorithm relies on Generalized Advantage Estimation (GAE), which uses a weighted average of the advantages for the different n-step returns to reduce its variance. There is also an entropy regularization term H is subtracted from the loss function to increase exploration in the training by incentivizing policies with more uniform distributions. Besides being a more stable approach in comparison to DDQN, it also has the advantage that the learned policy is stochastic, so it is not necessary to introduce noise for exploration.

Algorithm 2 PPO-Clip with multiple environments – based on (Graesser and Keng 2019, 177–78; Weng et al. 2020)

1: Initialize actor (θ_A) and critic (θ_C) networks with random weights
2: Initialize the buffer D, and parameters such as β, ε, and K
3: For epoch in K epochs do
4: For each training environment
5: Reset training environment
6: Collect an episode using θ_A and store in D
7: Compute $\pi'(a|s)$ with θ_A
8: Compute values V_{tar}^{π} with θ_C and trajectory
9: Compute advantages A_{GAE}^{π}
10: For each minibatch m
11: Compute new values V_{new}^{π} with θ_C
12: Compute $\pi(a|s)$ with θ_A
13: Compute the probability ratio $r(\theta_A)$ using $\pi'(a|s)$ and $\pi(a|s)$
14: Compute the entropy H with θ_A
15: $J^{CLIP}(\theta_A) = \mathbb{E}[\min(r(\theta)A_{GAE}^{\pi}, clip(r(\theta_A), 1-\varepsilon, 1+\varepsilon)A_{GAE}^{\pi})]$
16: $L_{pol}(\theta_A) = J^{CLIP}(\theta_A) - \beta H$
17: $L_{val}(\theta_C) = MSE(V_{new}^{\pi}, V_{tar}^{\pi})$
18: Update θ_A using $L_{pol}(\theta_A)$
19: Update θ_C using $L_{val}(\theta_C)$
20: Empty D
21: Test the policy during one episode for each test environment

2.4.2.2. Agent observation

In P2, the observation is centered on the agent. The example in Figure 34 shows the observation of a single agent with a perception grid of shape (15, 15), the depth of the observation (d_{agent}) with 6 layers. It also considers $k = 2$, and $l = 2$, which results in a perception of 4 neighbor agents.

[16] The base implementation of PPO is derived from the library Tianshou 0.3.0 (Weng et al. 2020). This implementation was extended with custom code to support multi-agent and parameter sharing.

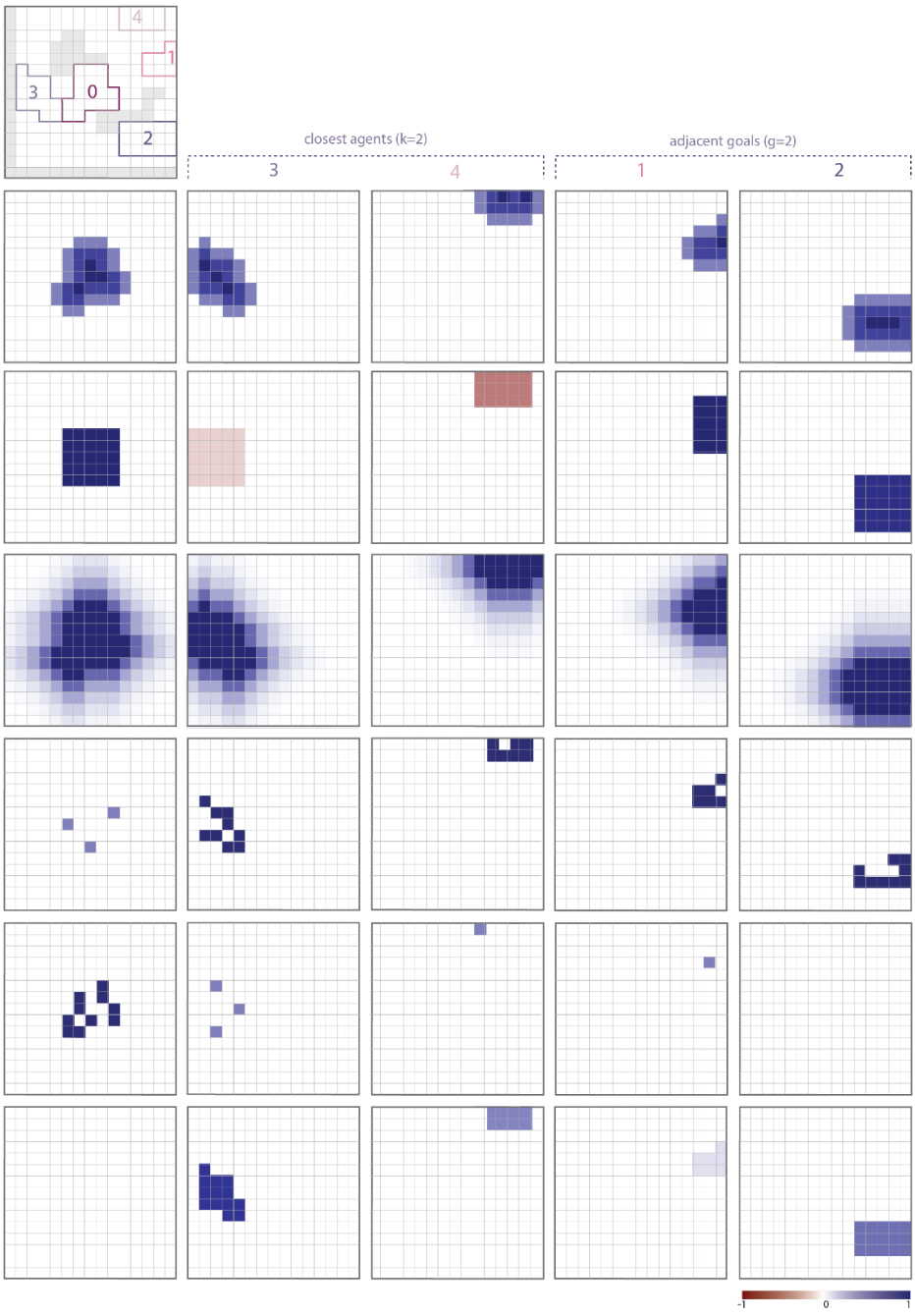


Figure 34: Observation for agent 0 with action grid of shape (5, 5), perception grid of shape (15, 15), k=2, and l=2. Row 1: grid with obstacles (agents are added just for visualization). The following rows are based on the g agents that share a relationship of adjacency with agent 0 (1, and 2) and its k closest neighbors that do not share a relationship of adjacency (3 and 4). Row 2: classification of the cells: structure (dark blue), surface (medium blue), and offset (light blue); row 3: score of the agent with the respect to the area over the action grid; row 4: soft adjacency values based on Manhattan distance; row 5:indication of folding cells: L-folds (light blue), U-folds (dark blue); row 6: indication of the daylit cells; row 7: the current utility of the agent represented in the internal cells of the agent. By Author.

In terms of depth, the agent observes its information layers plus the ones for the l agents that are related to it by its adjacency goal and the k closest agents (at the current step of the simulation) that do not share a relationship of adjacency with agent i. In terms of 2-D space, the observation

grid is smaller than the environment grid, so it slices all the information layers available to the agent. Therefore, from the perspective of agent i, all the information in observation o_i is sliced based on the perception grid of i.

More generally, the observation o_i is a multi-dimensional array of shape $(w_{per}, h_{per}, 1 + (1 + k + l) \times d_{agent})$, where the number of layers is independent of the number of agents in or the shape of the environment. It consists of

- A clipped environment grid with obstacles (w_{per}, h_{per})
- a multi-dimensional array o_i^{state} of shape $(w_{per}, h_{per}, d_{agent})$ with state information about the agent i with
    - Internal state of agent i with cell classification with structure, surface, and offset (w_{per}, h_{per})
    - g^{area} (w_{per}, h_{per})
    - g^{adj} (w_{per}, h_{per})
    - g^{fold} (w_{per}, h_{per})
    - g^{lit} (w_{per}, h_{per})
    - The agent cells scaled by the utility f (w_{per}, h_{per})
- a multi-dimensional array $o_i^{g\ states}$ of shape $(w_{per}, h_{per}, l \times d_{agent})$ with state information (like the one listed for o_i^{state}) for all the l agents that share a relationship of adjacency with agent i.
- a multi-dimensional array $o_i^{k\ states}$ of shape $(w_{per}, h_{per}, k \times d_{agent})$ with state information (like the one listed for o_i^{state}) for all the k-closest agents (at the current step of the simulation) that do not share a relationship of adjacency with agent i.

Particularly, we adopted an action grid of size (5, 5) and a perception grid of size (15, 15) for P2. Besides, we used $k = 4$, $l = 3$, and $d_{agent} = 6$. Therefore, the observation for each agent contains 7 layers for itself plus 6 layers for each of the 7 other agents, which results in an observation o_i is a multi-dimensional array of shape $(15, 15, 49)$.

2.4.2.3. Simple neural network for parameter sharing

P2 uses a neural network with a shared body (initial layers) and separate network heads (the final layers) for the actor and for the critic. This architecture allows the model to build a common representation based on the state information, then establish custom features for their distinct tasks in the independent portion of the neural network.

Table 8 Architecture of Neural Network for P2

	Layer Type	Input shape	Output shape	Kernel	Stride	Padding	Activation
Shared Body	Convolution 2D	(b, 15, 15, 49)	(b, 15, 15, 64)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 64)	(b, 15, 15, 32)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 32)	(b, 15, 15, 16)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 16)	(b, 15, 15, 8)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 8)	(b, 15, 15, 4)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Flatten	(b, 15, 15, 4)	(b, 900)	-	-	-	-
	Fully Connected	(b, 900)	(b, 256)	-	-	-	Linear
Critic	Fully Connected	(b, 256)	(b, 128)	-	-	-	ReLU
	Fully Connected	(b, 128)	(b, 64)	-	-	-	ReLU
	Fully Connected	(b, 64)	(b, 1)	-	-	-	Linear
Actor	Fully Connected	(b, 256)	(b, 128)	-	-	-	Relu
	Fully Connected	(b, 128)	(b, 26)	-	-	-	Linear

2.4.2.4. Training

The training setting relies on 64 environments with 16 agents that are randomly initialized and that interact in randomly generated environments during multiple episodes of 128 steps (see Figure 35 and Figure 36). To keep track of the robustness of the performance during training, there are 32 testing environments with different number of agents: 4 environments with 4 agents, 8 environments with 8 agents, 8 environments with 12 agents, 8 environments with 16 agents, and 4 environments with 20 agents. To see more details of this training setting, check Table 6. Besides, P2 and P3 were trained with multiple spatial goals, both to check the technical feasibility of the training and to support the design of custom agents that converge to specific spatial types in the larger combinatorial space of the restricted polyominoes

For each training or testing episode, the simulation engine randomly initializes the environment with the following steps (see Figure 35):

- randomly initialize a new board grid with sizes between $(8, 8)$ and $(31, 31)$ with padding of 7, which results in a shape between $(22, 22)$ and $(45, 45)$.
- Populate the board with random obstacles that are either x-aligned or y-aligned series of cell.

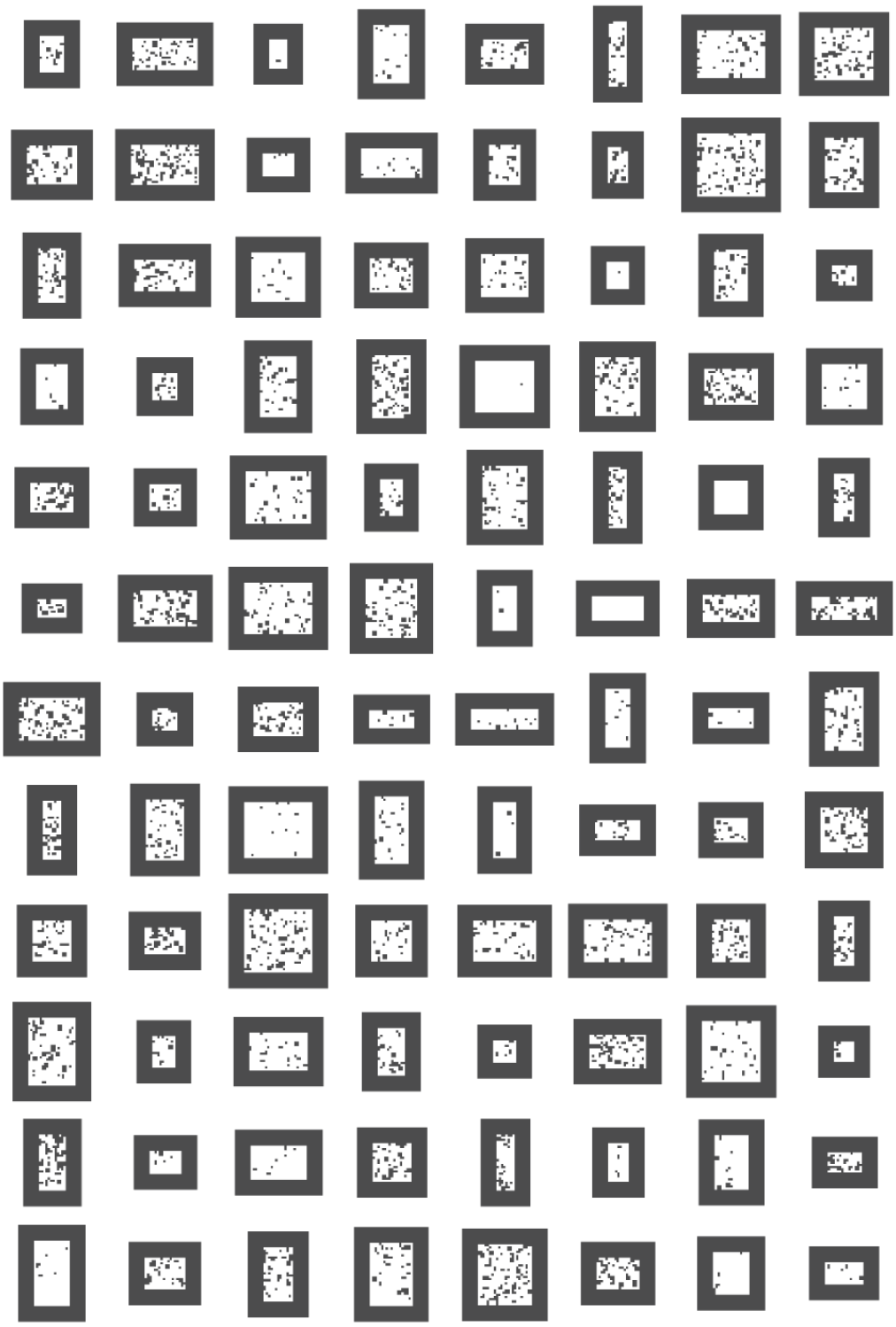


Figure 35 96 examples of boards generated by the engine. By Author.

For the setting of the agents (see Figure 36), the engine randomly

- initializes the area and adjacency targets

- Initializes the layers for the n agents and its initial random positions
- runs several steps with random actions so the agents settle in a random initial configuration

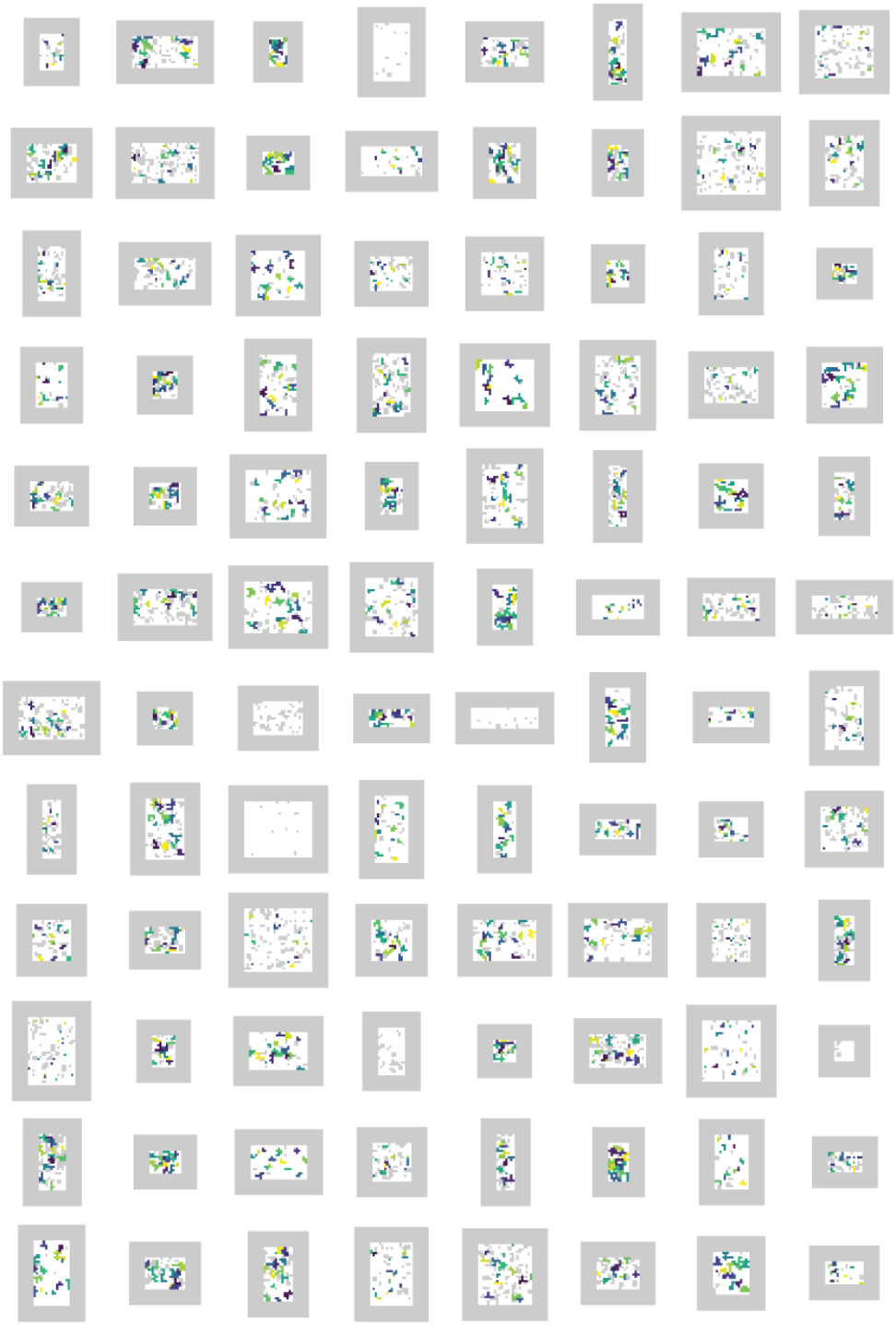


Figure 36 96 examples of agent initialization by the engine. By Author.

To test the feasibility of having custom agents with different spatial behaviors, the model was trained with different objectives: Adjacency and Area; Adjacency, Area, Not Folding (5), and Daylit (75%); Adjacency, Area, and Not Folding (3); Adjacency, Area, and Not Folding (5).

The graphs in Figure 37 and Figure 38 show that the training process was stable for all the different goals. Besides, while time was not recorded, each of these training sessions took between 4 and 5 days due to the use of multiple training environments and larger batch sizes.

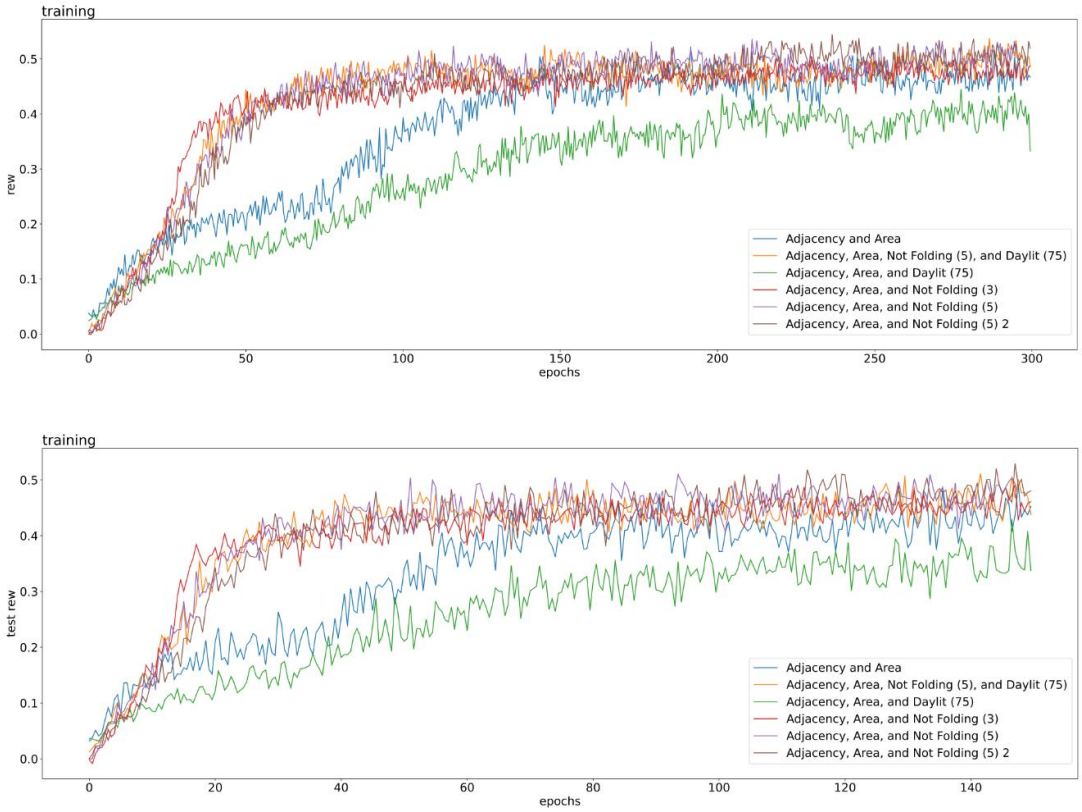


Figure 37 Training graphs with training and test rewards for P2. By Author.

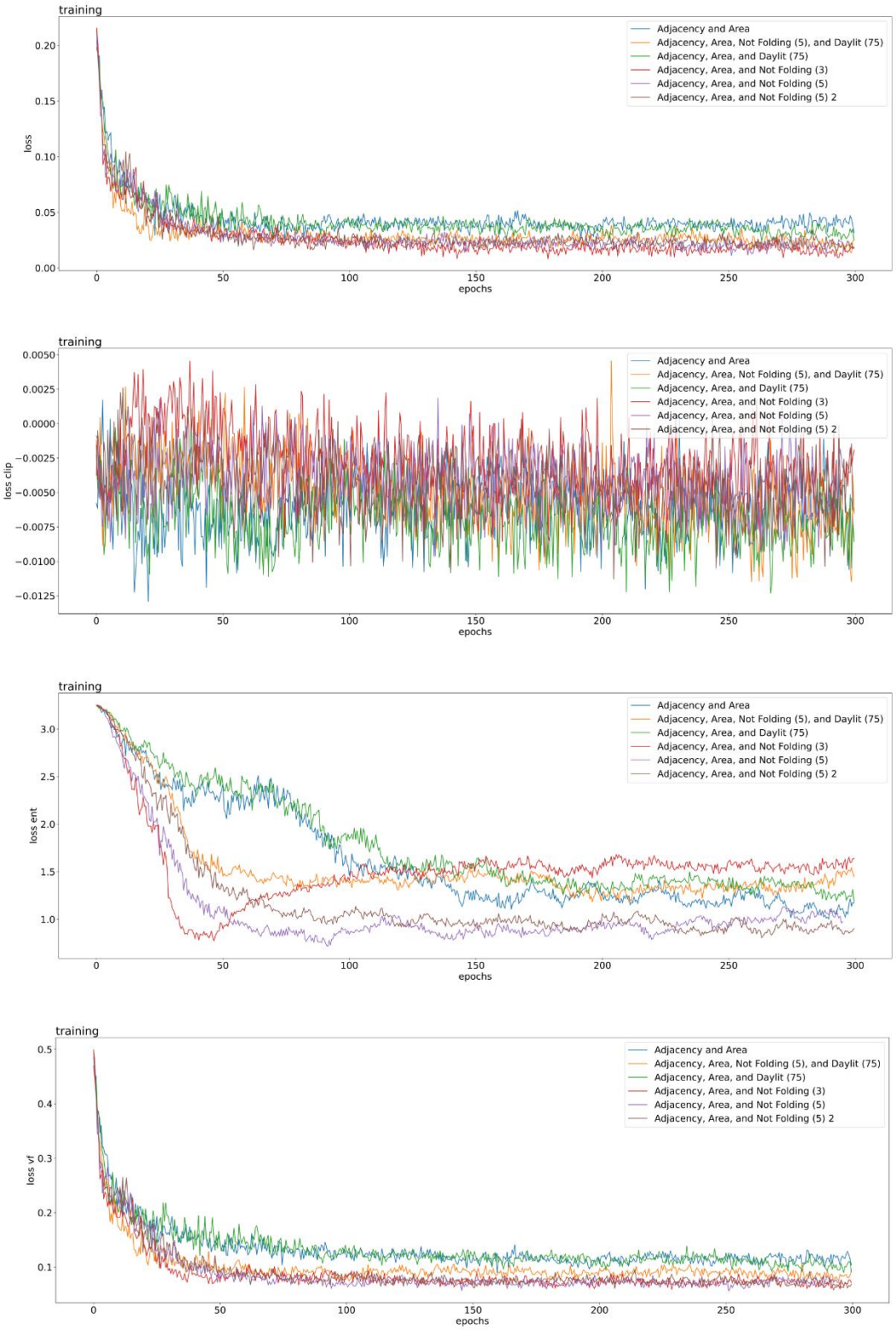


Figure 38 Training graphs with losses for P2. By Author.

2.4.2.5. Lessons learned

The use of the actor-critic framework with a simpler parameter sharing method in P2 accelerated the training process, made it more stable, and reduced the noise of the inference with a stochastic policy. The agent-centric representation enabled the training and deployment of successful policies in environment of variable sizes, which is critical for the scalability in the application of the system. Besides, P2 was able to learn policies based on different combination of goals, which resulted in layouts of varied types. Some of these policies were successful in addressing all the goals, others opted for sacrificing the performance in one of the goals to succeed in the others.

2.4.3. P3

The development of P3 intends to explore some variations in the training setup that can potentially improve the performance. The main changes are the development of an algorithm to generate more sophisticated and realistic obstacles in the environment and variations in the training parameters.

2.4.3.1. Training

The training setting relies on 16 agents that are randomly initialized and that interact in randomly generated environments during multiple episodes of 256 steps. In contrast to the previous, version, where the environment was generated by placing small obstacles over the grid, this version attempts to create more realistic boards. It uses an algorithm that excavates an initial board full of obstacles using the number of adjacent empty spaces as a weight for random selection (see Algorithm 3).

Algorithm 4 Generate environment

1: Initialize parameters $0.75 \leq p_{space} \leq 0.95, 0.5 \leq e \leq 2, 5 \leq n_{init} < 20, 15 \leq h_{env} < 45, 15 \leq w_{env} < 45$
2: initialize a new board B of shape (h_{env}, w_{env}) with obstacles
3: initialize a list A of length $(h_{env} \times w_{env})$ with zeros. It counts the number of adjacent cells that are empty
4: populate the grid A with n_{init} empty cells
5: update the list A with $+1$ for each obstacle adjacent to an empty cell
6: while percentage of empty cells is smaller than p_{space}
7: sample the next empty cell based on the probability $\frac{A^e}{\sum_i A^e{}_i}$
8: update the list A with a 0 for the empty cell
9: update the list A with $+1$ for each obstacle adjacent to the empty cell
10: pad B

After initializing the environment, the engine randomly

- initializes the area and adjacency targets
- Initializes the layers for the n agents and its initial random positions
- runs several steps with random actions so the agents settle in a random initial configuration

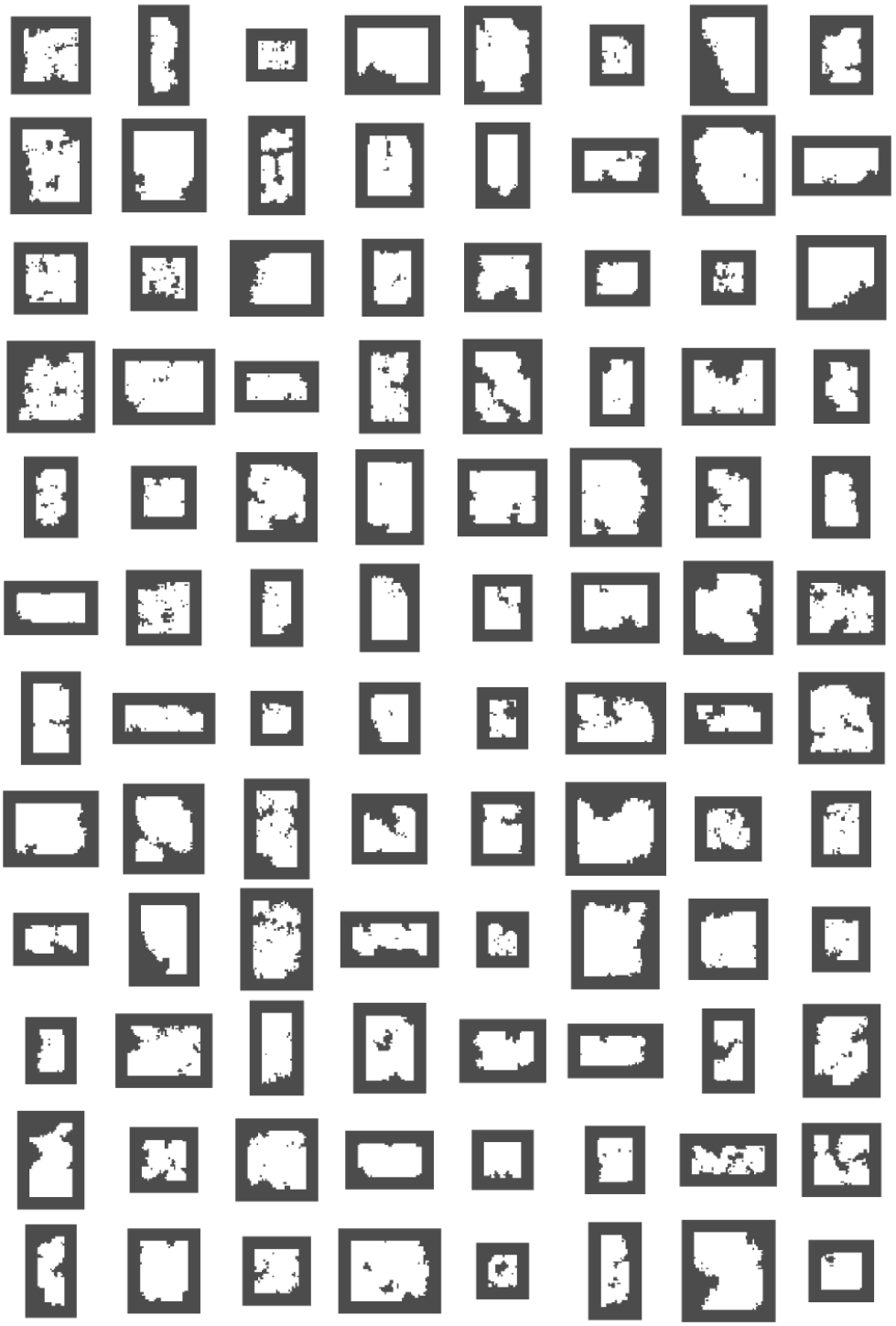


Figure 39 96 examples of boards generated by the engine. By Author.

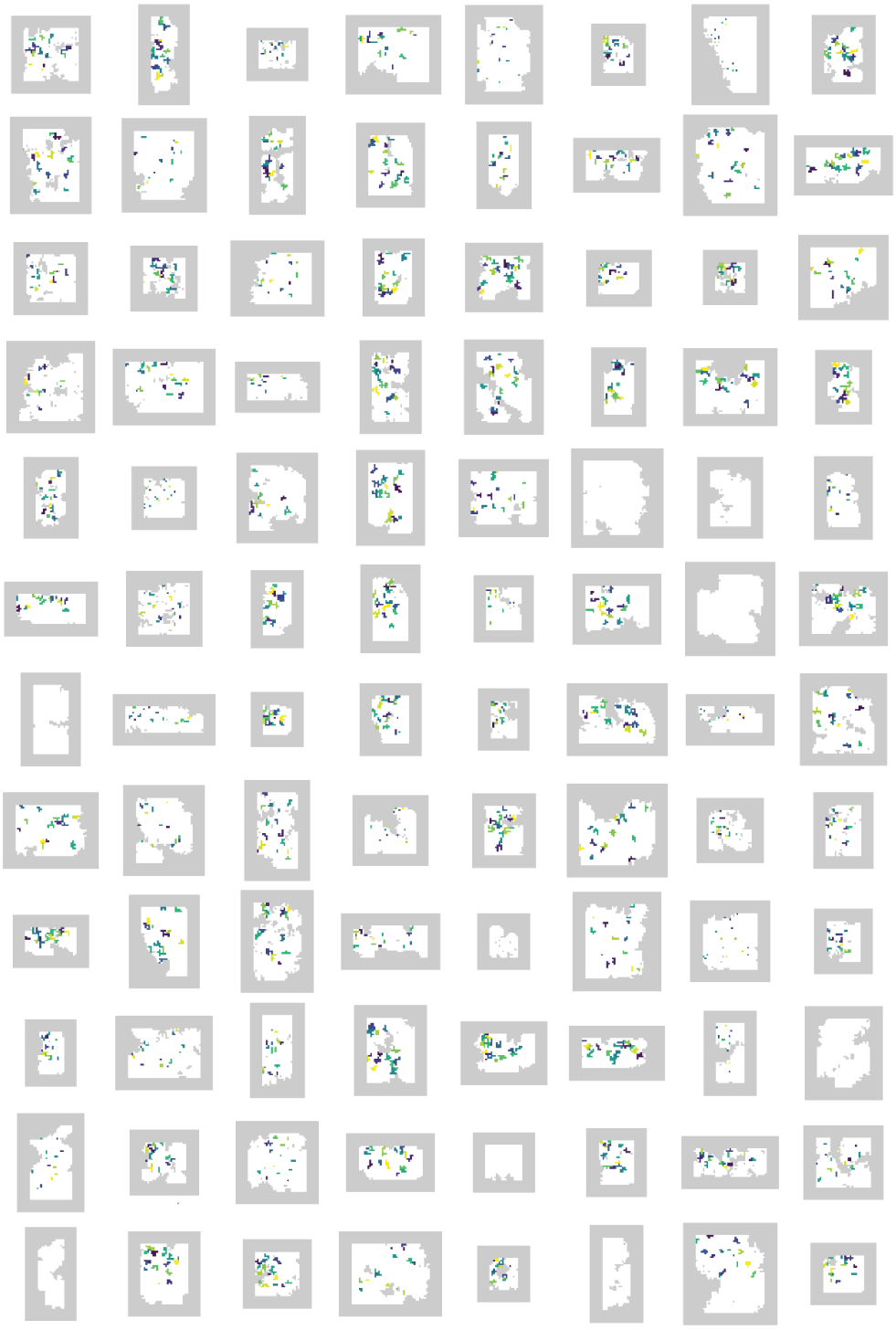


Figure 40 96 examples of agent initialization by the engine. By Author.

In contrast to P2, the number of steps in the episode was increased duplicated (128 to 256) while the number of training environments was reduced by half (64 to 32). The intention is to preserve the number of steps per epoch is the same but expose the agents to more experience in stable configurations. The different spatial behaviors are tested again in this new configuration to

confirm the feasibility of learning custom policies. The model was trained with different objectives: Adjacency and Area; Adjacency, Area, Not Folding (5), and Daylit (50%), Adjacency, Area, Not Folding (5), and Daylit (75%); Adjacency, Area, and Not Folding (5). Notice that the Adjacency, Area, Not Folding (3) objective was removed from the list and replaced by Adjacency, Area, Not Folding (5), and Daylit (50%). Notice that Adjacency, Area, Not Folding (5) was trained over all the 300 epochs to enable the comparison between P2 and P3. Due to time restriction, the remaining policies were only trained over 200 epochs.

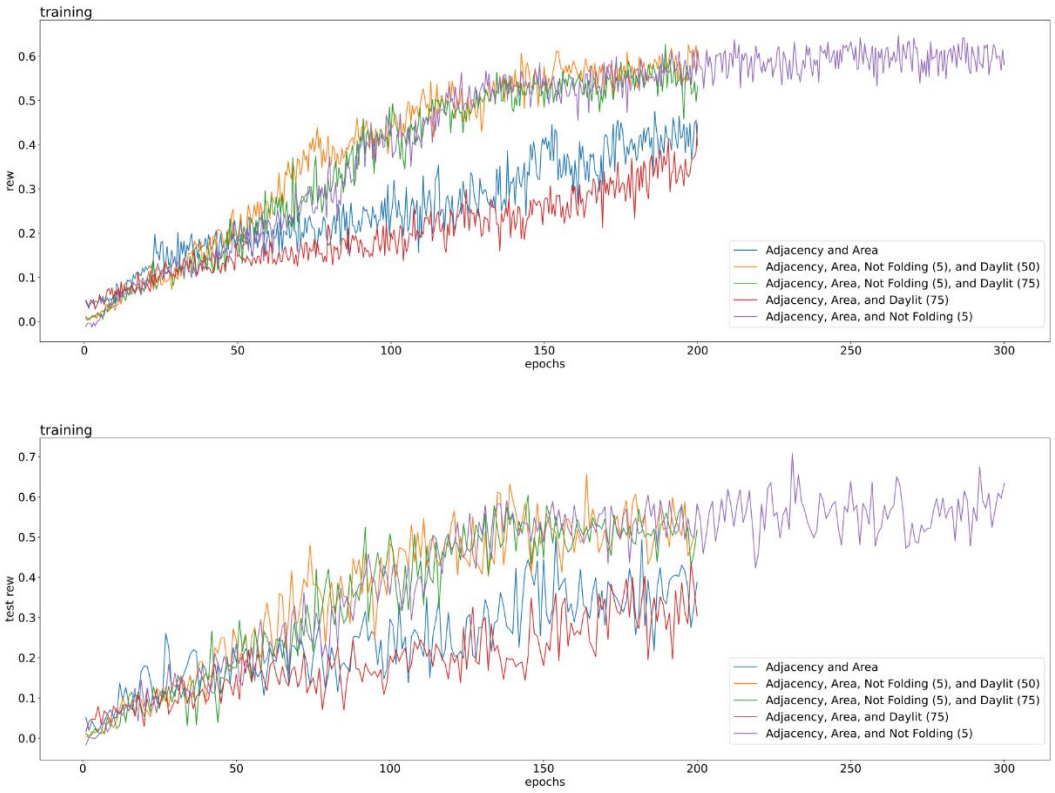


Figure 41 Training graphs with training and test rewards for P3. By Author.

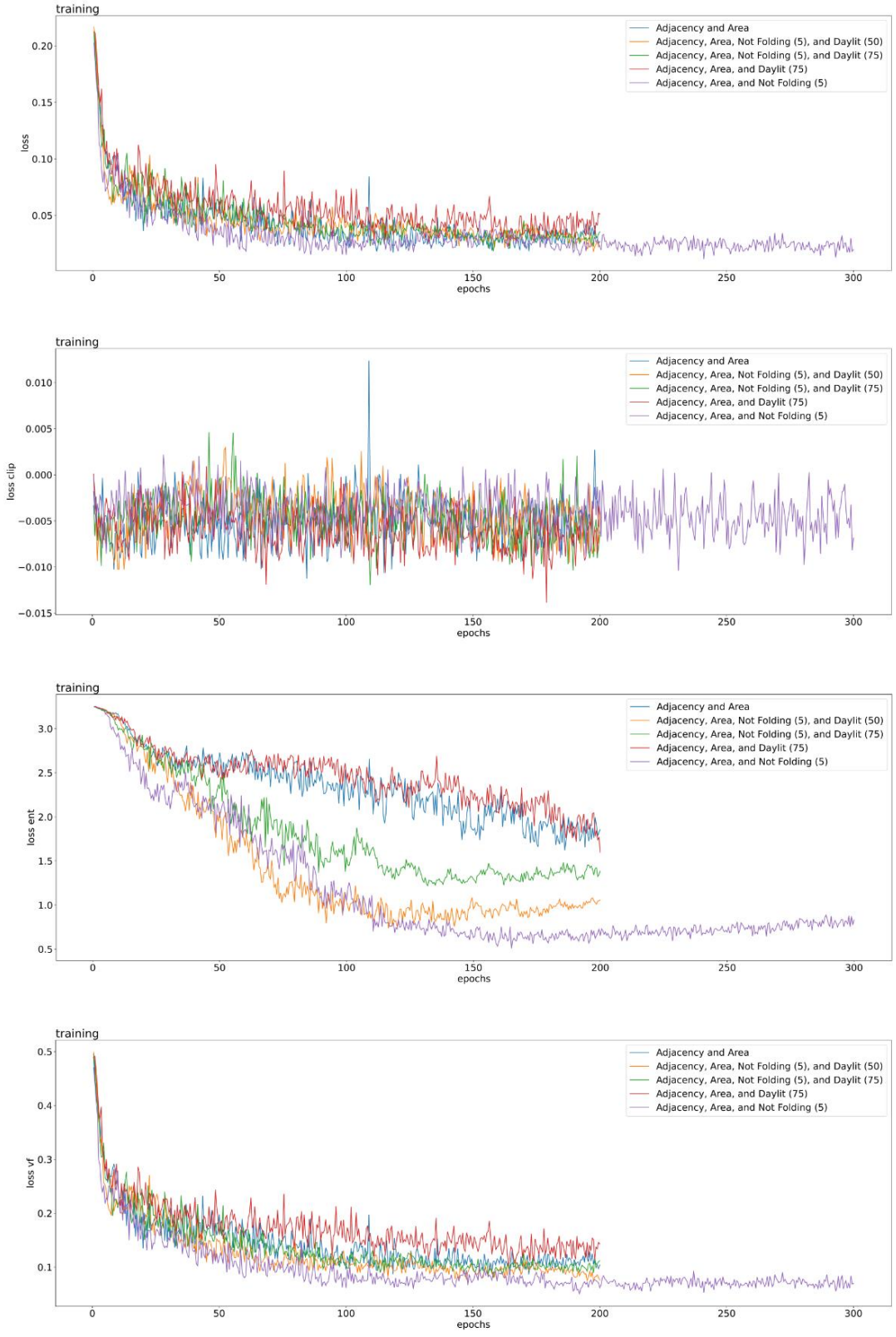


Figure 42 Training graphs with losses for P3. By Author.

3. Evaluating the design artifact

3.1. Introduction

While the previous chapter proved that the design artifact can be constructed and trained, this chapter attempts to answer the main research question: can agent-based modeling and current deep reinforcement learning techniques support the design of custom agents for a spatial synthesis based on fine-grained interaction?

Three design cases are used to test the performance of the different iterations of the prototype considering aspects of machine learning, the research framework of generative design with fine-grained interaction and the problem of spatial synthesis. From the perspective of machine learning, these design cases are necessary to provide a common ground to test if agents trained in varied and stochastic environments can perform well in unseen design problems. While a policy trained specifically for the design case could perform better, policies with good generalization can support design tools that minimize the need of training on-demand. With the respect to the research framework of fine-grained interaction, the design cases evaluate the capacity of the artifact to support step-by-step formation of valid designs in face of adversity, such as site conditions and design perturbations. For the scope of the application, the performance of the artifact is evaluated not only with spatial synthesis problems with constraints and aspects of a real design program but also to unconventional ones that can result in speculative configurations.

The chapter is organized according to the three test cases and followed by a discussion:

1. **Design case 1** comprehends single-family house to be implemented in three different sites
2. **Design case 2** includes a single-story complex of art galleries located in a site between urban lots with small buildings and an existing park
3. **Design case 3** encompasses a complex of eight houses to be placed on a large and empty site
4. **Discussion** provides a synthesis about the results, about qualities and capacities identified in the prototype.

3.2. Design Case 1: single family housing on small site

The first design case is a single-family house to be implemented in three different sites. This case was created to evaluate P1 with a single spatial goal but was also used to evaluate the following versions of the prototype. Design case 1 consists of twelve agents that represent the spaces of a two-bedroom house with a free floor plan (see Figure 43).

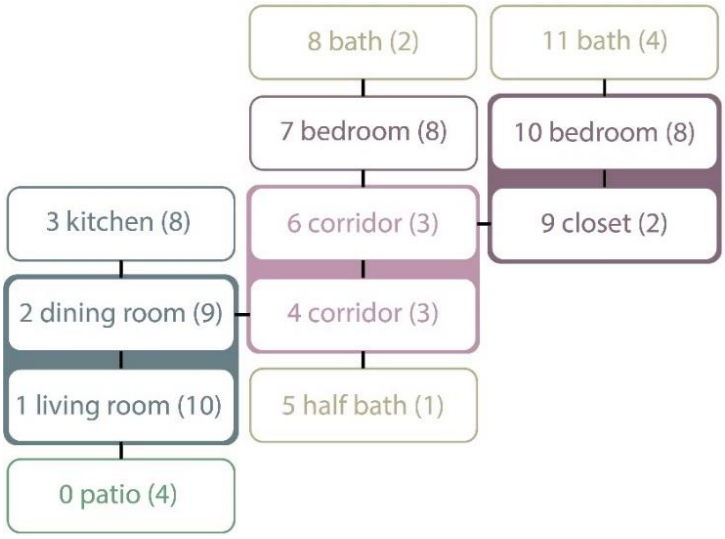


Figure 43 Program of the house in case 1. The leading number is used to identify the spaces in the next images. The numbers in parenthesis indicate target area. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces. By Author.

For each episode, the agents interact for 6500 steps, which are divided in three stages that are separated by two phases of perturbations:

- Stage 1 (0-1999): ε -greedy policy with $\varepsilon = 0.05$ (DDQN) or regular policy (PPO)
- Perturbation 1 (2000-2099): random action selection.
- Stage 2 (2100-3099): ε -greedy policy with $\varepsilon = 0.05$ (DDQN) or regular policy (PPO)
- Perturbation 2 (4100-4499): random action selection.
- Stage 3 (4500-6499): ε -greedy policy with $\varepsilon = 0.05$ (DDQN) or regular policy (PPO)

In stage 1, the initialization places the empty agents in random positions. In stage 2 and three, the initial state is based on the randomization of the final solution of the previous state. From the perspective of the agents, these initializations are much more challenging than addressing a typical human interaction, such as sequential addition, edition, or removal agents. However, the assumption here is that a policy that can recover from more complex states will be better to deal with the iterative construction by a designer.

The agents were tested for the house program in three different sites with obstacles (Figure 44): (1) a site on the top of the hill, (2) a site with a central obstacle, (3) a site with two stream banks connected by a bridge.



Figure 44 Three different sites: (1) site on the top of the hill, (2) site with central obstacles, (3) site with stream and bridge. By Author.

The policy Adjacency, Area, and Not Folding (5) from the different iterations of the prototype was applied to the design case 1 to evaluate their performance and the resulting spatial patterns. Also, the algorithm used to translate the design representation of the agent polyominoes into 3D architectural layout models with doors and windows is described in the next chapter.

3.2.1. P1

Figure 45, Figure 46, and Figure 47 show some snapshots of the resulting configuration for the three environments over 6500 steps using the policy Adjacency, Area, and Not Folding (5). Overall, the policy converged to the the basic elements spatial synthesis (adjacency and area) but was also able to address the formation of more conventional room layouts with finer adjustments to address obstacles and conflicts. The agents improved the performance of the house layout configuration either from random initialization or from a random perturbation in all the scenarios. After the perturbations, the agents were able to re-organize in different spatial configurations, which shows the potential for direct interaction with the designer.

In a visual and qualitative analysis, it is possible to observe that the agents developed a functional and consistent behavior. Besides, some interesting spatial patterns that emerged from the interaction of the agents with local situations and from randomness, despite not being explicitly described in the objectives. For example,

- In the environment on the top of the hill, the agents often defined L-shapes for the bathrooms to adapt to the irregular boundary (Figure 45, 879 and 882, and Figure 48)
- an agent created a protuberance to increase the area and to adapt to local obstacles (Figure 45, patio at 3898 or kitchen at 5128)
- A small courtyard emerged between the sectors of the house (Figure 47, at 6216, and 6219).
- The bridge in scenario three incentivized a separation of the social and private areas of the house to the two banks, which made the satisfaction of adjacency easier by constraining the associated agents on one side of the bank.

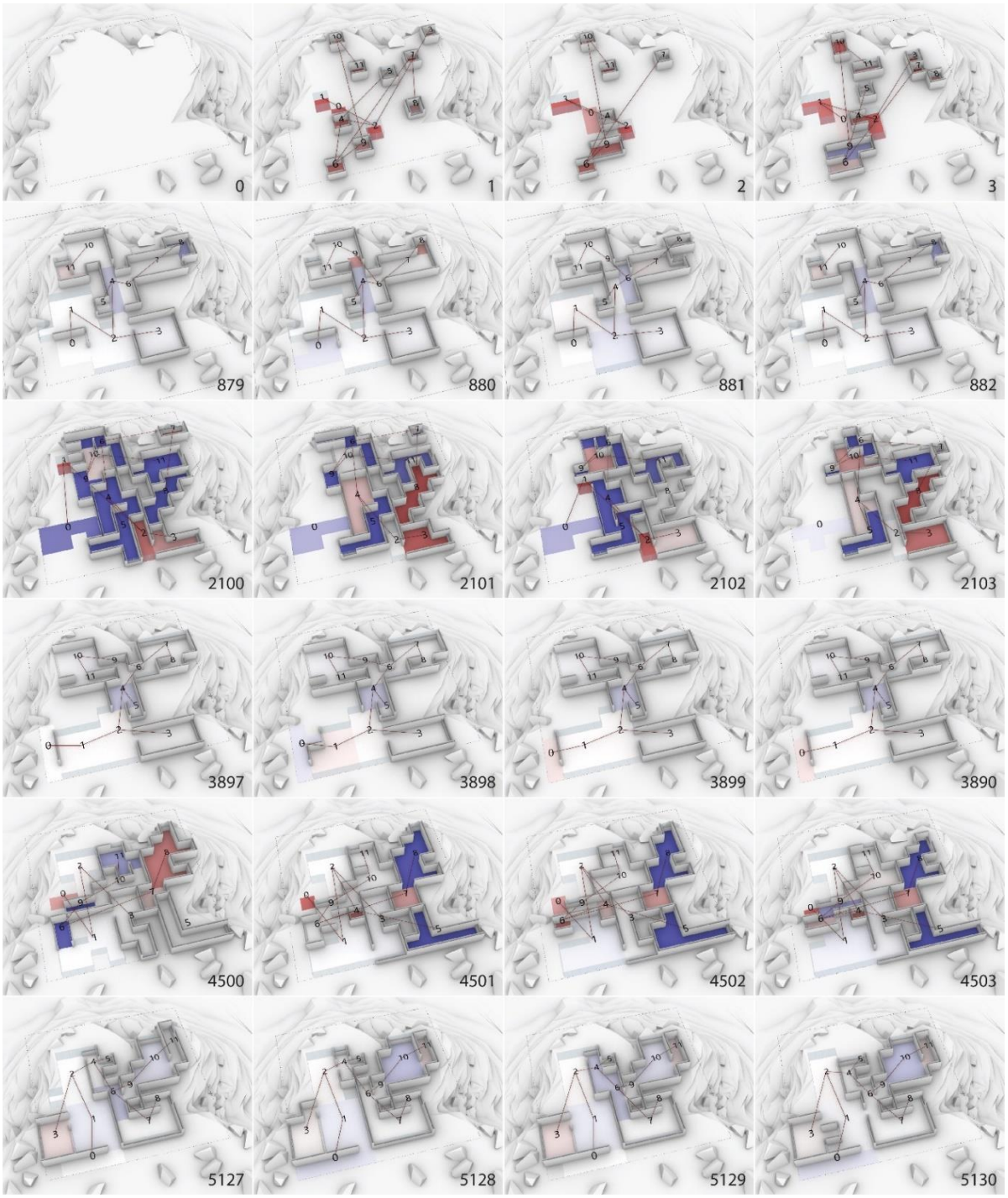


Figure 45 Rows 1-6: simulation of 12 agents from P1 in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

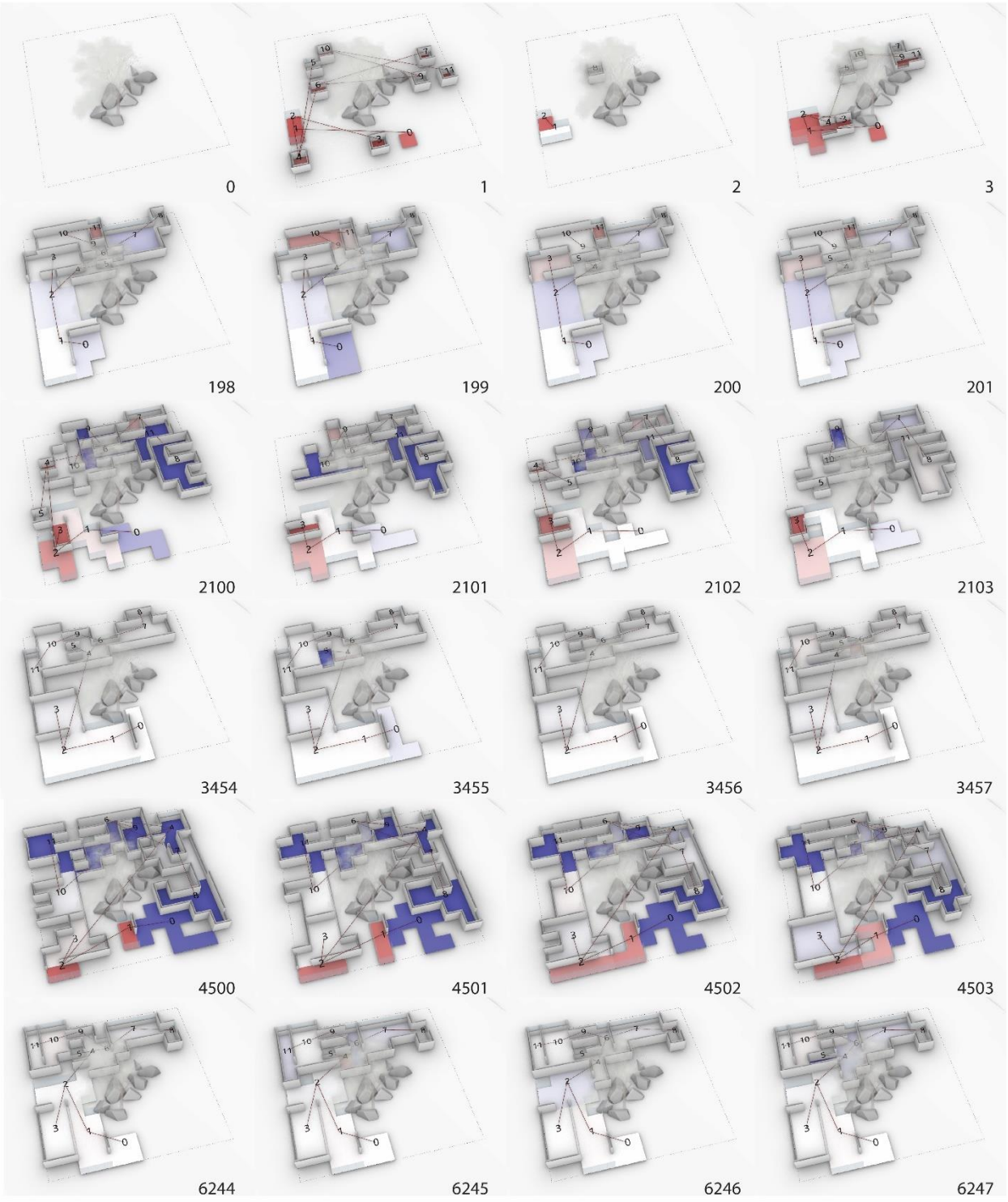


Figure 46 Rows 1-6: simulation of 12 agents from P1 in a terrain with a large tree and rocks as obstacles at the center. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

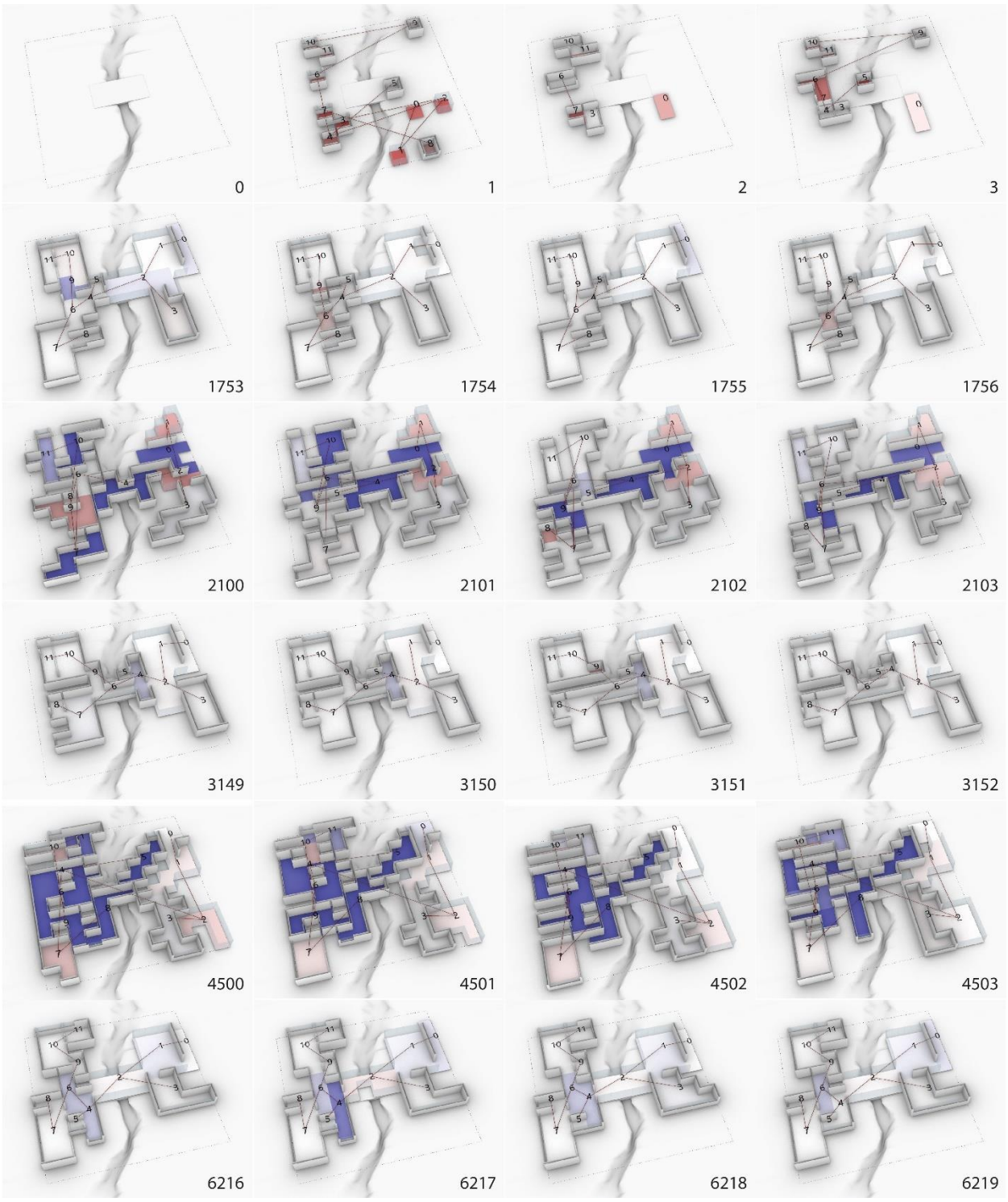


Figure 47 Rows 1-6: simulation of 12 agents from P1 in a terrain with a bridge over a stream. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.



Figure 48 Example of layout generated by P1. Artistic rendering reproduced with permission of Jinmo Rhee.

Overall, the main achievement is that the policy of P1 was able to generalize the knowledge from the training environment with 6 random agents, random goal parameters, and random obstacles, to specific design cases with 12 agents. The resulting policy enabled the agents to react to and recover from random perturbations in the simulation and generated a variety of spaces that satisfied the multiple spatial goals.

The performance of the agent was consistent among the different scenarios, with the average of the adjacency utility f^{adj} between .922 and .955 in the selected run. The average of .922 in the site with central obstacles indicates that agent is not always able to converge to a configuration that satisfies all adjacencies over many steps (e.g., 3454-3457). The average of the f^{area} is between .891 and .909, which probably reflects the irregular and limited open spaces available for the agents in all the scenarios. The average of f^{fold} is between .931 and .945, which is reflected in the dominance of rectangular or quasi-rectangular agent polyominoes in the layouts.

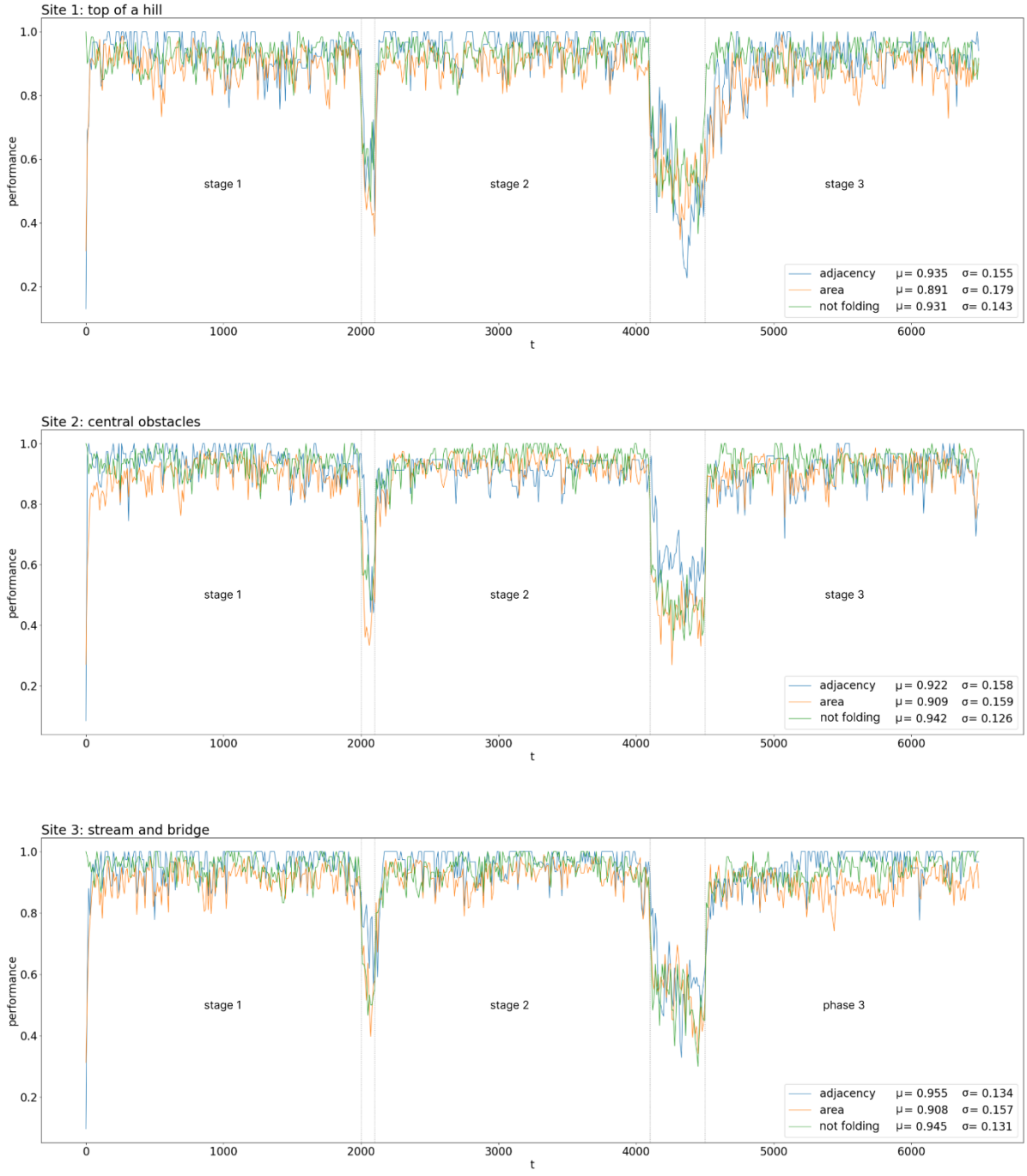


Figure 49 Graphs with the performance of the agents from P1 (y axis) over an episode (x axis) for each of the three environments. The areas between dotted lines indicate the periods of perturbation. The means (μ) and standard deviations (σ) consider the three stages with ε-greedy action selection but not the periods with random perturbation. By Author.

3.2.2. P2

In contrast to P1, P2 was trained with 16 agents but it was for a shorter period. It also used training sites of different sizes, and it also relied on a much simpler neural network and in more restricted agent-centered percept information. Despite all these differences, P2 achieved a superior performance, while generating houses with the same spatial type of a cluster of rectangular and L-shaped rooms. The Figure 50, Figure 51, and Figure 52 show examples of the PPO agents trained in P2 generating the layout for the house.

Another aspect that is important to mention is how the resulting policy addresses exploration directly, so no random action selection is necessary. Notice in the graphs in Figure 53 that the agents engage in changing the configuration, which adds perturbation to the performance when the adjacency score is lower but become more stable as the score gets closer to 1. Comparing the graphs in Figure 49 and Figure 53, it seems that even in the phases where the agents are changing the layout, the PPO agents in P2 have a much stabler behavior than the DDQN agents in P1.

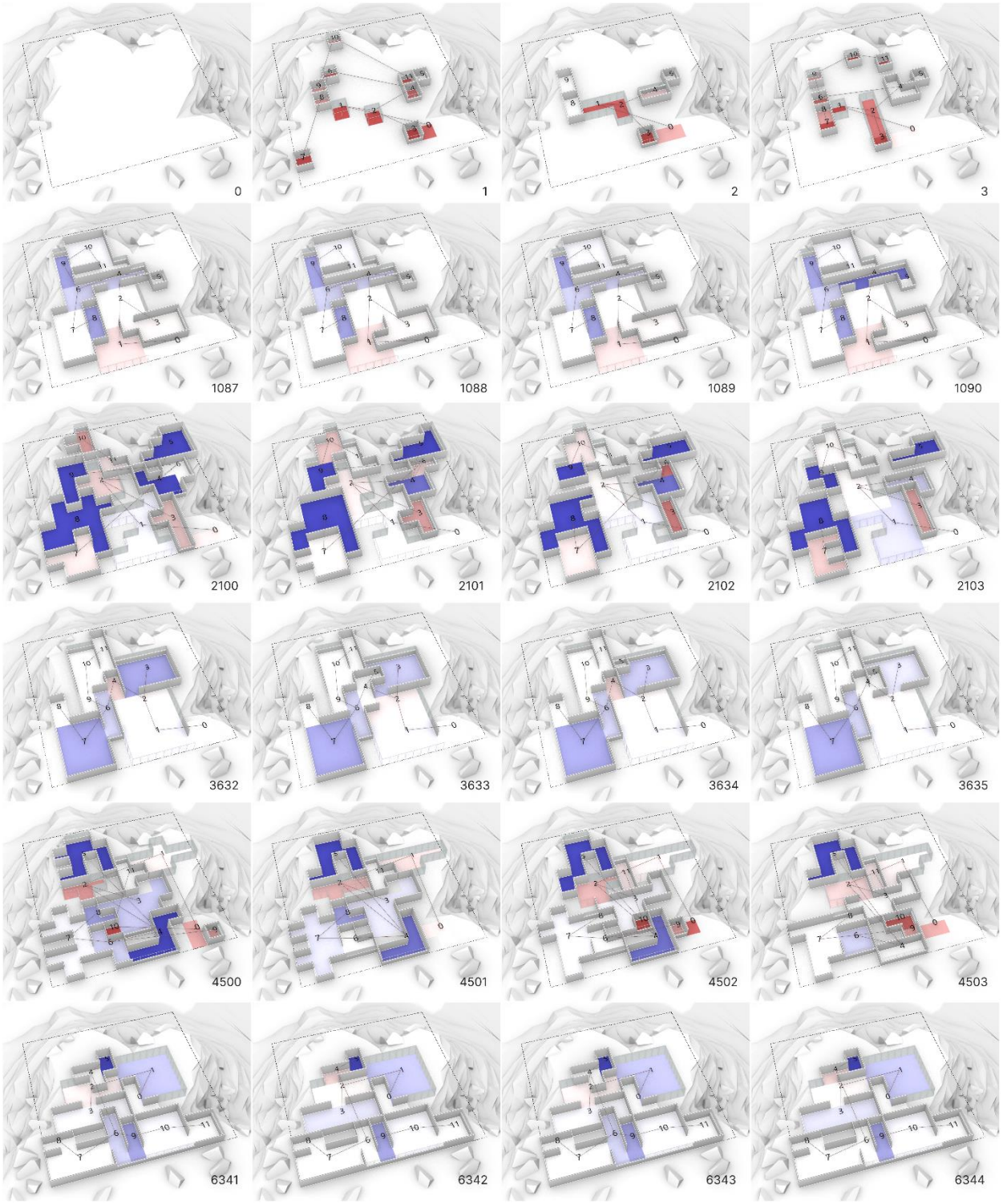


Figure 50 Rows 1-6: simulation of 12 agents from P2 in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

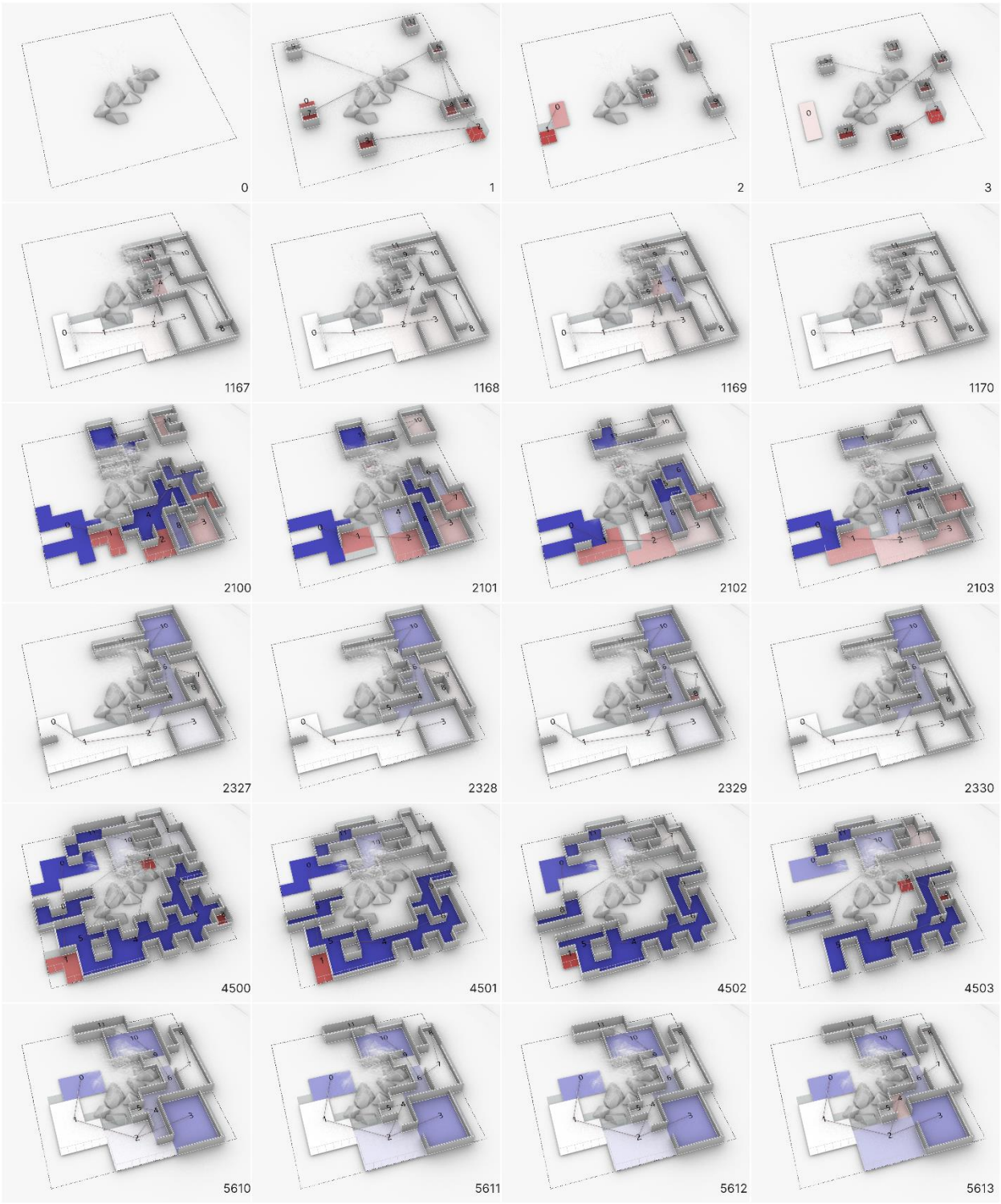


Figure 51 Rows 1-6: simulation of 12 agents from P2 in a terrain with a large tree and rocks as obstacles at the center. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

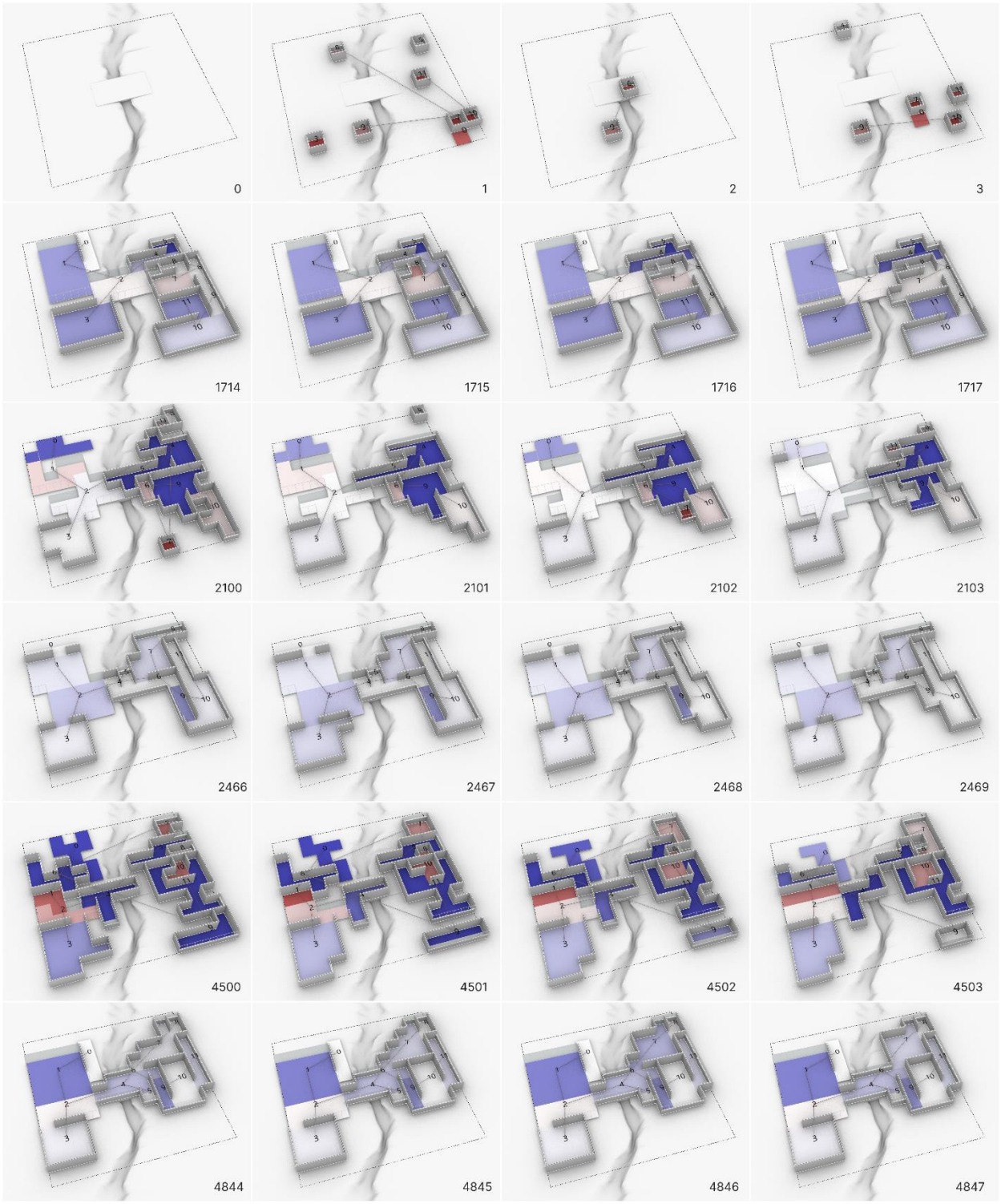


Figure 52 Rows 1-6: simulation of 12 agents from P2 in a terrain with a bridge over a stream. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

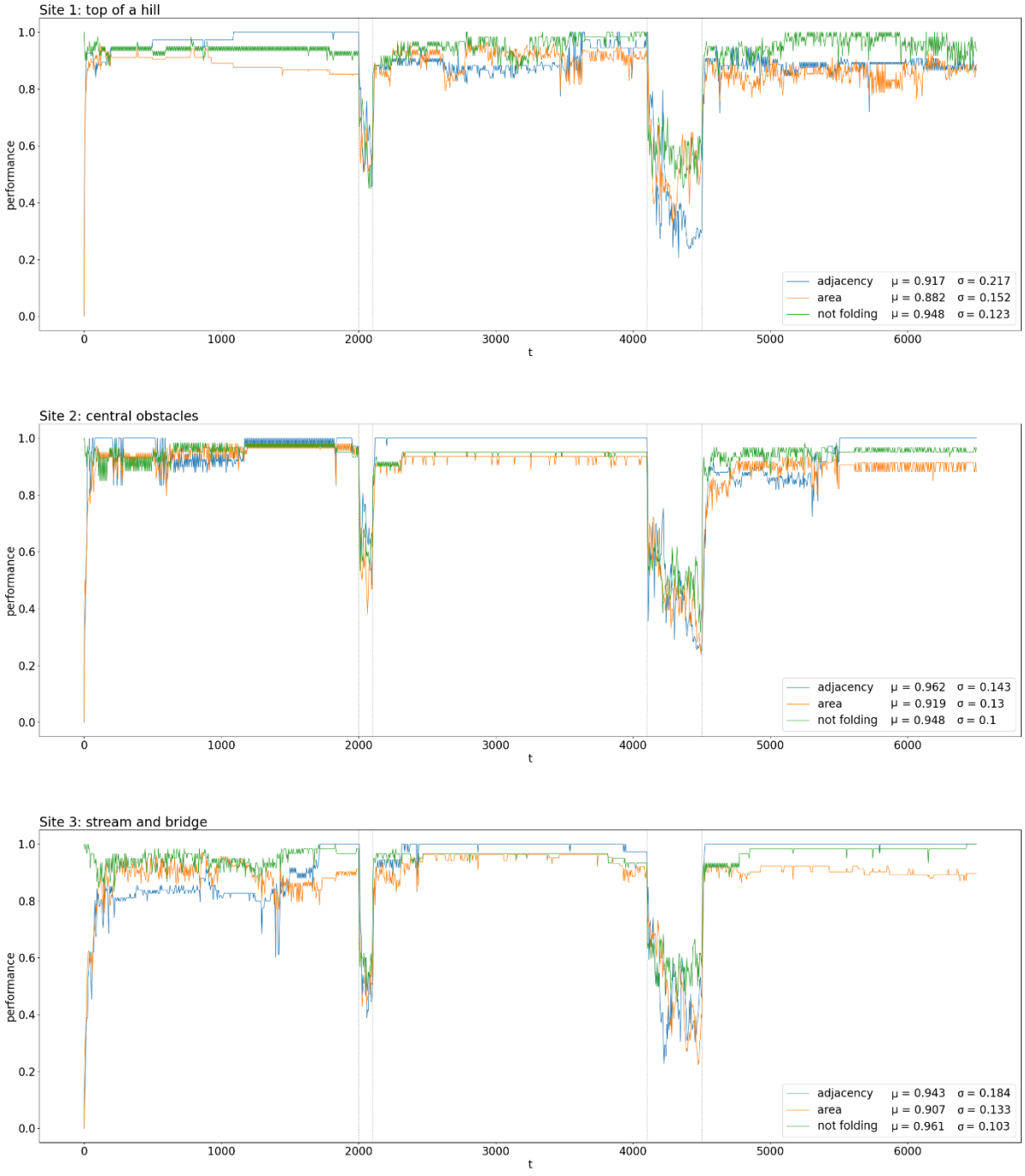


Figure 53 Graphs with the performance of the agents from P2 over one episode for each site. The areas between dotted lines indicate the periods of perturbation. The means (μ) and standard deviations (σ) consider the three stages with the policy but not the periods with random perturbation. By Author.

The policy was also evaluated 50 times in each of the environments. The results are on the following graphs (Figure 54). Notice that on average, the agents converge to a good configuration over a few iterations, even after random perturbations.

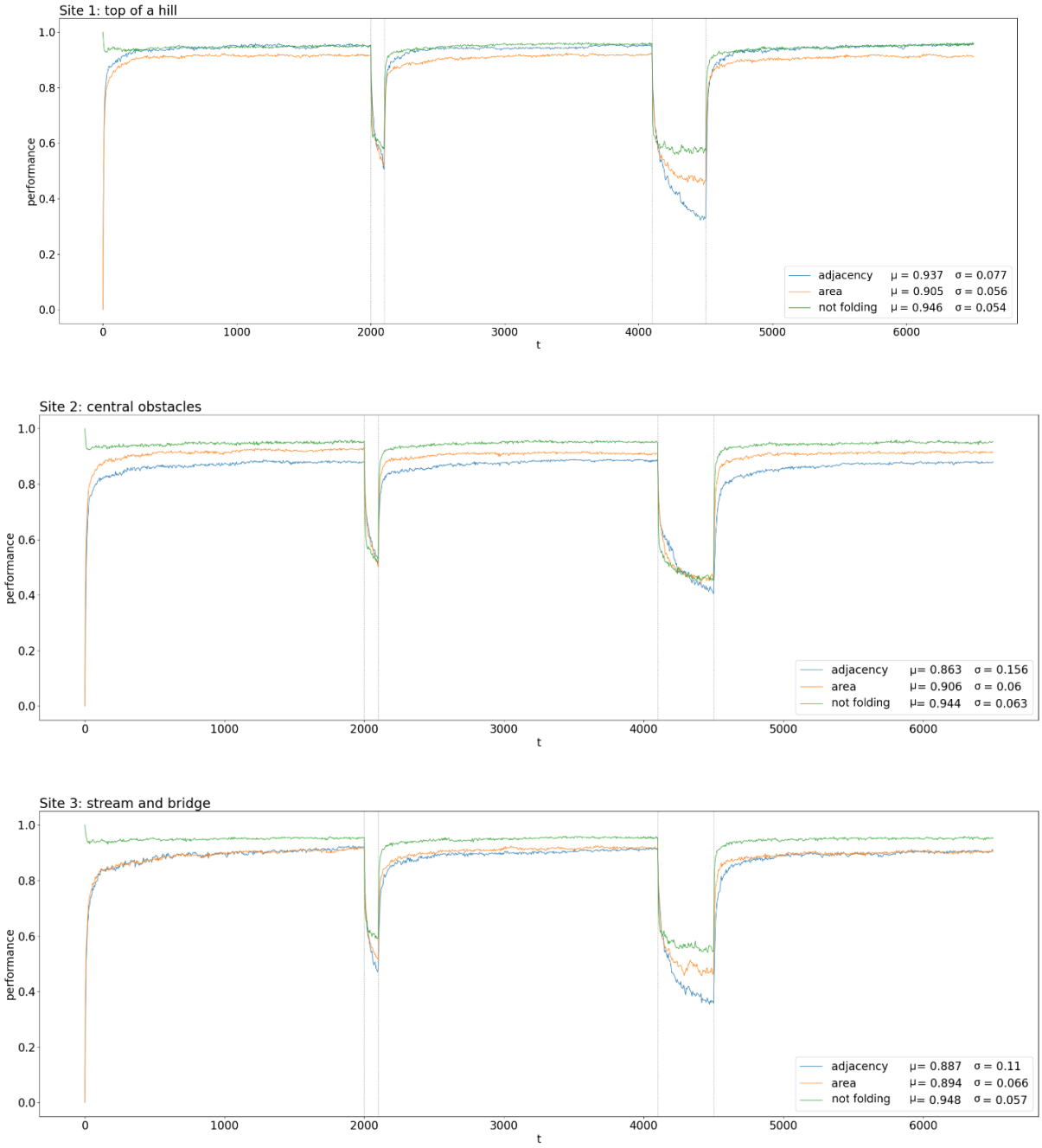


Figure 54 Graphs with the average performance of the agents over 50 episodes for each of the three environments. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the three stages with policy but not the periods with random perturbation. By Author.

3.2.3. P3

P3 is like P2 but with a few adjustments in the training settings. Overall, the learned policy also converged to good results in the three scenarios either from random initialization or from a random perturbation in all the scenarios.

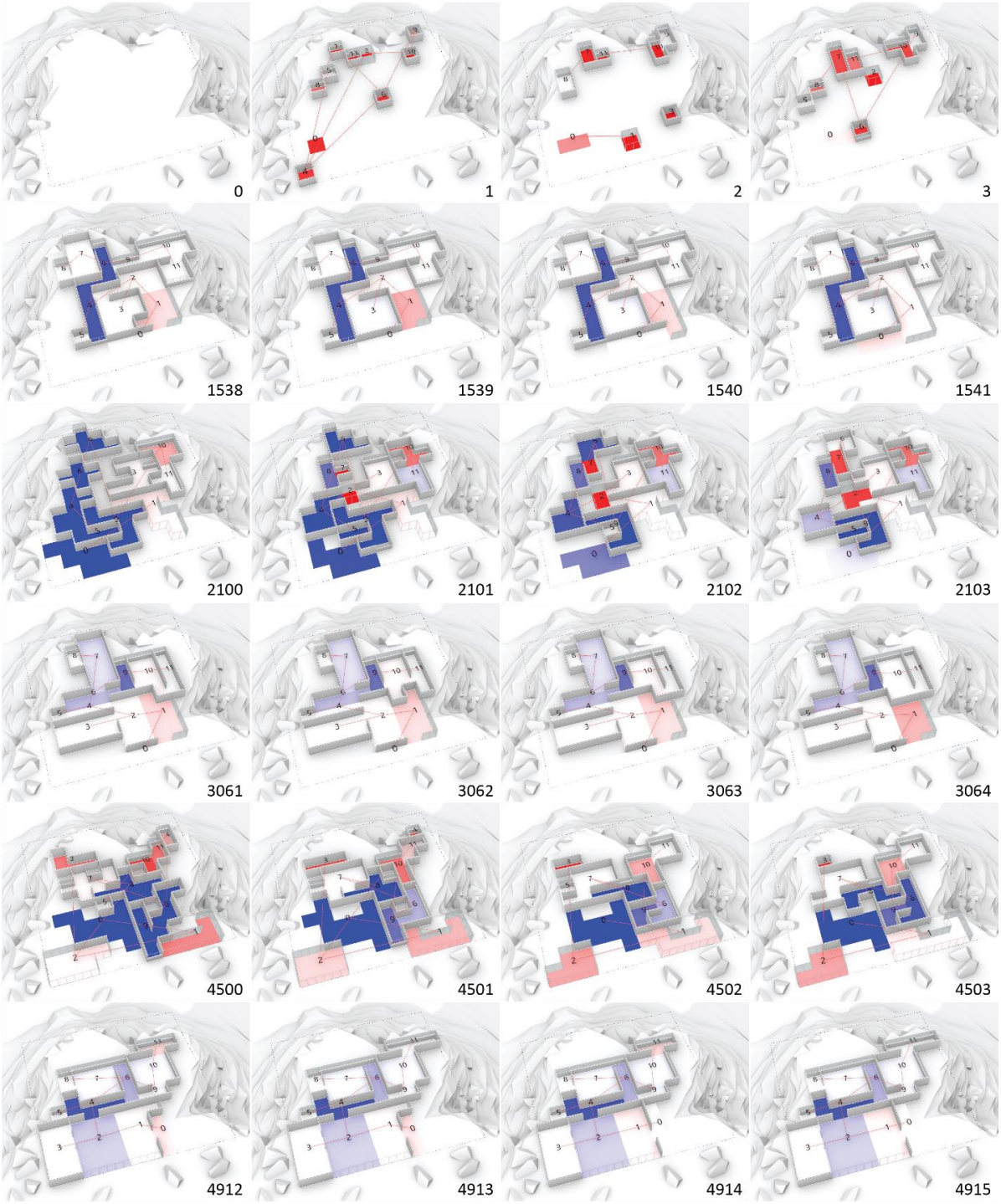


Figure 55 Rows 1-6: simulation of 12 agents in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

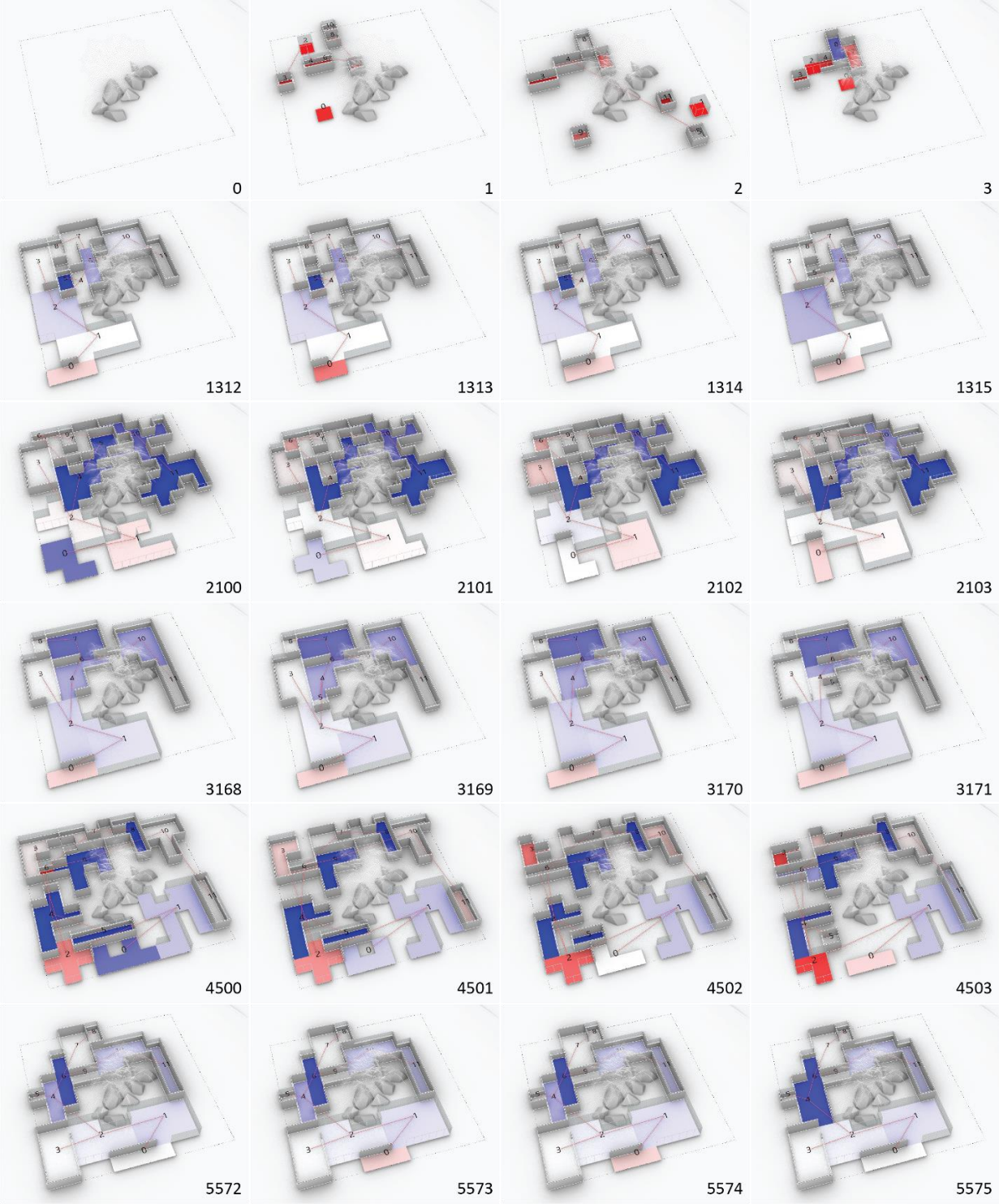


Figure 56 Rows 1-6: simulation of 12 agents in a terrain with a large tree and rocks as obstacles at the center. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.

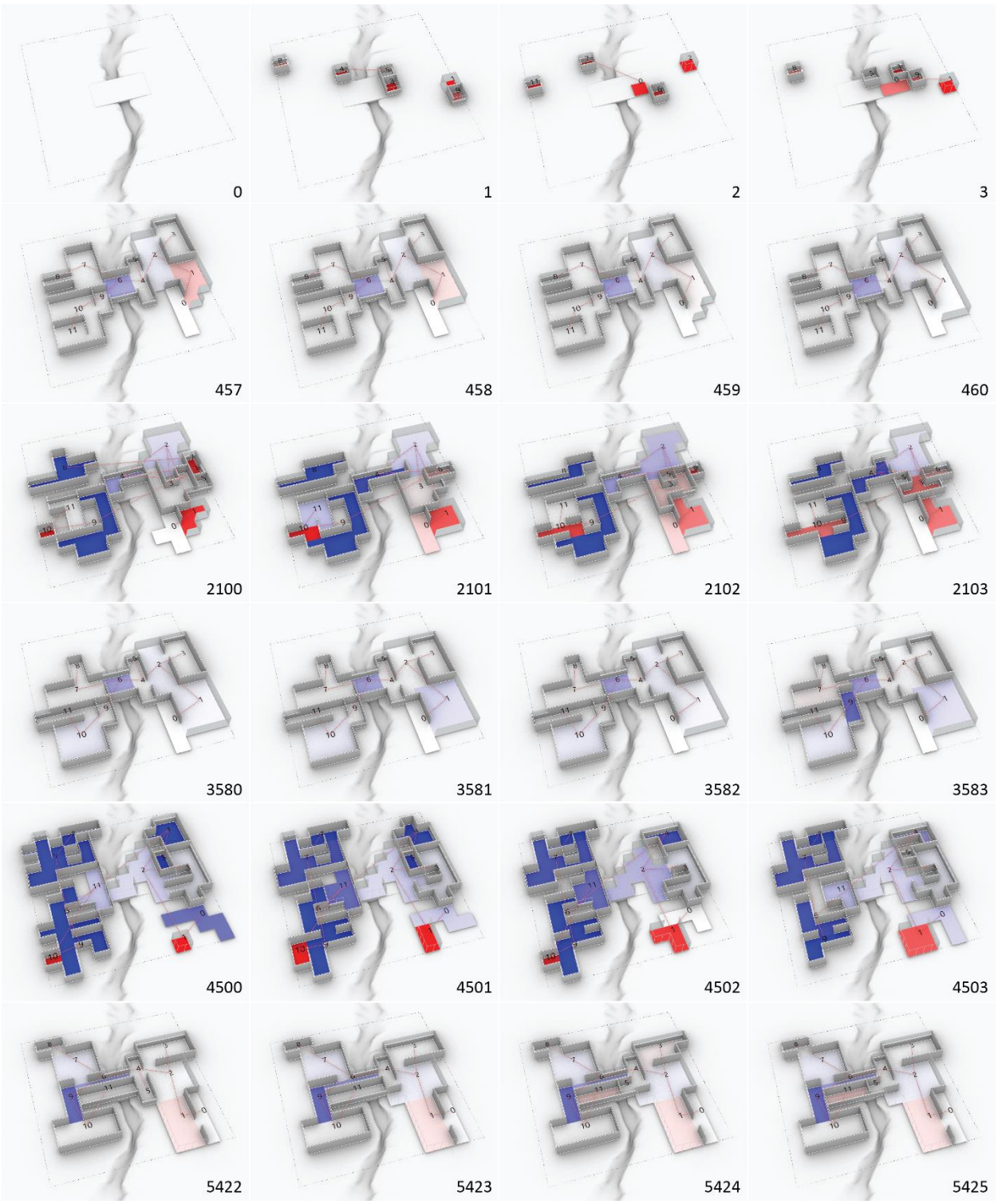


Figure 57 Rows 1-6: simulation of 12 agents in a terrain with a bridge over a stream. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. By Author.



Figure 58 Graphs with the performance of the agents from P3 over one episode for each site. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the three stages with the policy but not the periods with random perturbation. By Author.

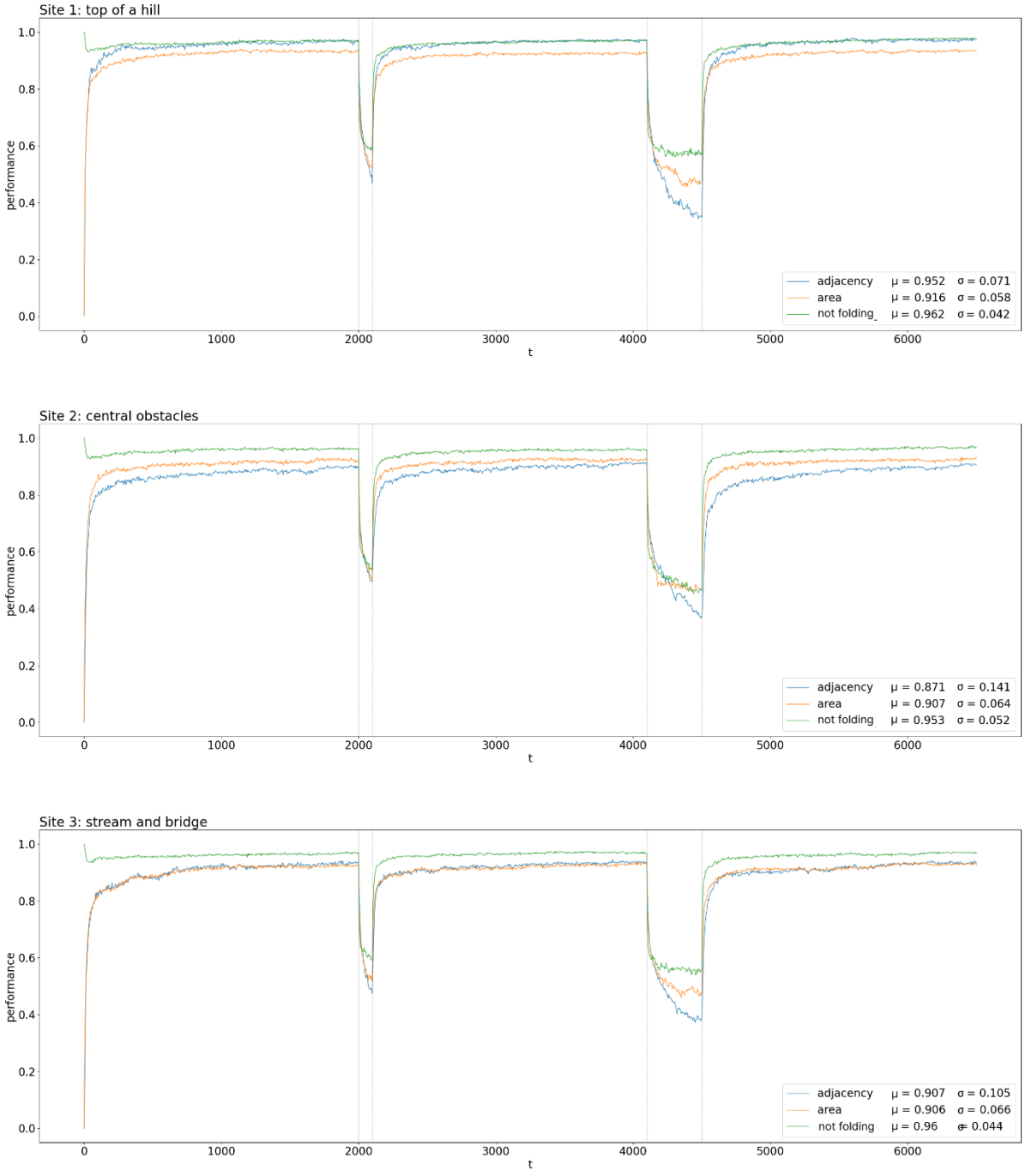


Figure 59 Graphs with the average performance of the agents from P3 over 50 episodes for each site. The areas between dotted lines indicate the periods of perturbation. The means (μ) and standard deviations (σ) consider the three stages with the policy but not the periods with random perturbation. By Author.

Considering the average performance of 50 runs, it is possible to see in Table 9 that the adjustments in P3 slightly improved the performance of the agent with the respect to all metrics and to all sites.

Table 9 Comparison of average utility of P2 and P3 for house environments.

	top of the hill			central obstacles			stream and bridge		
	$P2$	$P3$	$\frac{P3-P2}{P2}$	$P2$	$P3$	$\frac{P3-P2}{P2}$	$P2$	$P3$	$\frac{P3-P2}{P2}$
adjacency	0.937	0.952	1.6%	0.863	0.871	0.9%	0.887	0.907	2.3%
area	0.905	0.916	1.2%	0.906	0.907	0.1%	0.894	0.906	1.3%
not folding	0.946	0.962	1.7%	0.944	0.953	1.0%	0.948	0.960	1.3%

3.3. Design Case 2: art gallery on a large site

Design case 2 is a single-story complex of galleries for local and regional artists, with a focus on the promotion of visual arts and on integrating it to the cultural, gastronomical, and bohemian life of the city. It was designed to test the capacity of generalization of the different policies trained in P2. It consists of an art gallery complex that is modelled by 18 agents with more connections and a loop – see Figure 60. The floorplan of the complex should have a fluid configuration, which provides opportunities to evaluate the typological implications of the policies derived from different spatial goals in P2 and P3.

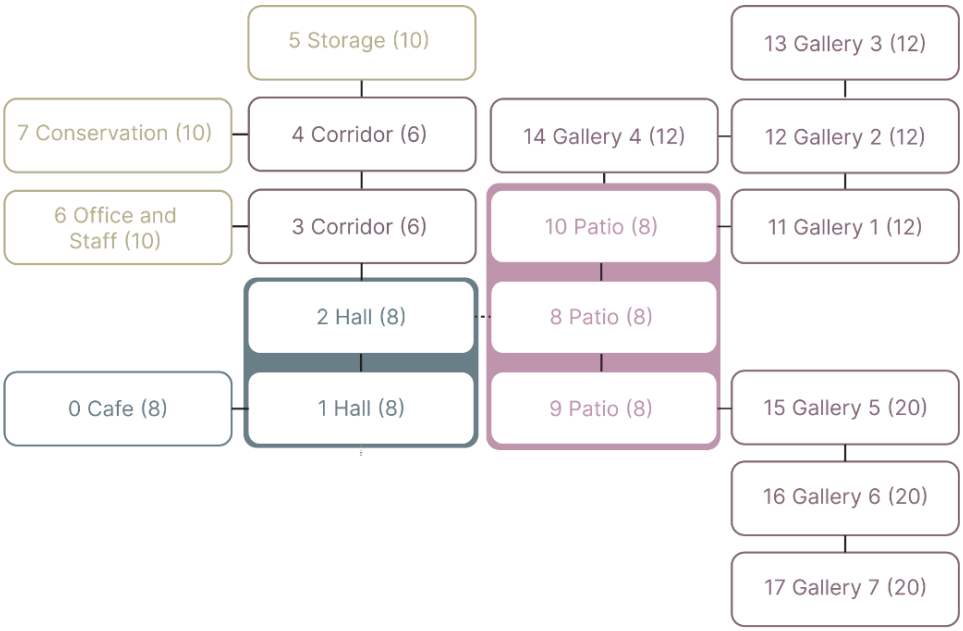


Figure 60 Program for art gallery. The leading number is an identifier of the spaces. The numbers in parenthesis indicate target area in terms of cell units. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces. By Author.

The site for the museum is located between urban lots with one to two-story buildings and an existing park (see Figure 61). As a result, it has an irregular boundary facing back of buildings, fences, and many natural elements such as boulders or trees. Some natural elements are also present inside the site and should be preserved.



Figure 61 Site for the art gallery. Dotted red line represents the boundary and obstacles for the agents. By Author.

Like in the design case 1, I opted for testing the performance of the agents over several steps to evaluate the convergence of the agent to good configurations under different conditions. However, considering the larger number of policies and the good performance of the training, the number of total steps was reduced and there is only one phase with random action selection. More specifically, the agents interact in the environment for 2200 steps, which are divided in two stages that are separated by a single perturbation:

- Stage 1 (0-999): trained policy.
- Perturbation 1 (1000-1199): random action selection.
- Stage 2 (1200-2199): trained policy.

For the design case 2, the analysis focuses on the performance of the different policies trained in P2 and P3. This analysis considers both their convergence to good states and the emerging spatial patterns. Besides, each policy was executed 100 times. In the first 50 cases, the engine initializes the simulation with empty agents at random position. In the last 50 cases, a spring layout method from the library NetworkX ("Spring_layout" n.d.) is applied to the adjacency graph of the program over 50 iterations to optimize the initial placement of each agent. This method uses the Fruchterman-Reingold algorithm (Fruchterman and Reingold 1991) to simulate a force-directed representation of the adjacency network, treating edges as springs and nodes as repelling objects, until the positions in the positions are close to an equilibrium. The idea is to evaluate how much the initial knowledge about the topology can contribute to better convergence in comparison to the strategies developed by the agents with limited view of the environment.

3.3.1. P2

The policy Adjacency and Area generates interesting arrangements of polyominoes with multiple folds (see Figure 62), which result in a maze-like shape both for the room and for the layout. When the target area is larger, the agents converge to a rectangular shape with a few folds, so the galleries look like irregular rooms. In terms of adjacency, the behavior with and without the spatial layout optimization was critical. Without the optimization, the agents have a medium performance

in settling to positions that would guarantee the satisfaction of the adjacencies. With the optimization, the agents achieve excellent performance. This suggests that the agent was not able to develop the initial policy for negotiating positions. However, the agents are in a reasonable location, the policy successfully fine-tune the positions, negotiates the boundaries, and satisfy the required adjacencies.

In the policy Adjacency, Area, and Not Folding (maximum of 3 folds), adding a single fold to a room shape has a large impact in the score. Therefore, the agents opted for generating rows of single cells, which eliminates folds in the shape and reduces the distance between agents (see Figure 64). However, as the action grid restricts the extension to 5 cells, which is smaller than all the target areas in the art gallery program, this policy results in undersized rooms with excellent daylight performance. The agents had a good performance in terms of adjacency, which was further improved in the stages with the spatial layout initialization (Figure 65).

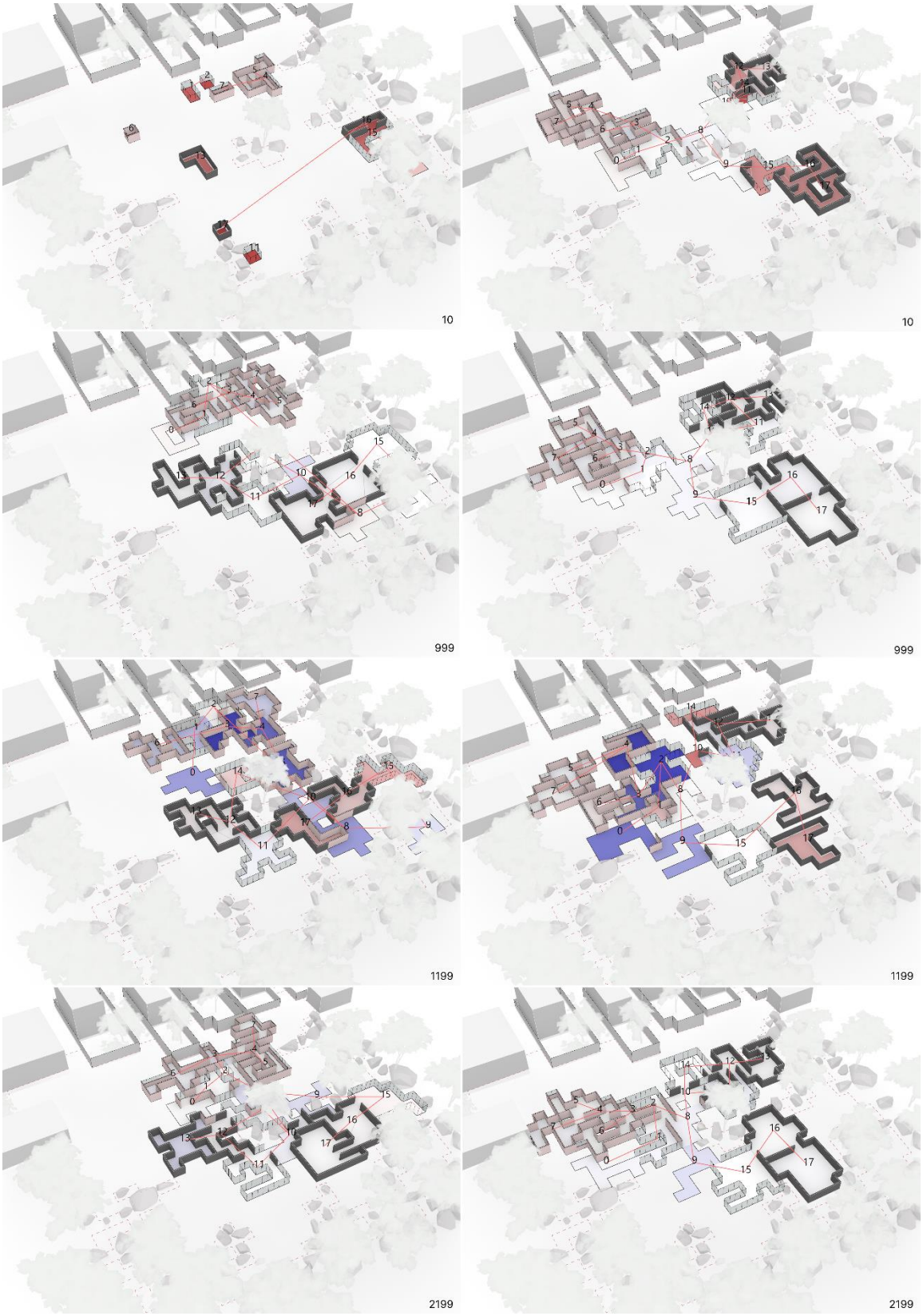


Figure 62 Agents trained for adjacency and area goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

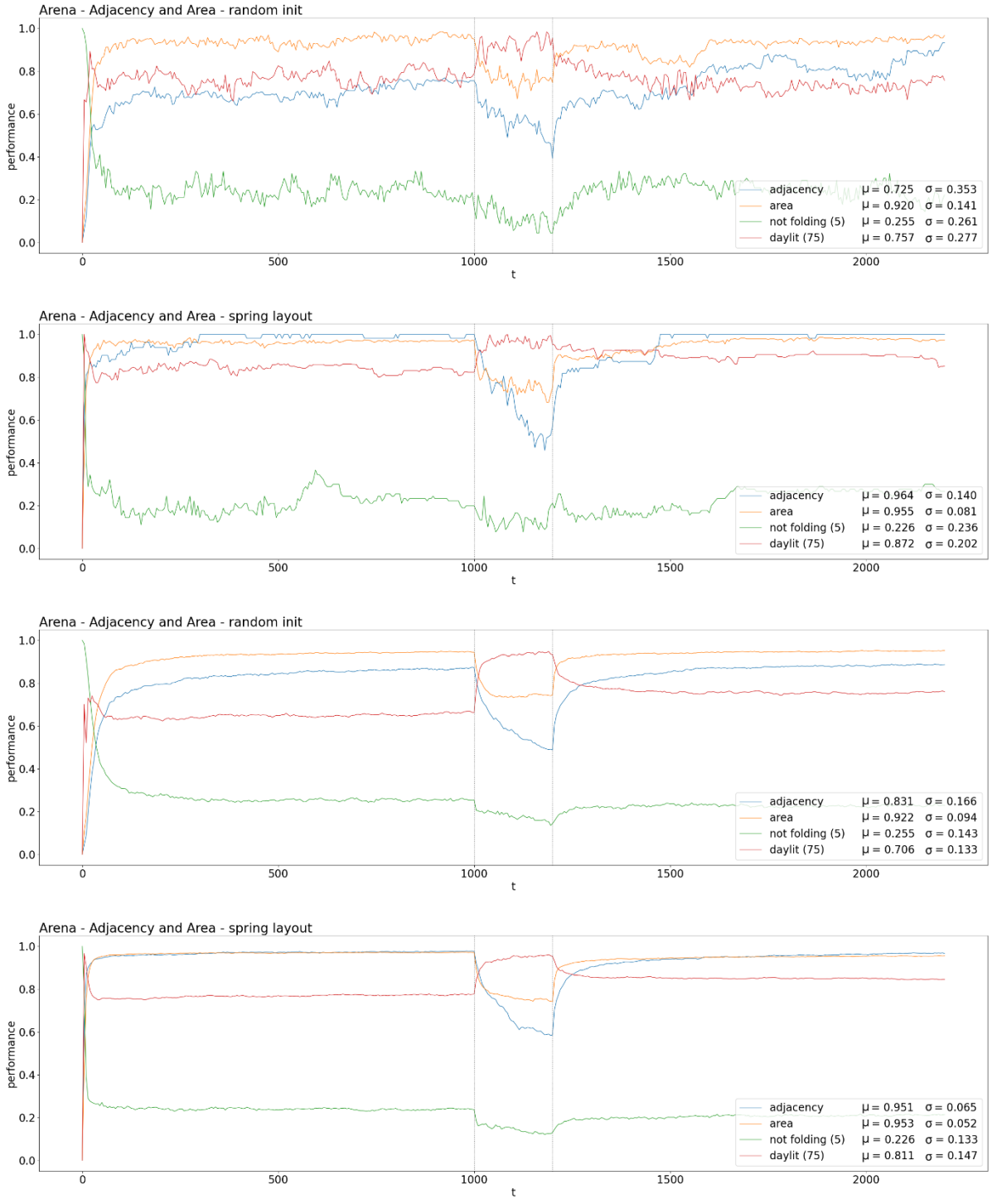


Figure 63 Performance of P2 with policy Adjacency and Area. Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

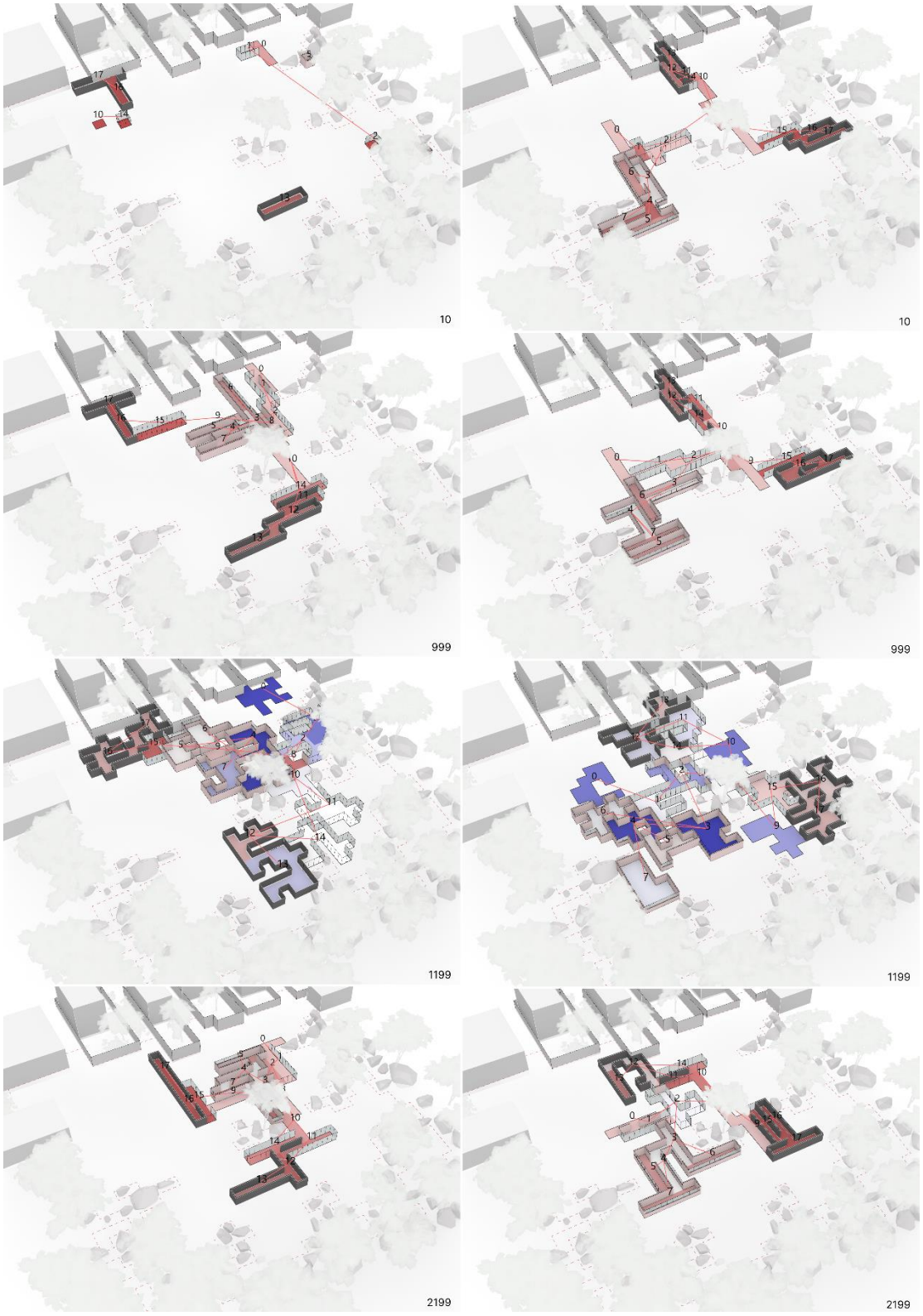


Figure 64 Agents from P3 trained for adjacency, area, and not folding (3) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

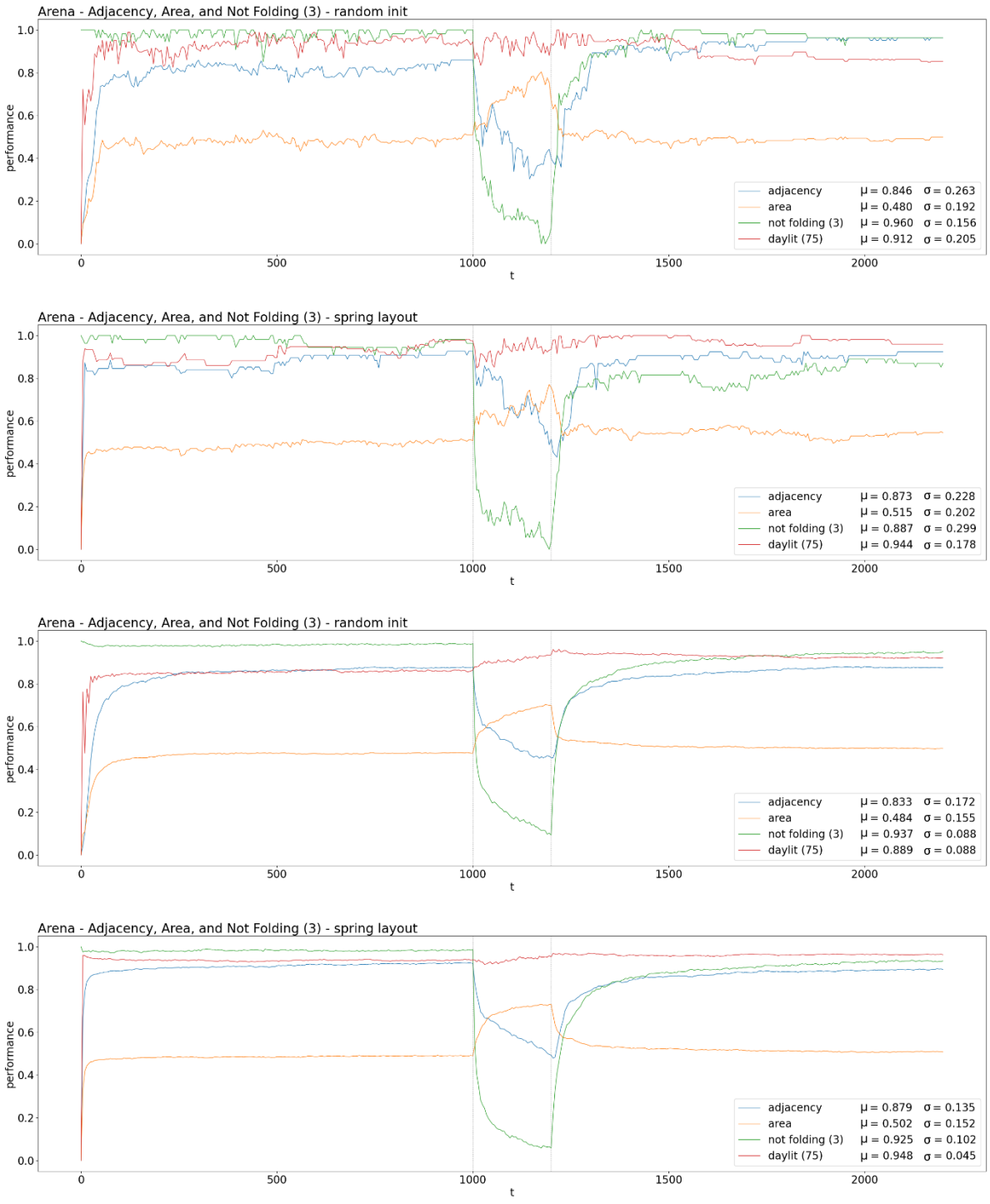


Figure 65 Performance of P2 with policy Adjacency, Area, and Not Folding (3). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

Like the performance in the design case 1, the policy Adjacency, Area, and Not Folding (maximum of 5 folds) resulted in arrangements of rectangular and L-shaped rooms. When the agents are initialized without any cells at random positions, the agents keep jumping to positions closer to the desired adjacencies, then they grow and adjust their shape, which results in a more compact layout and a good performance over all the three objectives (see Figure 70 – col 1, row 2). With the spring layout initialization used in this test, the agents are initialized according to the adjacency graph and positioned further away from the neighbors, so the agents must move closer to satisfy adjacency. Consequently, it results in a free form layout that expands from the center (hall and patios) to the gallery and administrative blocks in the periphery, which resulted in a better performance in terms of adjacency (see Figure 70 – col 2, row 2). In both initializations, after random perturbations the agents recover to a good configuration. See Figure 67 for more details about performance.

In the case of policy Adjacency, Area, and Daylit (75% of the room area) there is a strong pressure to increase the external walls adjacent to open spaces while the art gallery program requires large rooms. Thus, each agent tends to create irregular polyominoes with multiple concavities, which results in a room as a connection of a series of small spaces and niches (see Figure 68). This spatial pattern also creates small external patios in between the agents. In terms of average performance (see graphs in Figure 69), with random initialization or after random perturbation, the agents emphasize the daylit performance, and achieve a medium to good performance for adjacency and area. The spring layout initialization has a positive impact on all the goals considered by the policy, but it is particularly beneficial for the adjacency.

The policy Adjacency, Area, Not Folding (5), and Daylit (75%) imposes many restrictions and conflicts for the agents. Overall, daylight and adjacency are the dominant drivers of the policy. In phase 1, the agents expand into corridor sub-dimensioned shapes with a few folds. After the random perturbation, the agents improve their performance in terms of area but tend to add more folds to preserve the daylight level. Overall, daylit and adjacency values are great in all phases, but even better with the spring layout initialization.

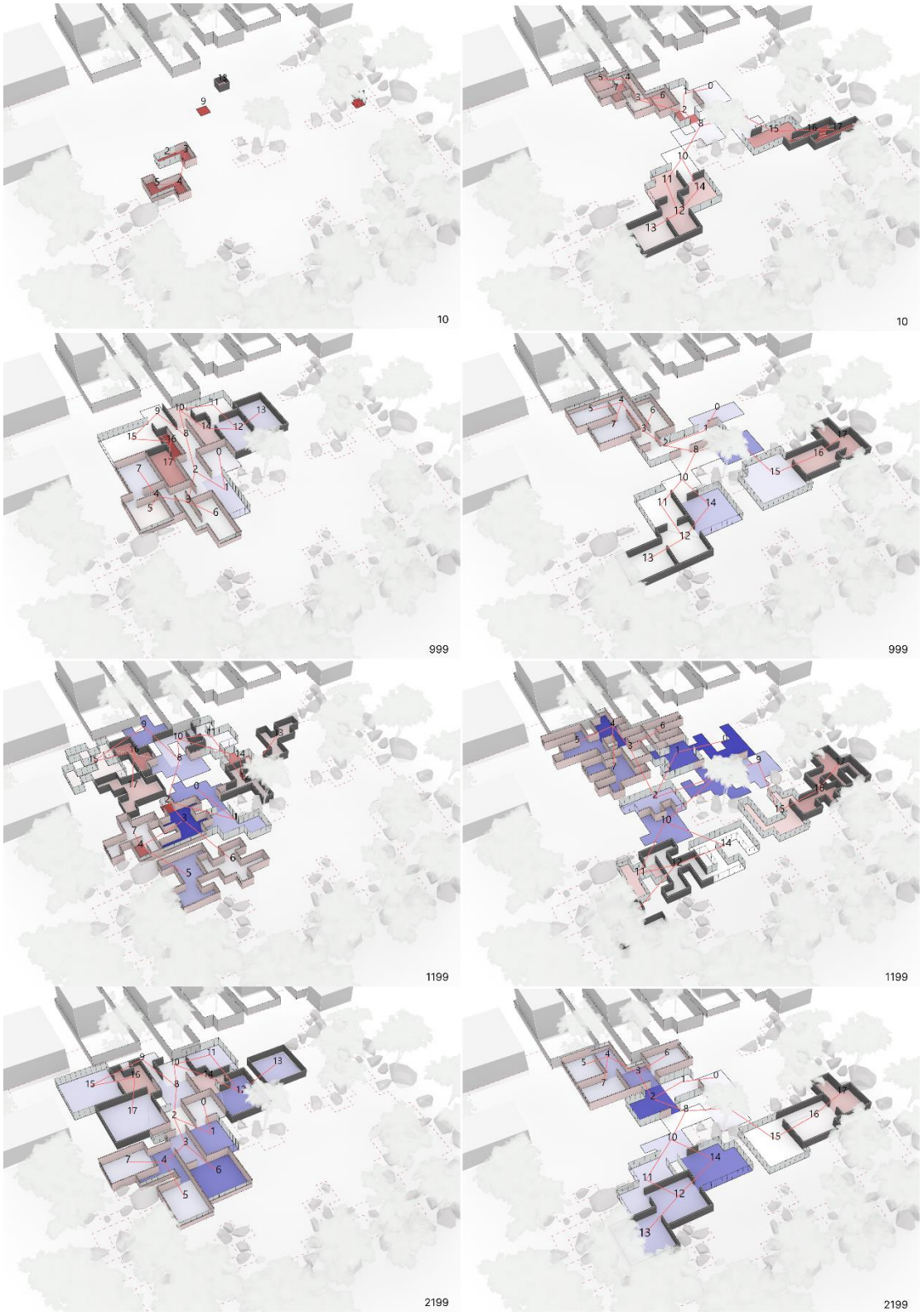


Figure 66 Agents trained for adjacency, area, and not folding (5) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

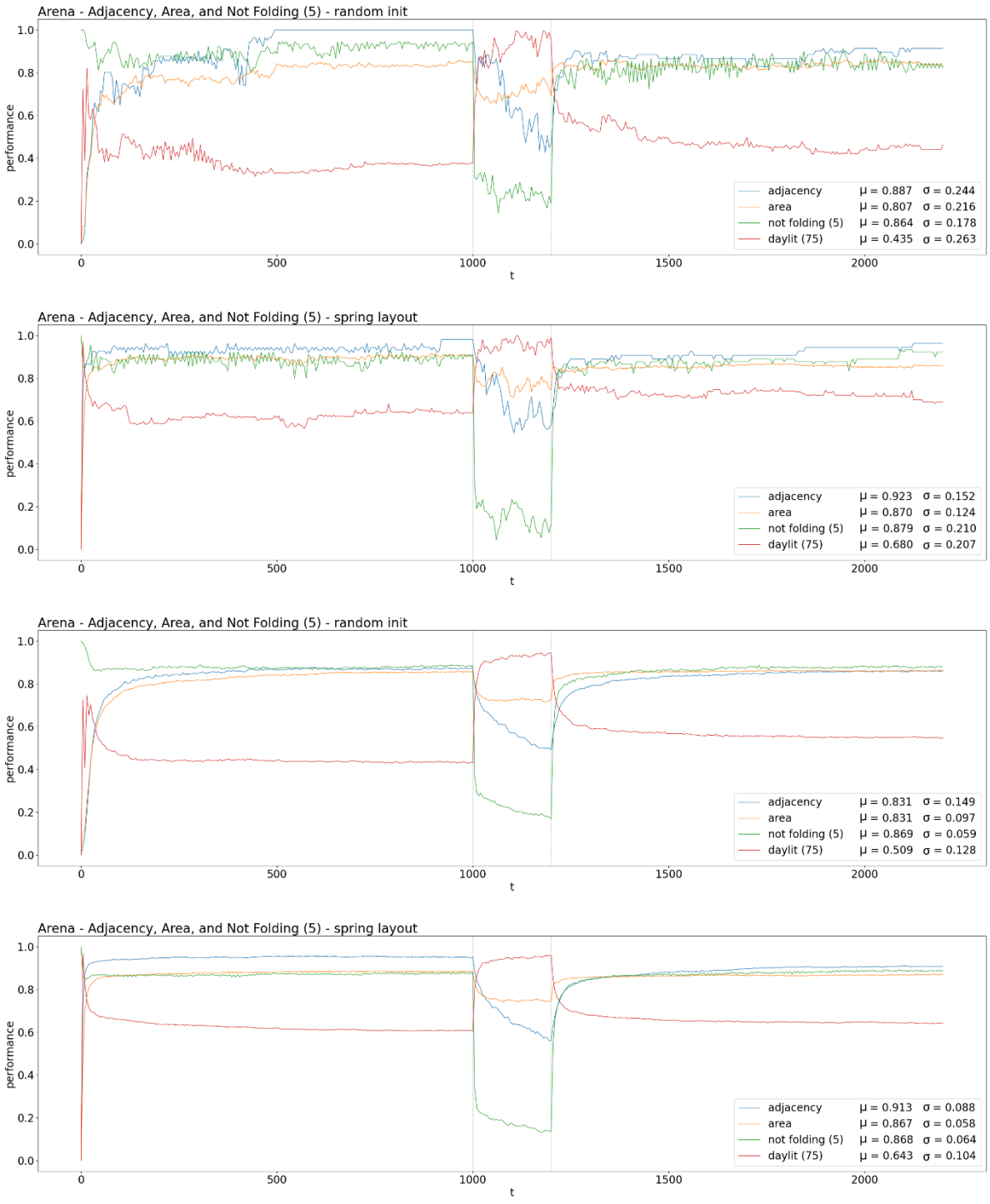


Figure 67 Performance of P2 with policy Adjacency, Area, and Not Folding (5). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

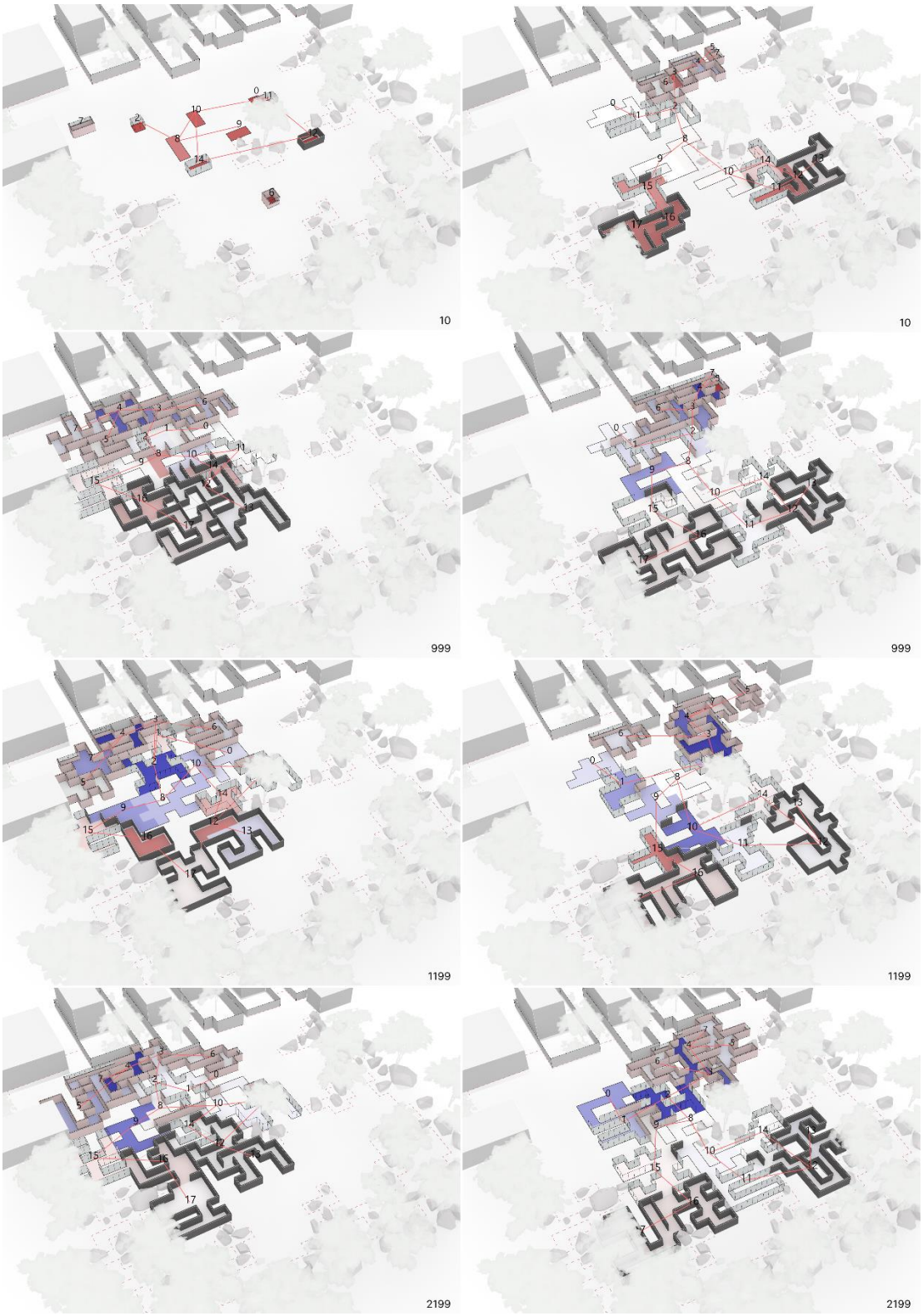


Figure 68 Agents trained for adjacency, area, and daylit (75%) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

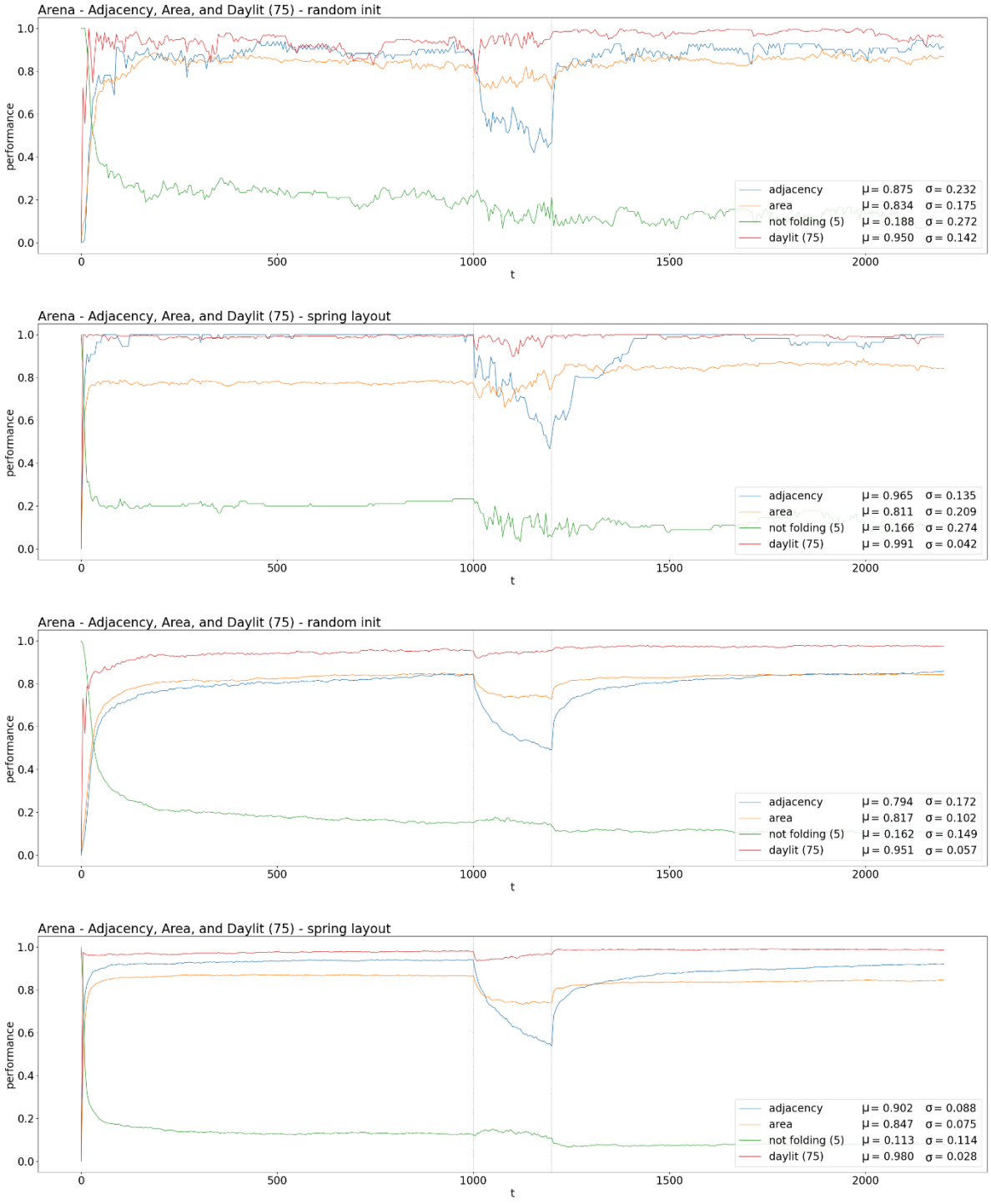


Figure 69 Performance of P2 with policy Adjacency, Area, and Daylit (75%). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

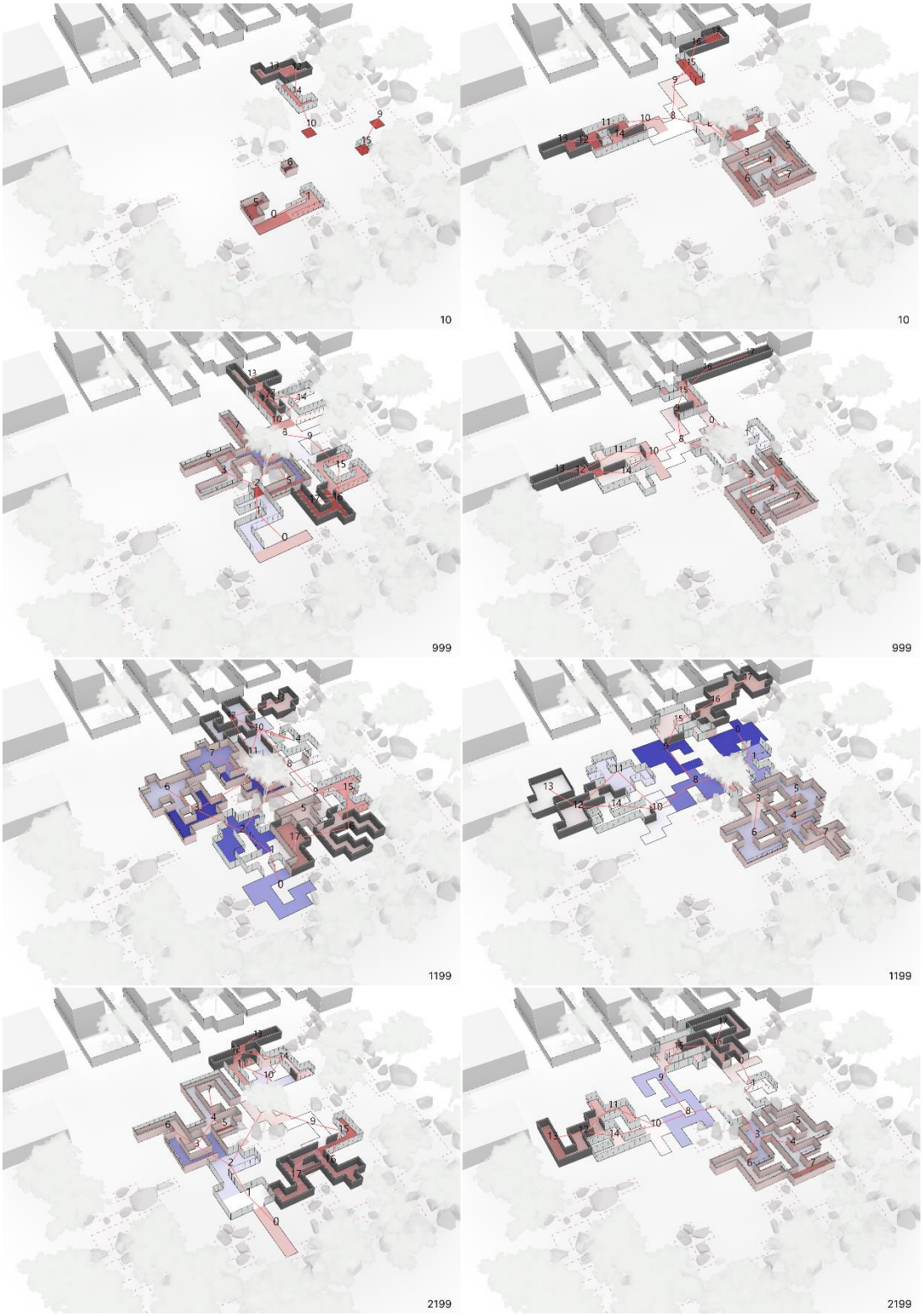


Figure 70 Agents trained for adjacency, area, not folding (5), and daylit (75%) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

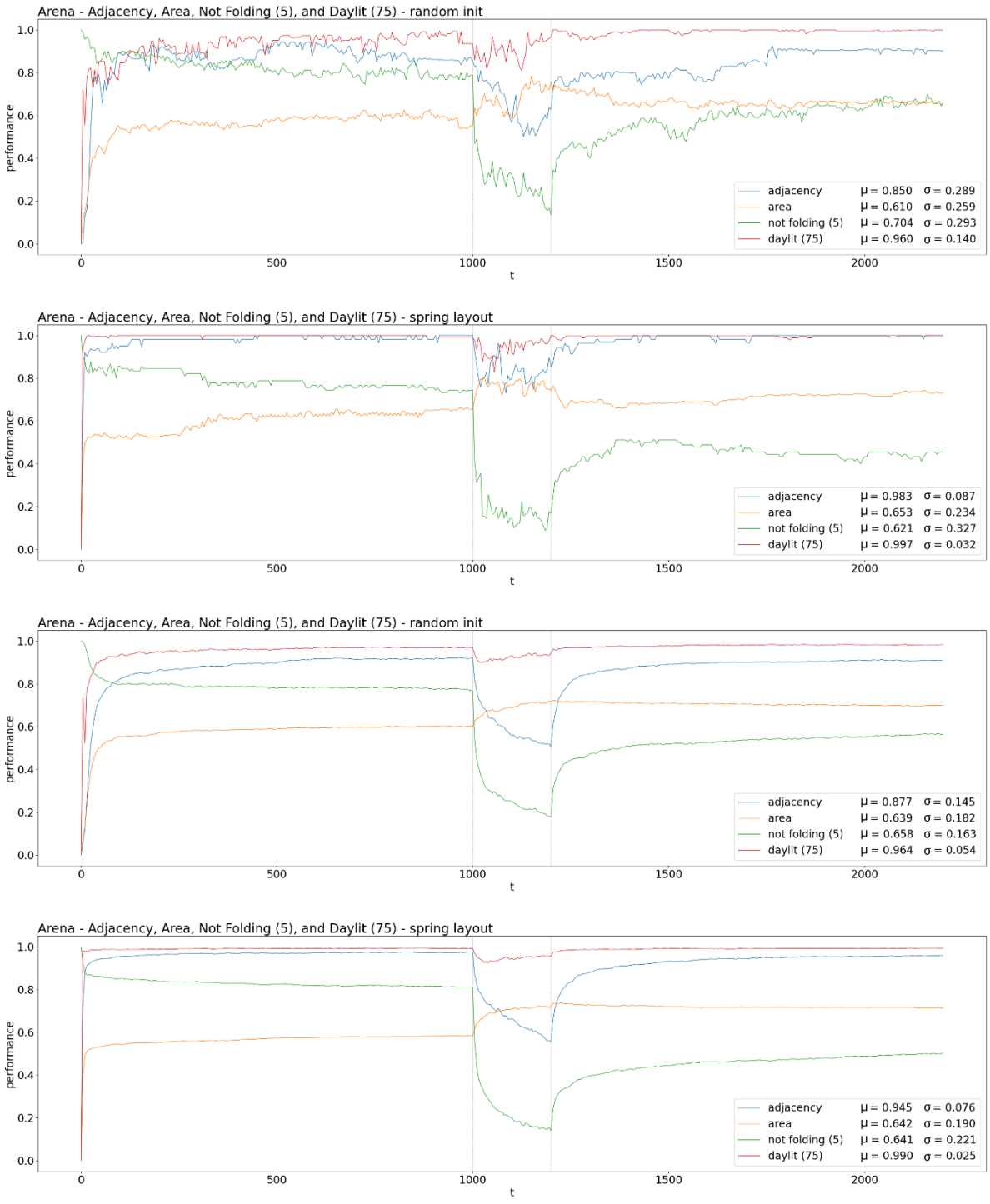


Figure 71 Performance of P2 with policy Adjacency, Area, Not Folding (5), and Daylit (75%). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). By Author.

3.3.2. P3

It is important to notice that except for Adjacency, Area, and Not Folding (5) all the policies were trained over 200 epochs, in contrast to the 300 epochs used in P3. Therefore, only the standard policies Adjacency, Area, and Not Folding (5) and the new Adjacency, Area, Not Folding (5), and Daylit (50%) of P3 will be analyzed for the design case 2. Both will be evaluated quantitatively and qualitatively, and the former will be used to compare the performance of P2 and P3. Overall, the other policies resulted in similar spatial patterns but with similar or slightly worse performance, probably from the shorter training session. The images and graphs for their performance are available in 7.8 Appendix H: Additional data for the design artifact evaluation.

Like in the previous prototype versions and case, the policy Adjacency, Area, and Not Folding (maximum of 5 folds) resulted in arrangements of rectangular and L-shaped rooms (see Figure 72). Also, while with the random initialization, the arrangement is more compact, the spring layout initialization stimulates a free layout based on a central core and centrifugal blocks with better performance overall (Figure 73). In both initializations, after random perturbations the agents recover to a good configuration. See for more details about performance. The table below describes the difference in performance between P2 an P3 for this policy. While the spring layout seemed to have a stronger impact for both versions, the changes implemented in P3 also contributed for a better performance in most of the metrics with or without spring layout (see Table 10). The exception is adjacency which had a slight decrease (-0.2%) for both cases. Another important note is that by stimulating a more disperse and centrifugal layout, the spring layout improves daylight performance of agents that were not trained for that (see example in Table 10).

Table 10 Average utility for the different utility components for the Adjacency, Area, and Not Folding (5) considering 50 samples for P2 with random initialization (P2R), P2 with spring layout (P2SL), P3 with random initialization (P3R), P3 with spring layout (P3SL).

	$P2R$	$P2SL$	$\frac{P2SL - P2R}{P2R}$	$P3R$	$P3SL$	$\frac{P3SL - P3R}{P3R}$	$\frac{P3R - P2R}{P2R}$	$\frac{P3SL - P2SL}{P2SL}$
adjacency	0.831	0.913	9.9%	0.829	0.911	9.9%	-0.2%	-0.2%
area	0.831	0.867	4.3%	0.833	0.891	7.0%	0.2%	2.8%
not folding (5)	0.869	0.868	-0.1%	0.876	0.895	2.2%	0.8%	3.1%
daylit (75%)*	0.509	0.643	26.3%	0.530	0.675	27.4%	4.1%	5.0%
*Not part of the goals of the policy but displayed here to analyze the general effect of the spring layout algorithm								

In Adjacency, Area, Not Folding (5), and Daylit (50%), the reduced parameter of daylight in comparison to the previous policies in P2, results in a policy that addresses daylight on the global arrangement and secondarily in the shape of the rooms. As a result, the rectangular arrangements look like the ones for Adjacency, Area, and Not Folding (5). However, the agents themselves organize a more disperse and centrifugal layout without the need of the spring layout algorithm. They also prevent the layout to be adjacent to borders of the site (see Figure 74). Overall, the agents opt for having an area slightly beyond or above the target area to improve the other metrics (see Figure 75).

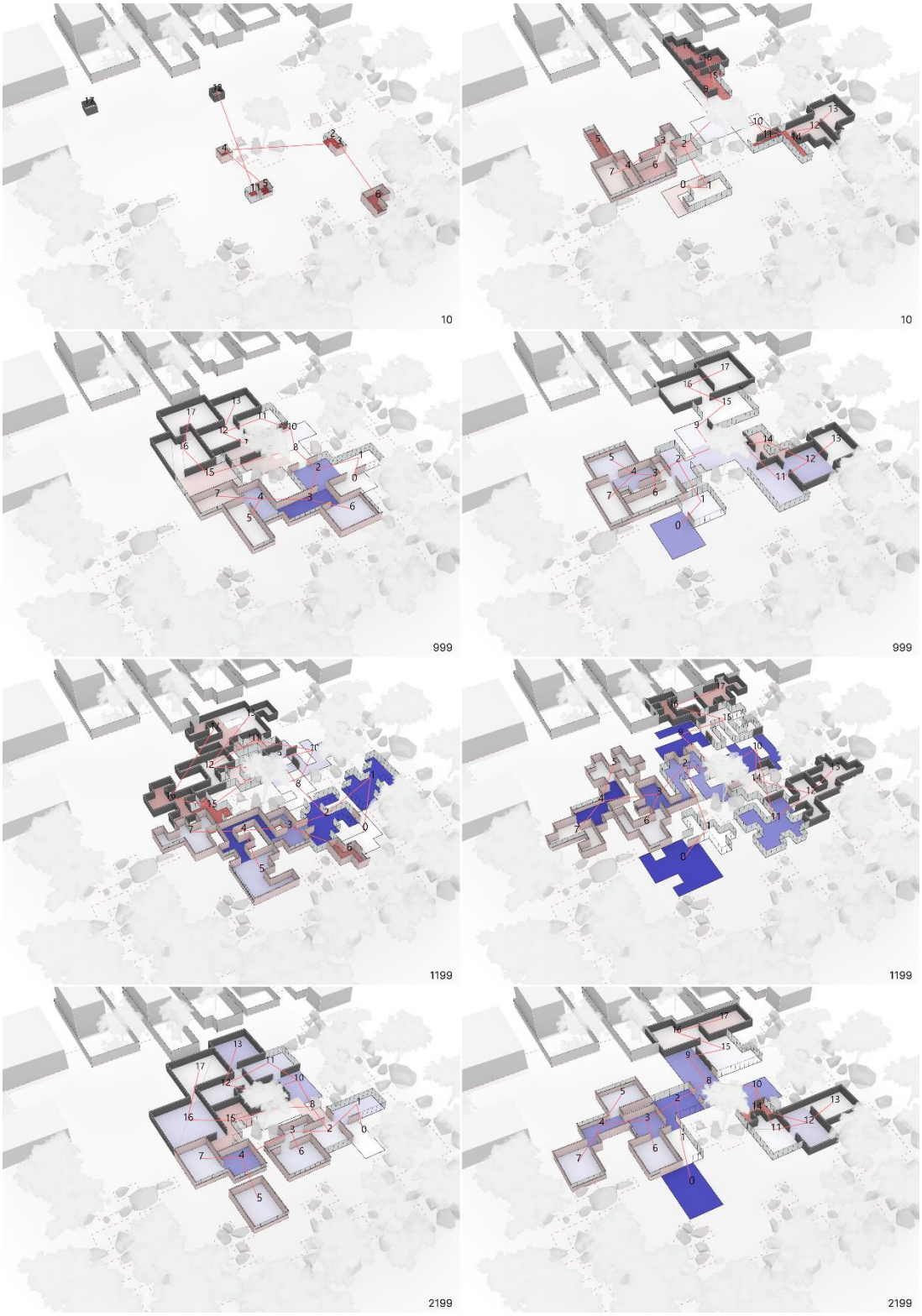


Figure 72 Agents trained for adjacency, area, and not folding (5) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles. Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

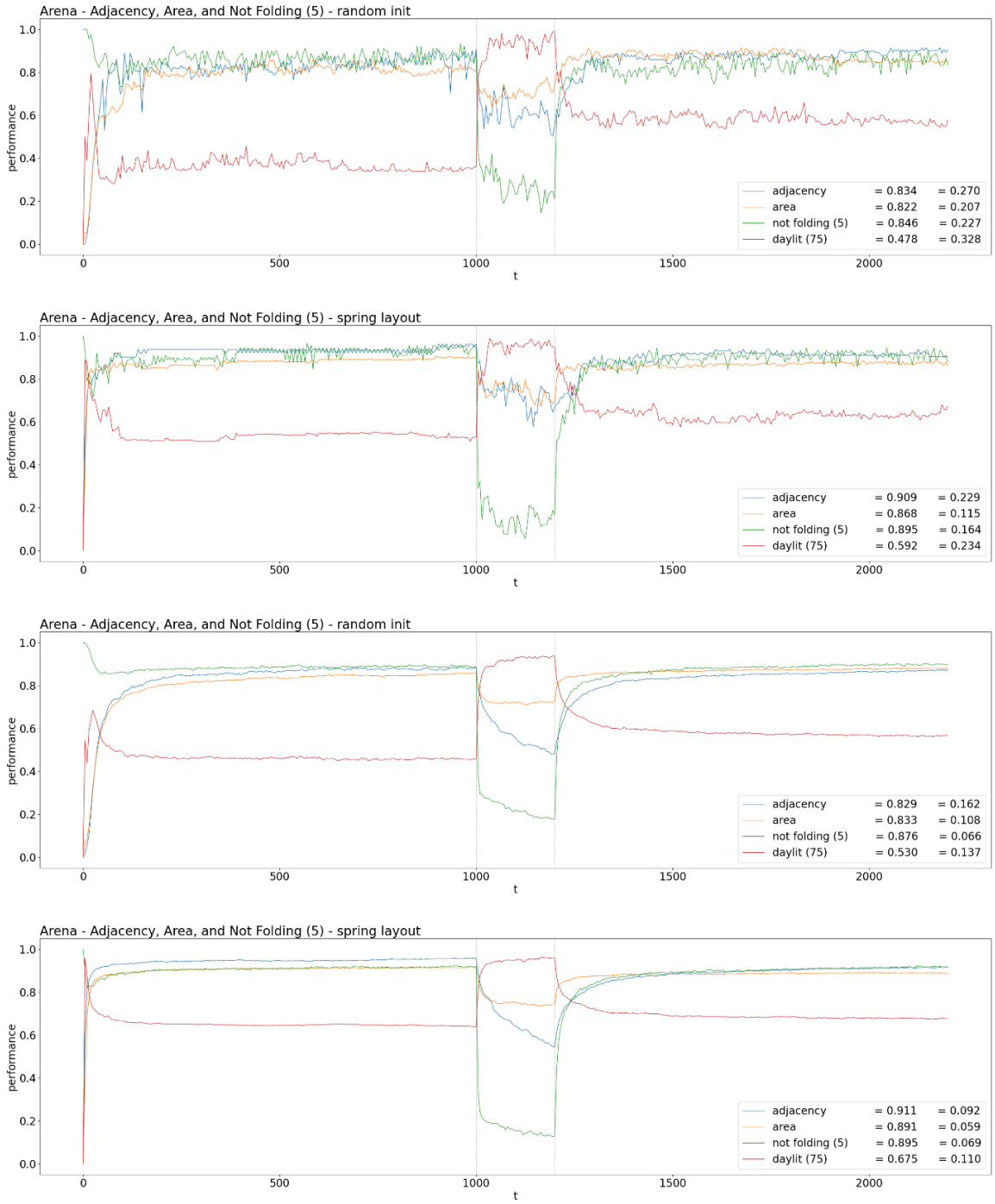


Figure 73 Performance of P3 with policy Adjacency, Area, and Not Folding (5). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

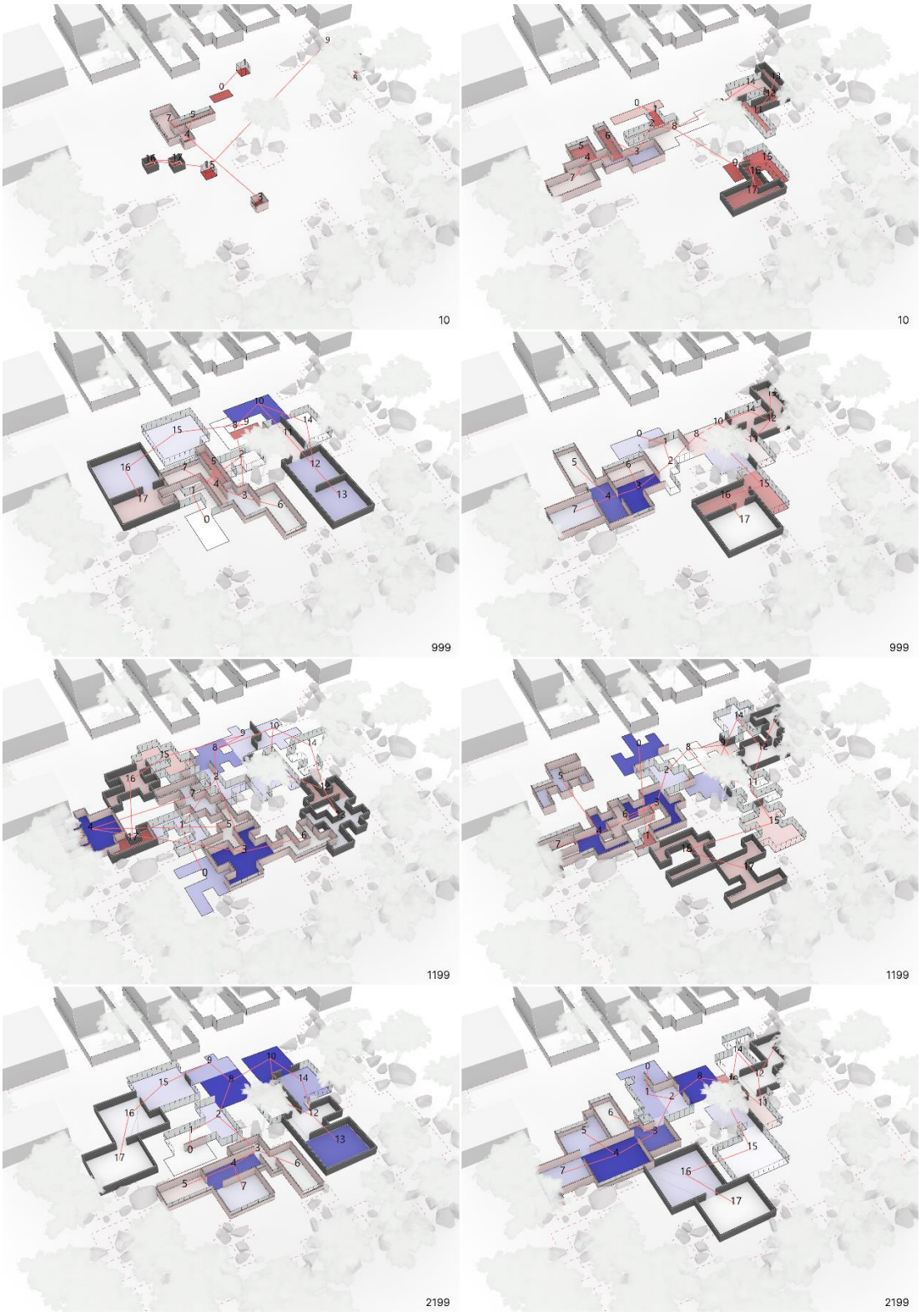


Figure 74 Agents trained for adjacency, area, not folding (5), and daylit (50%) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

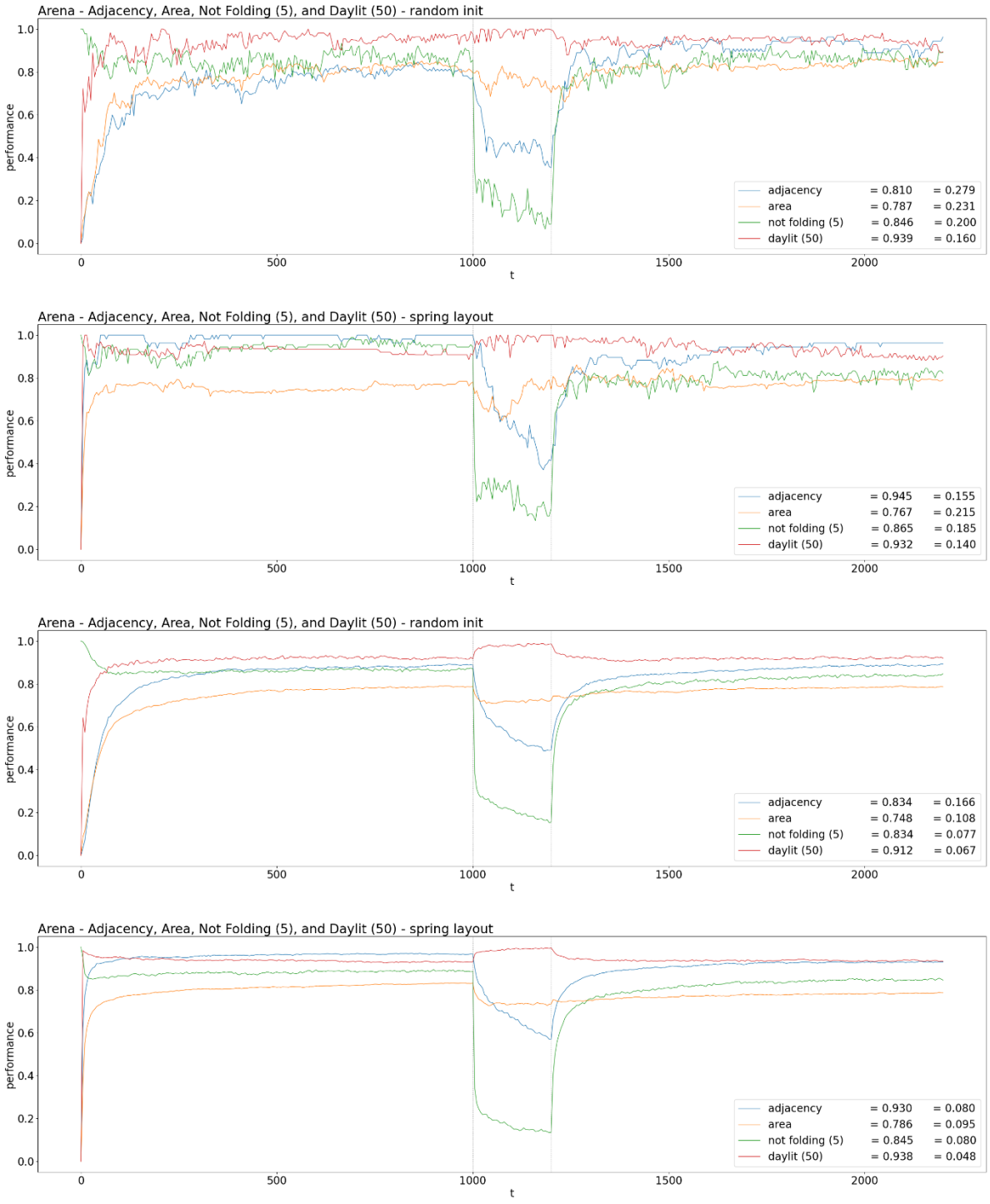


Figure 75 Performance of P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50%). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

3.4. Design Case 3: site-less house complex

This case comprehends the speculative design of a large single-floor complex composed of 8 units with the same area and adjacency specification as in the case 1. However, to understand the spatial qualities of policies that were not developed for the house floorplans or for conventional floorplans, the program is not assumed to be a house but an agglomeration of 8 buildings of the same type on a site. Four policies were selected to represent different spatial qualities

- P2 with policy Adjacency, Area, and Daylit (75)
- P2 with policy Adjacency, Area, and Not Folding (3)
- P3 with policy Adjacency, Area, and Not Folding (5)
- P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50)

The complex occupies an empty and flat site of 50×40 units. In total, there are 96 agents in the simulation, which are initialized with the spring layout algorithms and run for 2000 steps. This was repeated 50 times for each policy. The goal of this case is to

- evaluate if the policy can generalize to a design case with many more agents than in the random training
- evaluate the possibility of the system to support the parallel design of multiple buildings on a site
- visualize layout types that emerge from the interaction of many agents with a shared policy

For each of these policies, a matrix with the last state of each of the 50 runs of the policy is presented. Then, two options are selected for a more detailed representation with the following parameters

- Gride cell with 2m × 2m
- Floor to ceiling 2.70 meters
- Glazing from floor to ceiling on the living room and dining room
- Glazing from 2m to 2.70 meters on the other spaces

3.4.1. P2 with policy Adjacency, Area, and Daylit (75)

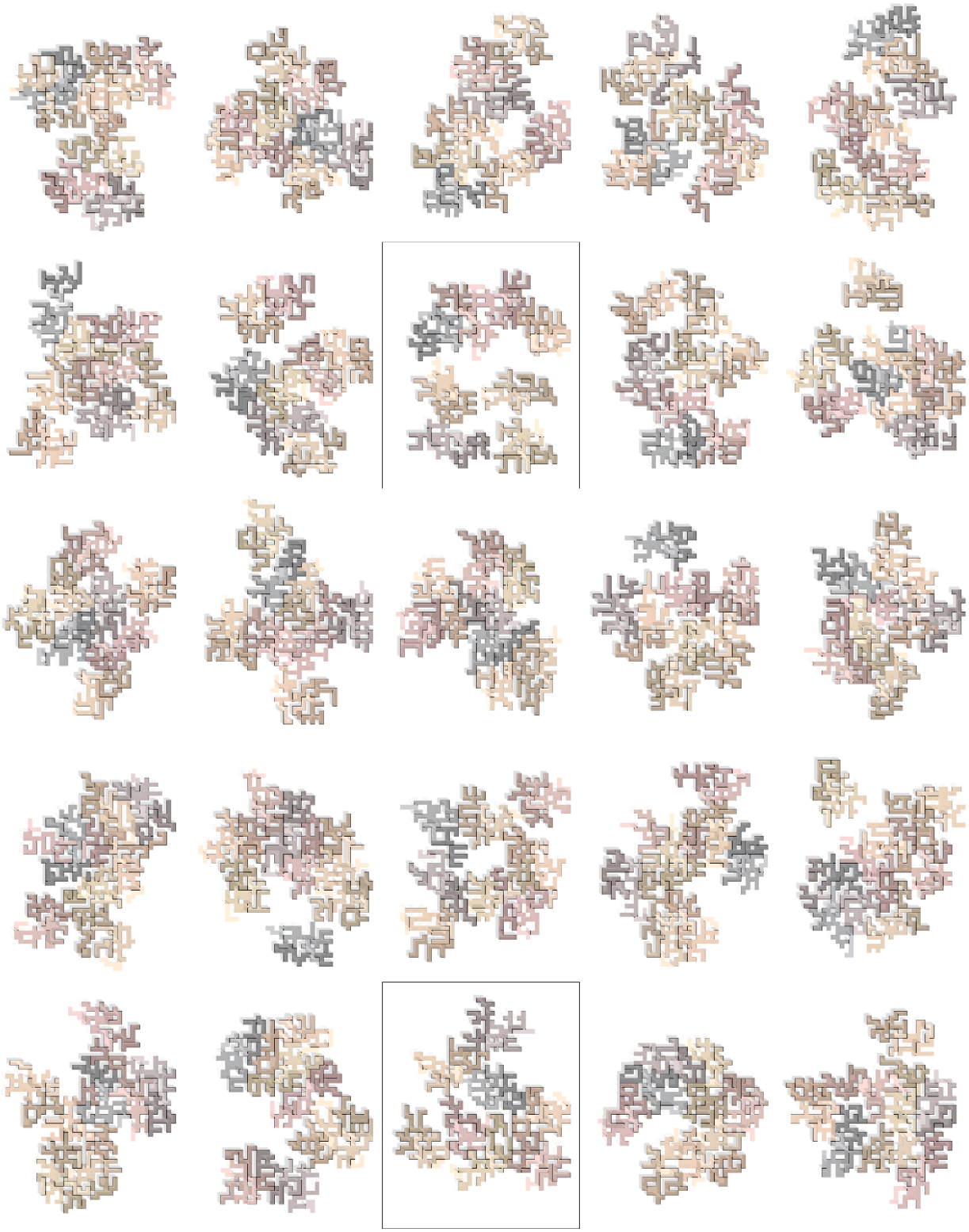


Figure 76 First 25 configurations produced by P2 with policy Adjacency, Area, and Daylit (75) over 2000 steps. The configurations highlighted were selected for a more detailed description. By Author.

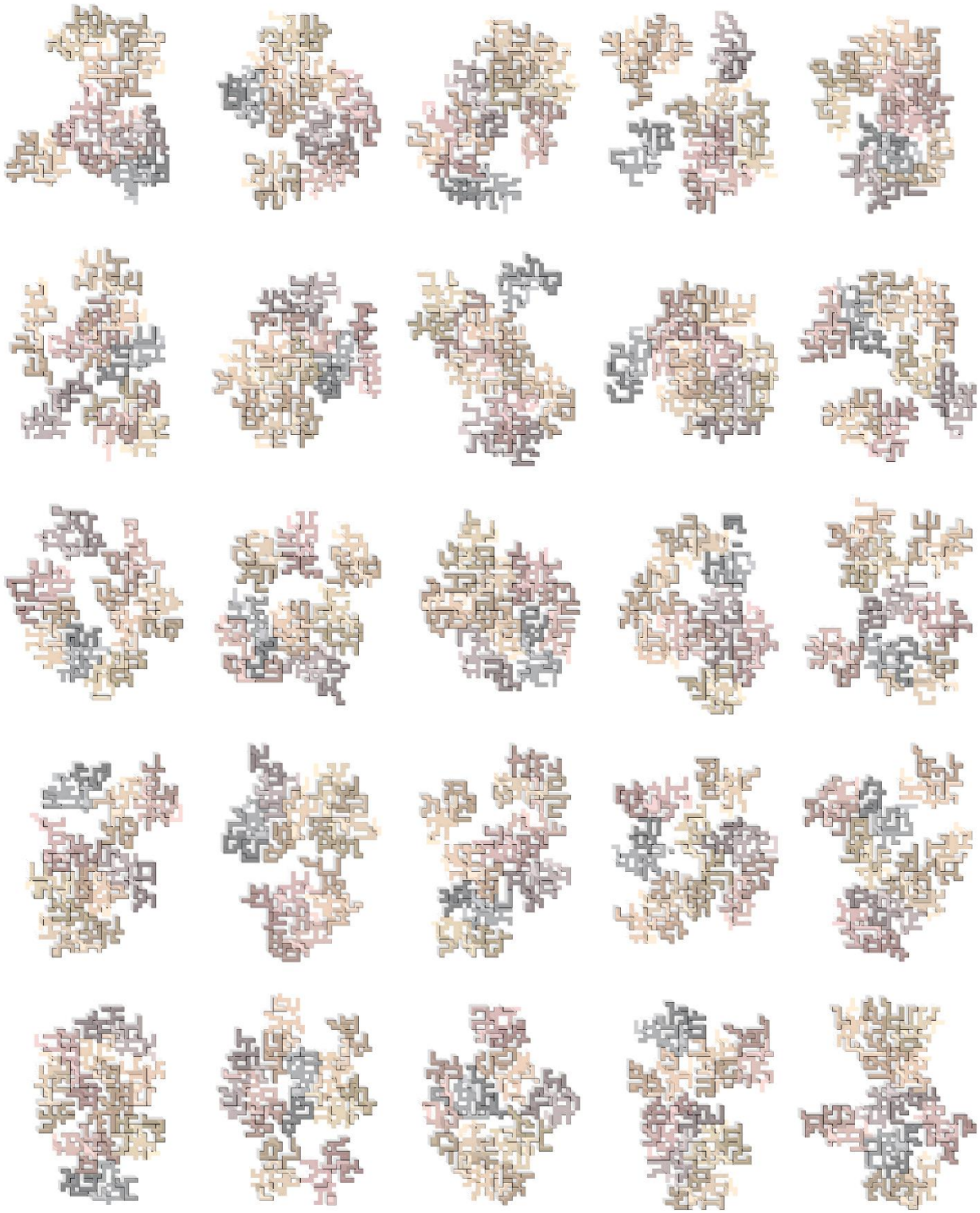


Figure 77 Last 25 configurations produced by P2 with policy Adjacency, Area, and Daylit (75) over 2000 steps. By Author.

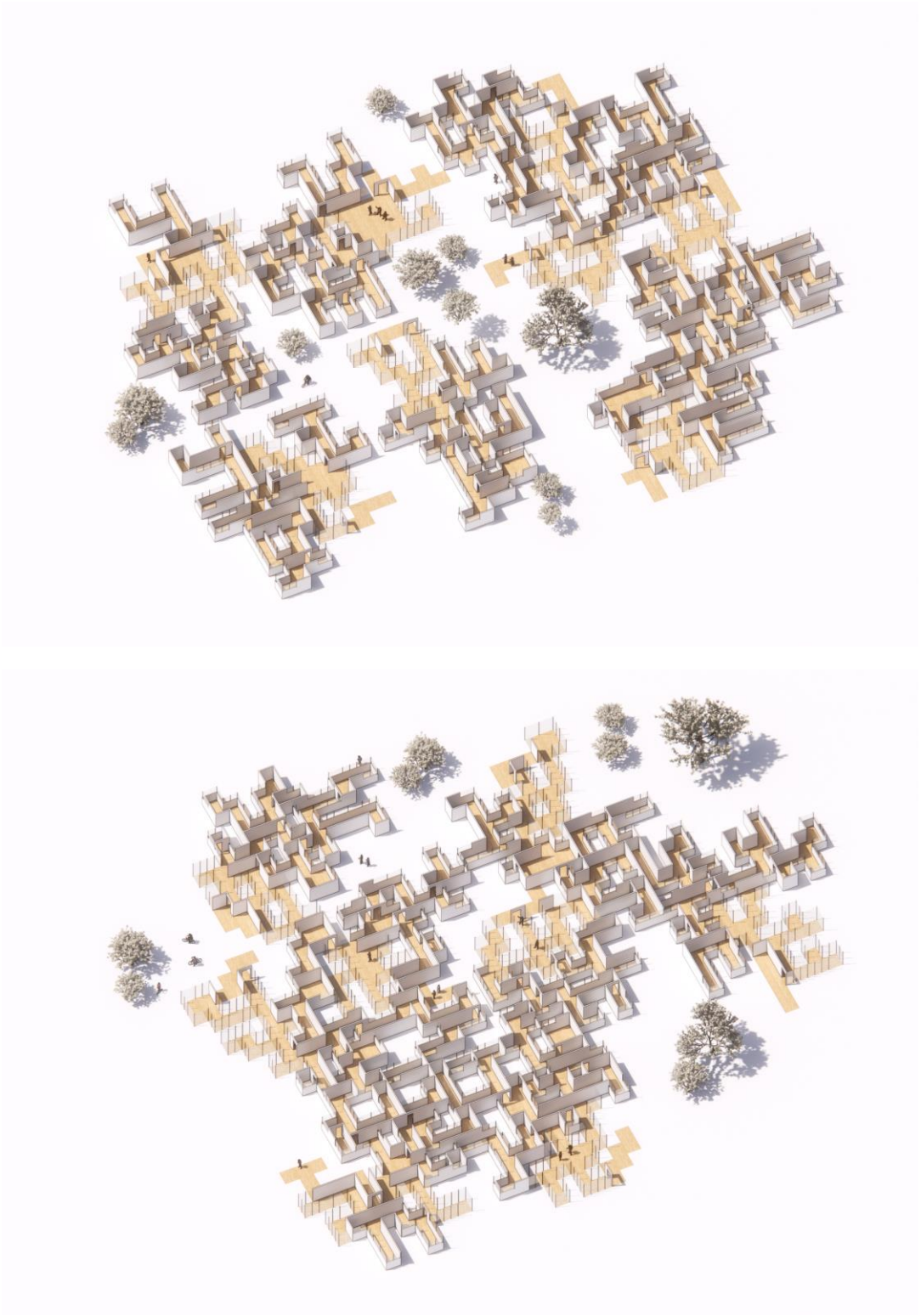


Figure 78 Selected configurations produced by P2 with policy Adjacency, Area, and Daylit (75) over 2000 steps. By Author.

3.4.2. P2 with policy Adjacency, Area, and Not Folding (3)



Figure 79 First 25 configurations produced by P2 with policy Adjacency, Area, and Not Folding (3) over 2000 steps. The configurations highlighted were selected for a more detailed description. By Author.



Figure 80 Last 25 configurations produced by P2 with policy Adjacency, Area, and Not Folding (3) over 2000 steps. By Author.

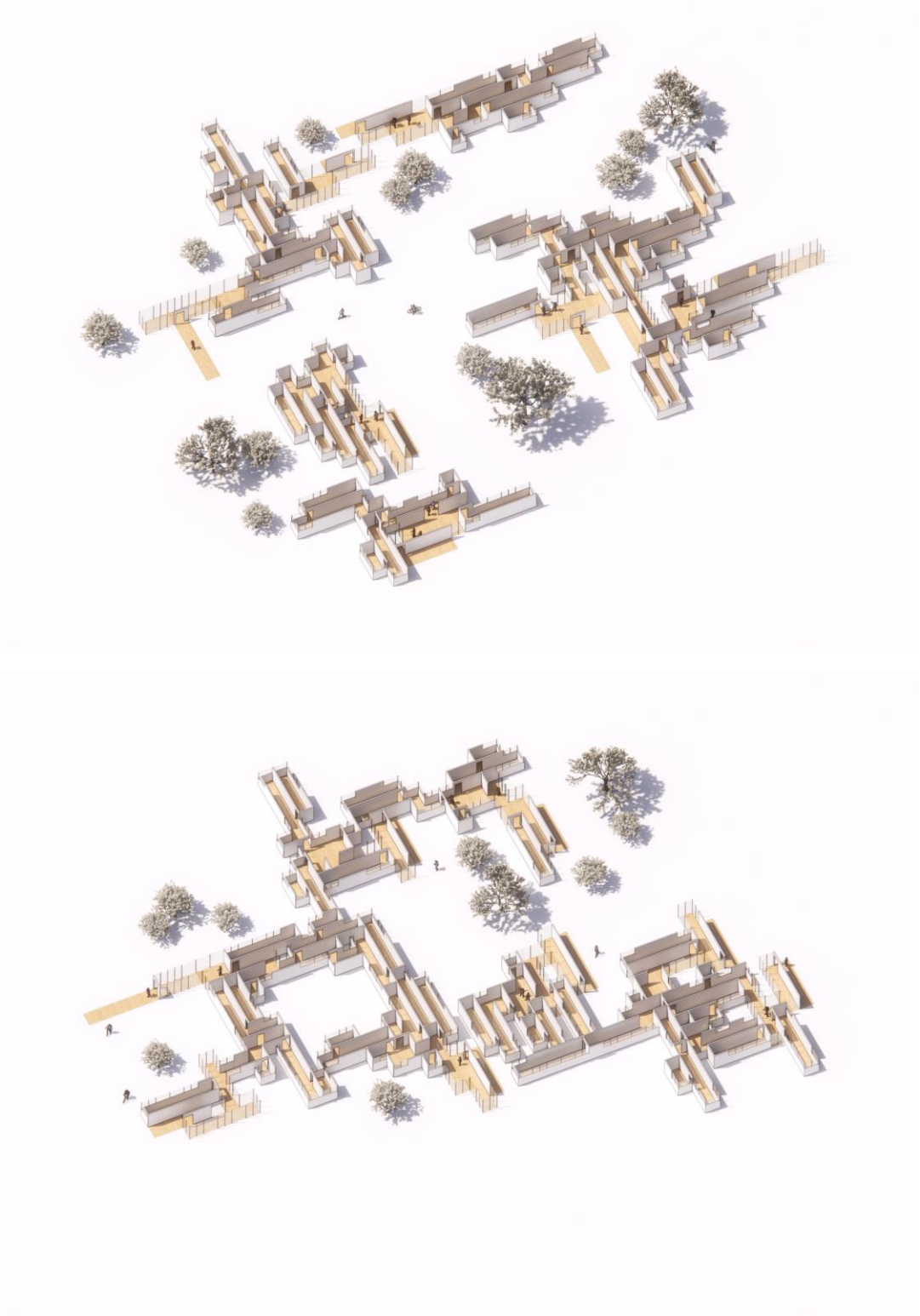


Figure 81 Selected configurations produced by P2 with policy Adjacency, Area, and Not Folding (3) over 2000 steps. By Author.

3.4.3. P3 with policy Adjacency, Area, and Not Folding (5)



Figure 82 First 25 configurations produced by P3 with policy Adjacency, Area, and Not Folding (5) over 2000 steps. The configurations highlighted were selected for a more detailed description. By Author.

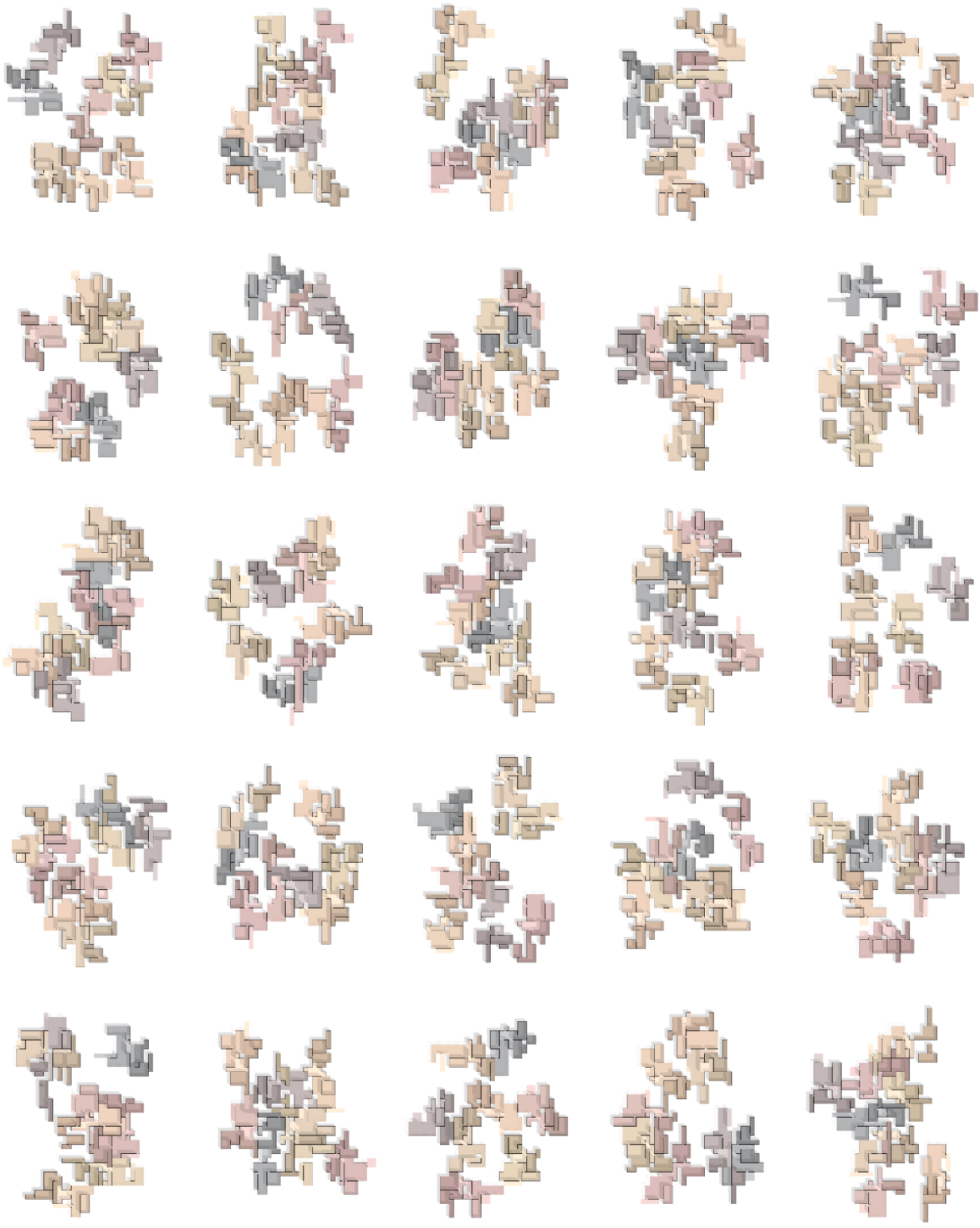


Figure 83 Last 25 configurations produced by P3 with policy Adjacency, Area, and Not Folding (5) over 2000 steps. By Author.

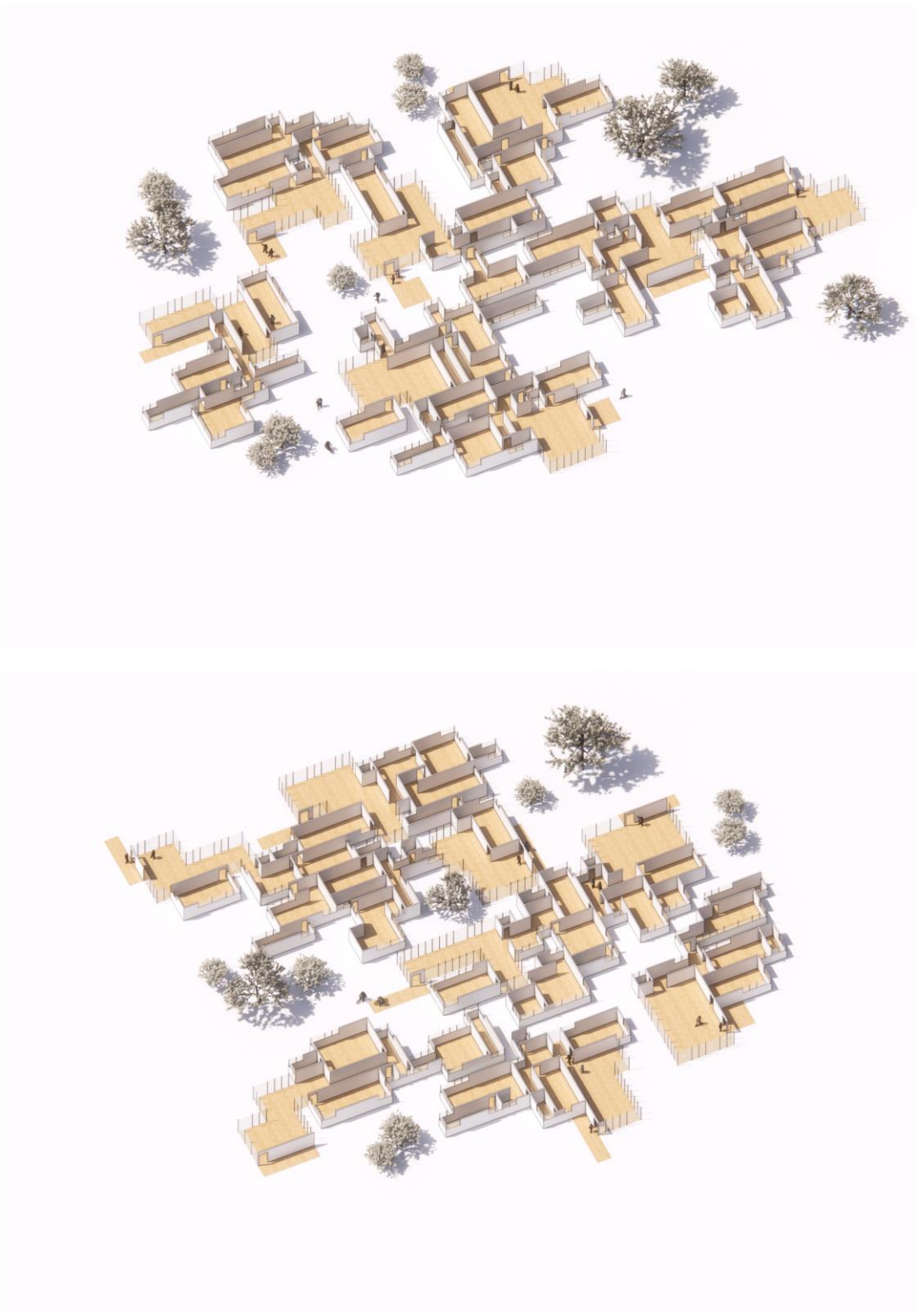


Figure 84 Selected configurations produced by P3 with policy Adjacency, Area, and Not Folding (5) over 2000 steps. By Author.

3.4.4. P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50)



Figure 85 First 25 configurations produced by P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50) over 2000 steps. The configurations highlighted were selected for a more detailed description. By Author.

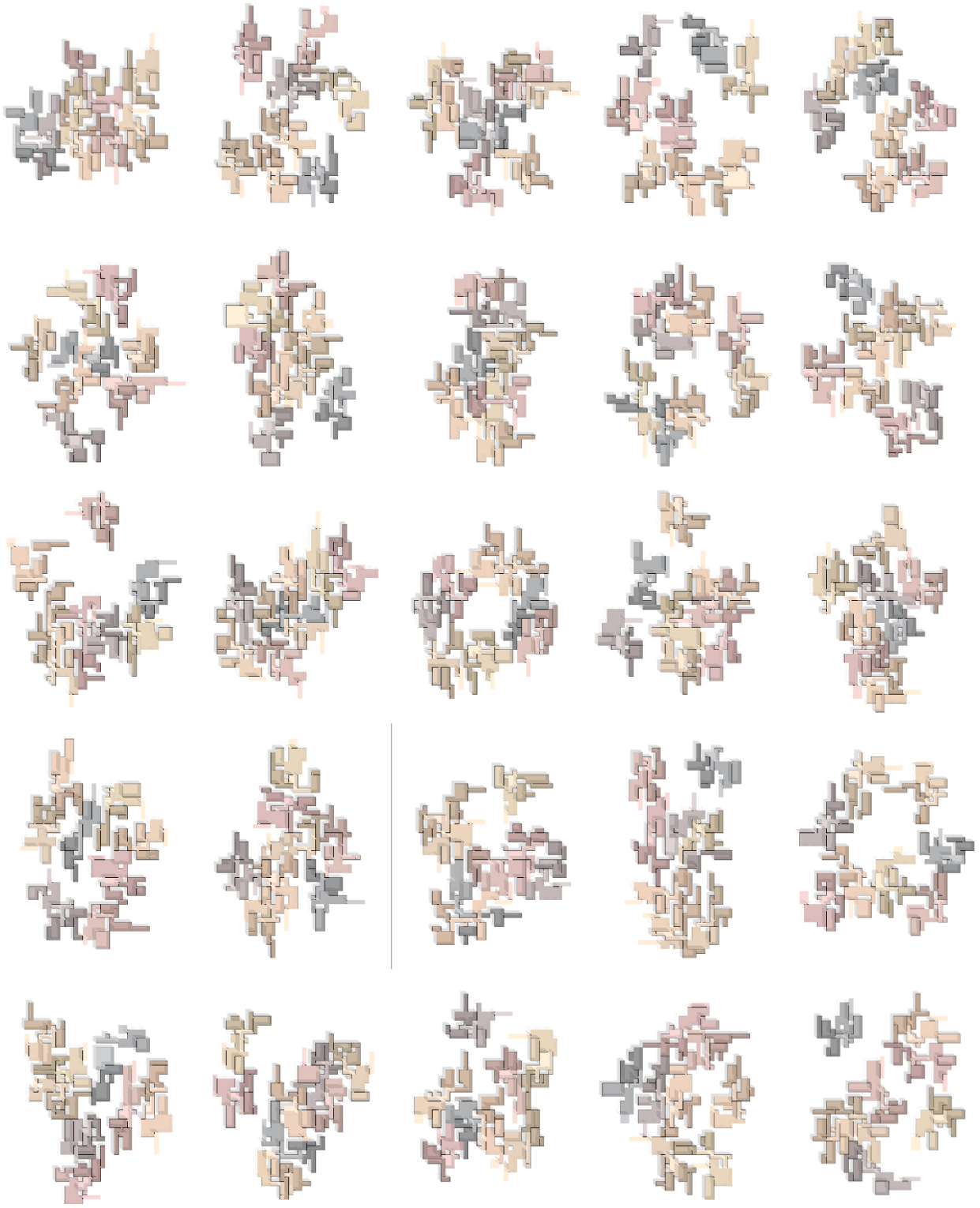


Figure 86 Last 25 configurations produced by P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50) over 2000 steps. The configurations highlighted were selected for a more detailed description. By Author.

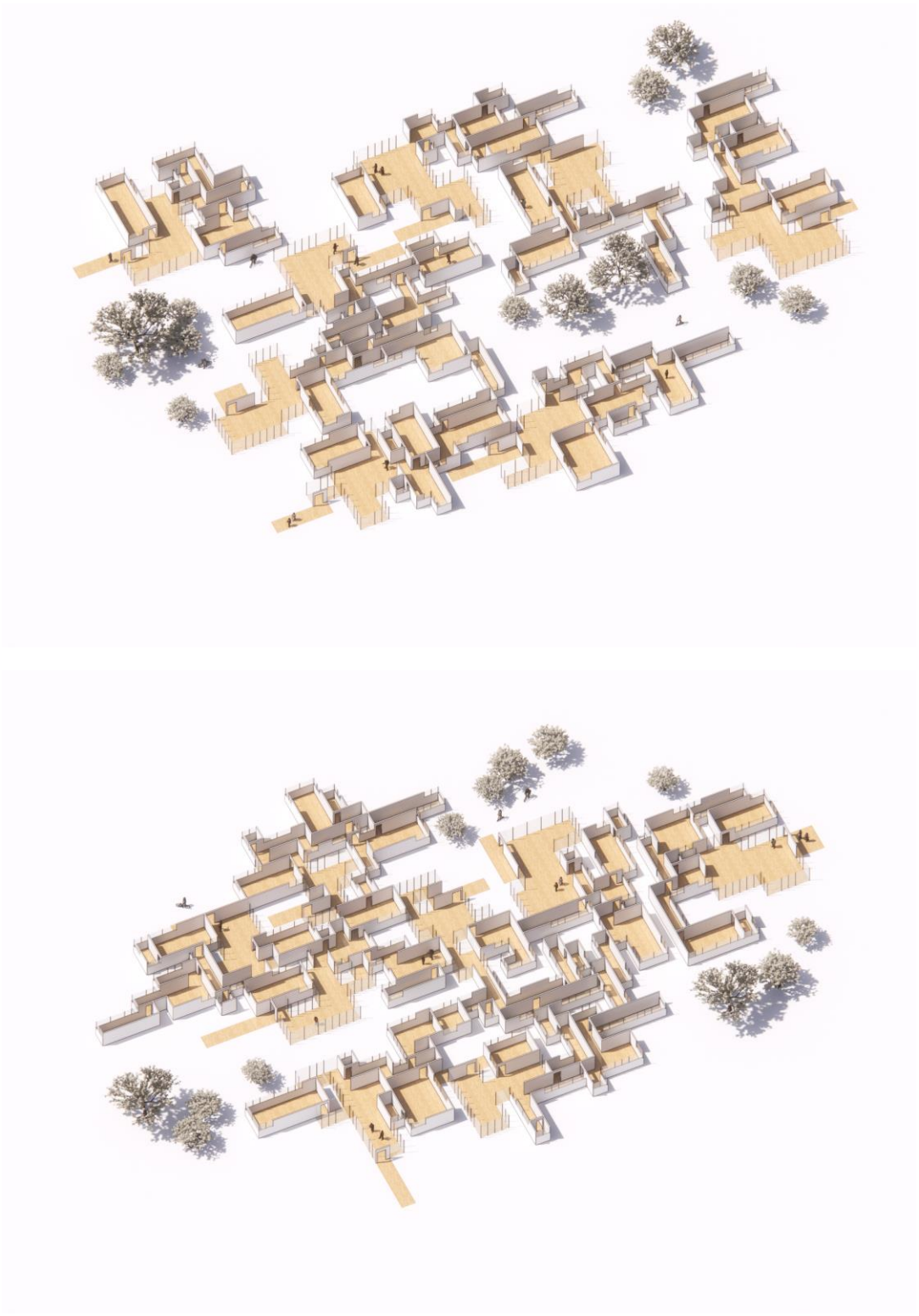


Figure 87 Selected configuration produced by P3 with policy Adjacency, Area, Not Folding (5), and Daylit (50) over 2000 steps. By Author.

3.4.5. Evaluation

Overall, the policies with the spring layout initialization achieved a great performance in case 3. The agents quickly converge to a good configuration (see Figure 88) and they spend most of the simulation doing small adjustments in the layout. In some cases, the agents were not able to solve high-level conflicts in terms of adjacencies that would require the collaboration of multiple agents, so there are a few cases where the building layouts ended up interwoven.

Overall, the typological results are consistent with the previous evaluations of the policy.

- P3 Adjacency, Area, and Daylit (75) increases the external extension of the external walls by creating irregular polyominoes with multiple concavities, which results in a configuration of corridors, small patios, and niches (see Figure 68).
- P3 Adjacency, Area, and Not Folding (maximum of 3 folds), generates rows of single cells restricted by the extension of 5 cells, which eliminates folds in the shape and reduces the distance between agents (see Figure 64). This policy results in an agglomeration of linear rooms. In the case of the grouped agents without internal walls, the union of the corridors result in other shapes such as rectangular, L-shaped, and S-shaped rooms. With 96 agents, large multiple quasi-rectangular patios are created between the building units, which contribute to good daylight. However, as the layout is based on an agglomeration of linear shapes, there are many cases where an internal room is mostly or completely surrounded by internal walls, which drastically reduce the daylight access.
- P3 Adjacency, Area, and Not Folding (5 folds) resulted in arrangements of rectangular and L-shaped rooms in a free form and centrifugal layout. Due to the limited space, some. However, with the large number of agents and limited space, buildings are connected, creating a series of irregular courtyards of different sizes.
- P3 Adjacency, Area, Not Folding (5), and Daylit (50%) results in room and complex configurations like the ones from the previous policy. However, each agent is attempting to expose more cells to the outdoor space, which result in small corridors and patios between buildings. Despite the use of the overhangs, the small patios successfully contributed to the better distribution of light.

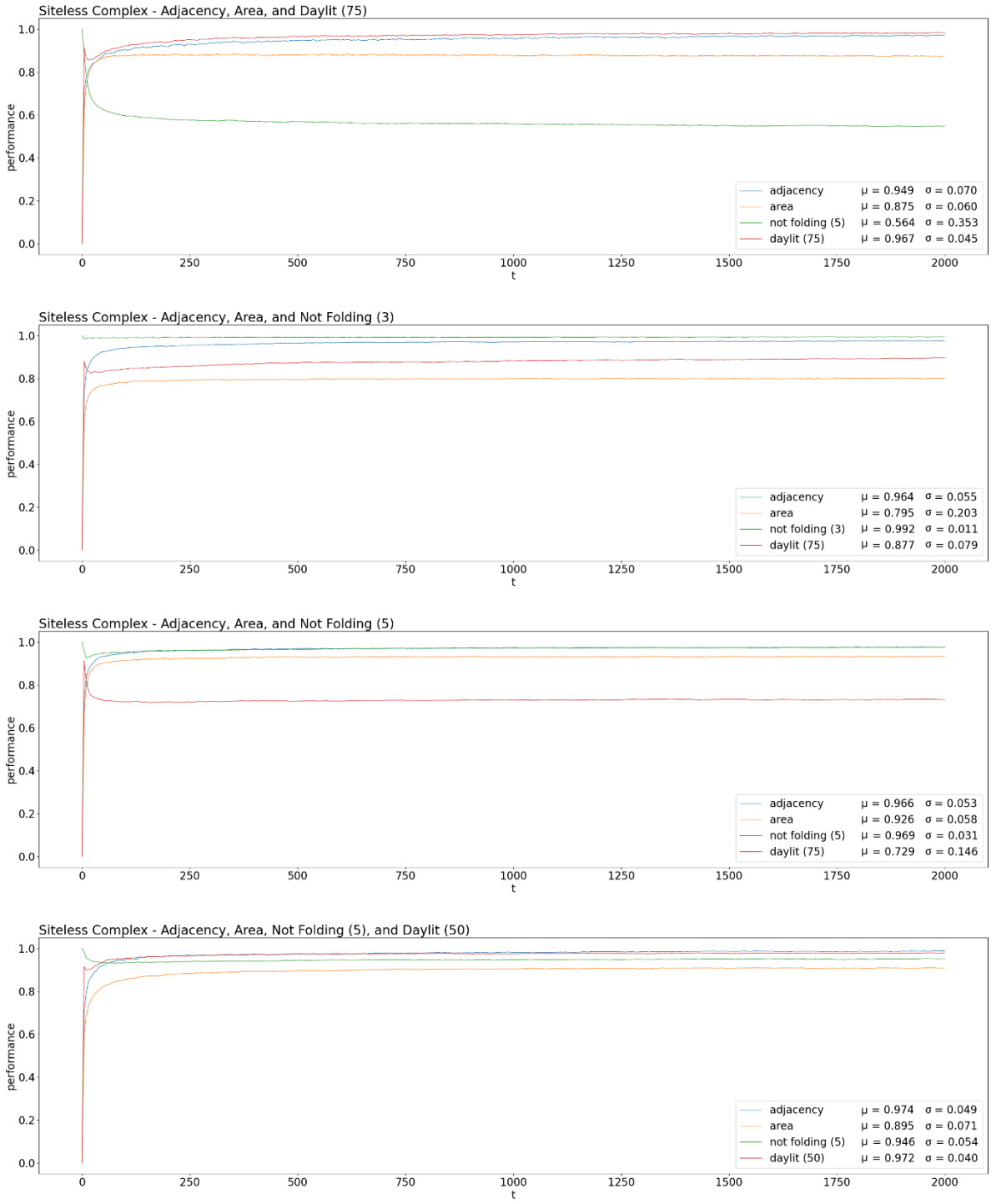


Figure 88 Average performance of different policies over 50 episodes and with spring layout initialization Row 1: P3 Adjacency, Area, and Daylit (75). Row 2: P3 Adjacency, Area, and Not Folding (3). Row 4: P3 Adjacency, Area, and Not Folding (5). Row 4: P3 Adjacency, Area, Not Folding (5), and Daylit (50). By Author.

3.5. Discussion on the design artifact

Overall, the policies tested in the last chapter were able to address the three design cases, were efficient with the respect to the different goals and were robust to the random perturbations in the simulations. Some of them were efficient with the respect to all the goals while others opted for emphasizing some of the sub-goals at the expense of another. In smaller versions of the layout problem, randomly initialized agents were able to solve topological properties of layouts based on the jumping strategy. However, as the grid and the number of agents increased, the performance of the agents largely benefited from the spring layout initialization.

The takeaway is that the research artifact is a proof by construction of the proposed framework. The combination of agent-based modeling and multi-agent deep reinforcement learning in the artifact solves many of the technical challenges discussed in the previous chapters and substantiates the research framework. The agent-based formulation of the artifact enables a distributed and granular construction of design alternatives with reduced assumptions about setup and order of the solution procedure. The multi-agent reinforcement learning method solves the problem of designing ecologies of spatial agents that can tackle multiple constraints and objectives while preserving the qualities of the agent-based formulation. After training, the agents are reflex agents (Table 13) because the execution only relies on the forward pass of small policy networks, which can be executed in real-time even for large number of agents.

4. An interactive simulation workflow for situational design exploration

The goal of this chapter is to demonstrate that the design artifact introduced in this dissertation enables designers to engage in an interactive simulation that support unstructured forms of design exploration and use *ad hoc* responses to the design circumstances. These are a some of the properties of the design artifact that supports designers inside the generative loop with fine-grained interaction:

- The computation is executed by distributed reactive agents, based on the forward pass of a small policy network, so it can happen in real-time.
- The granular formation of the representation is embedded in the formulation of the spatial agents as polyominoes and its respective action space, so it is also accessible by external control.
- The agents are trained with random environments to learn a policy with reduced the dependence on initialization and order of operations, which provides the freedom to designers to navigate in the design space with different strategies.

To explore how these and other properties of the design artifact can support novel forms of human-computer interaction and situated design exploration, the artifact was integrated to an interactive simulation workflow. This chapter describes this workflow with its communication mechanisms, interface, and commands. It demonstrates how designers' actions can be translated to events and integrated to an interactive simulation based on the design artifact.

4.1. General Workflow

We integrated the design artifact (1) presented in the previous chapters to a parametric model (2) and a game engine (3) to show its potential as an interactive simulation (Figure 89). For that integration it was necessary to create mechanisms of management and communication of data. In the next sections, we will describe the main components of the workflow with their respective mechanisms of communication, and then we emphasize the types of interaction that are supported by the game engine.

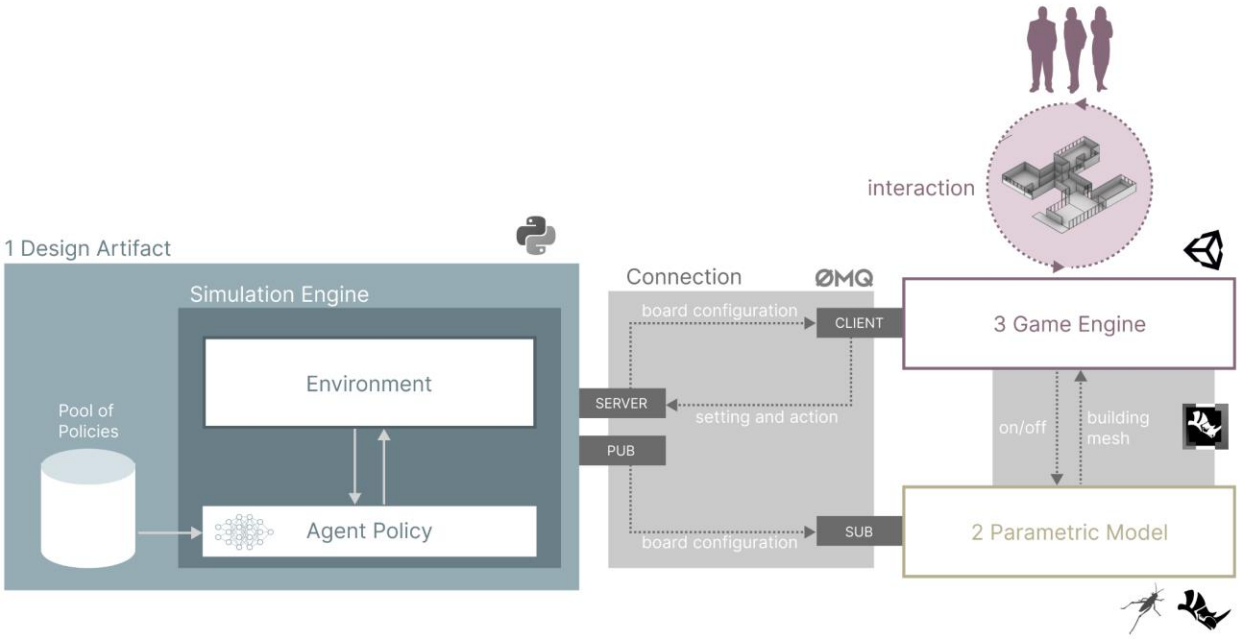


Figure 89 Overview of interactive simulation workflow. By Author.

4.2. Design artifact

The design artifact contains a simulation engine, which manages information about environment and agents, and the many functions used in their interaction, such as a transition function and a reward function. Besides, after training the agents in randomized environments, the artifact also contains a pool of agent policies, which can be deployed to generate different types of spatial arrangements.

For the interactive simulation workflow, the artifact was extended with functions to exchange data with the parametric model and the game engine, which rely on ZeroMQ, a messaging library that provides different messaging patterns. The connection between the construct and the game engine is defined by a REP socket, which allows the artifact and the game engine to work as server and client that exchange messages synchronously. The connection between the artifact and the parametric model is based on the publish-subscribe pattern. The artifact is the publisher, which sends the board updates, and the parametric model is the subscriber, which asynchronously keeps track of the messages.

The workflow is initialized with a pre-defined environment and agent configuration. The updates are defined by the interaction between designers, agents, and environment in the game engine. Therefore, besides the regular agent-environment update based on a policy, designers can also directly change the configuration of the board or even redefine the configuration of the spatial synthesis problem, providing guidance to the design exploration with event-driven computing.

The different forms of human-computer interaction will be described in detail in the section on the game engine. Each interaction results in the game engine sending a distinct message to the design artifact, with the specific request on how to update the board. For example, the game

engine can send a message telling the environment to take one step in the simulation with a certain policy or describing how designers changed the board. Each of these messages will trigger a function to implement the respective changes in the simulation loop in the artifact.

Some of these actions defined by the designer, such as adding or deleting a single agent cell, are compatible with the action space of the agents, so they can be solved with a single pass of the transition function in the simulation engine. Other actions, such as adding or deleting an agent, require further updates in the agent representation and in the problem specification. In the extreme, certain actions taken by the designer break the logic of the action space and require additional procedures to be taken. For example, when designers grab and move an agent in the game engine, there is an algorithm (Algorithm 5) that reinforces the board constraints and uses BFS to ensure that the new agent configuration is legal.

Algorithm 5 Move agent

receive i as the index of the agent being moved
receive d as a vector representing the movement of the agent i
define $pos_i^{tmp} \leftarrow pos_i + d$
constraint pos_i^{tmp} and d to represent a valid agent position and transformation inside the grid
$per\_grid_i^{tmp} \leftarrow per\_grid_i + d$
$act\_grid_i^{tmp} \leftarrow act\_grid_i + d$
if there is any empty cell in $act\_grid_i^{tmp}$:
    if $area(PnH_i) > 0$:
        $PnH_i^{temp} \leftarrow PnH_i + d$
        eliminate cells of PnH_i^{tmp} overlapping with other agents and with obstacles
        if $count(PnH_i^{temp}) > 0$:
            initialize PnH_i^{BFS} as a random cell sampled from PnH_i^{temp}
            expand PnH_i^{BFS} using BFS to flood fill the cells of PnH_i^{temp}
            $PnH_i \leftarrow PnH_i^{BFS}$
        else:
            $PnH_i \leftarrow \emptyset$
    $pos_i \leftarrow pos_i^{tmp}$
    $per\_grid_i \leftarrow per\_grid_i^{tmp}$
    $act\_grid_i \leftarrow act\_grid_i^{tmp}$

After the internal updates in the artifact, it sends a message with the resulting configuration back to the game engine. This message is also sent to the parametric model, which is responsible for the procedural generation of the architectural configuration.

4.2.1. Parametric model

The parametric model is responsible for generating the geometry for visualization, with floors, walls, windows, and openings for doors in Grasshopper, a parametric editor inside the NURBS-modeler software Rhinoceros 3D. This model was used for the visualization of solutions in the

previous chapters. The input information for the model is derived from two sources

- A custom asynchronous subscriber receives the configuration of the agents and obstacles from the design artifact
- Designers control parameters to customize the parametric model during the simulation

Based on these sources, the model generates the 3D model of the building based on the following steps (Figure 90):

- A) Extracting polyominoes and other information from the environment
- B) Converting the polyomino of each agent into a single mesh
- C) Combining agents that represent a single space and do not share internal partitions
- D) Extracting the boundary lines of the meshes, and differentiating internal and external walls
- E) Defining the rooms that do not have external walls, such as porches, based on user input
- F) Defining the glass walls for the external walls of the rooms selected based on user input
- For each room not specified by the user for special envelopes,
    - G) Defining the windows positions based on a percentage of glazing and on a heuristic to select external line – e.g., random selection or wall furthest away from the centroid of the house boundary
    - H) Defining the door positions between rooms that share a relationship of adjacency by randomly selecting one of the cell edges in the shared walls.

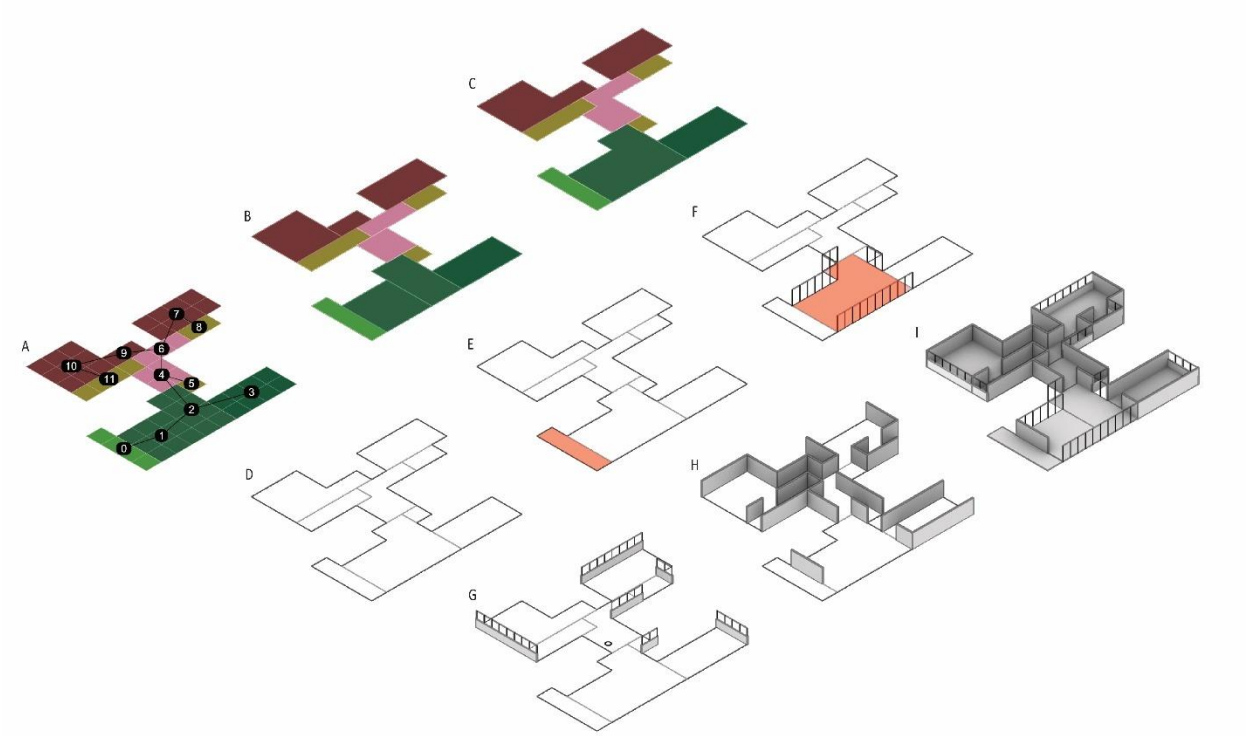


Figure 90 Basic steps for the parametric model derived from the environment information using design case 1 as an example. By Author.

After generating the building geometry, the model converts it into meshes, which are sent to and rebuilt by the host application (the game engine) using Rhino.Inside (Fragada, Baer, and Cook 2022).

4.3. Game Engine

The game engine is the design interface of the workflow. It contains geometric data about the site context, which is combined with the agent and environment data received from the design artifact and the parametric model to produce a visual representation of the design board on its canvas. Its main goal is to allow designers to guide the design generation by playing with the settings and configurations during the simulation. This interactive simulation relies on capturing and processing user action events, then sending them as messages to the design artifact. The game engine provides a menu with tools that support different ways in which the users can interact with the board and its components so that both the simulation and user events determine the flow of the program.

4.3.1. Interface

The visual interface is defined by four basic elements: (A) the name of the project, (B) the menu with the tools available for the designers, (C) the buttons to activate the connection with the parametric model, and (D) a canvas where the design geometry is displayed to the user and available for editing.

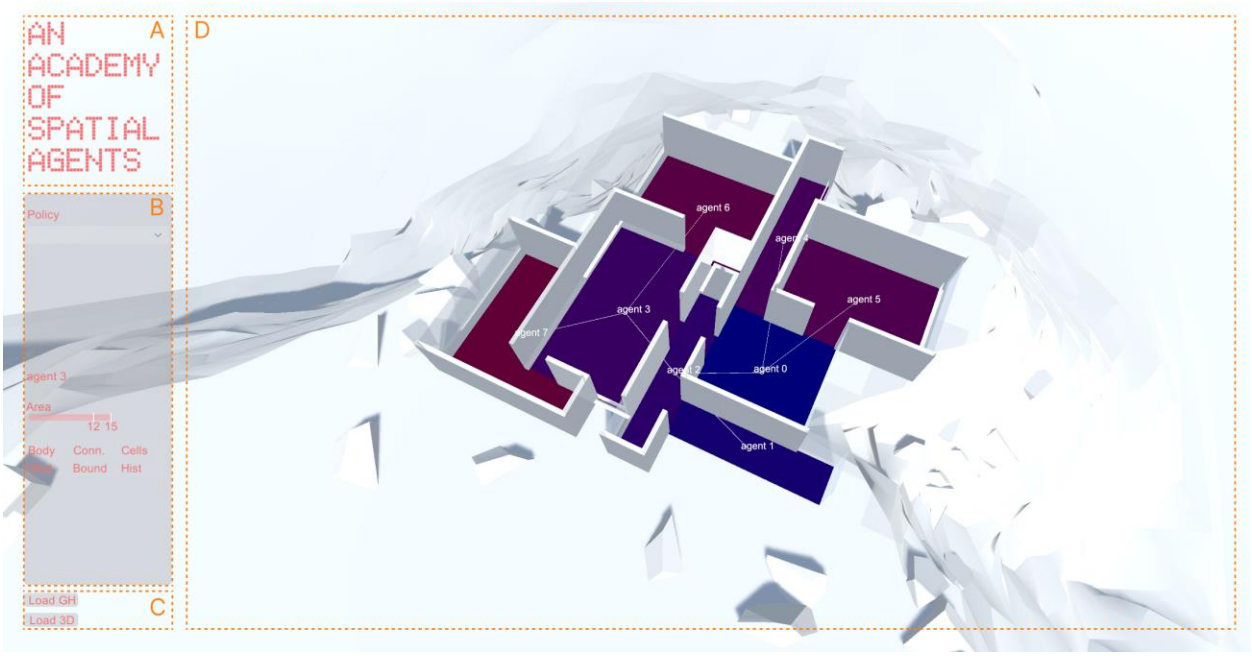


Figure 91Interface of the game engine with (A) title, (B) menu, (C) connection with parametric model, and (D) canvas. By Author.

4.3.2. Policy

Policy is a value list containing options for the function used to control the agents during the simulation. The options are

- None: no policy is used
- Random: a random policy is selected (see Figure 92)
- RL policies from the policy pool (see Figure 93), such as
    - RL – Adj Ar NF3: adjacency, area, and not-folding (3)
    - RL – Adj Ar NF5: adjacency, area, and not-folding (5)
    - RL – Adj Ar DL75: adjacency, area, and daylight access (75%)
    - RL – Adj Ar NF DL75: adjacency, area, not-folding (5), and daylight access (75%)

If no policy is selected, the simulation will pause. Otherwise, the simulation will run with the selected policy. Additionally, the user can pause the simulation at any point by holding the space bar.

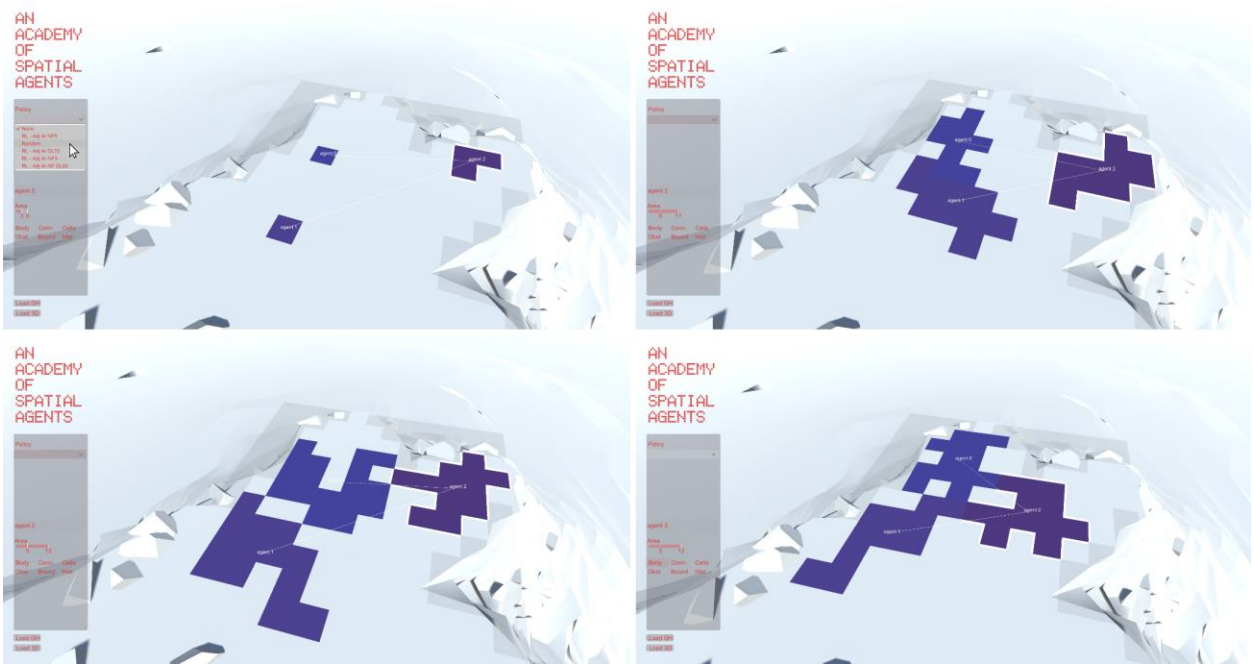


Figure 92 Example of random policy. Notice that there are multiple steps between snapshots. By Author.

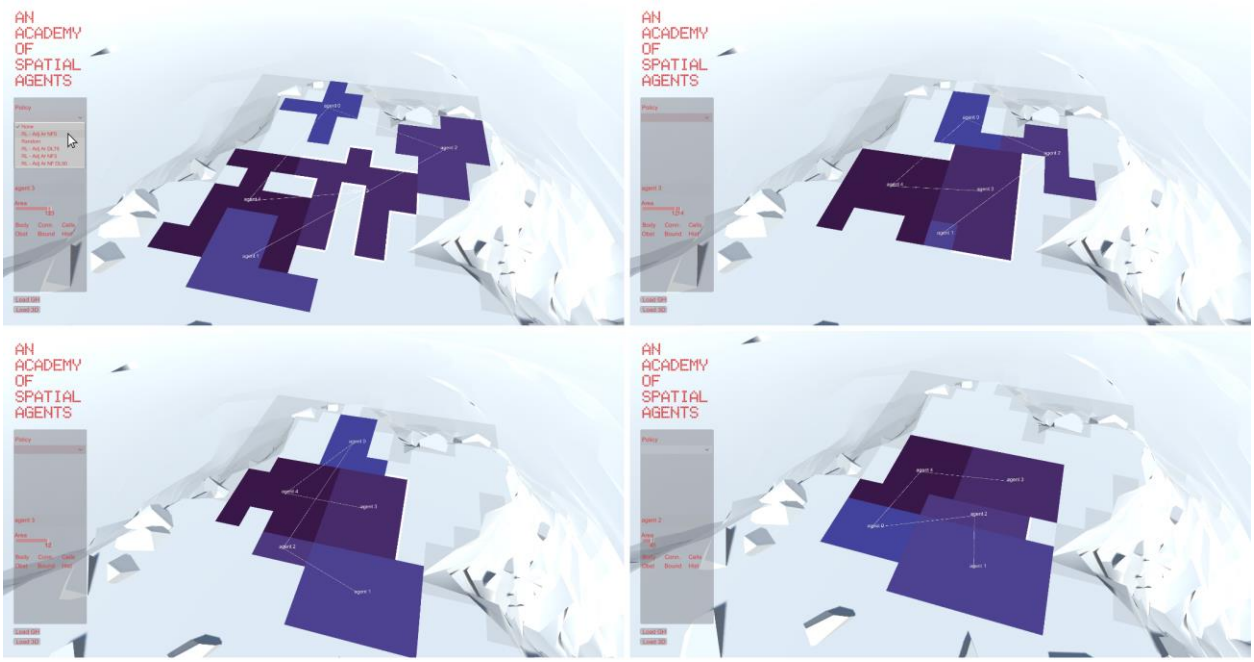


Figure 93 Example of simulation starting from a random arrangement and switching to a RL policy. Notice that there are multiple steps between snapshots. By Author.

4.3.3. Basic commands

Basic user interactions rely on selecting options in the menu and using the mouse combined with shortcut keys to do different actions. That combination will define what events will be triggered and processed by the game engine. Selection is the only command that is independent of the options of the menu. When designers right-click on the agent, it is highlighted, and two properties

of the agents appear on the menu:

- Name: initially defined as agent i but can be updated by the user.
- Area: a slider indicating the current area of the agent with a marker indicating the target area. The slider can be updated to indicate a change in the problem specification (see Figure 94).

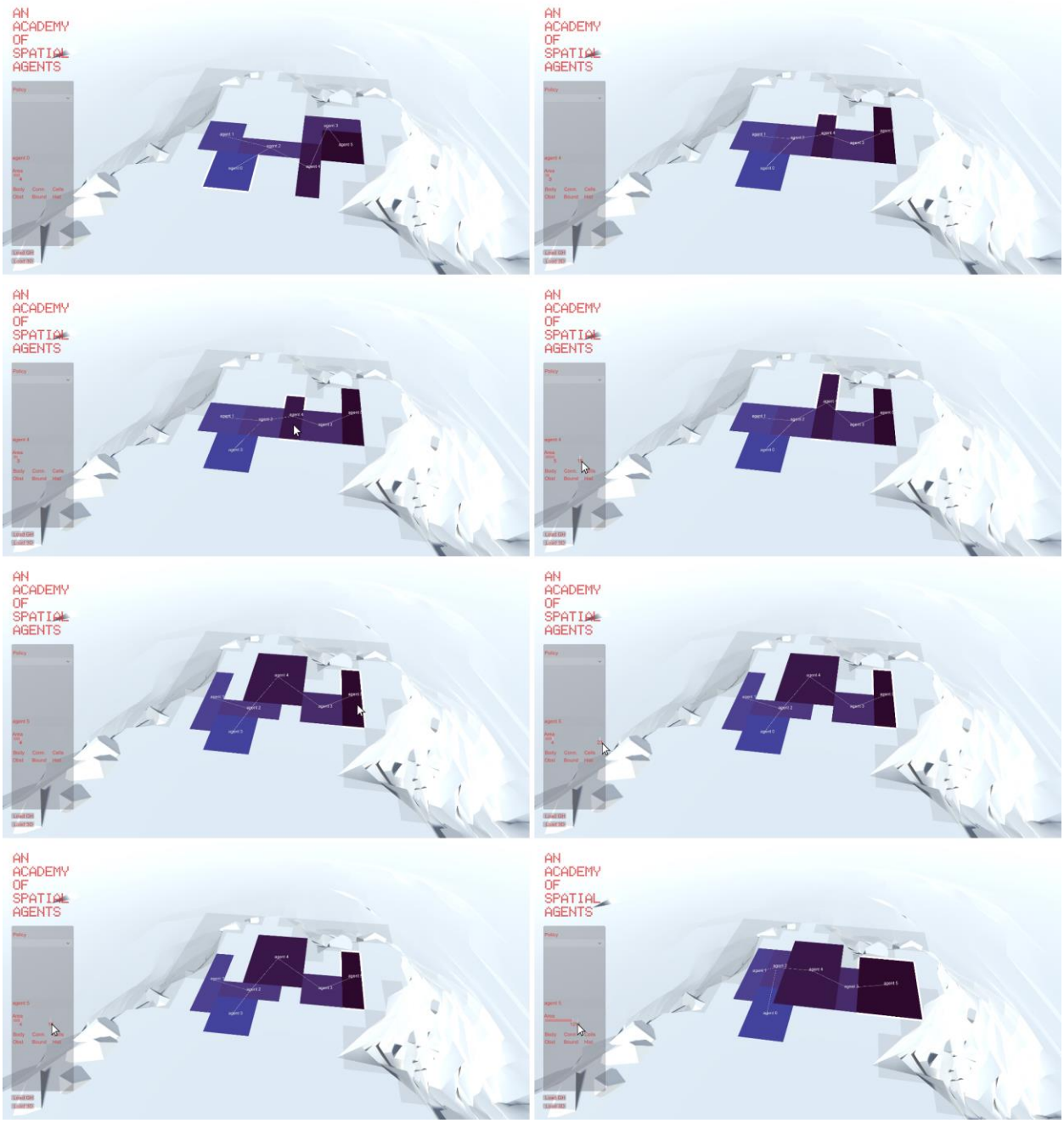


Figure 94 Example of changing the area of two agents as the simulation is running with a RL policy. By Author.

If there are no tabs selected, these are the available actions with the mouse

- Left click [on agent i] – hold – move – release [on agent j]: creates relationship of adjacency between i and j
- Right click [on agent i] – hold – move – release: moves agent i to another position (see Figure 95 and Figure 96)
- Ctrl + Right click [on agent i]: freezes or unfreezes agent, which defines if the agent will move or not during a simulation

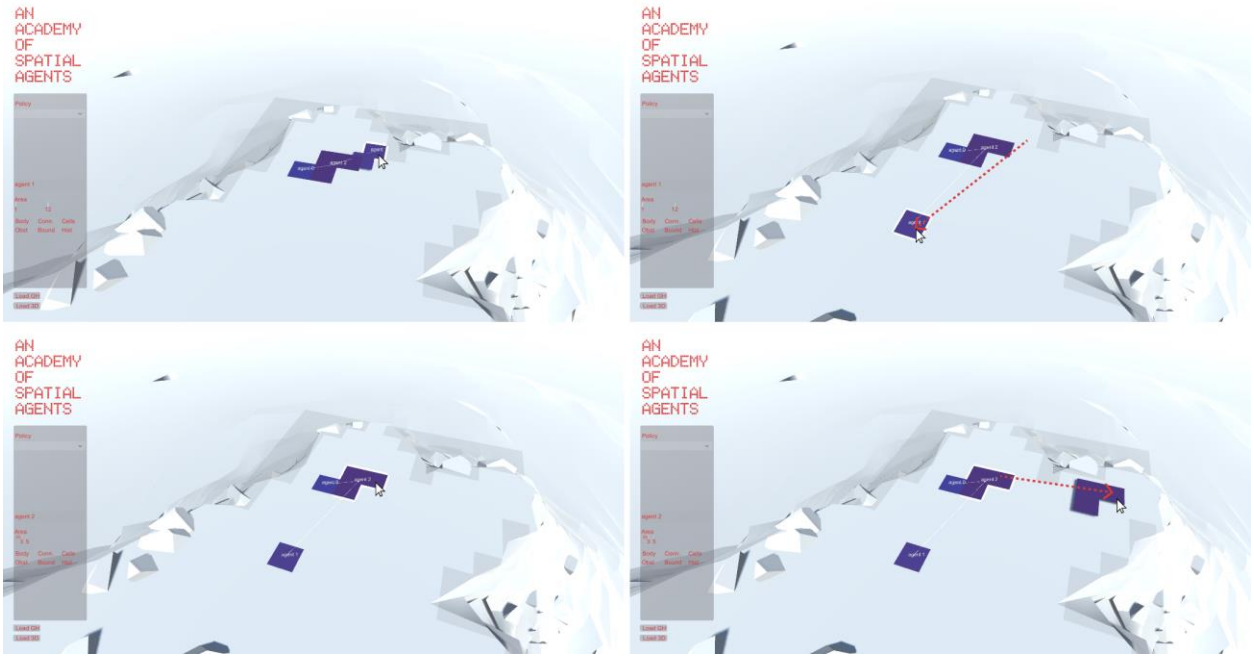


Figure 95 Example of moving an agent. By Author.

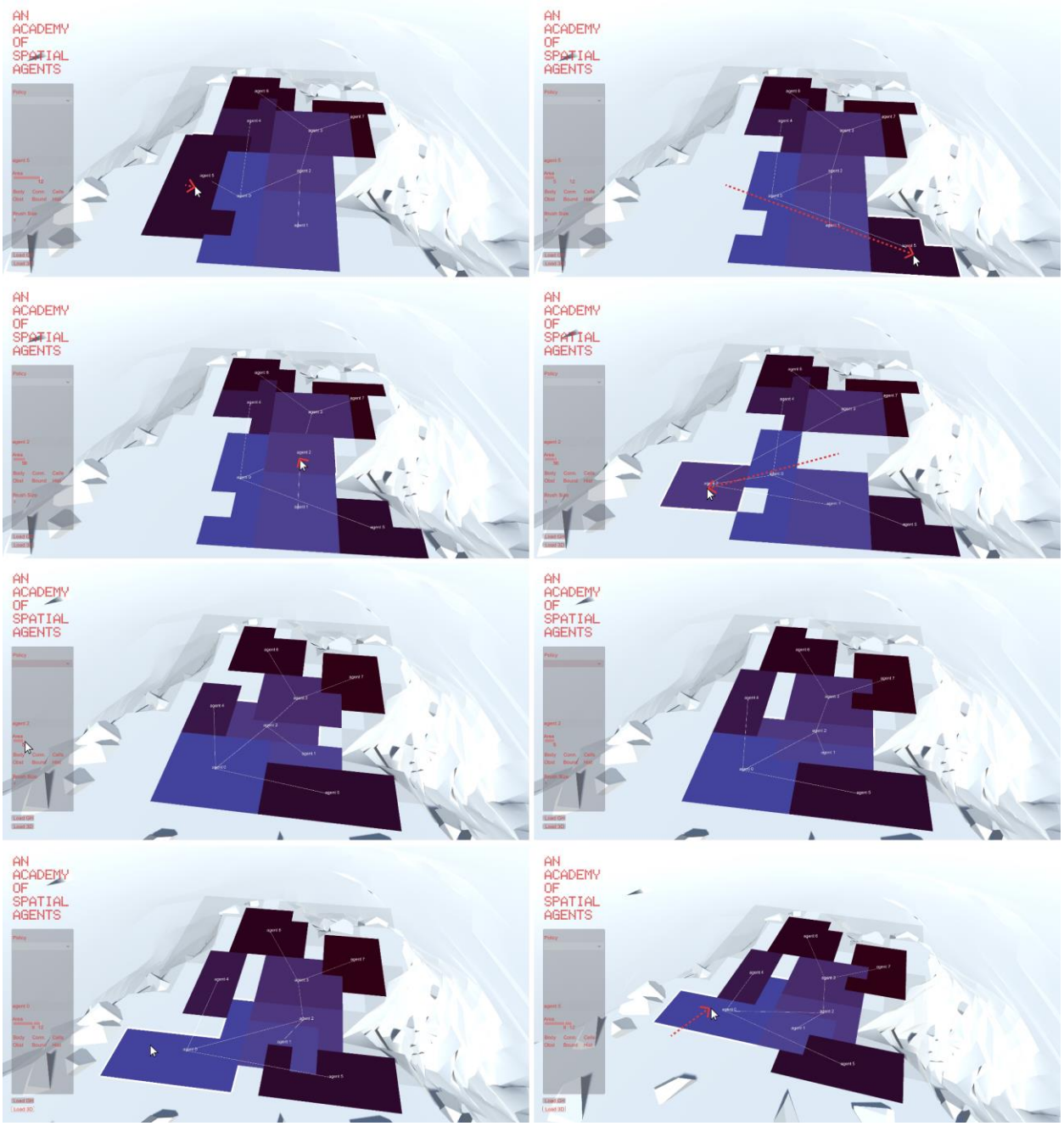


Figure 96 Example of moving agents during a simulation with a RL policy. By Author.

4.3.4. Agent body commands

If the Body tab is selected the user can control the number of agents in the simulation. These are the available commands:

- Left click [on board]: creates new agent on that position
- CTRL + Left click [on agent i]: deletes agent i

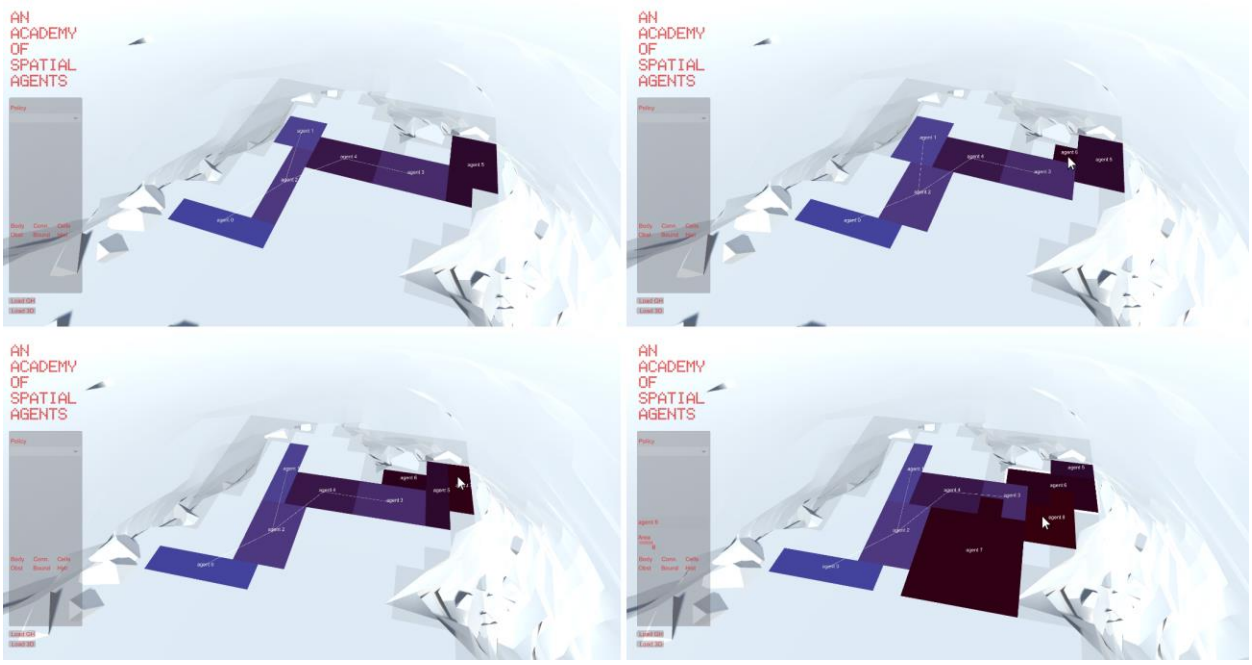


Figure 97 Example of adding agents during simulation with RL policy. By Author.

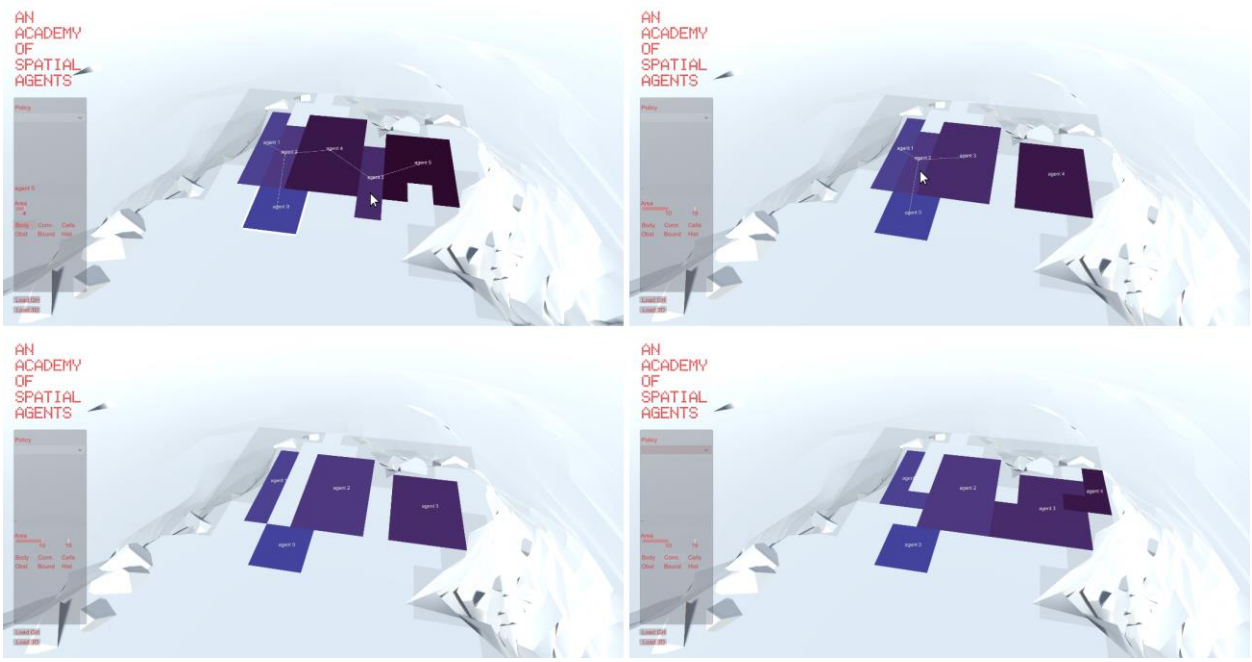


Figure 98 Deleting agents during a simulation with a RL policy. By Author.

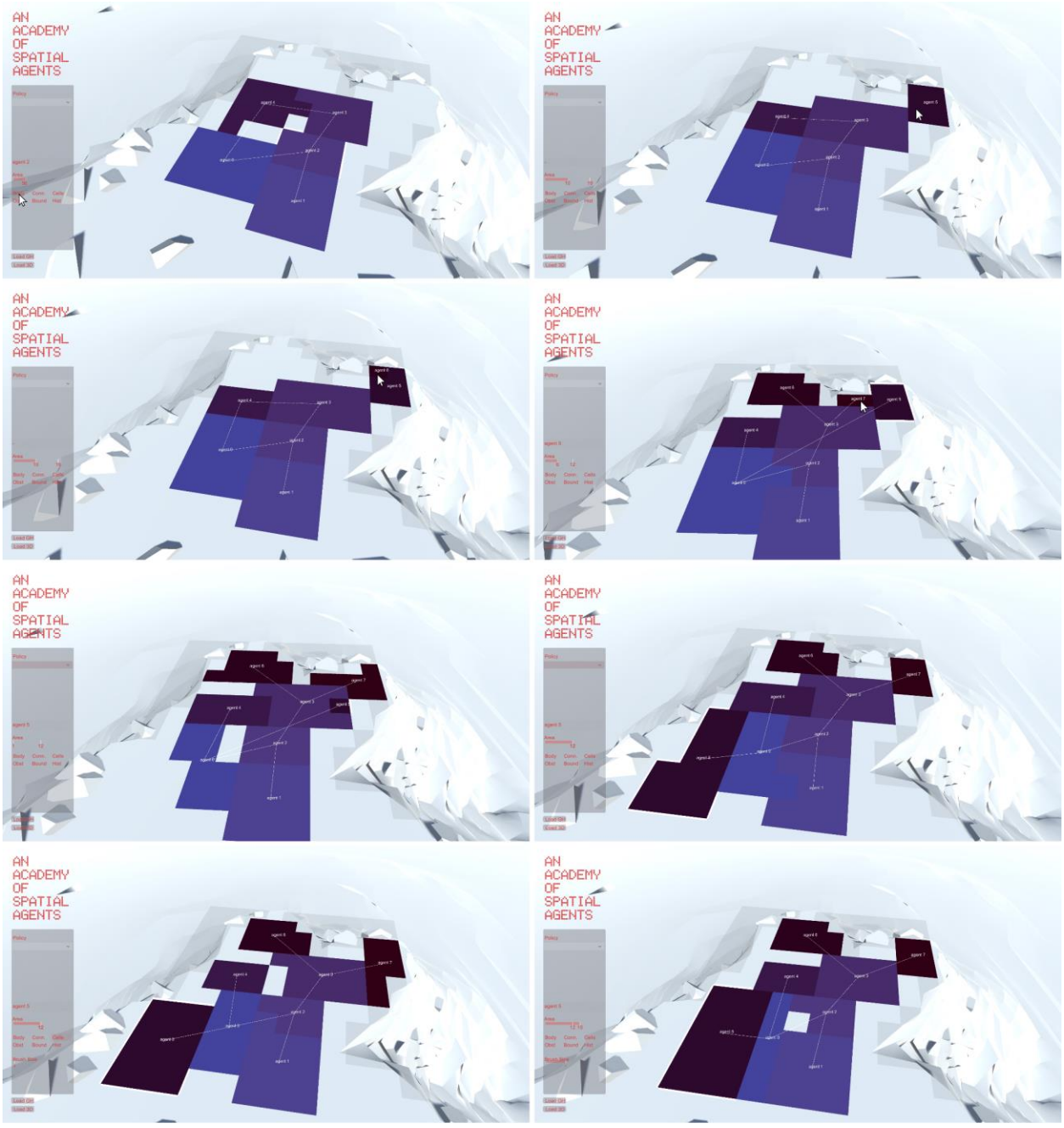


Figure 99 Adding agents and connecting them during a simulation with a RL policy. By Author.

4.3.5. Agent connection commands

If the Conn tab is selected, the user can control the connections between agents, which represent relationships of adjacency. These are the available commands:

- Left click [on agent i] – hold – move – release [on agent j]: creates relationship of adjacency between i and j
- Ctrl + Left click [on agent i] – hold – move – release [on agent j]: erases the relationship of

adjacency between i and j (if any)

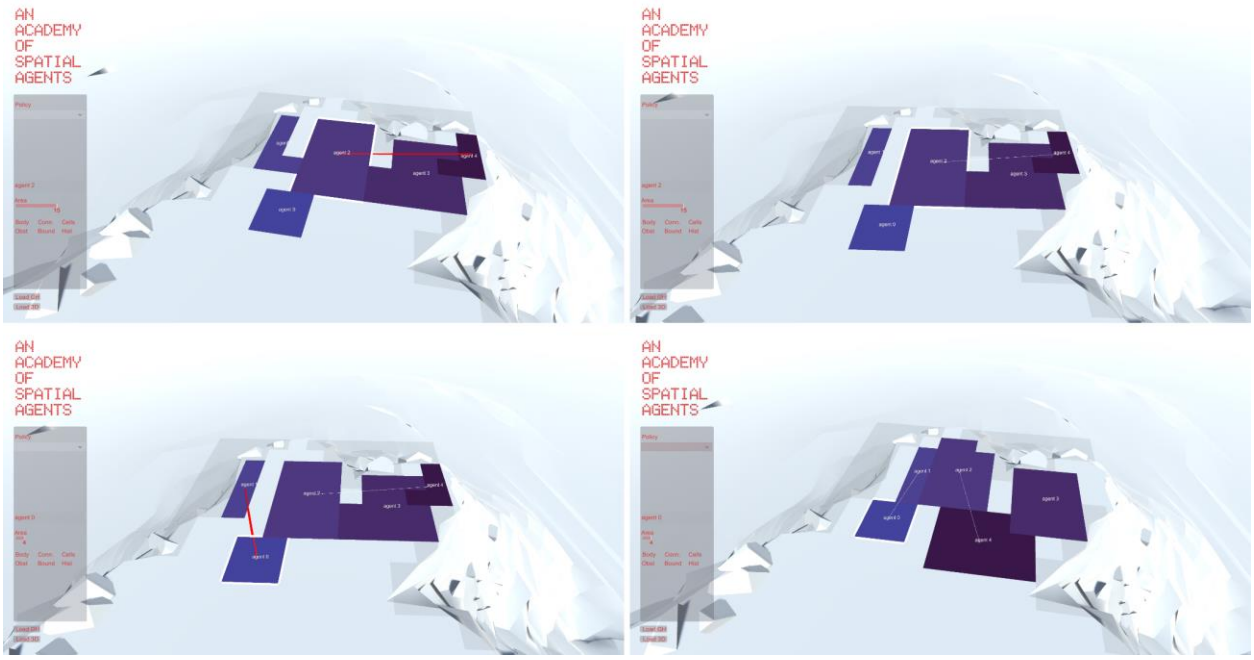


Figure 100 Example of creation of connections between agents as the simulation runs with a RL policy. By Author.

4.3.6. Agent cells edition commands

When the Cells tab is selected, it is possible to edit the cells of the agents. First, a slide indicates the number of cells in the Moore neighborhood of the position of the cursor that will be used as the brush to create or delete cells on the board. With that slider defined, these are the available commands

- Left click [on cells]: expands the cells of the selected agent under the area of the brush (if possible)
- CTRL + Left click [on cells]: erases the cells of the selected agent under the area of the brush (if possible)

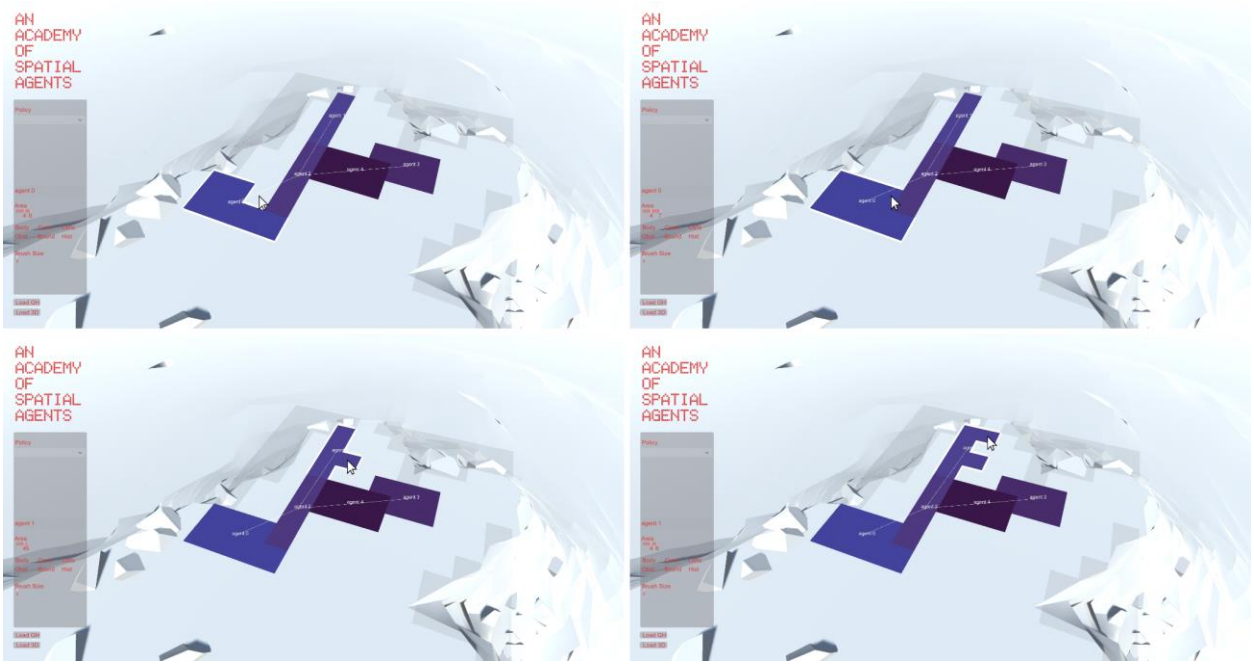


Figure 101 Example of adding cells to agents. By Author

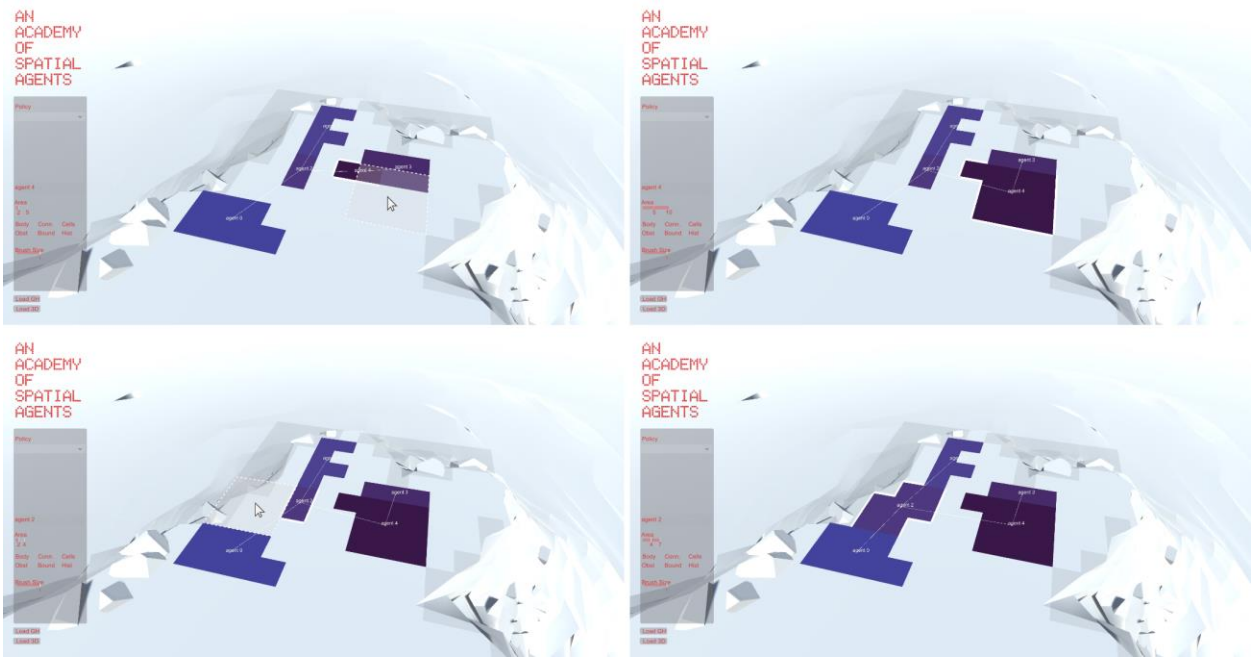


Figure 102 Example of adding cells to agents with larger brush. By Author

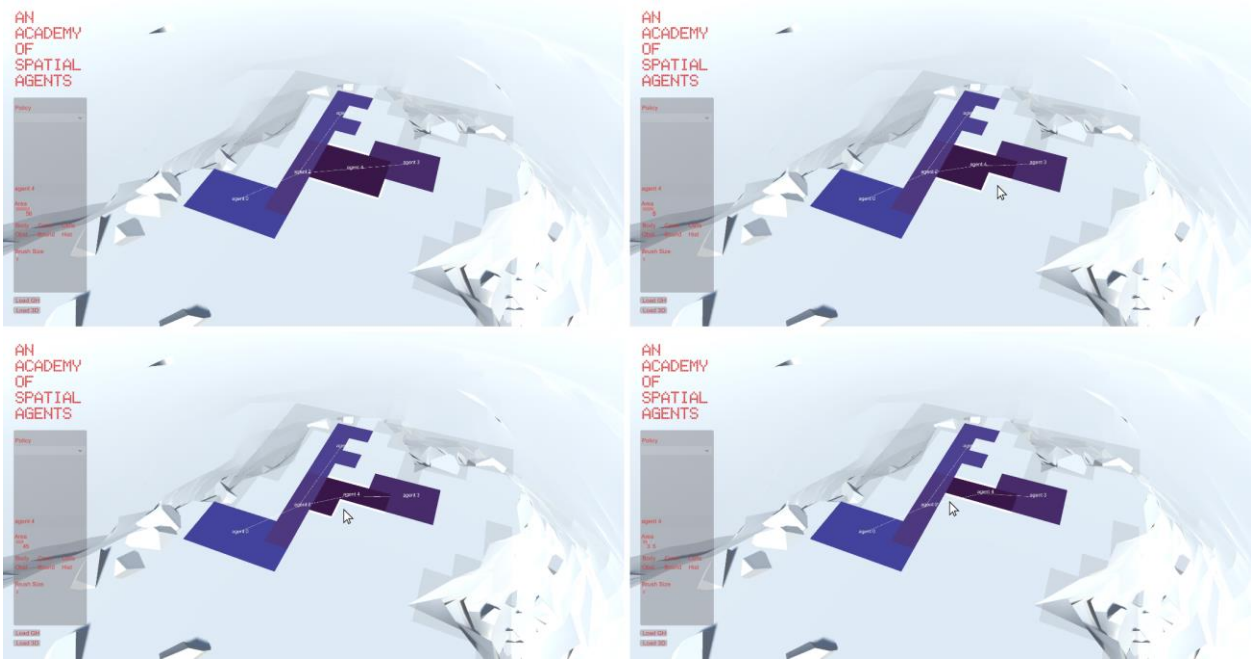


Figure 103 Example of deleting cells of the agents. By Author.

4.3.7. Environment obstacle commands

If the Obs tab is selected, designers can edit the obstacles on the board with these commands:

- Left click [on board]: creates new obstacle on cell
- CTRL + Left click [on agent]: deletes obstacle on cell (if any)

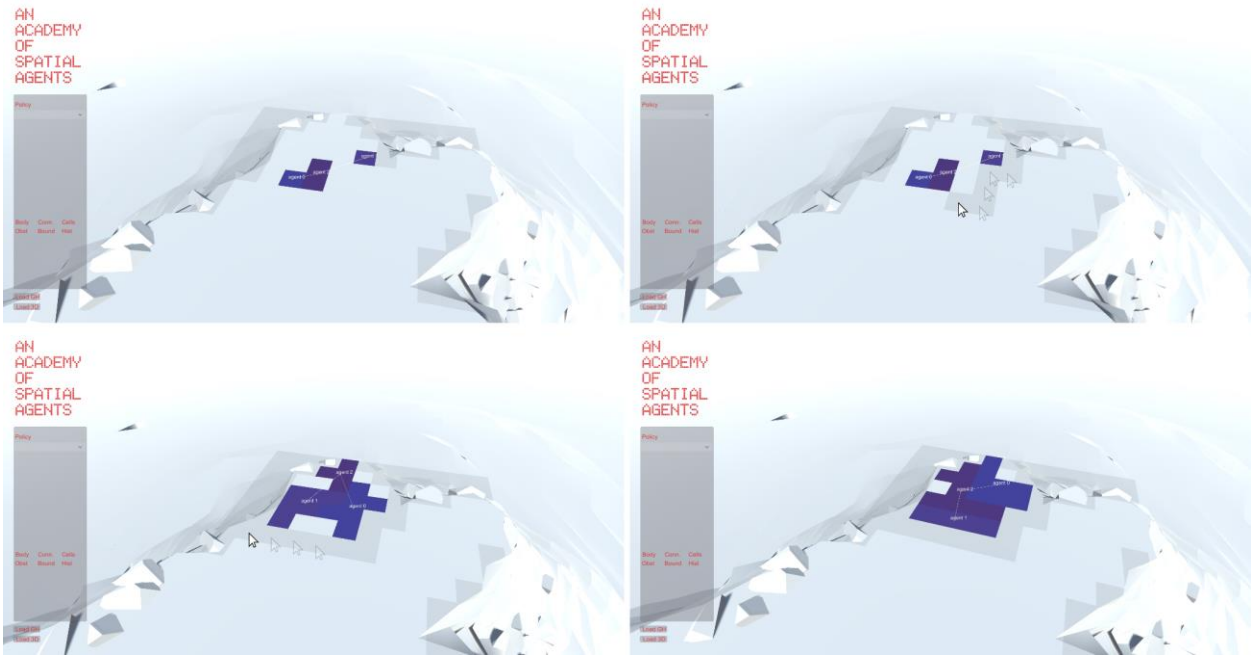


Figure 104 Example of adding obstacles to the environment during the simulation with RL policy. Transparent mouse cursors were added to show multiple steps in a single snapshot. By Author.

4.3.8. Environment boundary commands

When the Bound tab is selected, designers can edit the boundary of the board that is available to the agents. For that, these are the available commands:

- Left click on an edge of the boundary of the board – hold – move – release: expands or retracts the rectangular board by moving its edge
- Left click on a corner of the boundary of the board – hold – move – release: expands or retracts the rectangular board by moving its corner

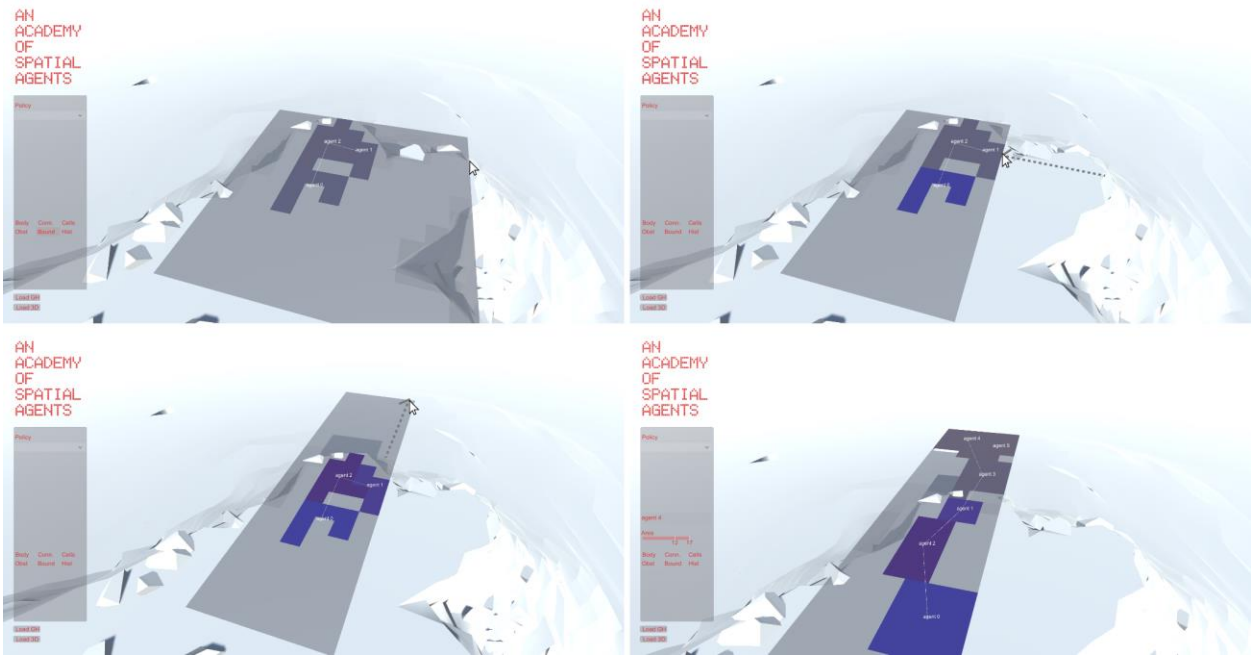


Figure 105 Example of changing boundary during simulation with RL policy. By Author.

4.4. Activating the parametric model

The button Load GH relies on Rhino.Inside to load the parametric editor Grasshopper using Unity as the host. With access to Grasshopper, it is possible to open the parametric model, which receives the cell configurations from the design artifact and builds a custom 3D model for the design. The button Load 3D activates the channel that transfers the 3D building meshes from to parametric model to the game engine. With the parametric model running simultaneously to the simulation, it becomes possible to interact with the agents and with the parameterization of the building as it is represented in 3D in the game engine. For example, in Figure 106 it is possible to see how the user can customize the 3D building as the simulation is running. Figure 107 and Figure 108 show examples of the interactive simulation running with user interaction and the parametric model activated.

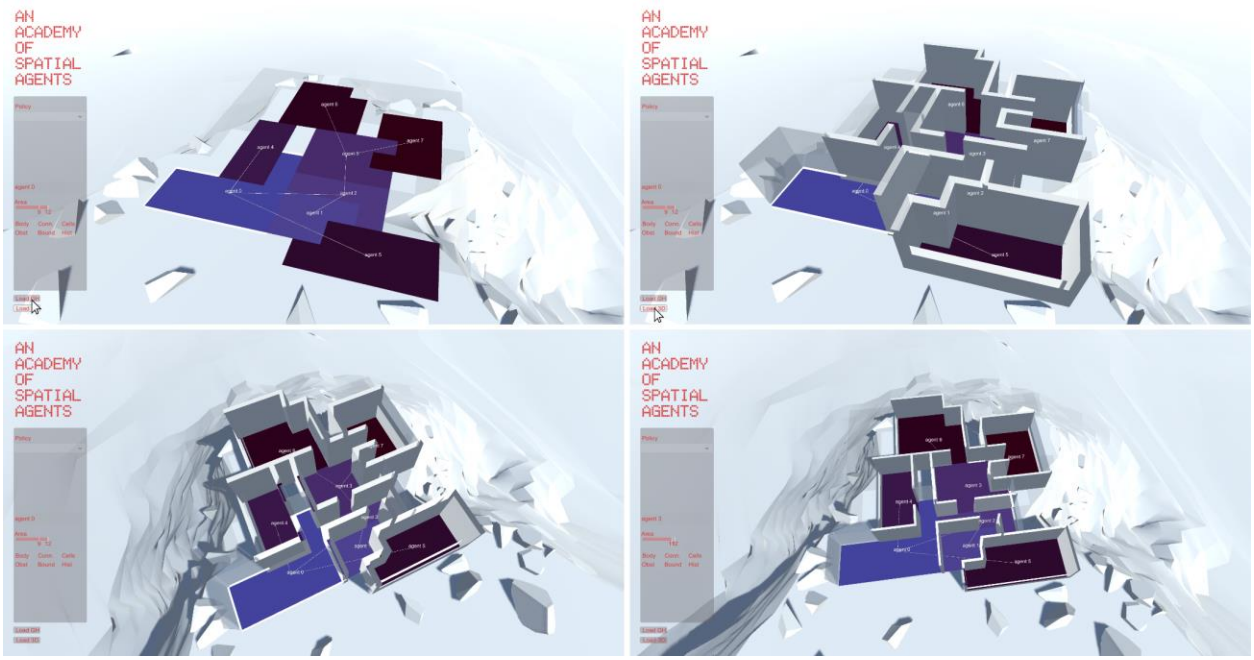


Figure 106 Example of activation of the parametric model and customization of the properties of the walls, such as thickness and height. By Author.

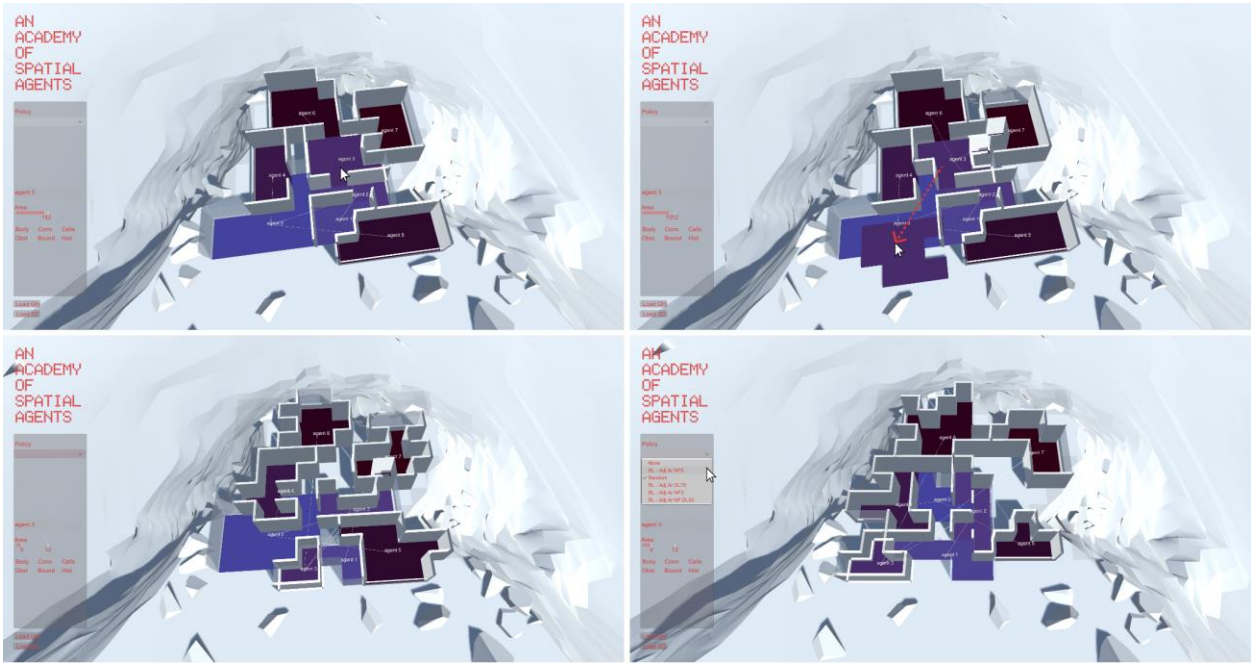


Figure 107 Moving an agent, then running he simulation with random policy with the parametric model on. By Author.

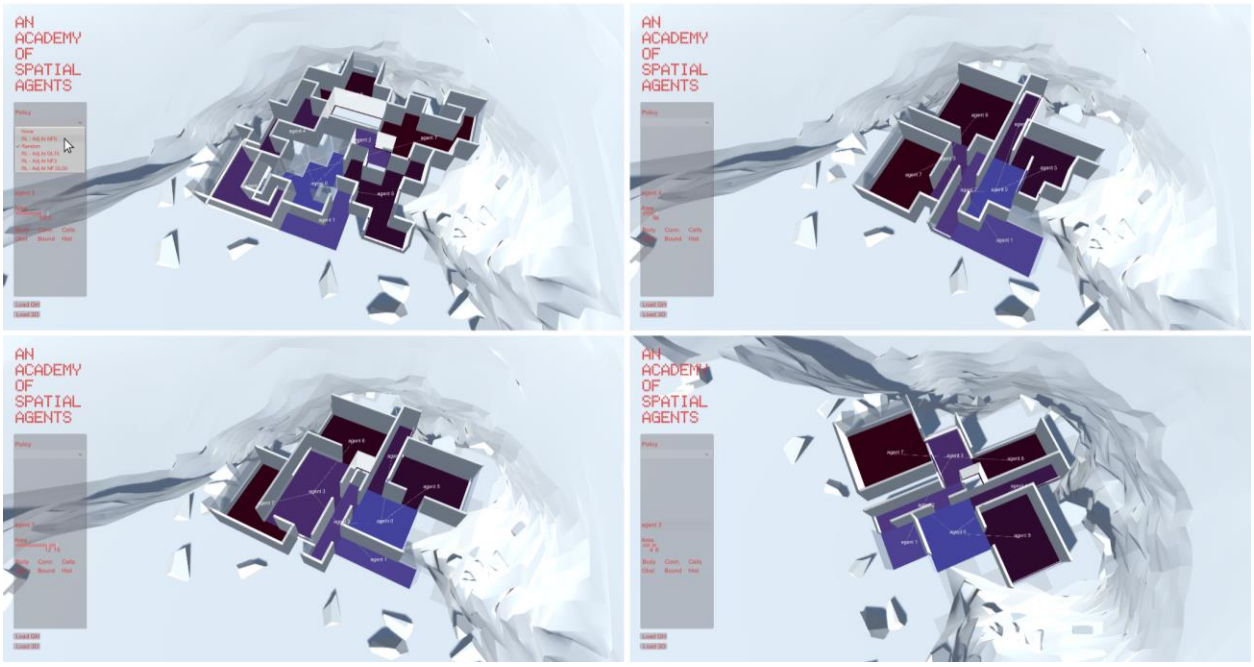


Figure 108 Starting from a random configuration and running a simulation with a RL policy and the parametric model activated. By Author.

4.5. Discussion

The interactive simulation workflow demonstrates the feasibility of integrating the agent-environment simulation of the design artifact to a generative design tool based on user events and control. The structure of the design artifact is designed for fine-granular interaction without front-loading a specific decision structure, so it is straightforward to support different forms of design interaction and *ad hoc* responses to the circumstances without changes in the structure of the simulation. Besides, the computation is distributed and reactive, so it becomes possible to enable designers to guide the formation of the design alternative in real-time. Overall, the workflow supports a form of design exploration based on designers and spatial agents responding to each other over time in a shared design representation. This demonstration is a step towards our original motivation of incorporating open-ended and conversational aspects of architectural composition, such as the exploration of emerging design perspectives with situated action.

5. Discussion, future steps, and conclusions

This research emerged from revisiting the problem of spatial synthesis and rebuilding it from the ground up based on a design theoretical perspective on interaction and on novel computational ideas that are still underexplored in the field. To the extent of our knowledge, the proposed framework and the design artifact constitute the first successful computational method for the design of an ecology of agents that can address multiple and custom spatial synthesis requirements with designers in the generative loop (see 7.7.8 A brief overview of reinforcement learning in architectural design). They also resulted in future objects of inquiry, which we will discuss under three categories

- Technical extensions
- Agent-based architectural design and morphology
- Situational design exploration and collaborative design

5.1. Technical extensions

Considering the positive results of the design artifact presented in this dissertation, there are many windows of opportunities for future research that rely on pushing the limits of the current technical setting. These are some of the aspects that can be changed to explore more variations of agent policies and potentially more complex layouts:

- Agent representation
    - extending the perception and action grids to provide more contextual information
    - increasing the action grid to allow shapes with higher resolution
    - extending the maximum number of adjacencies per agent
- Goals
    - leaving all the goal parameters free, including not folding or daylight access, to incentivize the existence of more heterogenous agents in the same simulation
    - adding logical operators to the rewards, such as “all” or “any”, so that certain agents can ignore specific goals when necessary
    - supporting shared goals among groups of agents, to create adaptable clusters of agents
- Training agents to discover long-term and collaborative behaviors by using
    - Longer training to enable long-term strategy and improve robustness to initialization
    - Curriculum for gradual skill acquisition, from single agent with simple tasks to multi-

agent addressing higher level strategies

- A massive number of agents

5.2. Agent-based architectural design and morphology

In architectural theory, bottom-up patterns of many similar elements, such as flocking patterns or group forms have been associated with novel approaches to architectural configuration, with open form and no compositional identity (Maki 1964; Allen 2012a). However, while many of these patterns are achievable with existing agent-based models, it is not straightforward to incorporate realistic architectural requirements to them. As a result, there is a gap between the exploration of bottom-up architectural form with relaxed design constraints and the development of practical agent-based tools that can support design practice and morphological analysis.

The methodology of agent design in the research artifact contributed to bridging this gap. This research provides a framework that supports the study and exploration of spatial configurations that emerge from local actions conditioned on the configuration of the model and of the simulation. These include the design constraints embedded in the formulation of the agent and of the environment, the design knowledge embedded in its training, and the specific configurations of a simulation.

The artifact successfully satisfied architectural requirements and captured different bottom-up organizational strategies with distributed policies and fine-grained control. As a result, it produced a diversity of site and layout configurations with functional relationships and consistent spatial qualities based on the proliferation of small and similar decentralized agents that grow and adapt to the circumstances. These aggregative patterns include clusters of linear rooms, irregular niches and corridors, conventional layout with quasi-rectangular rooms, etc.

Some of these can be identified as the compositional orders presented in Figure 2. For example, the policy Adjacency, Area, and Daylit (75) or Adjacency, Area, and Not Folding (3) resulted in layouts with non-hierarchical patterns associated with bottom-up architectural organization, such as mat-buildings and group forms based on clusters of linear rooms or maze-like modules.

In other cases, such as in Adjacency, Area, and Not Folding (5) or Adjacency, Area, Not Folding (5), and Daylit (50), the model produced variations of conventional architectural types that are not typically associated with agent-based modeling. These resemble variations of conventional compositions, such as free plan compositions with abstract geometry and hyper specialization of spaces and circulation systems. If the boundary of the environment restricts the agents, these same goals result in packing of quasi-convex polygonal rooms, such as the traditional floorplan layouts described by the pattern 191 (shape of indoor space) in A Pattern Language (Alexander et al. 1977, 883–92).

In terms of urban form, with more agents in the simulation and a larger site, specific cluster configurations that were not embedded directly into the policy emerged, such as a hierarchy of irregular courtyards, boulevards, and patios between housing units. These elements started to

display characteristics similar to the pattern 106 positive outdoor space (Alexander et al. 1977, 517–23), where building forms shape the outdoor space as a series of areas that are mostly convex and bounded. This also resonates with research on distributed computational methods to model vernacular morphology. For example, the Beady Ring is a discrete cellular model that captures the configuration of many urban settlements in Southern France (Hillier and Hanson 1984, 52–81). It uses pre-defined topological constraints to ensure that the aggregation of cells will result in dense clusters of buildings surrounded by rings of open spaces. In our design artifact, the agent restrictions

Overall, the resulting layouts with consistent spatial patterns produced by the design artifact demonstrate its capacity to explore architectural morphology through the lenses of agent-based computing. In this respect, there are some future challenges to be addressed.

The first is the incorporation of top-down compositional principles, such as regularity, axiality, symmetry and hierarchy into our agent-based framework. This is an interesting challenge for agent-based computing, because it relies on pre-defined spatial concepts that depend on the coordination of all the computational agents. It is not clear if algorithms based on multi-agent learning combined with global signals from the environment can address it or if global methods are necessary.

The second is the investigation of alternative forms of spatial representation and neural network architectures that can efficiently learn from them. The former would include 2.5 and 3-D grids, boundary models based on polygons or polyhedrons, meshes, or space partitions such as Voronoi or KD-trees. That would allow the artifact to model multi-story buildings, massing models, linked assemblies, etc. and therefore comprehend a larger scope of morphological spaces. Additionally, the mechanisms for post-processing could also be explored. Currently there is a single parametric model generating the 3D version of the spatial arrangement, so other versions created by different designers could be integrated to the interactive simulation workflow. Besides, the parametric model could also be replaced by a generative learning model, which could map the agent representation to a detailed floorplan representations according to a certain style.

Finally, it would be interesting to infer spatial goals from design examples that are familiar to designers instead of explicitly engineering a reward function. One alternative is to use example-based policy search derive the policy of the agents from these examples (Eysenbach, Levine, and Salakhutdinov 2021). A simpler alternative would be to train a neural network as a spatial goal for form outside of the reinforcement learning algorithm. On one hand, in this supervised learning setting, it would be necessary to have access to a curated dataset with the fitness of the agent form annotated. On the other hand, this would make the framework even more adaptive to existing design practices and would facilitate designers to create and share custom spatial agents. A vast production and socialization of agent-based models could incentivize novel forms of morphological investigations and collaborative design.

5.3. Situational design exploration and collaborative design

This research focused on a model of interaction with designers inside the loop sharing a fine-grained and local control of the sequential generation with computational agents. On one hand, this idea is inspired by the *status quo* of situational actions based on the affordances of the design tools to form design representations and ideas over time. This is a common practice in the discipline of architecture that is largely described in the literature of design theory. On the other hand, it is a disruptive idea because it uses novel forms of artificial intelligence to distribute the design decisions at the granular level of control between humans and computational agents, dissolving the boundary between synthesis (by computer) and exploration (by designers). With multi-agent spatial synthesis, novel spatial patterns can emerge from designers and computational agents continuously responding to and influencing each other over time in a shared design representation.

Overall, this is part of a larger trend of computational design research to explore alternatives to indirect generative methods based on coarse turn-taking between designers and computers.

For example, in design optimization, designers essentially define the preconditions for the design generation and later filter the favorite entries among the solutions produced by the computer. With the support of dashboard and graph visualizations, designers can also learn about the relationships between design parameters, fully formed configurations, and performance to pursue further design exploration (1.3.3.3). This type of performance-informed design space exploration workflow intends to use the feedback enable the optimization to support convergent and divergent thinking, and also the co-evolution of problem and solution (Wortmann 2018, 74–77). While there are research initiatives addressing more granular interactions, they face the challenge of restructuring the decision mechanisms used by optimization algorithms to enable a lower cognitive load with more opportunities for human improvisation (see 1.3.4.4).

Another example is generative learning (7.6.6), which relies on the idea of training a computational model as a mapping between design representations. In architectural design, this typically relies on mapping a pre-condition, such as a vector, a textual prompt, a graph, or a spatial representation, to representations of the target architectural configurations. To increase the granularity of the process, recent work explicitly trains one or more models to capture chains of decisions corresponding to different scopes of the spatial representation (Chaillou 2019) or to the assignment and placement of the different elements of the design (He, Huang, and Wang 2022). However, this approach still works with a sequence of coarse steps which are mostly pre-determined by the designers of the model. Besides, it requires having access or generating a large dataset with all the mappings forming the proposed decision structure.

Our agent-based approach does not adapt the structure of existing models to increase the granularity of the interaction. In contrast to the examples above, it builds the architecture of the generative model from the ground up based on a fine-granular model of interaction. It benefits from having the whole structure of the generative model designed for fine-granular interaction

without front-loading a specific decision structure. By combining agent-based modeling and multi-agent deep reinforcement learning, our method captures the decision-making structure of emerging and distributed agent policies on fine-granular spatial units. In this setting, it is straightforward to situate the designers in the generative process and to support coupling their thinking, perception, and action with spatial plays with distributed computational agents.

In this respect, there are some opportunities for future research.

The simulation where designer working and computational work on the shared design representation is currently limited by a single thread of design exploration. This could be amplified by exploratory strategies that are common in computer software and in other generative paradigms. For example, designers could build a catalogue with design alternatives collected during the simulation or generated by another algorithm. These alternatives could be retrieved at any point of the simulation to increase the capacity to explore different design scenarios over time and under uncertainty.

A second aspect to be developed is to evaluation of the proposed workflow with designers to investigate the types of situated action and play with the design representation that are supported by its structure of interaction. Further, it would be interesting to understand if these actions and plays can lead to emerging design hypotheses and perspectives. This type of evaluation could address both the case with a single designer and the case with a team of designers working on the same design problem.

5.4. Conclusion

Motivated by open-ended and conversational aspects of architectural composition, such as the exploration of emerging design perspectives with situated action, this research addressed the specific problem of creating a computational approach for spatial synthesis to support designers inside the generative loop with fine-grained interaction. The main outcomes are described in the following table.

Initially, the research used taxonomy building to organize the main ideas and paradigms of generative modeling and spatial synthesis, and to identify challenges in terms of design interaction. Then, it concentrated on developing a computational framework based on real-time interaction, granular control over the spatial representations, event-based computing, reduced assumption about central control strategies, and realistic architectural goals (see **1.4**). The development of a new framework was necessary as an alternative to the recent advancements in design optimization and deep learning in spatial synthesis, which mainly rely on other forms of interaction.

Table 11 Overview of outcomes of the chapters and selected sections in the appendix.

	Chapter / Section	Outcomes	Publication Type	Publication Reference
1	In pursuit of agent-based spatial synthesis for situational design	Construction of a taxonomy of generative models. Definition of a research framework. Definition of generative techniques for the research artifact.	Conference paper Conference paper Book Chapter	A (Veloso and Krishnamurti 2019) B (Veloso, Rhee, and Krishnamurti 2019) C (Veloso and Krishnamurti 2021a)
2	Designing and implementing the design artifact	Formulation of design artifact based on agent-based model and reinforcement learning. Implementation and training of three prototypes.	Conference paper Conference paper	D (Veloso and Krishnamurti 2021b) E (Veloso and Krishnamurti 2020)
3	Evaluating the design artifact	Evaluation of the capacity of the prototypes to address varied spatial goals and to produce coherent spatial patterns using three design cases. This is a proof by construction of the research framework.	Conference paper Conference paper Article	D (Veloso and Krishnamurti 2021b) E (Veloso and Krishnamurti 2020) F Under development
4	An interactive simulation workflow for situational design exploration	Integration of the design artifact to an interactive simulation workflow using a game engine and a parametric modeling editor.	Article	G Under development
5	Discussion, future steps, and conclusions	Description of future steps of the research in terms of technical improvement, architectural morphology, and situated design. Summary of the dissertation.		
	Appendices			
7.5	Appendix E: A review of multi-agent spatial synthesis	Review of spatial synthesis prototypes based on agent-based modeling and adjacent techniques. Identification of challenges.	Conference paper	B (Veloso, Rhee, and Krishnamurti 2019)
7.6.6	Generative Learning	Description and examples of the category generative learning in the proposed taxonomy.	Book Chapter	C (Veloso and Krishnamurti 2021a)

To prove the feasibility of this computational framework, we developed a research artifact that uses agent-based models, which can be robust, scalable, flexible, and can capture the dynamics of emerging and self-organizing systems. By distributing the computation and using simpler policies, agents enable designers to explore spatial patterns in large state spaces with event-driven computing, and real-time and fine-grained interactions. The artifact also integrates multi-agent reinforcement learning techniques to tackle a main bottleneck of existing spatial synthesis approaches based on agent-based computing: lack of a clear methodology of agent design that can support the multiple constraints and goals of architectural layouts.

In the artifact, multi-agent deep reinforcement learning is used as a training methodology for custom agents that can handle multiple architectural objectives while preserving the benefits of decentralized computing. This is based on training an ecology of spatial agents with parameter sharing and a parameterized reward function to support domain knowledge and enable

heterogeneous agent behaviors. To incentivize better policy generalization and reduce the dependence on initial design states and order of operations, the shared policy is trained in randomized environments and with different problem specifications.

The artifact succeeded in addressing multiple design cases: a house design in different sites, a museum on a complex site, and a complex with eight houses. The artifact also showed that by customizing their spatial goals, it is possible to produce variations of agent designs that result in radically different spatial patterns. Besides, the artifact also supports event-driven computing, real-time and fine-grained interaction. This was guaranteed by using reactive and distributed agents. We integrated these reactive agents to a parametric modeling editor and a game engine to build an interactive workflow with all the characteristics described in the research framework.

Overall, the construction of the framework, artifact, and the interactive workflow opened doors to different lines of future research, such as the investigation of agent-based architectural morphology, the further development of ecologies of agents that can handle more complex problems, and mainly, the evaluation of situational and collaborative design behavior supported by computational agents.

6. References

AI, OPEN. 2018. *Part 2: Kinds of RL Algorithms — Spinning Up Documentation*. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

Akin, O., Bharat Dave, and Shakunthala Pithavadian. 1987. "Problem Structuring in Architectural Design." *Carnegie Mellon University*. https://doi.org/10.1184/R1/6076109.v1.

Akin, Ömer. 1986. *Psychology of Architectural Design*. Pion Limited.

Akin, Ömer, and Ran Sen. 1996. "Navigation within a Structured Search Space." *Environment and Planning B: Planning and Design* 23: 421–42.

Akizuki, Yuta, Mathias Bernhard, Reza Kakooee, Marirena Kladeftira, and Benjamin Dillenburger. 2020. "Generative Modelling with Design Constraints - Reinforcement Learning for Object Generation." In *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th International Conference on Computer-Aided Architectural Design Researchin in Asia (CAADRIA)*, 1:445–54. Bangkok. http://papers.cumincad.org/cgi-bin/works/paper/caadria2020_306.

Alexander, Christopher. 1964. *Notes on the Synthesis of Form*. Harvard University Press.

Alexander, Christopher, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Algorta, Simón, and Özgür Şimşek. 2019. "The Game of Tetris in Machine Learning." *ArXiv:1905.01652 [Cs]*, May. http://arxiv.org/abs/1905.01652.

Allen, Stan. 2009. *Practice: Architecture, Technique+ Representation*. Routledge.

———. 2012a. "Field Conditions." In *Points+Lines: Diagrams and Projects for the City*, 1st ed., 90–103. New York: Princeton Architectural Press.

———. 2012b. *Points+Lines: Diagrams and Projects for the City*. 1st ed. New York: Princeton Architectural Press.

Anderson, Stanford. 1966. "Problem-Solving and Problem-Worrying." *Architectural Association*.

As, Imdat, and Prithwish Basu. 2021. *The Routledge Companion to Artificial Intelligence in Architecture*. 1st ed. Routledge Companions. New York: Routledge.

Ashby, W. Ross. 1956. *An Introduction to Cybernetics*. Chapman&Hall. London.

Babaoglu, Ozalp, Geoffrey Canright, Andreas Deutsch, Gianni A. Di Caro, Frederick Ducatelle, Luca M. Gambardella, Niloy Ganguly, Márk Jelasity, Roberto Montemanni, and Alberto Montresor. 2006. "Design Patterns from Biology for Distributed Computing." *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1 (1): 26–66.

Banham, Reyner. 1980. *Theory and Design in the First Machine Age*. The MIT Press.

———. 1990. "A Black Box." *New Statesman & Society* 3: 122.

Bayraktar, Mehmet Emin, and Gülen Çağdaş. 2013. “Fuzzy Layout Planner: A Simple Layout Planning Tool for Early Stages of Design.” In *Computation and Performance: Proceedings of the 31st ECAADe Conference*, edited by Stouffs Rudi and Sariyildiz Sevil, 1:375–81. Delft: TUDelft.

Bazalo, Frano, and Tane J. Moleta. 2015. “Responsive Algorithms.” In *Emerging Experience in Past, Present and Future of Digital Architecture: Proceedings of the 20th International Conference on Computer-Aided Architectural Design Researchin in Asia (CAADRIA)*, edited by Y. Ikeda, C. M. Herr, D. Holzer, S. Kaijima, M. J. Kim, and M. A. Schnabel, 209–18. Hong Kong: CAADRIA.

Bellman, Richard E. 1957. *Dynamic Programming*. Princeton University Press.

Bidgoli, Ardavan, and Pedro Veloso. 2018. “DeepCloud. The Application of a Data-Driven, Generative Model in Design.” In *Recalibration: On Imprecision and Infidelity. Proceedings of the 38th ACADIA Conference*, edited by Phillip Anzalone, Marcella Del Signore, and Andrew J. Wit, 176–85. Mexico City: Universidad Iberoamericana. http://papers.cumincad.org/cgi-bin/works/paper/acadia18_176.

Bilgin, Enes. 2020. *Mastering Reinforcement Learning with Python: Build next-Generation, Self-Learning Models Using Reinforcement Learning Techniques and Best Practices*. 1st edition. Packt Publishing.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. New York: Springer-Verlag.

Bohnacker, Hartmut, Benedikt Gross, Julia Laub, and Claudius Lazzeroni. 2012. *Generative Design: Visualize, Program, and Create with Processing*. Princeton Architectural Press.

Bonebau, Eric, Marco Dorigo, and Guy Theraulaz. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Studies in the Sciences of COmplexity. New York: Oxford University Press.

Bonsai. 2017. “Deep Reinforcement Learning Models: Tips & Tricks for Writing Reward Functions.” *Medium* (blog). November 16, 2017. https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge university press.

Bratko, Ivan. 2011. *Prolog Programming for Artificial Intelligence*. Harlow: Pearson Education.

Brownlee, Jason. 2012. *Clever Algorithms: Nature-Inspired Programming Recipes*. lulu.com. http://www.cleveralgorithms.com/nature-inspired/index.html.

Cagan, Jonathan, Jonathan I. Campbell, Susan Finger, and Tetsuo Tomiyama. 2005. “A Framework for Computational Design Synthesis: Model and Applications.” *Journal of Computing and Information Science in Engineering* 5 (9): 171–81.

Calixto, Victor, and Gabriela Celani. 2015. “A Literature Review for Space Planning Optimization Using an Evolutionary Algorithm Approach: 1992-2014.” In *Project Information for Interaction: Proceedings of 19th SIGraDi Conference*, 2:662–71. São Paulo: Blucher.

Cardon, Dominique, Jean-Philippe Cointet, and Antoine Maziéres. 2018a. “Neurons Spike Back.” Neurons Spike Back. 2018. https://neurovenge.antonomase.fr/.

———. 2018b. “Neurons Spike Back: The Invention of Inductive Machines and the Artificial Intelligence Controversy.” *Réseaux*, Machines Prédictives, 211 (5): 173–220. https://doi.org/10.3917/res.211.0173.

Carpo, Mario. 2013. *The Digital Turn in Architecture 1992-2012*. Chichester: John Wiley & Sons.

———. 2017. *The Second Digital Turn: Design Beyond Intelligence*. Cambridge: The MIT Press.

Carta, Silvio. 2021. “Self-Organizing Floor Plans.” *Harvard Data Science Review* 3 (3). https://doi.org/10.1162/99608f92.e5f9a0c7.

Carta, Silvio, Stephanie St Loe, Tommaso Turchi, and Joel Simon. 2020. “Self-Organising Floor Plans in Care Homes.” *Sustainability* 12 (May): 4393. https://doi.org/10.3390/su12114393.

Causevic, Siwei. 2020. “A Structural Overview of Reinforcement Learning Algorithms.” Medium. June 18, 2020. https://towardsdatascience.com/an-overview-of-classic-reinforcement-learning-algorithms-part-1-f79c8b87e5af.

Chaillou, Stanislas. 2019. “AI + Architecture: Towards a New Approach.” Master’s thesis, Cambridge: Harvard. https://view.publitas.com/harvard-university/ai-architecture-thesis-harvard-gsd-stanislas-chaillou.

Chasioti, Eleni. 2020. “‘Gameplay with Encoded Architectural Tilesets: A Computational Framework for Building Massing Design Using the Wave Function Collapse Algorithm.’” Master’s thesis, University College London.

Chen, Liang. 2012. “Agent-Based Modeling in Urban and Architectural Research: A Brief Literature Review.” *Frontiers of Architectural Research* 1 (2): 166–77.

Christensen, Jesper Thøger. 2014. “The Generation of Possible Space Layouts.” In *Fusion: Data Integration at Its Best.: Proceedings of the 32nd ECAADe Conference*, edited by Emine Mine Thompson, 239–46. Newcastle upon Tyne: eCAADe.

Churchman, Charles W., Russel L. Ackoff, and E. Leonard Arnoff. 1957. *Introduction to Operations Research*. New York: Wiley.

Clancey, William J. 1997. *Situated Cognition: On Human Knowledge and Computer Representations*. 1st ed. Cambridge: Cambridge University Press.

Coates, Paul, and Robert Thum. 1995. *Generative Modelling*. University of East London. http://roar.uel.ac.uk/948/.

Colquhoun, Alan. 1969. “Typology and Design Method.” *Perspecta*, 71–74.

Corona-Martínez, Alfonso. 2003. *The Architectural Project*. 6. Texas A&M University Press.

Coulom, Rémi. 2006. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search.” In *Proceedings of the 5th International Conference on Computer and Games*. Turin, Italy.

Cross, Nigel. 1982. “Designerly Ways of Knowing.” *Design Studies* 3 (4): 221–27.

———. 2001. “Designerly Ways of Knowing: Design Discipline versus Design Science.” *Design Issues* 17 (3): 49–55.

———. 2006. *Designerly Ways of Knowing*. London: Springer.

———. 2011. *Design Thinking: Understanding How Designers Think and Work*. Berg.

Danhaive, Renaud, and Caitlin Mueller. 2015. "Combining Parametric Modeling and Interactive Optimization for High-Performance and Creative Structural Design." In *Future Visions - Proceedings of the International Association for Shell and Spatial Structures*. Amsterdam: IASS.

Davis, Daniel. 2013a. "A History of Parametric." *Daniel Davis* (blog). 2013. https://www.danieldavis.com/a-history-of-parametric/.

———. 2013b. "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture."

De Wolf, Tom, and Tom Holvoet. 2005. "Emergence Versus Self-Organisation: Different Concepts but Promising When Combined." In *Engineering Self-Organising Systems: Methodologies and Applications*, edited by Sven. A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal. Lecture Notes in Computer Science. Berlin: Springer. https://doi.org/10.1007/11494676_1.

Derix, Christian, and Asmund Izaki. 2013. "Spatial Computing for the New Organic." *Architectural Design* 83 (2): 42–47.

Di Caro, Gianni. 2004. "Ant Colony Optimization and Its Application to Adaptive Routing in Telecommunication Networks." PhD Dissertation, Institut de Recherches Interdisciplinaires et de D´eveloppements en Intelligence Arti cielle.

Downey, Allen B. 2012. *Think Complexity*. Sebastopol: O'Reilly Media. http://thinkcomplex.com.

Eastman, Charles M. 1970. "Representations for Space Planning." Edited by C. L. Lawson. *Communications of the ACM* 13 (4).

Eastman, Charles N. 1975. *Spatial Synthesis in Computer-Aided Building Design*. Elsevier Science Inc.

Egor, Gavrilov, Schneider Sven, Dennemark Martin, and Koenig Reinhard. 2020. "Computer-Aided Approach to Public Buildings Floor Plan Generation. Magnetizing Floor Plan Generator." *Procedia Manufacturing*, The 1st International Conference on Optimization-Driven Architectural Design (OPTARCH 2019), 44 (January): 132–39. https://doi.org/10.1016/j.promfg.2020.02.214.

El-Khaldi, Maher (Maher Sami). 2007. "Mapping Boundaries of Generative Systems for Design Synthesis." Thesis, Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/39323.

Eysenbach, Ben, Sergey Levine, and Russ R Salakhutdinov. 2021. "Replacing Rewards with Examples: Example-Based Policy Search via Recursive Classification." In *Advances in Neural Information Processing Systems*, 34:11541–52. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2021/hash/5ffaa9f5182c2a36843f438bb1fdbdea-Abstract.html.

Fang, Zhihao, Yuning Wu, Ammar Hassonjee, Ardavan Bidgoli, and Daniel Cardoso-Llach. 2020. "Towards a Distributed, Robotically Assisted Construction Framework." In *Proceedings of the 40th Annual Conference of the Association of Computer Aided Design in Architecture (ACADIA)*, edited by Brian Slocum, Viola Ago, Shelby Doyle, Adam Marcus, Maria Yablonina, and Matias

del Campo, 1:320–29. Online and Global: Acadia Publishing Company. http://papers.cumincad.org/cgi-bin/works/paper/acadia20_320.

Fernando, Ruwan. 2014. “Space Planning and Preliminary Design Using Artificial Life.” In *Rethinking Comprehensive Design: Speculative Counterculture: Proceedings of the 19th International Conference on Computer-Aided Architectural Design Researchin in Asia (CAADRIA)*, edited by Ning Gu, Shun Watanabe, Halil Erhan, Matthias Hank Haeusler, Weixin Huang, and Ricardo Sosa, 657–66. Hong Kong: CAADRIA.

Fischer, Thomas, and Christiane M. Herr. 2001. “Teaching Generative Design.” In *Proceedings of the 4th Conference on Generative Art*. Politechnico di Milano University.

Floreano, Dario, and Claudio Mattiussi. 2008. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Foerster, Jakob, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. “Counterfactual Multi-Agent Policy Gradients.” *ArXiv:1705.08926 [Cs]*, December. http://arxiv.org/abs/1705.08926.

Foster, David. 2019. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play*. Sebastopol: O’Reilly Media.

Fragada, Luis E., Steve Baer, and William F. Cook. 2022. “Rhino.Inside® Samples.” Code Repository. GitHub. 2022. https://github.com/mcneel/rhino-developer-samples.

Fruchterman, Thomas M. J., and Edward M. Reingold. 1991. “Graph Drawing by Force-Directed Placement.” *Software: Practice and Experience* 21 (11): 1129–64. https://doi.org/10.1002/spe.4380211102.

Gänshirt, Christian. 2007. *Tools for Ideas: Introduction to Architectural Design*. Basel: Birkhäuser.

Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. 2016. “Image Style Transfer Using Convolutional Neural Networks.” In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2414–23. Las Vegas, NV, USA: IEEE. https://doi.org/10.1109/CVPR.2016.265.

Gausa, Manuel. 2010. *OPEN: Arquitectura y Ciudad Contemporánea. Teoría e Historia de Un Cambio*. Barcelona: Actar.

Gausa, Manuel, Vicente Guallart, Willy Müller, Federico Soriano, Fernando Porras, and José Morales. 2003. “Field.” In *The Metapolis Dictionary of Advanced Architecture*, 220–22. Barcelona: Actar.

Gero, John S. 1994. “Towards a Model of Exploration in Computer-Aided Design.” In *Proceedings of the IFIP TC5/WG5.2 Workshop on Formal Design Methods for CAD*, edited by John S. Gero and Enn Tyugu, 315–36. IFIP Transactions. Tallinn: Elsevier.

Gips, James, and George Stiny. 1980. “Production Systems and Grammars: A Uniform Characterization.” *Environment and Planning B: Planning and Design* 7: 399–408.

Glanville, Ranulph. 2009. “A (Cybernetic) Musing: Design and Cybernetics.” *Cybernetics & Human Knowing* 16 (3–1): 175–86.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. “Generative Adversarial Nets.” In *Proceedings of NIPS: Advances in Neural Information Processing Systems*, 2672–80. Montreal: Curran Associates, Inc.

Goodfellow, Ian, Bengio Yoshua, and Courville Aaron. 2016. *Introduction to Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goodhouse, Andrew, ed. 2013. *When Is the Digital in Architecture?* Montreal: CCA and Sternberg Press.

Gorbman, Yasha J., Abraham Yezioro, and Isaac G. Capeluto. 2009. “Computer-Based Form Generation in Architectural Design - a Critical Review.” *International Journal of Architectural Computing* 7 (4): 535–53.

Graesser, Laura, and Wah Loon Keng. 2019. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Data & Analytics Series. Addison-Wesley Professional.

Gumin, Maxim. 2016. “Mxgmn/WaveFunctionCollapse.” C#. https://github.com/mxgmn/WaveFunctionCollapse.

Guo, Zifeng, and Biao Li. 2017. “Evolutionary Approach for Spatial Architecture Layout Design Enhanced by an Agent-Based Topology Finding System.” *Frontiers of Architectural Research* 6 (1): 53–62.

Gupta, Jayesh K., Maxim Egorov, and Mykel Kochenderfer. 2017. “Cooperative Multi-Agent Control Using Deep Reinforcement Learning.” In *Autonomous Agents and Multiagent Systems*, edited by Gita Sukthankar and Juan A. Rodriguez-Aguilar, 66–83. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-71682-4_5.

Hao, Hua, and Jia Ting-Li. 2010. “Floating Bubbles.” In *New Frontiers: Proceedings of the 15th International Conference on Computer-Aided Architectural Design Researchin in Asia (CAADRIA)*, edited by B. Dave, A. I. Li, N. Gu, and H.-J. Park, 175–83. Hong Kong: CAADRIA.

Harding, John. 2016. “Evolving Parametric Models Using Genetic Programming with Artificial Selection.” In *Complexity & Simplicity: Proceedings of the 34th ECAADe Conference*, 1:423–32. Oulu: eCAADe.

Harding, John, and Cecilie Brandt-Olsen. 2018. “Biomorpher: Interactive Evolution for Parametric Design.” *International Journal of Architectural Computing* 16 (June): 144–63. https://doi.org/10.1177/1478077118778579.

Harding, John, and Christian Derix. 2011. “Associative Spatial Networks in Architectural Design: Artificial Cognition of Space Using Neural Networks with Spectral Graph Theory.” In *Design Computing and Cognition’10*, edited by John S. Gero, 305–23. Dordrecht: Springer.

Hassel, Hado van, Arthur Guez, and David Silver. 2016. “Deep Reinforcement Learning with Double Q-Learning.” In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2094–2100. Phoenix: AAAI.

He, Feixiang, Yanlong Huang, and He Wang. 2022. *IPLAN: Interactive and Procedural Layout Planning*. https://doi.org/10.1109/CVPR52688.2022.00764.

Heess, Nicolas, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, et al. 2017. “Emergence of Locomotion Behaviours in Rich Environments.” *ArXiv:1707.02286 [Cs]*, July. http://arxiv.org/abs/1707.02286.

Henrion, Max. 1978. “Automatic Space-Planning: A Postmortem?” In *Conference Proceedings*, 175–91. New York: North-Holland.

Henrique, Gonçalo C., Ernesto Bueno, Daniel Lenz, and Victor Sardenberg. 2019. “Generative Systems: Interwining Physical, Digital and Biological Processes, a Case Study.” In *Architecture in the Age of the 4th Industrial Revolution: Proceedings of the 37th ECAADe and 23rd SIGraDi Conference*. Vol. 1. Porto: eCAADe-SIGraDi-FAUP.

Herr, Christiane, and Thomas Fischer. 2019. *Design Cybernetics and CAAD Research: Aspects of Our Shared Interests*.

Herr, Christiane M., and Ryan C. Ford. 2015. “Adapting Cellular Automata as Architectural Design Tools.” In *Emerging Experience in Past, Present and Future of Digital Architecture: Proceedings of the 20th Proceedings of the 23rd International Conference on Computer-Aided Architectural Design Researchin in Asia (CAADRIA)*, 169–78. Daegu: Kyungpook National University.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David SIlver. 2018. “Rainbow: Combining Improvements in Deep Reinforcement Learning.” In *The Thirty-Second AAAI Conference on Artificial Intelligence*, 3215–22. New Orleans: AAAI.

Hillier, Bill, and Julienne Hanson. 1984. “The Logic of Space.” In *The Social Logic of Space*, 52–81. Cambridge: Cambridge University Press.

Hindupur, Avinash. n.d. “Hindupuravinash/the-Gan-Zoo.” Repository. The GAN Zoo. Accessed January 27, 2020. https://github.com/hindupuravinash/the-gan-zoo.

Holland, John H. 1995. *Hidden Order: How Adaptation Builds Complexity*. New York: Basic Books.

———. 1998. *Emergence: From Chaos to Order*. New York: Basic Books.

———. 2012. *Signals and Boundaries: Building Blocks for Complex Adaptive Systems*. Cambridge: The MIT Press.

Homayouni, Hoda. 2000. “A Survey of Computational Approaches to Space Layout Planning (1965-2000).” *Department of Architecture and Urban Planning University of Washington*.

Hosmer, Tyson, Panagiotis Tigas, Dave Reeves, and Ziming He. 2020. “Spatial Assembly with Self-Play Reinforcement Learning.” In *Distributed Proximities: Proceedings of the 40th Annual Conference of the Association of Computer Aided Design in Architecture (ACADIA)*, edited by Brian Slocum, Viola Ago, Shelby Doyle, Adam Marcus, Maria Yablonina, and Matias del Campo, 1:382–93. Online and Global: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/acadia20_320.

Hsu, Ying-Chun, and Robert J. Krawczyk. 2003. “New Generation of Computer Aided Design in Space Planning Methods: A Survey and a Proposal.” In *Proceedings of the 8th CAADRIA Conference*, edited by K. Keatruangkamala and W. Nakapan, 101–16. Bangkok: Rangsit University.

Huang, Chenyu, Genjia Zhang, and Jiawei Yao. 2022. “Energy-Driven Intelligent Generative Urban Design, Based on Deep Reinforcement Learning Method With a Nested Deep Q-R Network.” In *POST-CARBON - Proceedings of the 27th International Conference of the Association for Computer Aided Architectural Design Research in Asia (CAADRIA)*, edited by Jeroen van Ameijde, Nicole Gardner, Kyung Hoon Hyun, Dan Luo, and Urvi Sheth, 233–42. Sydney: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/caadria2022_239.

Huang, Chien-hua. 2021. “Reinforcement Learning for Architectural Design-Build - Opportunity of Machine Learning in a Material-Informed Circular Design Strategy.” In *Proceedings of the 26th International Conference of the Association for Computer Aided Architectural Design Research in Asia (CAADRIA)*, edited by A. Globa, J. van Ameijde, A. Fingrut, N. Kim, and T.T.S. Lo, 1:171–80. Hong Kong,: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/caadria2021_118.

Huang, Weixin, and Hao Zheng. 2018. “Architectural Drawings Recognition and Generation Through Machine Learning.” In *Recalibration on Imprecision and Infidelity: Proceedings of the 38th Annual ACADIA Conference*, edited by Phillip Anzalone, Marcella Del Signore, and Andrew J. Wit, 156–65. Mexico CIty: ACADIA.

Ireland, Tim. 2009a. “Emergent Space Diagrams: The Application of Swarm Intelligence to the Problem of Automatic Plan Generation.” In *Joining Languages, Cultures and Visions: Proceedings of the 13th CAAD Futures Conference*, edited by Temy Tidafi and Tomás Dorta, 245–58. Montréal: Les Presses de lUniversité de Montréal.

———. 2009b. “Sniffing Space II: The Use of Artificial Ant Colonies to Generate Circulation Patterns in Buildings.” In *Joining Languages, Cultures and Visions: Proceedings of the 13th CAAD Futures Conference*, edited by Temy Tidafi and Tomás Dorta, 214–27. Montréal: Les Presses de lUniversité de Montréal.

———. 2010. “Stigmergic Planning.” In *LIFE in:Formation, On Responsive Information and Variations in Architecture: Proceedings of the 30th ACADIA Conference*, edited by Aaron Sprecher, Shai Yeshayahu, and Pablo Lorenzo-Eiroa, 183–89. New York: ACADIA.

———. 2015a. “A Cell Inspired Model of Configuration.” In *Computational Ecologies: Design in the Anthropocene: Proceedings of 35th ACADIA Conference*, edited by Lohns Combs and Chris Perry, 136–47. New York: ACADIA.

———. 2015b. “An Artificial Life Approach to Configuring Architectural Space.” In *Real Time: Proceedings of 33rd ECAADe Conference*, edited by Bob Martens, Gabriel Wurzer, Thomas Grasi, Wolfgang E. Lorenz, and Richard Schaffranek, 2:581–90. Wien: eCAADe.

Jacq, Alexis. n.d. “Neural Transfer Using PyTorch — PyTorch Tutorials 1.5.1 Documentation.” Tutorial. Neural Style Transfer Using Pytorch. Accessed June 25, 2020. https://pytorch.org/tutorials/advanced/neural_style_tutorial.html.

Janssen, Patrick, and Rudi Stouffs. 2015. “Types of Parametric Modelling.” In *Emerging Experience in Past, Present and Future of Digital Architecture: Proceedings of the 20th CAADRIA Conference*, edited by Yasushi Ikeda, Christiane M. Herr, Dominik Holzer, Sawako Kaijima, Mi Jeong Kim, and Marc Aurel Schnabel, 157–66. Hong Kong: CAADRIA.

Jones, John C. 1992. *Design Methods*. 2nd ed. Wiley.

Joyce, Sam, and Ibrahim Nazim. 2021. “Limits to Applied ML in Planning and Architecture: Understanding and Defining Extents and Capabilities.” In *Towards a New, Configurable Architecture. Proceedings of the 39th ECAADe Conference*, edited by V. Stojakovic and B. Tepavcevic, 1:243–52. Novi Sad, Serbia: eCAADe.

Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, et al. 2021. “Highly Accurate Protein Structure Prediction with AlphaFold.” *Nature*, July, 1–11. https://doi.org/10.1038/s41586-021-03819-2.

Kennedy, James, Russel C. Eberhart, and Yuhui Shi. 2001. *Swarm Intelligence*. San Francisco: Morgan Kaufmann Publishers.

Knecht, Katja, and Reinhard König. 2010a. “Generating Floor Plan Layouts with Kd Trees and Evolutionary Algorithms.” In *Proceedings of the 13th Generative Art Conference*, 238–53. Milan: Domus Argenia Publisher. http://www.generativeart.com/.

———. 2010b. “Generating Floor Plan Layouts with K-d Trees and Evolutionary Algorithms.” In *Proceedings of the 13th Generative Art Conference*, 238–53. Milan. http://www.generativeart.com/.

Knight, Terry, and George Stiny. 2001. “Classical and Non-Classical Computation.” *Arq: Architectural Research Quarterly* 5 (4): 355–72.

Kocsis, Levente, and Csaba Szepesvári. 2006. “Bandit Based Monte-Carlo Planning.” In *Machine Learning: ECML 2006*, edited by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, 282–93. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/11871842_29.

Koenig, Reinhard. 2011. “Generating Urban Structures: A Method for Urban Planning Supported by Multi-Agent Systems and Cellular Automata.” *Przestrzen I Forma*, no. 16: 353–76. https://doi.org/10.31219/osf.io/s4crf.

Koenig, Reinhard, Matthias Standfest, and Gerhard Schmitt. 2014. “Evolutionary Multi-Criteria Optimization for Building Layout Planning: Exemplary Application Based on the PSSA Framework.” In *Fusion: Proceedings of the 32nd ECAADe Conference*, 2:567–74. Newcastle upon Tyne, England, UK.

Kolarevic, Branko. 2003. “Digital Morphogenesis.” In *Architecture in the Digital Age: Design and Manufacturing*, 12–28. London: Spoon Press.

———. 2005. “Digital Morphogenesis.” In *Architecture in the Digital Age: Design and Manufacturing*, edited by Branko Kolarevic, 12–28. London: Taylor & Francis.

Kotov, Anatolii, and Llija Vukorep. 2021. “Gridworld Architecture Testbed.” In *Towards a New, Configurable Architecture: Proceedings of the 39th Conference on Education and Research in Computer Aided Architectural Design in Europe*, edited by V. Stojakovic and B. Tepavcevic, 1:37–44. Novi Sad, Serbia: eCAADe. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_252.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. Cambridge: The MIT Press.

Krishnamurti, Ramesh. 1980. “The Arithmetic of Shapes:” *Environment and Planning B: Planning and Design*. https://doi.org/10.1068/b070463.

Krishnamurti, Ramesh, and Peter H. O’N. Roe. 1978. “Algorithmic Aspects of Plan Generation and Enumeration.” *Environment and Planning B: Planning and Design* 5 (2): 157–77. https://doi.org/10.1068/b050157.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. “ImageNet Classification with Deep Convolutional Neural Networks.” In *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Lawson, Bryan. 2005. *How Designers Think: The Design Process Demystified*. 4th ed. Amsterdam: Elsevier.

Leach, Neil. 2021. *Architecture in the Age of Artificial Intelligence: An Introduction to AI for Architects*. 1st edition. London: Bloomsbury Visual Arts.

Liggett, Robin S. 1980. “The Quadratic Assignment Problem: An Analysis of Applications and Solution Strategies.” *Environment and Planning B: Planning and Design* 7 (2): 141–62. https://doi.org/10.1068/b070141.

Liggett, Robin S. 1985. “1 - Optimal Spatial Arrangement as a Quadratic Assignment Problem.” In *Design Optimization*, edited by John S. Gero, 1–40. Academic Press. https://doi.org/10.1016/B978-0-12-280910-1.50006-4.

———. 2000. “Automated Facilities Layout: Past, Present and Future.” *Automation in Construction* 9 (2): 197–215.

Lin, Cheng, Tingxiang Fan, Wenping Wang, and Matthias Nießner. 2020. “Modeling 3D Shapes by Reinforcement Learning.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, 545–61. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-58607-2_32.

Liu, Jie, Tim Dwyer, Guido Tack, Samuel Gratzl, and Kim Marriott. 2020. “Supporting the Problem-Solving Loop: Designing Highly Interactive Optimisation Systems.” *ArXiv:2009.03163 [Cs]*, September. http://arxiv.org/abs/2009.03163.

Lobos, Danny, and Dirk Donath. 2010. “The Problem of Space Layout in Architecture: A Survey and Reflections.” *Arquitetura Revista* 6 (2): 136–61.

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.” In *Advances in Neural Information Processing Systems*, 6382–93.

Lucan, Jacques. 2012. *Composition, Non-Composition: Architecture and Theory in the Nineteenth and Twentieth Centuries*. EPFL Press.

Lynn, Greg. 1999. *Animate Form*. New York: Princeton Architectural Press.

Maki, Fumihiko. 1964. “Investigations into Collective Form.” *Washington: Architecture School of Washington University*.

Mandelbrot, Benoit B. 1982. *The Fractal Geometry of Nature*. Vol. 1. WH freeman New York.

March, Lionel. 2011. “Forty Years of Shape and Shape Grammars, 1971 – 2011.” *Nexus Network Journal* 13 (1): 5–13. https://doi.org/10.1007/s00004-011-0054-8.

Meyboom, AnnaLisa, and Dave Reeves. 2013. “Stigmergic Space.” In *Adaptive Architecture: Proceedings of the 33rd ACADIA Conference*, edited by Philip Beesley, Michael Stacey, and Omar Khan, 199–206. Toronto: Riverside Architectural Press.

Mintrone, Alessandro, and Alessio Erioli. 2021. “Training Spaces - Fostering Machine Sensibility for Spatial Assemblages through Wave Function Collapse and Reinforcement Learning.” In *Towards a New, Configurable Architecture - Proceedings of the 39th ECAADe Conference*, edited by V. Stojakovic and B. Tepavcevic, 1:17–26. Novi Sad, Serbia: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_148.

Mitchell, Tom. 1997. *Machine Learning*. Boston: McGraw-Hill.

Mitchell, William. 1977. *Computer-Aided Architectural Design*. New York: Mason Charter.

———. 1990. *The Logic of Architecture: Design, Computation, and Cognition*. Cambridge: The MIT Press.

———. 1998. “Antitectonics: The Poetics of Virtuality.” In *The Virtual Dimension: Architecture, Representation and Crash Culture. Princeton Architectural Press, New York*, 205–17.

Mitchell, William, Robin Liggett, and Thomas Kvan. 1987. *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers*. New York: Van Nostrand Reinhold Company.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. “Asynchronous Methods for Deep Reinforcement Learning.” *ArXiv:1602.01783 [Cs]*, June. http://arxiv.org/abs/1602.01783.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. “Playing Atari with Deep Reinforcement Learning.” *ArXiv Preprint ArXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski. 2015. “Human-Level Control through Deep Reinforcement Learning.” *Nature* 518 (7540): 529.

Møller, Tobias N., and Jonas A. Billeskov. 2019. “Expanding Wave Function Collapse with Growing Grids for Procedural Content Generation.” Project. Copenhagen: Aalborg University Copenhagen.

“Monoceros.” 2021. Text. Food4Rhino. January 21, 2021. https://www.food4rhino.com/en/app/monoceros.

Morales, Miguel. 2020. *Grokking Deep Reinforcement Learning*. 1st ed. Manning Publications. https://www.manning.com/books/grokking-deep-reinforcement-learning.

Mordvintsev, Alexander, Christopher Olah, and Mike Tyka. 2015. “Inceptionism: Going Deeper into Neural Networks.” *Google AI Blog* (blog). July 13, 2015. http://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

Narahara, Taro. 2017. “Collective Construction Modeling and Machine Learning: Potential for Architectural Design.” In *Sharing Computational Knowledge!: Proceedings of the 35th ECAADe Conference*, edited by Antonio Fioravanti, Stefano Cursi, Salma Elahmar, Silvia Gargaro, Gianluigi Loffreda, Gabriele Novembri, and Armando Trento, 341–48. Rome: eCAADe.

Nauata, Nelson, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. 2020. “House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation.” In *Proceedings of the European Conference on Computer Vision (ECCV)*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, 162–77. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-58452-8_10.

Navlakha, Saket, and Ziv Bar-Joseph. 2011. “Algorithms in Nature: The Convergence of Systems Biology and Computational Thinking.” *Molecular Systems Biology* 7 (1): 546.

———. 2015. “Distributed Information Processing in Biological and Computational Systems.” *Communications of the ACM* 58 (1): 94–102.

Newell, Allen. 1967. “Heuristic Programming: Ill-Structured Problems,” 81.

Newell, Allen, and Herbert A. Simon. 1972. *Human Problem Solving*. New Jersey: Prentice Hall.

Newton, David. 2019. “Generative Deep Learning in Architectural Design.” *Technology | Architecture + Design* 3 (2): 176–89. https://doi.org/10.1080/24751448.2019.1640536.

———. 2021. “Toward Dynamic and Explorative Optimization for Architectural Design.” In *Routledge Companion to AI in Architecture*, edited by Imdat As and Prithwish Basu, 280–300. Routledge.

Niazi, Muaz, and Amir Hussain. 2011. “Agent-Based Computing from Multi-Agent Systems to Agent-Based Models: A Visual Survey.” *Scientometrics* 89 (2): 479–99. https://doi.org/10.1007/s11192-011-0468-9.

Nilsson, Nils J. 2009. *The Quest for Artificial Intelligence*. Cambridge University Press.

O’Connor, Jennifer, Eleanor Lee, Francis Rubinstein, and Stephen Selkowitz. 1997. “Tips for Daylighting with Windows.” Ernest Orlando Lawrence Berkeley National Laboratory University of. https://facades.lbl.gov/sites/all/files/tips-for-daylighting-1997.pdf.

Otto, Frei. 2003. *Occupying and Connecting*. Edition Axel Menges.

Oxman, Rivka. 2006. “Theory and Design in the First Digital Age.” *Design Studies* 27 (3): 229–65.

———. 2008. “Digital Architecture as a Challenge for Design Pedagogy: Theory, Knowledge, Models and Medium.” *Design Studies* 29 (2): 99–120.

Oxman, Rivka, and Robert Oxman. 2014. “From Composition to Generation.” In *Theories of the Digital in Architecture*, 55–61. New York: Routledge.

Ozbulak, Utku. 2017. “Pytorch CNN Visualizations.” GitHub. 2017. https://github.com/utkuozbulak/pytorch-cnn-visualizations.

Pearson, John E. 1993. “Complex Patterns in a Simple System.” *Science* 261 (5118): 189–189. https://doi.org/10.1126/science.261.5118.189.

Pickering, Andrew. 2010. *The Cybernetic Brain: Sketches of Another Future*. University of Chicago Press.

Picon, Antoine. 2004. “Architecture and the Virtual: Towards a New Materiality.” *PRAXIS: Journal of Writing Building*, no. 6: 114–21.

Protzen, Jean-Pierre, and David J. Harris. 2010. *The Universe of Design: Horst Rittel’s Theories of Design and Planning*. Routledge.

Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. 2004. “Graphical Modeling Using L-Systems.” In *The Algorithmic Beauty of Plants*. New York: Springer-Verlag.

Puusepp, Renee. 2014a. “Agent-Based Models for Computing Circulation.” In *Design Agency: Proceedings of the 34th ACADIA Conference*, edited by David Gerber, Alvin Huang, and Jose Sanchez, 43–52. Cambridge: Riverside Architectural Press.

———. 2014b. “Spatial Agglomerates.” In *Rethinking Comprehensive Design: Speculative Counterculture: Proceedings of the 19th CAADRIA Conference*, edited by Ning Gu, Shun Watanabe, Halil Erhan, Matthias Hank Haeusler, and Weixin Huang, 585–94. Hong Kong: CAADRIA.

Radford, Antony D., and John S. Gero. 1988. *Design by Optimization in Architecture, Building, and Construction*. New York: Van Nostrand Reinhold Company.

Rashid, Tabish, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.” *ArXiv:1803.11485 [Cs, Stat]*, June. http://arxiv.org/abs/1803.11485.

Reinhart, C F. 2005. “A Simulation-Based Review of the Ubiquitous Window-Head-Height to Daylit Zone Depth Rule-of-Thumb,” 10.

Reinhart, Christoph. 2014. *Daylighting Handbook I*. 1st edition. Building Technology Press.

Retsin, Gilles. 2019. “Discrete Architecture in the Age of Automation.” *Architectural Design*, Special Issue, 89 (2): 144.

Reynolds, Craig. 1987. “Flocks, Herds and Schools: A Distributed Behavioral Model.” In *ACM SIGGRAPH Computer Graphics*, 21:25–34. ACM.

———. 1999. “Steering Behaviors for Autonomous Characters.” In *Game Developers Conference*, 1999:763–82.

Rhee, Jinmo, and Pedro Veloso. 2021. “Generative Design of Urban Fabric Using Deep Learning.” In *Proceedings of the 26th International Conference of the Association for Computer Aided Architectural Design Research in Asia (CAADRIA)*, edited by A. Globa, J. van Ameijde, A. Fingrut, N. Kim, and T.T.S. Lo, 1:31–40. Hong Kong.

Rittel, Horst WJ. 1972. *On the Planning Crisis: Systems Analysis of the" First and Second Generations"*. Institute of Urban and Regional Development Berkeley.

Rittel, Horst WJ, and Melvin M. Webber. 1973. “Dilemmas in a General Theory of Planning.” *Policy Sciences* 4 (2): 155–69.

Rosa, Joseph. 2003. *Next Generation Architecture: Folds, Blobs, and Boxes*. New York: Rizzoli.

Rowe, Peter G. 1987. *Design Thinking*. Cambridge: The MIT Press.

———. 2017. *Design Thinking in the Digital Age*. The Incidents. Cambridge: Harvard University Graduate School of Design.

Ruiz-Montiel, Manuela, Javier Boned, Juan Gavilanes, Eduardo Jiménez, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. 2013. “Design with Shape Grammars and Reinforcement Learning.” *Advanced Engineering Informatics* 27 (2): 230–45. https://doi.org/10.1016/j.aei.2012.12.004.

Russell, Stuart J., and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson Series in Artificial Intelligence. Hoboken: Pearson.

Rutten, David. 2013. “Galapagos: On the Logic and Limitations of Generic Solvers.” *Architectural Design* 83 (2): 132–35. https://doi.org/10.1002/ad.1568.

Saha, Nirvik, John Haymaker, and Dennis Shelden. 2020. “Space Allocation Techniques (SAT).” In *Distributed Proximities - Proceedings of the 40th Annual Conference of the Association of Computer Aided Design in Architecture (ACADIA)*, edited by B. Slocum, V. Ago, S. Doyle, A. Marcus, M. Yablonina, and M. del Campo, 1:248–57. CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/acadia20_248.

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. “Evolution Strategies as a Scalable Alternative to Reinforcement Learning.” *ArXiv:1703.03864 [Cs, Stat]*, September. http://arxiv.org/abs/1703.03864.

Schön, Donald. 1983. *The Reflective Practitioner: How Professionals Think In Action*. New York: Basic Books.

———. 1992. “Designing as Reflective Conversation with the Materials of a Design Situation.” *Knowledge-Based Systems* 5 (1): 3–14. https://doi.org/10.1016/0950-7051(92)90020-G.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. “Proximal Policy Optimization Algorithms.” *ArXiv:1707.06347 [Cs]*, August. http://arxiv.org/abs/1707.06347.

Schumacher, Patrik. 2008. “Parametricism as Style - Parametricist Manifesto.” 2008. https://www.patrikschumacher.com/Texts/Parametricism%20as%20Style.htm.

Shoham, Yoav, and Kevin Leyton-Brown. 2009. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical FOundations*. Cambrisge University Press.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, et al. 2017. “Mastering the Game of Go without Human Knowledge.” *Nature* 550 (7676): 354–59. https://doi.org/10.1038/nature24270.

Silver, David, Satinder Singh, Doina Precup, and Richard S. Sutton. 2021. “Reward Is Enough.” *Artificial Intelligence* 299 (October): 103535. https://doi.org/10.1016/j.artint.2021.103535.

Simon, Herbert A. 1973. “The Structure of Ill Structured Problems.” *Artificial Intelligence* 4 (3–4): 181–201.

———. 1975. “Style in Design.” *Spatial Synthesis in Computer-Aided Building Design* 9: 287–309.

———. 1996. *The Sciences of the Artificial*. 3rd ed. Cambridge: The MIT Press.

Simon, Joel. 2017. “Evolving Floorplans.” 2017. http://www.joelsimon.net/evo_floorplans.html.

Simonyan, Karen, and Andrew Zisserman. 2015. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” *ArXiv:1409.1556 [Cs]*, April. http://arxiv.org/abs/1409.1556.

Sims, Karl. 1994a. “Evolving 3D Morphology and Behavior by Competition.” In *Artificial Life IV*, edited by Rodney A. Brooks and Pattie Maes, 12. A Bradford Book.

———. 1994b. “Evolving Virtual Creatures.” In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH ’94*, 15–22. Not Known: ACM Press. https://doi.org/10.1145/192161.192167.

Sjoberg, Christian, Christopher Beorkrem, and Jefferson Ellinger. 2017. “Emergent Syntax: Machine Learning for the Curation of Design Solution Space.” In *Disciplines Disruption: Proceedings of the 37th ACADIA Conference*, edited by Takehiko Nagakura, Skylar Tibbits, Mariana Ibañez, and Caitlin Mueller, 552–61. Cambridge, MA: ACADIA.

Smith, Shane I., and Chris Lasch. 2016. “Machine Learning Integration for Adaptive Building Envelopes: An Experimental Framework for Intelligent Adaptive Control.” In *Proceedings of the 36th ACADIA Conference: Posthumans Frontiers*, edited by Kathy Velikov, Sandra Manninger, Matias del Campo, Sean Ahlquist, and Geoffrey Thün, 98–105. Ann Arbor: ACADIA.

Song, Yanan, and Philip F. Yuan. 2021. “A Research on Building Cluster Morphology Formation Based on Wind Environmental Performance and Deep Reinforcement Learning.” In *Towards a New, Configurable Architecture - Proceedings of the 39th ECAADe Conference*, edited by V. Stojakovic and B. Tepavcevic, 1:335–44. Novi Sad, Serbia: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_150.

“Spring_layout.” n.d. Documentation. NetworkX - Network Analysis in Python. Accessed May 26, 2022. https://networkx.org/documentation/stable/reference/generated/networkx.drawing.layout.spring_layout.html.

Steadman, Philip. 1973. “Graph Theoretic Representation of Architectural Arrangement.” *Architectural Research and Teaching*, 161–72.

———. 1983. *Architectural Morphology: An Introduction to the Geometry of Building Plans*. Pion: Pion.

Stevens, Garry. 2002. *The Favored Circle: The Social Foundations of Architectural Distinction*. The MIT Press.

Stiny, George. 1980. “Introduction to Shape and Shape Grammars.” *Environment and Planning B: Urban Analytics and City Science* 7 (3): 343–51. https://doi.org/10.1068/b070343.

———. 2006. *Shape: Talking about Seeing and Doing*. Cambridge: The MIT Press.

Suchman, Lucy. 2006. *Human-Machine Reconfigurations: Plans and Situated Actions*. 2nd ed. Learning in Doing: Social, Cognitive and Computational Perspectives. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511808418.

Sutton, Richard S., and Andrew G. Barto. 2018a. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge: The MIT Press. http://incompleteideas.net/book/.

———. 2018b. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge: The MIT Press.

Sutton, Richard S, David McAllester, Satinder Singh, and Yishay Mansour. 2000. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press. https://papers.nips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

Syp, Marc. 2011. "Space Planning with Physics." 2011. http://www.marcsyp.com/space-planning/.

Terry, J. K., Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, et al. 2021. "PettingZoo: Gym for Multi-Agent Reinforcement Learning." *ArXiv:2009.14471 [Cs, Stat]*, June. http://arxiv.org/abs/2009.14471.

Terry, Justin K., Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. 2021. "Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning." *ArXiv:2005.13625 [Cs, Stat]*, February. http://arxiv.org/abs/2005.13625.

Thurner, Stefan, Rudolf Hanel, and Peter Klimek. 2018. *Introduction to the Theory of Complex Systems*. Oxford : New York: Oxford University Press.

Tomasetti, Thornton. 2019. "Design Explorer." Building Performance Study - Demo. 2019. http://tt-acm.github.io/DesignExplorer/.

Tran Luciani, Danwei, Jonas Löwgren, and Jonas Lundberg. 2019. "Designing Fine-Grained Interactions for Automation in Air Traffic Control." *Cognition, Technology and Work* 22 (4): 685–701. https://doi.org/10.1007/s10111-019-00598-9.

Turing, Alan M. 1952. "The Chemical Basis of Morphogenesis." *Philosophical Transactions of the Royal Society of London*, Biological Sciences, 237 (641): 37–72.

Veloso, Pedro. 2014. "Exploring the Bubble Diagram." In *Design in Freedom: Proceedings of the 18th SIGraDi Conference*, edited by Fernando G. Amen, 1:115–19. São Paulo: Blucher. http://cumincad.scix.net/cgi-bin/works/paper/sigradi2014_128.

Veloso, Pedro, Gabriela Celani, and Rodrigo Scheeren. 2018. "From the Generation of Layouts to the Production of Construction Documents: An Application in the Customization of Apartment Plans." *Automation in Construction* 96 (9): 224–35.

Veloso, Pedro, and Ramesh Krishnamurti. 2019. "From Black Box to Generative System." In *BLACK BOX: Articulating Architecture's Core in the Post-Digital Era*. Pittsburgh: ACSA. https://www.researchgate.net/publication/333089518_From_Black_Box_to_generative_System.

———. 2020. "An Academy of Spatial Agents." In *Anthropologic - Architecture and Fabrication in the Cognitive Age. Proceedings of the 38th ECAADe Conference*, edited by Liss C. Werner and Dietmar Köring, 2:191–200. Berlin: CUMINCAD.

———. 2021a. “Mapping Generative Models for Architectural Design.” In *Routledge Companion to AI in Architecture*, edited by Imdat As and Prithwish Basu, 29–58. Routledge.

———. 2021b. “Self-Learning Agents for Spatial Synthesis.” In *Proceedings of the 5th International Symposium on Formal Methods in Architecture (5FMA)*, edited by Sara Eloy, David L. Viana, Franklim Morais, and Jorge V. Vaz. Lisbon: Springer Nature.

Veloso, Pedro, Jinmo Rhee, and Ramesh Krishnamurti. 2019. “Multi-Agent Space Planning: A Literature Review (2008-2017).” In *Hello, Culture!: Proceedings of 18th CAAD Futures Conference*, edited by Ji-Hyun Lee, 52–74. Daejeon, Korea.

Vishal, Singh, and Ning Gu. 2012. “Towards an Integrated Generative Design Framework.” *Design Studies* 33 (2): 185–207. https://doi.org/10.1016/j.destud.2011.06.001.

Von Buelow, Peter. 2012. “ParaGen: Performative Exploration of Generative Systems.” *Journal of the International Association for Shell and Spatial Structures* 53 (December): 271.

Vrachliotis, Georg, Joachim Kleinsmanns, Martin Kunz, and Philip Kurz, eds. 2017. *Frei Otto: Thinking by Modeling*. Leipzig: Spector Books.

Wang, Dasong, and Roland Snooks. 2021a. “Artificial Intuitions of Generative Design: An Approach Based on Reinforcement Learning.” In *Proceedings of the 2020 DigitalFUTURES*, edited by Philip F. Yuan, Jiawei Yao, Chao Yan, Xiang Wang, and Neil Leach, 189–98. Singapore: Springer. https://doi.org/10.1007/978-981-33-4400-6_18.

———. 2021b. “Intuitive Behavior - The Operation of Reinforcement Learning in Generative Design Processes.” In *Projections - Proceedings of the 26th International Conference of the Association for Computer Aided Architectural Design Research in Asia (CAADRIA)*, edited by A. Globa, J. van Ameijde, A. Fingrut, N. Kim, and T.T.S. Lo, 1:101–10. Hong Kong,: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/caadria2021_118.

“Wasp.” 2017. Text. Food4Rhino. September 19, 2017. https://www.food4rhino.com/en/app/wasp.

Weinzapfel, Guy, and Steven Handel. 1975. “IMAGE: Computer Assistant for Architectural Design.” In *Spatial Synthesis in Computer-Aided Building Design*, by Charles M. Eastman, 61–68. London: Applied Science Publishers.

Weinzapfel, Guy, Timothy E. Johnson, and John Perkins. 1971. “IMAGE: An Interactive Computer System for Multi-Constrained Spatial Synthesis.” In *Proceedings of the 8th Design Automation Workshop*, 101–8. ACM.

Weinzapfel, Guy, and Nicholas Negroponte. 1976. “Architecture-by-Yourself: An Experiment with Computer Graphics for House Design.” In *Proceedings of the 3rd Annual Conference on Computer Graphics and Interactive Techniques*, 10:74–78. Philadelphia: Association for Computing Machinery. https://doi.org/10.1145/563274.563290.

Weng, Jiayi, Minghao Zhang, Alexis Duburcq, and Kaichao You. 2020. “Tianshou v0.3.0.” Repository. GitHub. 2020. https://github.com/thu-ml/tianshou/releases/tag/v0.3.0.

Wibranek, Bastian, Yuxi Liu, Niklas Funk, Boris Belousov, Jan Peters, and Oliver Tessmann. 2021. “Reinforcement Learning for Sequential Assembly of SL-Blocks - Self-Interlocking Combinatorial Design Based on Machine Learning.” In *Towards a New, Configurable*

Architecture - Proceedings of the 39th ECAADe Conference, edited by V. Stojakovic and B. Tepavcevic, 1:27–36. Novi Sad, Serbia: CUMINCAD. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_247.

Wilensky, Uri, and William Rand. 2015. *An Introduction to Agent-Based Modeling: Modeling Natural, Societal and Engineered Complex Systems with Netlogo*. Cambridge: The MIT Press.

Williams, Ronald J. 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." *Machine Learning* 8 (3): 229–56. https://doi.org/10.1007/BF00992696.

Winograd, Terry, and Fernando Flores. 1986. *Understanding Computers and Cognition: A New Foundation for Design*. Massachusetts: Addison-Wesley.

Witten Jr, T. A., and Leonard M. Sander. 1981. "Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon." *Physical Review Letters* 47 (19): 1400.

Wolfram, Stephen. 2002. *A New Kind of Science*. Champaign: Wolfram Media.

Woodbury, Robert. 2010. *Elements of Parametric Design*. New York: Routledge.

Woodbury, Robert F., and Andrew L. Burrow. 2003. "Notes on the Structure of Design Space." *International Journal of Architectural Computing* 1 (4): 517–32. https://doi.org/10.1260/147807703773633518.

———. 2006. "Whither Design Space?" *Artif. Intell. Eng. Des. Anal. Manuf.* 20 (2): 63–82. https://doi.org/10.1017/S0890060406060057.

Wooldridge, Michael. 2009. *An Introduction to Multiagent Systems*. 2nd ed. John Wiley & Sons.

Wortmann, Thomas. 2017a. "Model-Based Optimization for Architectural Design: Optimizing Daylight and Glare in Grasshopper." *Technology| Architecture+ Design* 1 (2): 176–85. https://doi.org/10.1080/24751448.2017.1354615.

———. 2017b. "Surveying Design Spaces with Performance Maps: A Multivariate Visualization Method for Parametric Design and Architectural Design Optimization" 15 (1): 38–53.

———. 2018. "Efficient, Visual, and Interactive Architectural Design Optimization with Model-Based Methods." PhD Dissertation, Singapore: Singapore University of Technology and Design.

Zellner, Peter. 1999. *Hybrid Space: Generative Form and Digital Architecture*. New York: Rizzoli.

Zhang, Aston, Zack C. Lipton, Mu Li, Alex J. Smola, Brent Weness, Rachel Hu, Shuai Zhang, Yi Tay, Anirudh Dagar, and Yuan Tagar. 2021. "Dive into Deep Learning." On-line book. Dive into Deep Learning. June 2021. https://d2l.ai/index.html.

Zheng, Hao. 2018. "Drawing with Bots: Human-Computer Collaborative Drawing Experiments." In *Learning, Prototyping, and Adapting: Proceedings of the 23rd International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA)*, edited by Tomohiro Fukuda, Weixin Huang, Patrick Janssen, Kristof Crolla, and Suleiman Alhadidi. Hong Kong: CAADRIA.

7. Appendices

7.1. Appendix A: Notes on architectural composition

Traditionally, the existence of a formal order in architectural practice is associated with the term composition. The basic idea of composition refers to the organization of parts and relations to form a whole susceptible to some overall ordering principle. In the context of architectural design, it is associated with the core process of architecture conception in which each part of the design is determined by its own spatial configuration, by connections with other spaces, and by the functions they have in relation to the whole (Corona-Martínez 2003, 131). Composition comprehends not only the set of desired relation among the spatial elements but also the set of strategies to implement these relations. In this sense, designers often impose spatial constraints and context-specific heuristics to restructure an architectural problem and converge to specific types of solutions.

7.1.1. Origins of architectural composition

Along the history, the professional understanding of architectural composition has not been situated under the objective framework of problem-solving. Instead, it has been developed as ongoing conversation of individuals and groups with the discipline of architecture itself, leading to spatial solutions beyond the fundamental requirements of space. Composition has always been loaded with multiple subtleties and ideological divergences. As design ideologies change, there is a recurrent re-formulation of the spatial logic and used to order space. Prof. Jacques Lucan, an expert researcher on composition, states that "in architecture, to compose is to conceive a building following principles or rules that certain architects have attempted to set forth, in particular by means of the teaching they practice or the articles and books they publish" (2012, 6). As the idea of composition is built based on local paradigms – the consensus of an architect or a group of architects for a typical design solution, it is subjected to constant redefinitions, ubiquitous use, and even denial.

According to Lucan (2012, 21–23), Leon Battista Alberti introduced the term composition for painting in 1435 in the treatise "De picture". However, the first architectural use of the word composition only appeared in 1734, with Robert Morris' Lectures on Architecture. It was related to the asymmetrical design principles of the British Picturesque. In the 18th and 19th centuries, composition – but also disposition and distribution - became dominant in the architectural theory to indicate the design of building masses and layout. Facing the functional specialization of rooms, increasingly complexity of circulation systems, and constraints of the site, composition was the activity of organizing architectural space according to functionality and a body of aesthetic principles.

After its introduction in the 15th century for painting, the term composition transitioned to the territory of the architectural theory to indicate the design of building masses and layout. One of the most notorious systematic compositional methods was developed by Jean-Nicolas-Louis Durand and presented his "Precis des leçons d´architecture données à l´École Polytechnique"

(1802). It provided a hierarchical decomposition of buildings into construction materials, building elements, and building parts (Lucan 2012, 29–50). A combinatorial method described the assembly of materials into elements, elements into parts and, finally, parts into a complete building, following a simplified version of composition with grids and axes of composition.

It is also important to notice that here was a clear distinction between a more pragmatic (and even algorithmic) approach to composition of the Polytechnique and the approach consolidated in architecture by the École des Beaux-Arts and propagated to other schools in the Beaux-Arts system. In the latter, the design of buildings was also subscribed to the academic principles of regularity, axiality, symmetry and hierarchy, ensuring an organic unit of the whole. However, it was more tacit and personal, establishing an ambiguous relationship with architectural theory. As Lucan points out, "throughout the nineteenth century, the leading figures in Ecole des Beaux-Arts instructions felt no particular need to set down the specific principles guiding them, and all the more so as theory was often viewed with suspicion by the atelier heads themselves" (Lucan 2012, 155)

7.1.2. Composition and non-composition over the last century

In the first half of the 20th century, architectural composition was subjected to drastic changes. Particularly with the modern avant-gardes, many design artifices were used to disrupt and neglect the closed order underlying the academic composition.

While modernist architects frequently declared the overcoming of composition, there is no consensus about the success of this endeavor. For instance, Banham (1980) claim that the design methods and construction ideals of modern architecture have sustained a symbolic and aesthetic relationship with the new technologies rather than adopting new operative principles. In this view, while the elements and aesthetics principles of composition changed with the avant-garde, the underlying abstract compositional method was still pervasive. Similarly to Banham, Corona-Martínez (2003) argues that despite the loss of principles and elements of classical composition, functionalism maintained the underlying compositional method and updated it with the use of abstract geometry and the hyper specialization of the spaces and circulation systems. He understands that modernist composition oscillates between the extremes of functionally specialized spaces and continuous space with strategies such as free-plan or transparency.

This argument can be illustrated with the four categories of composition proposed by the French-Swiss architect Le Corbusier as an opposing alternative to the traditional house (Figure 2 - top). :

- "The first type shows each organ rising up next to its neighbor, in accordance with an **organic reasoning**".
- "The second type shows the compression of organs within a **rigid envelope**, absolutely pure".
- "The third type furnishes, with a visible framework (**skeleton structure**), a simple envelope, clear, transparent as a network; it allows the creation of useful volumes of rooms different

on each floor in form and quantity".

- "The fourth type attains, on the outside, the pure form of the second type; inside it had the advantages, the characteristics of the first and third" (Lucan 2012, 411).

Lucan understands the impact of the modernist spatial compositional principles from a different perspective. He describes a tension between strategies of composition and strategies against composition in the modernist and contemporary architecture. He even introduces the terms "open order" and "non-composition" to describe these conscious attempts in the last century to move beyond the closed-order principles of composition (Lucan 2012, 7). Along different chapters, Lucan presents a series of strategies used by modernist architects to begin a disruption with academic composition and look for an open order. These examples include dispersion of irregular disposed parts, free plan, the rupture of the enclosure with free wall planes, and collage and assemblage of forms.

Following Lucan's chronology, while traditional and postmodernist architects rescued and adapted the classical composition as a design tool for the contemporary design, the following generations of modernist and contemporary architects took a further step to find alternatives for composition. In this context of, Lucan (2012) associates the increasingly diagrammatic approach to architectural design with the increasing dissolution of the ordered assemblage of figural objects and composition. These include neutral grids, large-scale and high-density aggregative structure that enables growth and functional change (mat-building), use of diagrams as a syntactical exploration of form, diagrams to formalize design operations and program montage. These categories contain different views on the creation and organization of spatial arrangements in an architectural project, distinct both from the academic and modernist compositional strategies.

In this context, the taxonomy introduced by Fumihiko Maki in "Investigations in Collective Form" (Maki 1964) addresses the interest in collective form and the departure from the top-down design approach. Maki contraposes the composition with figural objects with a bottom-up approach based on growth and adaptation processes. To explain this contrast, Maki proposes three categories of architectural form (Figure 2 - middle):

- compositional form: figural elements that are conceived and determined by rules of composition. Notice that it applies both to academic and modernist free-form composition
- mega form: a main open-ended or hierarchical megastructure that guides and regulates all the small and changing elements of the collective form
- group form: generated by a set of small, decentralized elements that are able proliferate, grow, and adapt to the circumstances, preserving its base form

Another relevant taxonomy for this discussion is presented by Stan Allen (2009) who understands that there is a continuum between classical, modern, and even contemporary compositional strategies as they are generated by top-down rules and are based on the organization of figural

objects. While the rules of composition that govern these modernist compositions of fragments may be new, the underlying (classic) assumption that composition is concerned with the arrangement of a connection among these parts persists (Allen 2012a; 2009, 239).

Allen establishes four categories of compositional strategies (Figure 2 – bottom left):

- axial symmetry refers to the classical composition, based on axiality, symmetry and hierarchy in pursuit of an organic whole
- peripheral composition refers to free form and centripetal organizational associated with modernism
- collision refers to fragments of shapes associated with deconstructivism
- linked assemblies are probably associated with aggregative structures – such as the mat-buildings and mega-structures described by Lucan

Allen's text is a provocative manifesto for a new class of non-compositional organizational strategies that he defines as field-conditions. In opposition to all the above cited forms of composition, field-conditions are bottom-up organization of many similar parts. These parts are aggregate small bodies with no compositional identity or meaning. These aggregates are ruled by some form of local forces of the system, such as with iron fillings, schools, swarms, or crowds. It is important to note that with the definition of field-conditions, Allen not only connects diagrammatic strategies in art and architecture with complexity theory – particularly with self-organizing systems –, but also with agent-based modelling.

While the description of field-conditions as a non-compositional methodology is still speculative in "From Object to Field", it acquires a more pragmatic tone in the text "The tick 2-d" (2009). While in the former he exemplifies field conditions with clusters, open clusters, mosaics, and twigs, in the latter, he adds patterns such as patchwork, loose grid, striated, field vectors, and felt (Figure 2 – bottom right). Besides, he also associates mat-building with the diagrammatic work of contemporary architectural firms such as OMA, MVRDV or SANAA, and situate them under the umbrella of field-conditions.

The message with this brief review of the idea of composition and some of its taxonomies is that the organization of spatial configurations in architecture is not only a problem-solving activity, but an activity that relies on configurational patterns discussed and shared inside the profession. While these patterns are can respond to larger technological and social issues, they are mainly validated inside the profession, with its changing architectural ideologies and methods.

7.2. Appendix B: The influence of classic AI methods on generative models

Framing spatial configuration as problem-solving activity made it compatible with the methods of emerging fields, such as symbolic artificial intelligence (AI) and operations research (OR), and with the computational and technical infrastructure of the time: labs equipped with mainframes of limited graphical, interactive, and parallel computing.

Operations research was created during World War II to provide a scientific basis for decision making in military operations and was later adapted for general management (Churchman, Ackoff, and Arnoff 1957). It is one of the fields that rely on modeling a decision problem with mathematical functions. There is a function that maps a set of decision variables to a certain objective (an objective function) and the constraints, which are equalities or inequalities based on these variables. This is the general definition of an optimization problem:

Table 12 Definition of the general optimization problem

Minimize or maximize an objective function	$f_0(x_0, \ldots, x_{n-1}) \rightarrow \mathbb{R}$
Subject to constraints	$f_1(x_0, \ldots, x_{n-1}) \leq 0$
	...
	$f_m(x_0, \ldots, x_{n-1}) \leq 0$

This formulation provides both the design space (the possible values of the decision variables) and a way to measure the value of any solution (the output of the objective function for any legal combination of the decision variables). Even when certain values are not available, they should be approximated or specified by the decision maker to fit this mathematical structure. Then, optimization techniques, such as analytic methods or algorithms, can find configurations of the variables that maximize or minimize the objective function subject to the constraints.

In contrast, Symbolic AI had a larger ambition of providing a model not only for solving problems but also for understanding how humans solve problem. This subject was highly influenced by the pioneer work of Herbert Simon in the fields of behavior economics and cognitive science and Newell and Simon's information processing theory (IPS). IPS interprets both humans and computers as symbol-processing agents that, abstracted from physiological bases, can solve problems with limited knowledge of the environment and memory (Newell and Simon 1972).

While in a mathematical optimization the design space is defined by decision variables, in symbolic AI it is defined by symbolic states of the design alternatives, which are represented as nodes in a decision network that is often referred to as state space. There are three types of nodes: the initial state, intermediate states, and a goal state. The initial state represents the initial configuration at the beginning of the problem solving, while the goal state is a state that contains solutions for the problem. The remaining states are intermediate states. Computational descriptions or tests can characterize states and detect the goal state. In terms of navigation, there are operators that are represented by the edges in the network that transform one state into another.

In contrast to optimization, where the goal is to find the variables in a pre-determined design space that result in the optimal solution, Symbolic AI relies on a search algorithm that uses heuristics to

systematically expand the nodes, which results in a search tree. A heuristic in this context represents any form of useful knowledge that used for expanding the search towards the goal states. Therefore, in a problem formulated as a search, a symbolic decision maker (human or a computer) starts at an initial state of the problem, then iteratively perceives information in the problem environment and uses custom heuristics to systematically choose a transformation among the available edges that would eventually take to a satisfactory state (the goal). Heuristic search became "(…) the primary weapons in the armory of 1960s AI researchers and to the establishment of problem-solving as the canonical AI task."(Russell and Norvig 2021, 106),

Despite the difference between the original logic of search and optimization, there has been a long effort to understand them and other related computational methods as members of a family of problem solvers. For example, Newell (1967) situates the different optimization and search strategies in a space defined by two dimensions: generality and power. More general methods require less information from the problem environment and can address larger sets of problems in a domain. In his example, a method that can locate warehouses is less general than one that can solve all physical resource location problems (Newell 1967, 12). In contrast, the evaluation of a method in terms of power comprehends its capacity to deliver solutions, which may comprehend the probability of reaching a solution in the domain, the quality of the solutions, the amount of computation used to find a solution (Newell 1967, 12–13).

For Newell, there is a correlation between power and generality of computational methods. More powerful methods tend to require more information from the task environment and therefore are less general. For example, heuristic search is more general (and less powerful) than optimization techniques such as linear programming, which require specific mathematical structures for the problem formulation.

Recent overviews of AI also provide a more pragmatic understanding of the techniques for problem solving. For Russell and Norvig (2021), the standard model of AI consists of building rational computational agents that can solve problems but do not intend to model the behavior of humans solving problems. These self-contained problem-solving agents are an abstraction for constructs that operates autonomously and adapts itself to changes in a given task environment. The basic structure of an AI agent has sensors to perceive the environment and actuators to execute actions (Figure 109). An internal agent program mediates these two components, defining a map between percepts and actions.

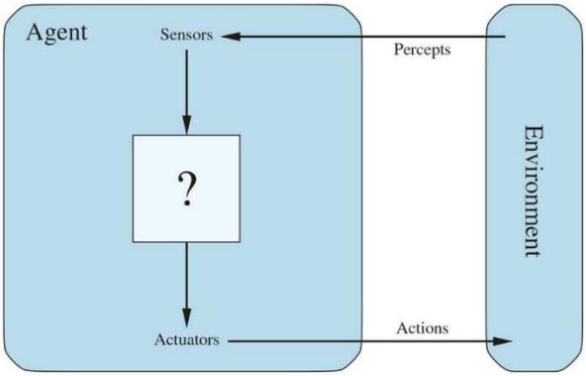


Figure 109 Basic structure of an AI agent (From Russell and Norvig 2021, 37). Reproduced by permission of Pearson Education, Inc.

For the authors, the notion of rationality in AI is generally based on consequentialism, where an agent is evaluated by the consequences of its behavior (Russell and Norvig 2021, 39–40). AI agents should act to achieve the best outcome or expected outcome in the task environment with the respect to its goals, which is evaluated using a performance measure. More specifically, the authors state that, "for each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has" (Russell and Norvig 2021, 40). They present four basic classes of agents[17] (Figure 110 and Table 13), characterized by an increasing of complexity of its internal program and components.

Table 13 Structure of standard agents in artificial intelligence - based on Russell and Norvig 2021, 47–54.

	Reflex agents	**Model-based reflex agents**	**Goal-based Agents**	**Utility-based agents**
Sensor model	Y	Y	Y	Y
Actuator	Y	Y	Y	Y
Transition model	N	Y	Y	N/Y
Performance measure	-	-	Goal	Utility function
Program	Condition-action rules that responds to percepts	Condition-action rules	Search or planning for best action	Maximize the utility or expected utility

[17] The authors also present a fifth class, called Learning agents (Russell and Norvig 2021, 56–58), which is related to the branch of AI called Machine Learning and will be addressed in the chapters ahead.

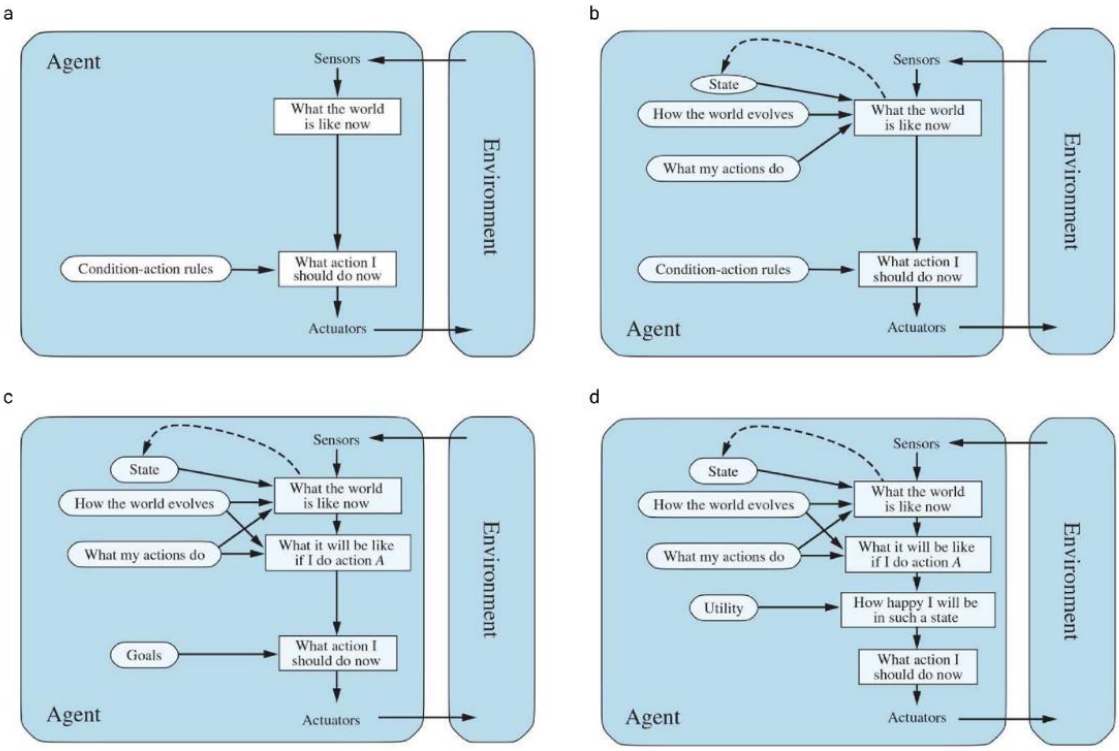


Figure 110 Diagrams with the structure and of four basic AI agents (From Russell and Norvig 2021, 50, 52, 54, 55): a) reflex agent; b) model-based reflex agent; c) goal-based agent; d)utility-based agent. See description in the text below. Reproduced by permission of Pearson Education, Inc.

a) **Reflex agents** rely on rules conditioned on the sensor observation of the environment to select actions (condition-action rules)

b) **Model-based reflex agents** combine the sensor observations of the environment with an internal transition model of the world – i.e., an understanding of how the world works and how its actions affect it – to keep track of the states of the world

c) **Goal-based agents** use algorithms that combine the goal information combined with the transition model of the world to search for the best actions.

d) **Utility-based agents** use the sensors, an internal function that measures its performance, and potentially a transition model, to select the best actions

Russell and Norvig's textbook provide a vast repertoire of techniques. Heuristic search algorithms the foundation of the goal-based agents. Optimization algorithms are also part of the goal-based agents for the cases where only the configuration of the final solution and not the path to this solution matters. Besides, optimization algorithms are also included in some cases of the utility-based agents.

7.2.1. Search as a model for architectural design

Classic AI methods rely on an assumption of objective rationality, or at least, of approximated rationality, to provide anticipated knowledge about the problem to be solved, the consequences

of the decision making, and ultimately all the elements required for decidability. Under the notion of rational decision making, problems can be treated as well-defined – i.e., they can be formulated to a point in which it can be written down in a paper or program and the solutions are findable without the need of additional information (Rittel 1972, 391–93; Rittel and Webber 1973, 160).

When Hebert Simon provides a list of characteristics for well-defined problems, he describes the necessary conditions required to translate a problem to a feasible heuristic search (1973, 183–84). However, he also understands that this objective rationality for decision making is an idealization because real problems are not well- but ill-defined. So, a problem-solver should decompose a real-problem in well-structured parts before processing them (H. A. Simon 1973).

This ambition of treating a search as a general model for understanding how humans solve problem was also influential in architectural design. In the "Sciences of the Artificial" (1996b – originally published in 1968), Simon described a body of knowledge for disciplines that are concerned with the synthesis, such as engineering or architecture. Simon considered that architectural problems, such as the spatial design of a house, as part of the residual class of ill-defined or ill-structured problems (H. A. Simon 1973). This class is defined in opposition to well-defined problems, so it refers to problems that have "little structure in terms of their operational parameters: goals, legal operations, alternatives to be considered, and evaluation functions" (Ö. Akin 1986, 5). These problems can only be partially formulated, so "both the ends and the means of the solution are unknown at the outset of the problem-solving exercise" (Rowe 1987, 40).

Simon interpreted the act of design based the idea of a limited or bounded rationality. According to him, designers are trained to break ill-defined design problems down into pieces and provide structure to them, which allowed them solve the resulting well-defined sub-problems and coordinate the results to form the whole (H. A. Simon 1973). In this context, designers are considered as specific symbolic machine and the logic of design is a search parameterized by previous steps of the process, by personal preferences and heuristics defined by the designer (H. A. Simon 1975, 297, 300–301). With no access to ultimate metrics to generate optimal solutions, designers search in an unbounded space of design alternatives based on personal aspirations and style. Style here refers both to the custom characteristics of the resulting design and of the search that generated it, such as the order of the design process, the re-use of previous solutions, and the imposition of autonomous constraints (H. A. Simon 1975, 297–301). Overall, design is not considered a drawing-centered problem but essentially a multi-step decision problem. It is a search that is customized by the designer's style and forms of representation and that results in solutions that are good enough or "satisficing" (a merge of satisfying and sufficing) rather than optimal (H. A. Simon 1996, 119–21).

The formalization of the design process in the frame of information-processing was very influential over the last decades in design cognition and computational design. Some of the researchers even pushed them further to incorporate ill-defined aspects of design.

Akin adopted the framework of information-processing and the concept of search as tools to

analyze specific characteristics of architectural design (1986). For example, he considers that due to the nature and complexity of design, the search for satisficing design solutions in architecture usually relies on local heuristic exploration in limited representation domains. Akin *et al* (O. Akin, Dave, and Pithavadian 1987) also considers that there are also other components in the design, such as problem structuring, which establishes an initial setup for a problem to be solved by further solving sub-phases, such as generation, testing, and selection. In this sense, problem structuring and problem solving are co-evolutionary components of design.

Woodbury and Burrow (2003; 2006) adopt the model of designers as information processing systems to model the search for satisficing design solutions with computational tools. They define design space exploration as

> "(…) the idea that computers can be used to help designers by representing many designs, arraying them in a network structure (the space part) and by assisting designers to make new designs and move amongst previously discovered designs in the network (the exploration part)." (2003, 519)

In this sense, exploration comprehends the search in the design space mediated by human-computer interaction mechanisms.

Gero (1994) considers that when the different variables and their relationships of the design process are well-defined, a search then "determines feasible, satisficing or, in appropriate circumstances, optimal values for structure variables, which produce desired behavior" (1994, 317). However, a large part of non-routine design processes involves other elements, such as exploration or learning. Learning is associated with reviewing and restructuring the knowledge used in the design (1994, 315). Exploration is related to reconfiguring of the problem variables (1994, 318–19). Therefore, in contrast Woodbury and Burrow's definition, which relies on computational mechanisms to search in the design space, Gero associates exploration with creating or modifying the different design state spaces themselves.

7.2.2. Search as a model for generative models

One of the first reviews of generative models for spatial synthesis (W. Mitchell 1977) was largely influenced by ideas from Symbolic AI and in particular by the logic of search proposed by Simon. For example, for Mitchell, the definition of a computational generative model is basically a search for design solutions (Figure 111). The generative model operates with discrete steps, so the design space accessed in the computational process can be codified in a network. In this context, the network structure of a problem described by Newell and Simon becomes Mitchell's state-action graph. The edges represent allowable design operations of the generative model that modify a partial solution, which is represented as moving from a node of the graph to another (W. Mitchell 1977, 43–46).

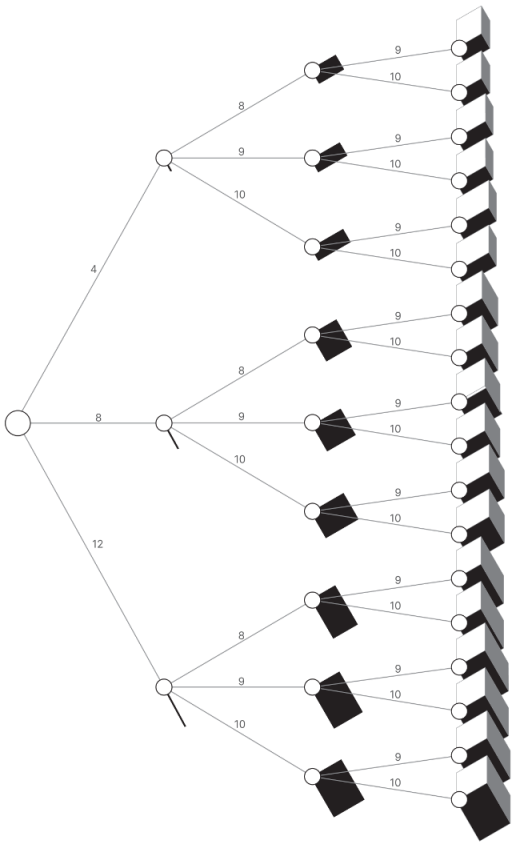


Figure 111 State-action graph. Redrawn by Author, based on (W. Mitchell 1977, 45).

The nodes are the set U of all potential solution states of the problem in the state-action graph. There are two subsets of nodes: a subset G of all goal solutions that meet the desired criteria and the subset S of all solutions generated for consideration by the generative model during the design process. With design elements formalized as nodes, edges, and goals upfront, algorithms can be used to search for design alternatives that are in the goal set – i.e., solutions in the intersection of S and G.

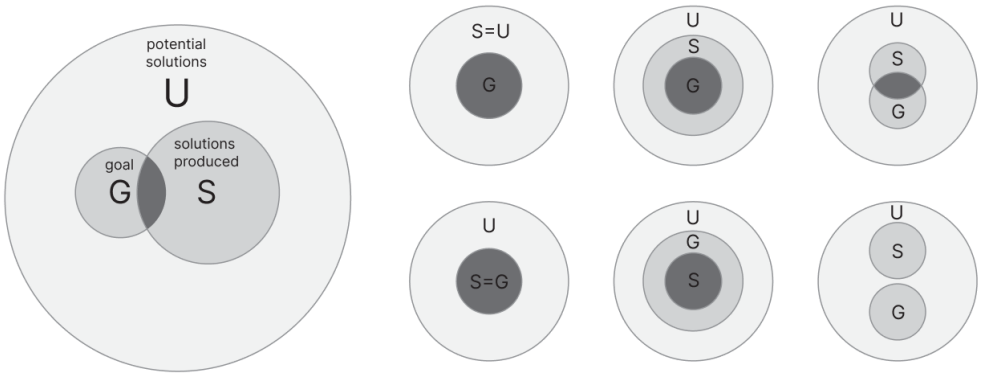


Figure 112 Left: Venn diagram showing the relations between sets U, G, and S in a generative system; Right: six different relationships between solution generation and solution evaluation procedures. The darker shaded areas indicate satisfactory solutions that are actually produced. Redrawn by author, based on (W. Mitchell 1977, 49, 57)

Nevertheless, due to the situational and subjective nature of architectural creativity and to the fact that there is usually an extremely large space of possible architectural configurations (Ö. Akin and Sen 1996, 422), this formalization is not a trivial process. In practice, many of the parameters of an architectural configuration, such as the incorporation of implicit design factors, or the variety of spatial shapes, had to be simplified or downplayed to create functional computational models. So, while Mitchell's definition of a generative models was heavily influenced by Simon's general theory of search, he acknowledged that the AI techniques of the period privileged well-defined aspects of design, which may not be feasible in practice (1977, 60–62).

7.2.3. Generative models for spatial synthesis

Fields such as symbolic artificial intelligence (AI) and operation research (OR) provided methods to automate the generation of architectural configurations with computers. Mitchell's overview of the spatial synthesis procedures comprehends not only their respective computational methods (search and optimization) but a larger spectrum of solutions organized by their strength – according to Newell's (1967, 12–14) understanding of algorithm power. While weak synthesis procedures, such as randomly generating solutions, require none or little knowledge about the problem, strong procedures produce results very quickly, but they depend on the expression of mathematical descriptions of objectives and constraints. His topics range from the weakest to the strongest methods for solution synthesis:

1. generate-and-test procedures: it is based on the systematic or random sampling and testing of the possible solutions (brute force generate-and-test and random generate-and-test)
2. improvement procedures: start with a solution and use strategies to improve it (simple improvement hill-climbing, greatest improvement procedures)
3. heuristic search procedures: a search/sequential decision process on a tree and complementary strategies to break the problem (BFS, DFS, heuristic information, branch selection, backtracking and planning routines).
4. nonlinear and linear programming: mathematical technique to maximize or minimize an objective function of several variables.
5. analytical procedures: use of analytical solution on several equations representing requirements.

One year after Mitchell's comprehensive review of spatial synthesis procedures, Henrion published "Automatic Space-planning: A Postmortem" (1978), which combines a brief review and an analysis of the spatial synthesis methods of the time. In the review, Henrion provides a tree-like classification of the automatic space-planners. This first two branches separate the two dominant classes: optimizers and satisficers. Optimizing systems are concerned with minimizing the cost of a solution (usually the circulation cost), while the satisficing systems focus on generating a solution that does not violate the existing constraints, such as adjacency or

dimensions. Henrion's classification divides these two classes in variations and descendants that combine both.

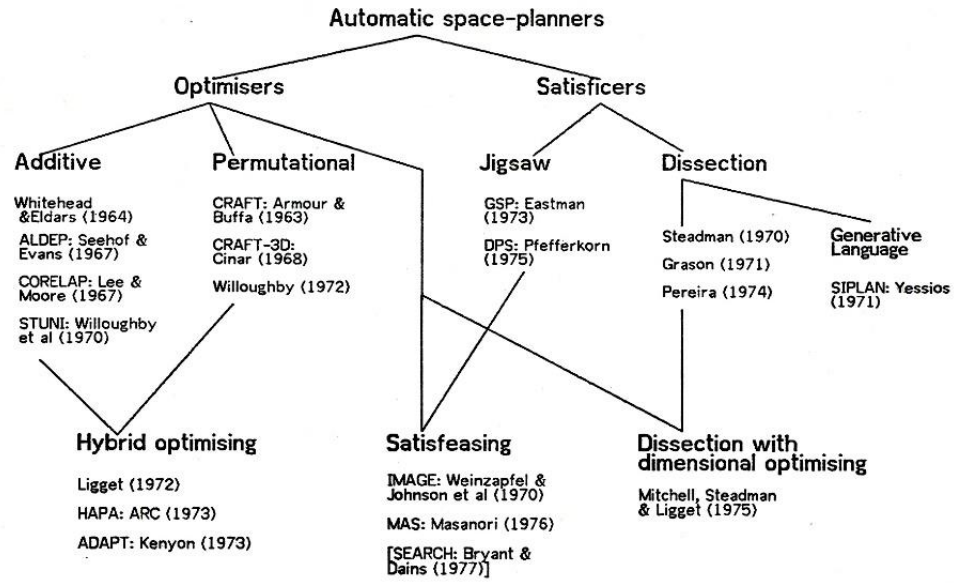


Figure 113 a classification of selected automatic space-planners (Henrion 1978, 177). Reproduced with permission of Elsevier Science &Technology Journals, from Automatic Space-Planning: A Postmortem? 1978; permission conveyed through Copyright Clearance Center,Inc.

The resulting subcategories are:

1. Additive [optimizer]: the spatial elements are added sequentially in an initial empty space with a heuristic that minimizes the costs based on the spaces already allocated.
2. Permutational [optimizer]: the spatial elements are added assigned initial locations and are interchanged until no further improvement is possible.
3. Hybrid optimizing [optimizer]: optimization strategies that use an additive heuristic with posterior refinement by permutation
4. Jigsaw [satisficer]: strategy to assign non-overlapping spatial units to existing locations, which can include heuristic to choose the space or location based in the shape constraints and resulting constraining of the solution.
5. Dissection [satisficer]: elements are formed by subsequent operations – such as division or addition – of an initial form forming a catalogue of solutions that follow certain constraints (such as adjacency).
6. Satisfeasing [hybrid]: strategy that recognizes that not all constraints in design can be satisfied, so it minimizes the degree of misfit by satisfying as many constraints as possible.
7. Dissection with dimensional optimizing [hybrid]: combination of the topological dissection of a floorplan with optimization of its dimensions.

In their book on design and optimization, Radford and Gero (1988) focused on the application of optimization techniques, such as calculus, linear programming and nonlinear programming, and dynamic programming, to engineering and architectural design. For them, optimization a specific area of research, distinguished from the concept of generation. They distinguished three types of symbolic models for architectural design:

1. simulation, where the computer predicts the performance of design decisions.
2. generation, where a set or decision rules is applied recursively.
3. optimization, where the computer prescribes a set of decisions to achieve a specified goal.

Robin Liggett analyzed the formulation of a space planning problem as specific type of combinatorial optimization problem called the quadratic assignment problem. She provided two main classes of solution techniques (1980; 1985):

1. constructive procedures: use rules to build a solution in a n-stage decision process to assign activities to a certain location, which results in decision processes in the form of tree searches.
2. improvement procedures: start with a solution and uses hill-climbing strategies to improve it iteratively, such as exchanging locations between pairs of activities.
3. combination/hybrid approaches: combination of constructive and iterative procedures – more specifically, Liggett proposed the use of constructive procedures for the initial placement with refinement by improvement.

More than a decade later, she extended her taxonomy with two categories related to metaheuristic algorithms (Liggett 2000):

4. simulated annealing: optimization heuristic inspired in the process of annealing in metallurgy and uses a decreasing temperature to reduce the exploration of worse/random solutions over time
5. genetic algorithms: class of optimization algorithms inspired in evolutionary biology that uses genetic operations, such as crossover and mutation, to explore populations of solutions.

Strategies in the weak and strong extremes of Mitchell's classification are absent (generate-and-test, linear programming, or analytical procedures). Constructive procedures incorporate the same idea as Mitchell's heuristic search procedure category. Moreover, Liggett expanded the iterative procedures with the new generation of metaheuristics for optimization that became widespread in the new spatial synthesis research of the 1990s.

Table 14 comparing categories of solution procedures in CAAD. Parenthesis are used to indicate child category or examples inside this that might justify its placement. Square brackets are used to indicate the parent category to which it belongs.

Clusters	Simon [1968] 1996	Mitchell 1977	Henrion 1978	Radford and Gero 1988	Liggett 2000
Search/Satisficers	Heuristic search	Heuristic search Exhaustive generate and Test [generate and test] Random generate and test [generate and test][18]	Jigsaw [satisficer] Dissection [satisficer]	Generation	Constructive procedures
Optimizers	Linear programming Dynamic programming Geometric programming	Random generate and test [generate and test][18] Improvement Nonlinear and linear programming	Additive [optimisers] Permutational [optimisers] Hybrid optimizing	Linear programming [optimization] Nonlinear programming [optimization] Dynamic programming [optimization]	Improvement procedures (hill climbing) Simulated annealing Genetic algorithm
Hybrid			"Satisfeasing" [optimisers][jigsaw] Dissection with dimensional optimizing [optimisers][dissection]		Hybrid approach
Other		Analytical procedures		Simulation	

[18] Generate and test is a general procedure in which the generation of the solution and the test process are completely independent – i.e., the solution is sampled and then evaluated (Newell and Simon 1972, 98). It can be used to look both for optimal and for satisficing solutions. For example, it can be used to generate all the elements of a search tree that satisfy a criterion, or it can sample solutions that perform better according to an objective function. Therefore, I opted for placing its random variant in both categories.

7.3. Appendix C: Beyond objective rationality in generative design?

7.3.1. The divide between black-box and glass-box design models

As discussed earlier, the research on spatial synthesis and generative models were largely informed by problem solving. Problem-solving relies on the notion of objective rationality or approximated rationality, which requires guarantees for decidability by anticipating knowledge about the problem at hand. It supports a systematic view of design and a glass box model is based on "a planned sequence of analytical, synthetic and evaluative steps and cycles" (Jones 1992, 50) and the use of generative techniques such as optimization or search.

However, generative models for spatial synthesis have been barely adopted in architectural education or professional practice, which also raised criticisms about their underlying assumptions. In this sense, Henrion (1978) suggested that the first generation of generative methods for spatial synthesis were not only technically inadequate but were also incompatible with architectural design methodologies, resulting in little practical application in real-world problems in architecture. For a future "resurrection of space-planning as a viable domain of AI systems"(1978, 182), he evoked the necessity of incorporating aspects of practice such as custom schemata for design, procedurally embedded design knowledge and the interaction between problem-solving and problem definition.

Decades later, Lobos and Donath (2010) provided a similar analysis of spatial synthesis techniques from the 2000s. They consider that these techniques rely on approaches from other fields, so they lack knowledge about problem and method definition that are specific to architecture and result in solutions that are not valuable (2010, 158–59). Therefore, they consider that the dominant engineering approach to spatial synthesis with an abstract definition of the architectural problem is the reason why space planning has not been successful in architectural design.

In contrast to the glass-box model, the black box model of design (Jones 1992, 46–47) depicts the designer as a magician that controls her actions unconsciously and respond to problems based on pattern recognition and experience. This design model comprehends conventional design methods where the designers heavily rely on exploring the affordances of design media and of personal and disciplinary constructs to guide design ideation. This provides the freedom to explore solutions based on design intuition, socially meaningful precedents, or a shared visual language rather than in empirical facts outside of the architectural discipline itself.

Despite being the dominant model in architectural education, the black-box model, with its personal bias, was also considered an element of social distinction and even a liability for the profession[19]. In the case of proponents of systematic design and computational methods, this

[19] For example, Stevens (2002) considers that by stablishing subsets of the design space that match a certain agenda and defining meaningful operations to explore these spaces, designers are operating in an economy of symbolic value to promote their social distinction. Banham's criticism of the professional and pedagogical practices in architecture (1990) also captures some sociological nuances. Particularly, he depicts the modus operandi of

critique has emphasized the cognitive limits of design methods centered on intuition. A pioneer of systematic methods of design, Alexander considers the designers' intuition and their reliance on a formal order as a sign of inability to solve complex problems (1964, 1). In the early days of spatial synthesis, Mitchell recognized that AI methods of the time could not address ill-defined aspects of design (1977, 60–62). Later, he would resonate Alexander's position and contrapose the formal complexity of generative design with the limited cognitive capacity of the conventional design process (W. Mitchell 1998). More recently, Kolarevic (2005) and Oxman and Oxman (2014) also support this antagonism by arguing that emerging generative strategies and their potential to generate complex forms are replacing the conventional design ideation and producing results that are more complex.

However, to design future generative models for spatial synthesis it is necessary not only to understand the affordances of new computational technologies in artificial intelligence and related fields but also to learn with this divide between glass-box and black-box, or equivalently, the divide between problem-solving and architectural creativity.

7.3.2. From an objective to a pragmatic rationality in design

The limits of objective rationality as an assumption of problem-solving or as a worldview have been largely discussed in AI and adjacent fields. For example, Winograd and Flores (1986) advocated for a notion of artificial intelligence beyond the rationalism and cognitivism of symbolic AI. They understand that computational agents in the real world should be able to operate inside a network of human relations and therefore, comprehend other dimensions of design, such as phenomenology, biology, and linguistic. Suchman (2006, 24–28) considers that the reliance of symbolic AI on planning as a cognitive model to determine purposeful action of a rational actor is limiting. She argues that real plans are never as systematic as the planning models from AI because they necessarily require *ad hoc* responses to the action of others and contingencies of a particular situation. According to Andrew Pickering (2010), the field of Cybernetics also challenges the rational view that the world is a knowable and representable place, and that the temporal process is not only graspable, but also causal and calculable. Instead, Cybernetics suggests an ontology of performance, where the world is a system perceived by the interactions between its constituent elements, so actions happen as performances of the components of the system over time.

In the case of architecture, one of the fields that have provided similar emphasis on situated action and cognition is design theory. Over the last decades, design theory has been moving further away from the objective model of problem solving to focus on other aspects of design, such as the rationality of existing practices, processes of argumentation, and aspects of communication and collaboration in design (Protzen and Harris 2010, 4–7). Particularly, the influence of Rittel's understanding of design problems as wicked problems is notorious in the current state of the

---

architecture as an exercise of an arcane, privileged, and unspoken aesthetic code, inculcated in the studios and glorified in the draftsmanship of architectural drawing.

design theories and methods (Protzen and Harris 2010, 5–6). Horst Rittel (1972) criticized many of the assumptions of the rational methods in design, mainly because they rely on a paradox of anticipating the consequences of contemplated action. In contrast to well-defined problems addressed by engineers or scientists, designers and planners address ill-structured problems, which he refers to as wicked problems, with the following characteristics (Rittel 1972, 392–93):

1. No definite formulation exists
2. Every formulation corresponds to a statement of the solution
3. There is no stopping rule
4. Instead of the Correct/false category, Good/bad should be applied
5. There is no exhaustive, enumerable list of permissible operations
6. There are many explanations for the same discrepancy
7. Every wicked problem can be considered a symptom of another problem
8. There is neither an immediate nor an ultimate test for the problem
9. It is one-shot operation (each problem requires a different operation)
10. It is unique
11. The wicked problem-solver has no right to be wrong

In a similar way, Stanford Anderson (1966) explicitly challenged the problem-solving approach to architectural design. He understands that architectural problems are intrinsically based on inconsistent representational domain of complex problems. Consequentially, objective design requirements or heuristics of synthesis cannot be completely determined *a priori*. Anderson contraposes the model of problem-solving with a design model named problem-worrying. Problem-worrying is based on the interaction of the designer with partial solutions in order to reformulate the design problem itself. According to Anderson:

> *I would see the architect's approach as a sequence of activities encompassing at least the following stages: generalized understanding of the problem; various formal proposals; study of the implications of the proposals; successive reformulation of the problem and proposals; and the final selection of a form for its appropriateness to the reformulated problem. In this case, one must judge not only the fit, but also how the problem has changed. And one must judge the fit not in terms of frictionlessness, but in terms of whether the friction is suited to the new problem formulation. Does the whole – reformulated problem and form – resisted criticism? (Anderson 1966, 10)*

Problem-worrying implies that the design problem cannot be fully structured in advance (as in well-defined problems) and that the design path cannot be modelled as a linear process based on anticipated consequences, because the designer learns new information about the problem and restructures it during the design process. Instead of anticipation and linearity, it places the

feedback between the designer, the design problem and solution, mediated by criticism as the logic of architectural design. Particularly, design feedback is situated as a tool for an open-ended reformulation of the problem and of partial solutions.

A strong critic of the rational methods of design, the pedagogue Donald Schön proposed an interpretation of architectural design as an activity founded on reflection in action (Schön 1983). In Schön's reflective model, there is no clear distinction between design analysis and synthesis as the process does not develop from a well-defined problem to a solution. Instead, the designer analyzes the problem while she is solving it, using fluid design representations. Schön's model of design situates cognitive processes such as reflection, perception, and action of the designer occur in cyclic process.

Nigel Cross understands that design deals with ill-defined problems and that design thinking is a constructive and synergetic activity. In this view, the design process is a co-evolution or a search for correspondence in the pair problem-solution (2006, 114). For Cross, a

> *"generative concept is not simply found in the problem as given, but largely created by the designer; it is not a matter of recognizing a pre-existing pattern in the data, but of creating a pattern that re-formulates the problem and suggests directions towards a solution" (Cross 2011, 74).*

Corona-Martinez (2003) interprets architectural design as continuous conversation between two types of representation: an internal representation (mental image) and some form of external representation, such as sketches or models. Therefore, the design is an iterative process of refinement that relies on the interaction between these two representations. Similarly, Christian Gänshirt (2007, 78–79) considers design as a cycle based on the feedback between perception, thought, and action mediated by visual and verbal design tools (algorithms, criticism, sketching, gesture, models, etc.).

Bryan Lawson also follows the same line of thought. In a chapter called "Route maps of the design process" (2005, 31–50), Lawson counters the discrete and directional mapping of design process proposed in the 1960s and 1970s with a three-dimensional map that contains both the idea of design cycle and of problem restructuring. The cycle is composed of evaluation, analysis and synthesis procedures, the restructuring is defined as a pair problem-solution aligned in axis of the cycle. This pair represents the negotiation between the design problem and solution.

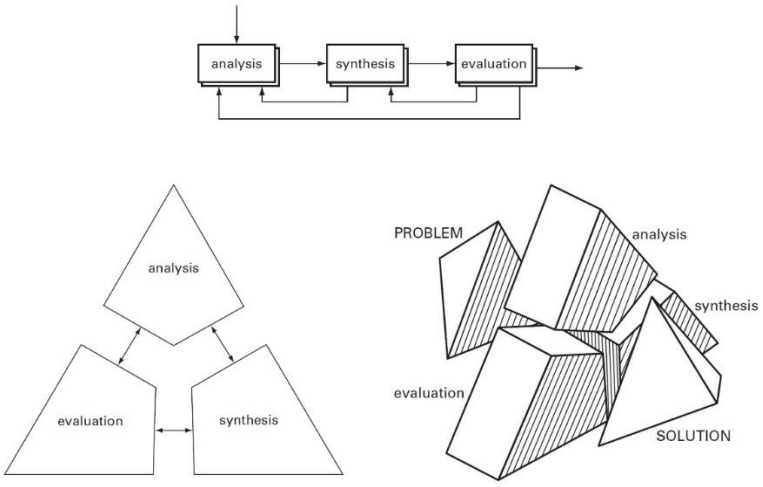


Figure 114 Route map of the design process (Lawson 2005, 38, 40, 49). Copyright ©2005 From How designers think: the design process demystified by Bryan Lawson. Reproduced by permission of Taylor and Francis Group, LLC, a division of Informa plc.

Ranulph Glanville (2009) exposed the connection between Schön's model of design and the cybernetic notion of conversation theory (CT, as proposed by the cyberneticist Gordon Pask). While Schön's design model situates feedback as the basic element to understand human-centered design, CT suggests a knowing model based on the interaction between agents (for example, human or AI agent). This connection is fundamental as it exposes a divide between a closed-world design, where a problem can be formalized and solved linearly, and an open-world design where acting on and understanding the design problem relies on feedback loops. Herr and Fischer (2019) also considers the cybernetic notion of conversation as a model for architectural creativity. They understand that the dialogical exchanges between different agents in the design are the essential elements to support novel insights, surprise, delight, and other qualities associated with open-ended design exploration.

In contrast to other fields where empirical facts and objective factors are dominant, in architecture they are necessary but not sufficient for determining good spatial configurations. Architectural creativity heavily relies on individual ideas or on design principles shared by other architects in publications to guide the design process without the need of anticipation or formal description. Architects work with shared precedents, aesthetic codes, visual language, design principles, methods, and rules. For example, Rowe (1987; 2017) argues that designers use heuristics reasoning strategies such as anthropometric analogies, literal analogies, environmental relations, typologies, and formal language to establish autonomous constraints in the design process. Cross considers that some kind of formal ordering principle has to be imposed to the design generator to make a solution possible (1982; 2001). Lawson not only defines a model for design problems with a taxonomy of the source of the different design constraints with a dimension of internal and external constraints but also emphasizes the importance of guiding principles to influence and set the mental context for each design process (2005, 106, 179).

7.3.3. Generative models beyond spatial synthesis

In the last decades, architects gained access to personal computers and to CAD, BIM, geometric modeling, and animation software that enabled the manipulation of geometry and design data. Programming was also rediscovered by designers with scripting languages and, specially, with the rise of graph-based parametric editors integrated in CAD systems in the 2000s. The access to computational hardware, software and programming marked the transition of computation from the research lab to the broader arena of design education and practice.

Generative models were part of large transformations in the architectural design field that have been predominantly interpreted under the notions of the term "digital" (Goodhouse 2013)[20]. Terms such as digital architecture, digital age, digital turn, or digital revolution have been widely adopted in design discourses and in the historiography to describe the disruptive technical and cultural condition. For example, the first digital turn in architecture (1992-2012, Carpo 2013) comprehended a paradigm shift in design education and practice, marked by the emergence of computational design models and a new conceptual vocabulary (Oxman 2006; 2008). In this digital turn, practical architects intertwined digital methods and aesthetic for the digital design (Rosa 2003; Zellner 1999; Kolarevic 2003). For example, the operations available in animation software, NURBS and parametric modeling were assimilated as a blend of method and design content to define novel forms of digital architecture (Lynn 1999; Schumacher 2008).

With access to tools to externalize design instructions and decisions, designers integrated generative models to changing trends and a varied number of topics, such as design aesthetics, mathematical forms, bio-inspiration, and building tectonics. Associated with digital fabrication, new advancements in architectural geometry, and simulation, the use of generative models supported a new repertoire of non-standard forms that were developed in research pavilions and, eventually, applied to the evaluation and synthesis of building surfaces, components, and envelope.

The review of the digital in architecture or the vast landscape of generative modeling is beyond the scope of this research. I will briefly identify the different trends and concepts associated with generative modeling in this period. While the concepts related to classic generative models will be analyzed in this chapter, other concepts will be discussed in the following chapters. For that, I organized many of the taxonomies related to generative models in this expanded setting. according to the following clusters: search/satisficers, rule-based modeling, optimization, parametric modeling, simulation, agent-based models, and others. As a result, I have in

[20] It is important to distinguish the terms computational and digital. Computational strictly refers to the use algorithms or models to perform operations on symbolic or numerical models. For example, computational design methods use these algorithms and models to represent, analyze and synthesize design alternatives. In contrast, the term digital has been used fluidly to describe a certain cultural condition or state of being related to the advent of certain information technologies, such as virtual reality or world-wide web.

Table 15,

1. the pedagogical categories for the teaching of generative systems proposed by Fischer and Herr (2001);
2. the concepts identified by Kolarevic (2003) to characterize the digital morphogenesis in the digital age;
3. the digital models of design (beyond conventional CAD systems) described by Oxman (2006)
4. the provisional taxonomy of generative systems created by El-Khaldi (2007, 112)
5. the main approaches to computer-based form generation in academic research and practice presented by Gorbman, Yezioro, and Capeluto (2009)
6. the categories of generative design technique presented by Vishal and Gu (2012) for an integrated framework;
7. the four significative techniques identified and used in workshop on generative systems by Henrique et al. (2019).

With the table, it is possible to identify patterns that contrast with the early generative models, which were developed for the synthesis of architectural layout with methods based on techniques from operations research (optimization) and symbolic artificial intelligence (heuristic search).

Rule-based models developed in the early days of generative models, such as shape grammars (1971), fractals (1975), and L-systems (1968), came to the forefront of the taxonomies. Usually, these models are associated with weak multi-step procedures, such as a deterministic or random derivations based on the rule set, to explore spatial patterns without much information from the task environment. Therefore, more systematic strategies to explore the design spaces of those rule-based models, such as search algorithms, became less common.

In contrast, optimization became a dominant technique for generative models with incorporation of black-box methods, such as metaheuristics and model-based algorithms. This tendency is also related to the incorporation of parametric modeling and simulations as part of the design toolkit. The former enables the use of modeling parameters as decision variables for optimization and the later provides access to objective functions related to building performance.

Table 15 Concepts associated with generative modeling (2001-2019. Parenthesis are used to indicate child category or examples inside this that might justify its placement. Square brackets are used to indicate the parent category to which it belongs.

Clusters	Fischer and Herr 2001	Kolarevic 2003	Oxman 2006	El-Khaldi 2007	Gorbman, Yezioro, and Capeluto 2009	Vishal and Gu 2012	Henrique et al. 2019
Search / Satisficers					Complete Enumeration		
Rule-based modeling	Algorithmic generation and growth (e.g., fractals, re-writing rules) Generative grammars (e.g., L-systems and shape-grammars)	Genetics (L-systems)	Generative: Grammatical transformative design model [Generative]	Shape Grammars L-Systems Fractals	Shape grammars and formal rule-based generation Cellular automata (L-systems) Case-based reasoning / expert systems	Shape grammars L-systems	Shape grammars L-systems
Optimization and performance	Algorithmic (re-) production (e.g., genetic algorithms, selective procedures).	Genetics (genetic algorithm) Performative Architecture	Evolutionary design model [Generative] Performance-based formation model [Performance] Performance-based generation model [Performance]		Space allocation Evolutionary methods Performance driven form generation (optimization)	Genetic algorithms	Genetic algorithms
Parametric modeling	Algorithmic generation and growth (parametric design)	Parametrics	Associative design formation model [Formation]	Parametric	Geometric constraint-based form generation		
Simulation		Metamorphosis Dynamics and fields of forces	Motion-based formation model [Formation]		Shape grammars and formal rule-based generation (field of forces)		
Agent-based modeling	Emergent systems, self-organization (e.g., cellular automata and swarm modelling)				Cellular automata	Cellular automata Swarm intelligence	Cellular automata
Others	Algorithmic generation and growth (data mapping)	Topology Non-Euclidean geometry NURBS Datascapes	Topological formation model [Formation]	Algorithm			

7.4. Appendix D: Agent-Based Computing

7.4.1. Artificial Intelligence and Multiagent Systems

In AI, agents are computational constructs that operate autonomously to solve a problem, which is described as a task environment (Russell and Norvig 2021, 42). Task environments have many properties that define the appropriate type of agent and techniques to solve them. The adoption of multiple agents is one of these properties. Due to the multiagent nature of the task or to the benefits of decoupling a planning problem into subproblems, multiagent task environments require multiple computational agents that can compete and/or collaborate (Russell and Norvig 2021, 43–45). Examples range from board games and card games to transportation and logistics, etc. While in some cases the problem itself is structured with multiple agents, in others the flexibility and autonomy of multiple agents are advantageous for the solving it.

Agents in a multiagent environment still have the same basic structure of a regular AI agent, which includes sensors, actuators, and potentially different internal agent program to map percepts to actions. Additionally, they can share information from their internal state through the shared environment and engage in different forms of collective activities.

One of the multidisciplinary fields intimately related to multiagent task environments and social agency in AI is multiagent systems (MAS), which comprehends elements from computer science, economics, and operations research. Its methods include algorithms for distributed constraint-satisfaction problems, distributed optimization, multiagent learning, and topics such as game theory, communication, and social choice (Shoham and Leyton-Brown 2009).

In MAS, agents are defined both by their capacity for autonomous decision-making and for interacting with other agents and engaging in analogues of social activities. According to (Wooldridge 2009, 4–5)

> *The idea of a multiagent system is very simple. An agent is a computer system that is capable of independent action on behalf of its user or owner. In other words, an agent can figure out for itself what it needs to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment. A multiagent system is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure. In the most general case, the agents in a multiagent system will be representing or acting on behalf of users or owners with very different goals and motivations. In order to successfully interact, these agents will thus require the ability to cooperate, coordinate, and negotiate with each other, in much the same way that we cooperate, coordinate, and negotiate with other people in our everyday lives.*

Particularly, Wooldridge (2009, 26–27) suggests the following list of capabilities to characterize agent intelligences:

> *Reactivity Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.*
>
> *Proactiveness Intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives.*

> *Social ability Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.*

Therefore, intelligence is defined within the paradigm of rationality dominant in AI, so an agent is considered an intelligent if it exhibits goal-oriented behavior. However, Wooldridge's list also emphasizes the importance of social ability and potential interaction with humans as part of problem solving.

Multiagent systems are particularly useful in cases where the task environment is structured with multiple agents or when central planning is not feasible computationally. Multiagent systems support agent architectures based on simple and reactive behaviors, which allow agents to explore intelligent behavior as a product of the interactions that the agent maintain in an environment. These agents react to changes in their local perception of the environment, so they can support robust and flexible strategies that can couple with real-time changes in the environment. Additionally, if a long-term strategies are required, it is also possible to create hybrid architecture that combines reactive and planning agents (Wooldridge 2009, 91–93).

7.4.2. Complex systems and agent-based modeling

In many fields, computation is used to model the behavior of real-world phenomena, such as fluid flow, pattern formation, reaction diffusion of chemicals, physics, markets, and even urban dynamics. In some cases, these phenomena can be described analytically and predicted with equations. In the cases of complex systems, the structure of the interaction between components changes over time because of the dynamics of the system. This requires replacing the analytical by an algorithmic description of the system that can capture time-varying interactions (Thurner, Hanel, and Klimek 2018, 7).

Agent-based modeling (ABM) is one of the methodologies used in the field of complex systems theory. It is based on the idea that the entities have a certain degree of autonomy and can be described algorithmically to capture the changing dynamics of the system. The presence of multiple agents that can take local decisions based on the local perception and interact in a shared environment is key to capture many aspects of complex systems. According to Wilensky and Rand (2015, 6),

> *Complex systems theory develops principles and tools for making sense of the world's complexity and defines complex systems as systems that are composed of multiple individual elements that interact with each other yet whose aggregate properties or behavior is not predictable from the elements themselves. Through the interaction of the multiple distributed elements an "emergent phenomenon" arises. The phenomenon of* **emergence** *is characteristic of complex systems.*

De Wolf and Holvoet (2005) overviewed the origins of the idea of emergence in complex systems from the dynamical systems to fields such as cybernetics, physics, biology, artificial intelligence, and artificial life. Emergence is described as the arising of novel and coherent patterns on the macro-level of the system from the interaction of its constituent elements at the micro-level, but that cannot be traced back to the individual parts at the latter (H. A. Simon 1996, 169–72; Holland 1998, 115–24; De Wolf and Holvoet 2005, 3; Wilensky and Rand 2015, 6). Emergence does not

rely on a central coordination or designer but only on this bottom-up relationship between the micro and macro levels of the system, so the system is typically robust to perturbations and flexible to react to different conditions of the environment.

For Holland, the notion of emergence also comprehends characteristics mentioned above, such as the bottom-up relationship between the micro and macro levels of the system, the existence of multiple components that interact, and the generation of a flux of recognizable patterns that can change over time resulting in perpetual novelty (Holland 1998, 3,4,115-122).

Additionally, Holland emphasizes the importance of the combination of procedures that freely generate possibilities coupled with constraints that limit these possibilities for emergence. This enables him to expand the umbrella of emergent systems and include not only phenomena typically associated with complex systems, such as ant colonies, immune system, or markets, but also AI problems and models, such as board games and neural networks (Holland 1998). For example, the state space in checkers or chess is a subset of all the possible arrangements of the pieces on the board. A small set of rules define how what moves are allowed for each piece. With these rules, it is possible to derive a transition function, which captures the possible state changes in the game. From the initial board configuration of the game, this transition function implicitly defines a search tree that can only reach legal configurations. Besides, rules can also define the strategies of the players in a game, which restrict the possible board configurations even further.

To model emergent systems that rely on the combination of freely generate possibilities with constraints, holland defines a broad class of models of constrained generating procedures (cgp), which consists of a network of mechanisms (Holland 1998, 125–42). Like an agent, each mechanism is a distributed computing unit with an internal transition function of an agent h that govern their behavior in a context. Given a stimulus as a set of input values I and the current state of the mechanism $S_h(t)$ at time t as input, this function uses if-then rules to computes the next state $S_h(t+1)$.

$$S_h(t+1) = f(I_{h1}(t), I_{h2}(t), \dots, I_{hk}(t), S_h(t)) \qquad \text{Equation 1}$$

The global state S of cgp is the union set $S_1 \cup S_s \dots \cup S_m$ with the local states of all the mechanisms. However, the mechanisms in the model are not isolated from the other mechanisms. To represent the structure of the interactions in complex systems, cgp are connected by links that constraint how they interact – i.e., the link indicates how the information from one mechanism flows to another mechanism as input. Besides, they can receive a "free input" from the environment, which is represented as a directed edge that points at the recipient agent.

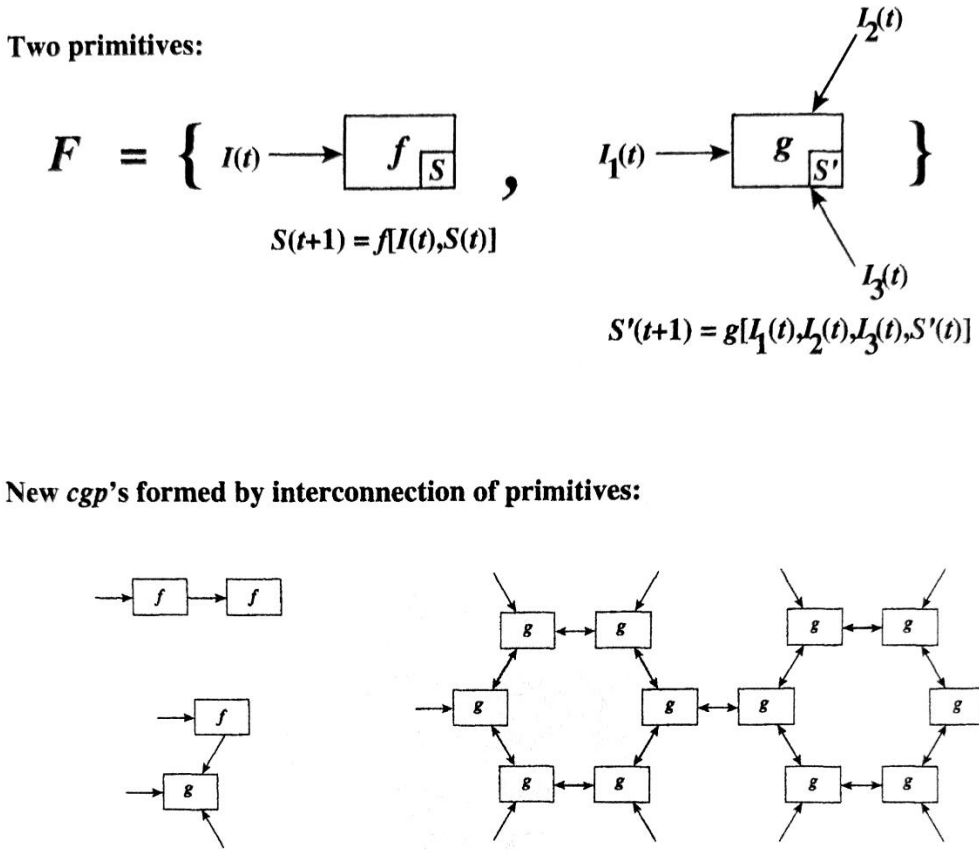


Figure 115 Constructing cgp's from a set of primitive mechanisms (Holland 1998, 135). From Emergence by John H. Holland, copyright © 1998. Reproduced with permission of Basic Books, an imprint of Hachette Book Group, Inc.

In some cases, such as in cellular automata, board games, or in neural networks, the structure of the network is fixed. Holland uses the term cgp-v to refer to the cases where the structure of the interaction is variable and time dependent. These comprehends a large set of the phenomena addressed in physics, biology, and social sciences, which rely on mobile agents.

Holland also described a general framework for complex adaptive systems (cas), where a system is described by a set of interacting and adaptive agents described in terms of rules (Holland 1995; 2012). According to him,

> *Adaptive agents are defined by an enclosing boundary that accepts some signals and ignores others, a "program" inside the boundary for processing and sending signals, and mechanisms for changing (adapting) this program in response to the agent's accumulating experience. Once the signal/boundary agents have been defined, they must be situated to allow for positioning of the relevant signals and boundaries. That is, the agents must be placed in a geometry that positions populations of agents of various kinds and localizes non-agent "resources." Within this geometry, agents must be able to form conglomerates that yield higher levels of organization with new boundaries so that the framework can capture hierarchical organization. (Holland 2012, 24)*

Overall, these agents are defined in terms of rules, hierarchical building blocks, and internal models to anticipate future consequences of its action implicitly (as in reflex agents) or explicitly (as in goal-based agents). For example, Holland describes a cas model named Echo, which

represents the interaction of evolving agents that are distributed in an environment with renewable resources (Holland 1995, 98–99).

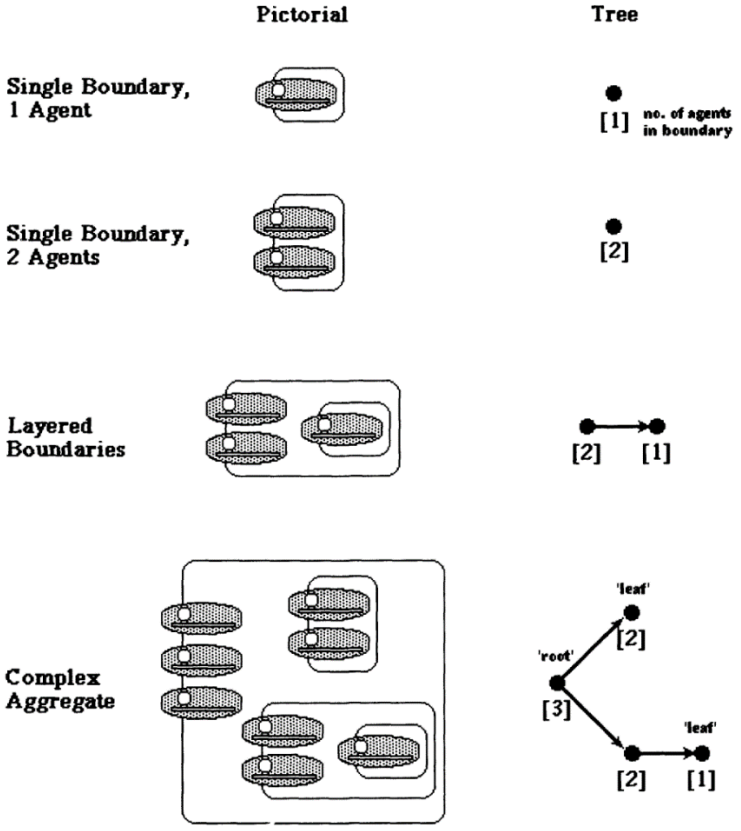


Figure 116 Boundaries and Tree representation of boundaries (Holland 1995, 118). From Hidden Order by John H. Holland, copyright © 1995. Reproduced with permission of Basic Books, an imprint of Hachette Book Group, Inc.

In the Echo model, agents collect resources and use control and tag mechanisms from its chromosome to exchange resources with other agents. The tagging mechanism result in a selective interaction with a variable linkage structure. When it acquires enough resources to make a copy of its chromosomes it can generate offspring. Besides, the interactions are also constrained by layered boundaries, which can directly host multiple agents or internal boundaries with other agents (see Figure 116). This enables the creation of a large range of building blocks formed by single agents, aggregate of agents, and multiagent. Agents can only interact with agents in the same boundary or with the ones in adjacent boundaries – i.e., the one directly external or internal to it (Holland 1995, 119).

With the model of cgp and the framework of cas, Holland captures a paramount property of ABM: situated computation. As the neighborhood of an agent constrains the possibilities of interaction between its parts, ABM can use simple rules and transition functions in the micro level to produce flows of coherent patterns on the macro-level of the system.

Another concept that is often associated with complex system and ABM is self-organization, whose meaning had inconsistent treatment over the history. De Wolf and Holvoet (2005) showed that

the term originated from Ashby's cybernetics and became a topic of areas such as physics, computer science, and system theory. According to them, "self-organization is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control" (De Wolf and Holvoet 2005, 7). Self-organization relies on autonomous control of the parts, such as with agents, and it leads to a dynamic increase in the order of the system behavior.

So, when modeling systems that exhibit properties such as emergence or self-organization, the use of agents as building blocks is a key to capture the underlying mechanisms. ABM, the agent is a unit of representation in the computational modeling of social dynamics related to natural or artificial phenomena (Wilensky and Rand 2015). ABM relies on modeling local interaction between agents in a shared environment. The interactions comprehend different components, such as agent-self, environment-self, agent-agent, environment-environment and agent-environment interaction (Wilensky and Rand 2015, 257–62). ABM may also include the user as an omniscient agent that has access to all agents, environments and can affect the behavior of the system. All these interactions typically depend on the configuration of agents and environment in a certain spatial representation, such as cellular grids, topological networks, or coordinate systems.

Examples of well-known ABM include cellular automata, game of life, forest fire, diffusion-limited aggregation, the segregation model, sugarscape, etc. (Downey 2012; Wilensky and Rand 2015).

7.4.3. Bio-Inspired Artificial Intelligence and Engineering

Despite their distinct goals, the boundary between AI and ABM is often blurred. While AI and related fields can look for complex systems as a source of inspiration for distributed methods of problem solving, ABM can use AI techniques to support research on complex systems.

One important area in this shared territory is bio-inspired artificial intelligence and engineering.

In Swarm Intelligence, Bonebau, Dorigo, and Theraulaz investigate "how to design adaptive, decentralized, flexible, and robust artificial systems, capable of solving problems, inspired by social insects" (1999, 7). They explore models of collective behavior of social insect colonies and other animal societies in tasks such as foraging, division of labor, corpse clustering, larval sorting, nest building, and cooperative transport. Then, extend these biologically motivated models to create decentralized problem-solving system for engineering methods and problems, such as optimization, multiagent systems, and collective robotics.

Over the 1990s, reverse engineering the mechanisms of ant societies that use pheromone to select the shortest path connecting their nest to a food site supported the development of many multi-agent algorithms for combinatorial optimization. These algorithms were unified under the framework of the ant colony optimization metaheuristic (ACO) by Dorigo, Di Caro, and Gambardella – see details in (Di Caro 2004, 1–4). ACO relies on "a set of ant-like agents, the use of memory and of stochastic decisions, and strategies of collective and distributed learning" (Di Caro 2004, 1).

In their book also named Swarm intelligence, Kennedy and Eberhart (2001) advocate for the

adoption of alternative models of artificial intelligence, based on emergent behaviors, and self-organization. Particularly, they focus on models motivated by social behavior and collective navigation, based on simple rules using homogeneous and often non-autonomous computational entities – which they refer to as particles (Kennedy, Eberhart, and Shi 2001, xix). They propose the particle swarm optimization algorithm (PSO), which relies on a swarm of candidate solutions that iteratively adapt their trajectory using individual and shared knowledge about the known best positions in the problem space (Kennedy, Eberhart, and Shi 2001, 287–325).

Floreano and Mattiussi describe a new artificial intelligence that "emerged at the turn of the millennium expanded its focus of attention from human brains and cognitive reasoning to a wider range of organisms, processes, and phenomena that occur at multiple spatial and temporal scales" (2008, xi–xii). They organize the theories and methods of this bio-inspired artificial intelligence in chapters based on the source of inspiration of models and algorithms, such as evolutionary systems, cellular systems, neural systems, developmental systems, immune systems, behavioral systems, and collective systems (Floreano and Mattiussi 2008). The book also includes models that are considered part of complex systems and ABM, such as cellular automata, game of life, or Boids, and many bio-inspired metaheuristics, such as ACO and PSO.

Particularly, biological models became an inspiration for engineering of distributed systems, such as large-scale computer and mobile networks due to their properties. Babaoglu *et al* (2006) understand that centralized engineering approaches based human specifications are not suitable to deal with the characteristics of these distributed task environments. In contrast, biological networks developed solutions for problems on a different scale of time and for mutable environments, which makes them less sensitive to general variations.

Navlakha and Bar-Joseph (2011; 2015) also investigate the potential of biological systems that perform tasks, such as coordinated decision making, leader election, routing and navigation. They contrast to the computationally intensive, fast, and deterministic approach of conventional engineering with biological systems. The latter tend to reduce the complexity of the communication and invest on randomized algorithms to operate in unreliable and noisy task environments. These systems are composed of a large scale of dynamic components and established strategies to be robust to failure of components and to adapt to external variables. Thus, while biological systems lack the central control and strong reliability of traditional design of computational distributed systems, they support desirable properties of adaptivity and robustness.

7.5. Appendix E: A review of multi-agent spatial synthesis

The field of spatial synthesis is grounded in classic Artificial Intelligence (AI) techniques and on optimization methods. However, there is an emerging niche of the field where agent-based techniques are employed in the generative models to augment exploration of spatial configurations in real-time, which I refer to as multi-agent spatial synthesis (MASS). In MASS, agents are autonomous entities that can use its local perception and its program to reason and act over the spatial representation. They can have different relationships with the spatial representations during the simulation, such as

1. Type 1: agents that embed the representation of a spatial unit, such as a room or a zone, so it can move and reshape it over time.
2. Type 2: agents that move and occupy a pre-existing spatial partition in the environment
3. Type 3: agents that are used as a reference for an algorithm that partition a space

Due to its emphasis on a continuous and interactive configuration of spatial patterns, MASS has huge potential for future decision making and design systems. However, it still has a secondary role in the literature. Existing spatial synthesis literature review papers (W. Mitchell 1977; Liggett 2000; Homayouni 2000; Hsu and Krawczyk 2003; Lobos and Donath 2010) have none or few examples of MASS, recent specialized literature review papers focus on classic techniques, such as evolutionary optimization (Calixto and Celani 2015) and a recent review of agent-based models in architectural research (Chen 2012) addresses collaborative design systems, but does not present MASS examples for building design.

This section addresses this gap by providing an initial categorization and a literature review of recent MASS work (2008-2017) within the scope of building design. It includes the allocation of human activities in architectural space, such as building parts on a site or rooms inside a building, using diagrams or technical representations. It comprehends agents with different complexity, from basic reflex agents that refine a pre-existing spatial arrangement to intelligent agents that can generates complete spatial patterns. It covers agent-based modeling, techniques adapted to control agents (such as physics simulation) and hybridizations of agents with conventional or manual methods. It excludes applications with cellular automata, which were already reviewed by a previous paper (C. M. Herr and Ford 2015), and allocation tasks that are strictly for urban design.

Different databases and online documentations were used to look for MASS prototypes, but the main source is CumInCAD[21]. In the next three sections, I organize the descriptions of sixteen research prototypes and two precedents according to categories that conciliate algorithmic and spatial characteristics: (1) agents as moving spatial units; (2) agents that occupy discrete spaces;

[21] CumInCAD is a cumulative index about publications in Computer-Aided Architectural Design (CAAD), which provides access to papers of the conferences organized by the sibling CAAD associations (ACADIA, CAADRIA, eCAADe, SIGraDi, ASCAAD and CAAD futures) and other related conferences.

and (3) agents that partition space.

After these descriptive sections, I analyze the prototypes based on their types and on the occurrence of features related to computational representation, evaluation, and control procedures. The goal is to characterize the prototypes under the proposed categories and to present trends, patterns, and challenges for future research.

7.5.1. Type 1: Agents as moving spatial units

Each agent of a system of type 1 incorporates an individual representation of space, which is modified to interact with other agents and to satisfy spatial objectives.

7.5.1.1. IMAGE (IMG)[22]

System description: IMG is an interactive computer system for multi-constrained synthesis of spatial arrangements (Weinzapfel and Handel 1975; Weinzapfel, Johnson, and Perkins 1971), and an important precedent for type 1.

System algorithm: Each spatial unit is a custom cuboid that moves in the space. These aim at satisfying (multiple) user-defined objectives and constraints: adjacent, keep out, overlap, relative position, ratio, width / depth / height, x / y / z position, distance, shared wall, enclose, next, on top of, floor, above, align, and visual access. A constraint graph stores cuboid units at the main nodes and relations as intermediate labelled nodes. A Least Mean Squares Fit (LMSF) is applied sequentially to all the spaces, changing the descriptors of a unit (dimension, location, and rotation) to achieve the best local improvement – i.e., minimum error regarding all the linear equations of the constraints.

7.5.1.2. Fuzzy Layout Planner (FLP)

System description: FLP (Bayraktar and Çağdaş 2013) is an editor to create and modify dynamic bubble diagrams for the early-stages of design. It adopts reactive spatial units and focuses on user manipulation.

System algorithm: Each agent is a rectangular bubble, which has minimum agency and is directly manipulated by the designer. As the user changes one of its dimensions, the bubble modifies the other dimension automatically to preserve the assigned area. A repulsion force ensures the packing of close bubbles.

[22] I refer to all the prototypes analyzed in this appendix by a name and an abbreviation. In cases where the authors defined the name in the paper, I adopt it. Else, I create a simple name based on the paper title, description, or characteristics of the prototype. The abbreviation of these names will be used to reference the prototypes in the tables in **7.5.4**.

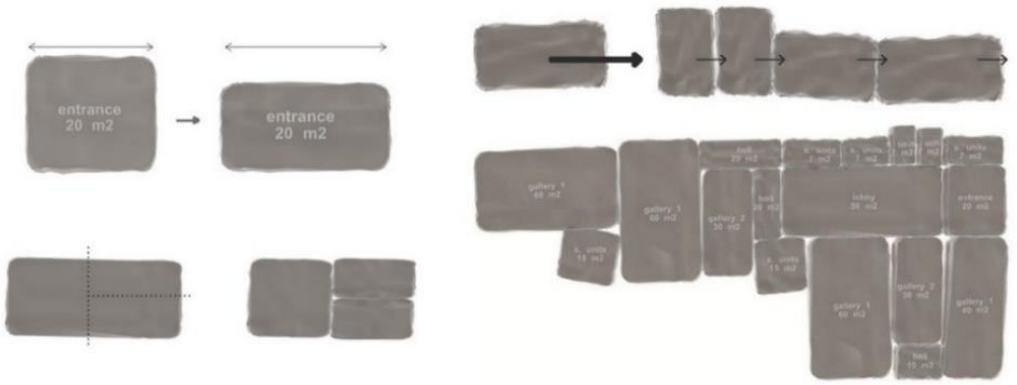


Figure 117 Fuzzy Layout Planner. Top left: dimensions of bubble adapt to preserve area; bottom-left: dividing bubbles; top-right: repulsion forces; bottom-right: packing bubbles (Bayraktar and Çağdaş 2013, 377–78). Reproduced with permission of the authors.

7.5.1.3. Sniffing Space I (SS1)

System description: SS1 (Ireland 2009a) is MASS hybrid of type 1 and 2. It consists of a self-organizing bubble diagram combined with loosely packed rectangles (Ireland 2010, 187–88).

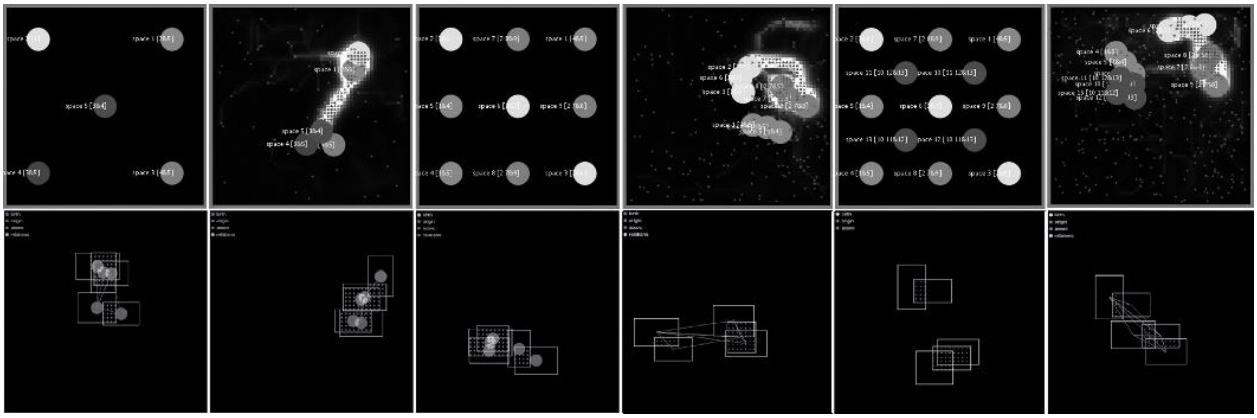


Figure 118 Sniffing Space. Top row: spatial association generating zones (Ireland 2009a, 255, 256). Bottom row: expansion of the model with rectangular boundaries (Ireland 2010, 187–88). Reproduced with permission of the author.

System algorithm: SS1 incorporates swarm intelligence – in contrast to the reactive agents of FLP, it uses ant foraging algorithms to define the interaction between two types of agents: (1) ant soldiers; and (2) an array of ant nests. The soldiers are produced by the nests, which are inter-connected by edges indicating association or adversity. The soldiers are points that navigate in the environment, producing and following two different pheromone trails depending on whether they are searching for associated nests or returning to their original nest. Each nest has a fixed circular body and a rectangular boundary with random dimensions x and y that satisfy its required area. The nests gradually follow visiting soldiers that are returning to associated nests, clustering with them and eventually interrupting the production of new soldiers. Additionally, they have three spatial behaviors: adapt x and y when they nestle, move away from overlapping adversary nests, and overlie with associated nests. It results in a loose packing of rectangles, defined by the

connections.

7.5.1.4. Actants (ACT)

System description: ACT (Ireland 2015b; 2015a) is a self-organizing system that loosely packs polygons.

System algorithm: It combines swarm intelligence and a physics engine. It is based on dynamic units called Actants, which are represented by soft body polygons composed of internal and boundary nodes that function as sensors and actuators. The nodes maintain the consistency of the shape by attraction-repel forces. The boundary nodes emit and track identity-based pheromone, detect other nodes, and move the Actant.

The system is based on a hunter-prey dynamic between different Actants. The pheromone differences in the environment might attract or repel boundary nodes, keeping the Actant in movement. After a period, if no associated Actant is found by following the pheromone gradients, one of the boundary nodes is selected for a direct search in the local vicinity. When an Actant wants to be found by its associates, it produces pheromone. When it is evading or seeking, it blocks pheromone production, preventing the agglomeration of non-associated Actants.

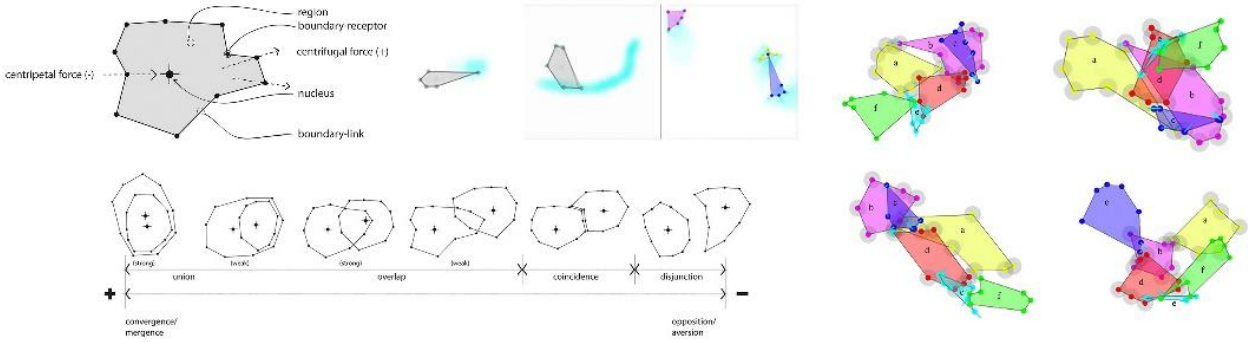


Figure 119 Actants. Top-left: an Actant and its components; bottom-left: scale of consolidation; top-center: Actants' vertices reacting to pheromone; right: different associations (Ireland 2015b, 585–88). Reproduced with permission of the author.

7.5.1.5. Floating Bubbles (FB)

System description: FB (Hao and Ting-Li 2010) is a self-organizing bubble diagram with optional rectangular boundaries.

System algorithm: Physics simulation and heuristics solve the adjacency between bubbles. Agents are represented as circular disks or, optionally, as rectangles, connected by adjacency edges. To solve the diagram, the system assigns two basic forces to the agent: attraction and repulsion. Attraction is proportional to the length of the vector between the boundaries of two connected bubbles. Repulsion is proportional to the overlapping area between two bubbles and pushes each bubble away. However, using only attraction and repulsion forces, the bubbles can get stuck in suboptimal arrangements. To avoid these situations, a heuristic moves a bubble towards a connected bubble when an adjacent requirement stays unsatisfied for too long. As the heuristic restarts the interactions in a slightly different state, eventually it reaches the equilibrium and

satisfy all the adjacencies.

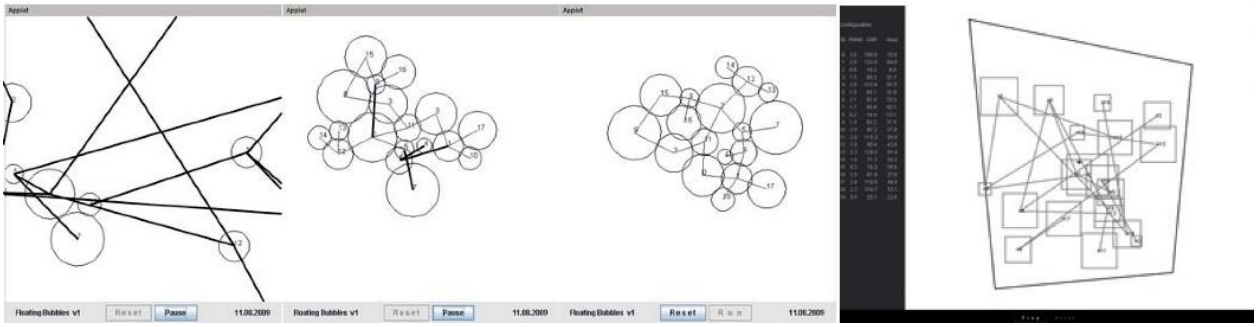


Figure 120 Floating Bubbles. Left: three stages in the basic floating bubble system; right: one stage of the generative process using squares (Hao and Ting-Li 2010, 177, 181, 182). Reproduced with permission of the authors.

7.5.1.6. Space-planning with real-time physics (RTF)

System description: RTF (Syp 2011) is a self-organizing, three-dimensional bubble diagram.

System algorithm: The user defines an abstract diagram with primary and secondary adjacencies among the elements of the architectural brief. The elements can also be connected to anchor objects, which represent important features of the environment. The specifications of the abstract diagram are translated to a physics simulation, where the elements are spheres, the edges are springs (with different forces for primary, secondary, and anchor connections), and the anchors are cylinders. The resulting bubble diagram is solved by packing the spheres using spring forces. To translate the bubbles to a floorplan diagram, the spheres are converted to squares in the three-dimensional space, which can be customized by the user.

7.5.1.7. Interactive Bubbles (IB)

System description: IB (Veloso 2014) is a self-organizing bubble diagram.

System algorithm: It combines a temperature heuristic with direct application of attraction and repulsion forces to organize circular agents. Attraction and repulsion forces solve the adjacency of connected bubbles and pack them inside a user-defined polygon. The overall behavior of each agent is a result of the sum of the vectors of the forces. A non-destructive heuristic uses varying temperatures to prevent suboptimal arrangements. The temperature is inversely proportional to the satisfaction of the objectives. An unsatisfied agent has higher temperature, reducing its area and its reaction to the surrounding forces. When the agent gets closer to satisfy its objectives, the temperature is reduced, recovering its original area and behavior. The user can manipulate the descriptors of the bubbles in real-time.

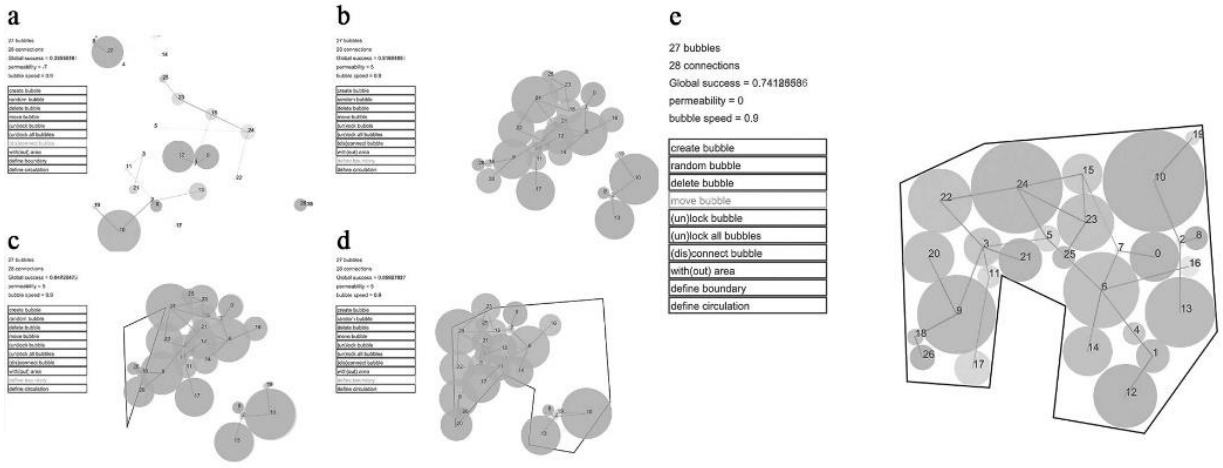


Figure 121 Interactive bubbles. a-b: simulation of the bubble diagram with changing temperature; c-e: simulation with the definition of a containment area (Veloso 2014). By Author.

7.5.1.8. Narrative Landscape (NL)

System description: NL (Christensen 2014) is a self-organizing system that packs convex polygons.

System algorithm: It uses a Markov chain combined with physics simulation to create and pack convex spaces. Initially, the user defines a set of spatial types with distinct geometric properties and specifies a transition matrix. The generative process starts by randomly sampling an initial space in that set and sequentially sampling the next spaces using the transition matrix. Different transition matrices and spatial properties will generate different chains of polygons. Each space is a soft body composed of a central particle and surrounded by child particles on the border. The behavior of the bodies is defined by the combination of four types of specialized springs: (1) a Markov spring connects the central particles of neighboring spaces to reinforce adjacency; (2) a set of collision springs attaches the central particle of a space to close particles of neighboring spaces to reinforce alignment; (3) a minimum distance spring preserves a minimum distance between spaces with different qualities; (4) a within distance spring reinforces adjacency for spaces with the same qualities that are not connected by Markov springs.

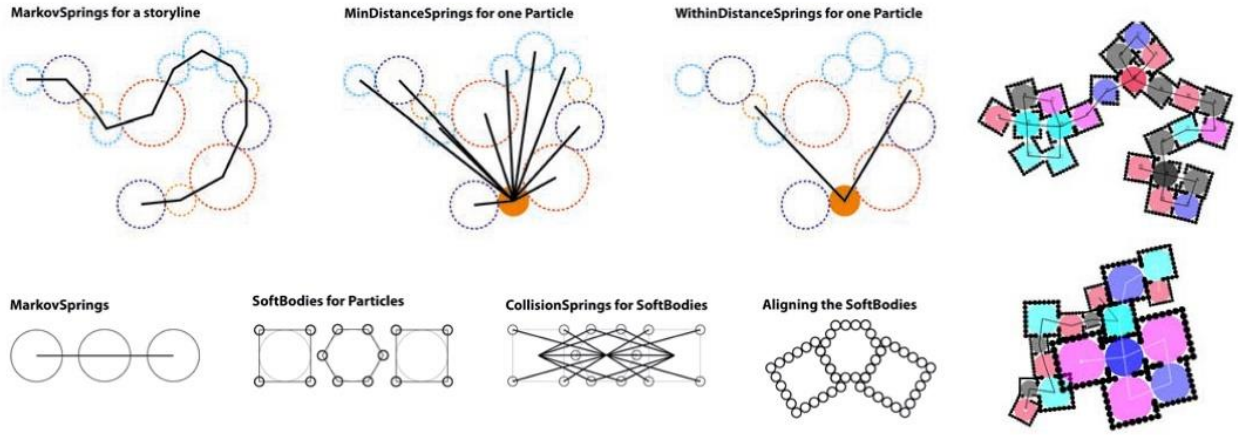


Figure 122 Narrative landscape. Left: soft bodies and classes for springs connecting particles; right: two layouts (Christensen 2014, 242, 245). Reproduced with permission of the authors.

7.5.1.9. Multi-agent + Evolutionary (MAE)

System description: MAE (Guo and Li 2017) is a MASS hybrid of type 1 and 2 that combines a 3D bubble diagram with a grid-based system to allocate activities for architectural layouts.

System algorithm: The simulation in MAE uses two types of agents: spheres that represent regular rooms and capsules that represent corridors, staircases, and linear rooms. The agents are placed on the environment and are connected based on adjacency requirements. They interact using rules such as attraction, repulsion, swap, and compression. In attraction, each agent is pushed closer to a connected agent by a vector $\boldsymbol{m}$, defined by the difference between their closest points. Repulsion uses the inverse of $\boldsymbol{m}$ to push overlapping agents apart. For most of the cases, attraction and repulsion only move the target agent. However, they can also affect the orientation and length if the target is a horizontal capsule-like agent. Pairs of connected agents use the swap rule to avoid crossing connections with other pairs. The compression rule is a heuristic that reduces the volume of the building by pushing the agents together along the simulation.

After the simulation, the resulting spaces are assigned to a 3D grid, forming sets of cells with the same room identities and sets of faces between the different rooms. This is the initial state for an optimization. At each iteration, a child solution is generated by mutation – room identities are swapped, and room shapes are changed by moving coplanar faces. The parent and child are evaluated by their room shapes, dimensions, aspect ratios and building shape. An annealing-based function defines the probability of accepting the child as the parent for the next iteration.

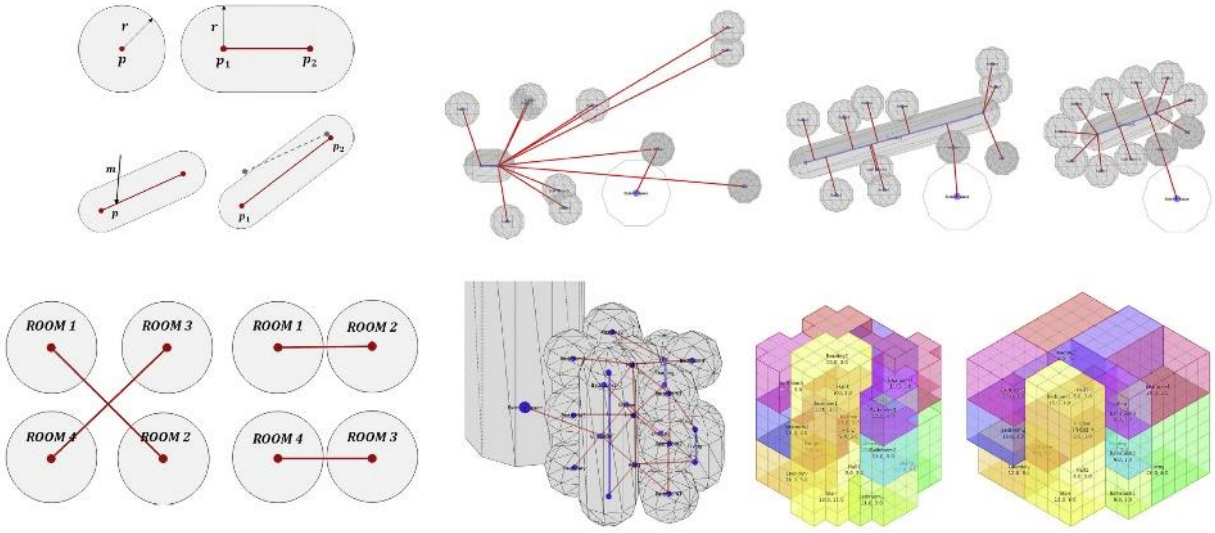


Figure 123 Multi-agent + Evolutionary. Top-left: spherical agent, capsule-like agent and push operation; bottom-left: swap operation; top-right: agents solving adjacencies; bottom-right: converting agents to a grid representation and optimizing the solution (Guo and Li 2017, 54–57). Reproduced with permission of the authors.

7.5.2. Type 2: Agents that occupy a space

Type 2 agents do not have a custom spatial shape. They navigate and allocate activities in a set of pre-defined spatial units, such as grids.

7.5.2.1. Sniffing Space II (SS2)

System description: SS2 (Ireland 2009b) is a grid-based system to allocate activities in the routes of an airport.

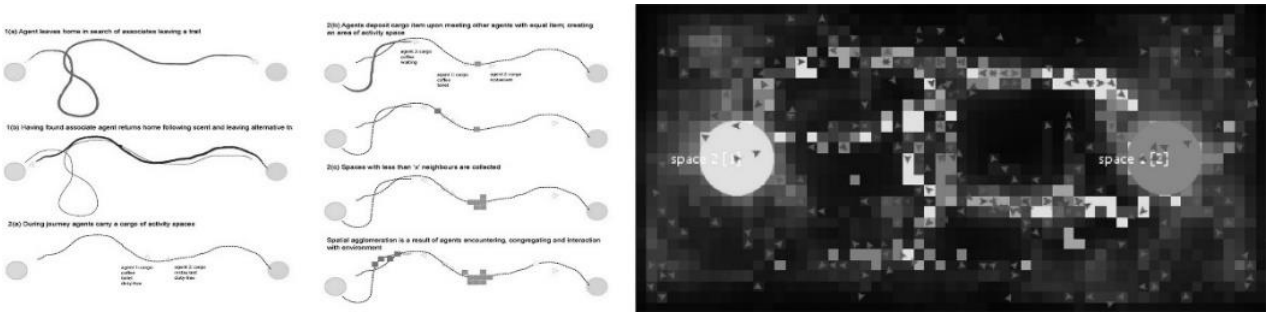


Figure 124 Sniffing Space II. Left: Routing between destination points; Right: cargo clustering (Ireland 2009b, 224–25). Reproduced with permission of the author.

System algorithm: It combines ant foraging with a cargo placement protocol to allocate activities on a grid. The ant foraging algorithm simulates passenger routing in an airport. Each ant has an agenda with a sequence of destinations or hubs, which represent specific spaces of the airport. Ants not only follow pheromones and navigate between hubs but also respond to other agents sharing similar qualities. These qualities are encoded as random activity cargos that are assigned to each agent. As agents with similar items meet on a trail, they deposit the cargo at their current location. Every time another agent with a similar cargo passes by this cluster of cargos, it also drops it. In contrast, a passing agent might collect dropped cargos that have less than n neighbors, preventing scattering of activities. The concentration of the cargos defines the activity cells on the routes between destination points.

7.5.2.2. Stigmergic Space Adjacency Software (SSAS)

System description: SSAS (Meyboom and Reeves 2013) is a self-organizing system that allocates different activities in a three-dimensional grid.

System algorithm: Agents represent different activities connected by adjacency relationships. Each agent has a specific RGB value and is compatible with agents within a close color range – meaning that they should be adjacent. The agents dispute the territory in a three-dimensional grid of cells. They expand to unoccupied cells with closest pheromone values and, after achieving the desired area, contract. At each node, the agent changes the local environment by diffusing its pheromone value, which attracts agents with compatible values. A regular cell carries a pheromone value and is influenced by the agents. However, the cell can be customized to impose a pre-defined value, stimulating spatial templates over which the agents will dispute the territory – such as circulation systems, boundary areas closer to the light exposure, etc. Also, the user can customize the set of nodes of the three-dimensional grid, creating topographies, a pre-defined *parti* or even voids to preserve views.

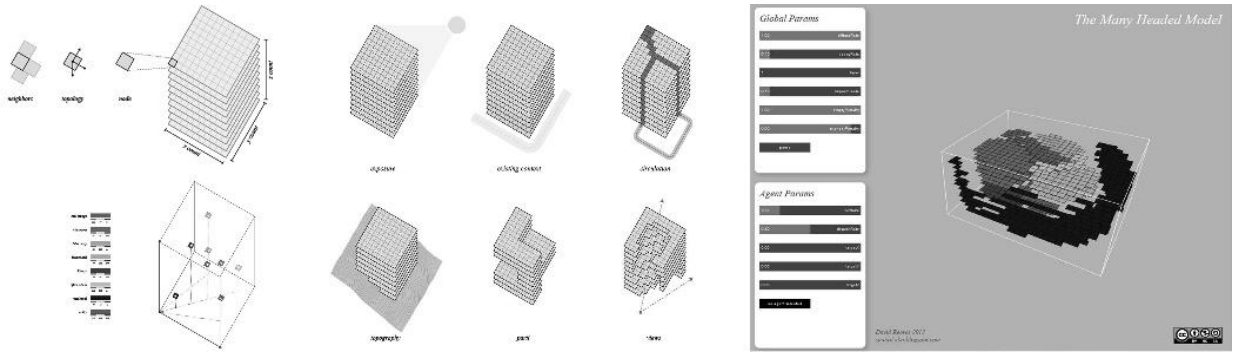


Figure 125 Stigmergic Space Adjacency Software. Left: grid topology and example of pheromone values; middle: templates for external influence and node masking; right: a solution (Meyboom and Reeves 2013, 201–4). Reproduced with permission of the authors.

7.5.2.3. Collective construction (CC)

System description: CC (Narahara 2017) is a multi-agent learning system that allocates different activities in a three-dimensional grid.

System algorithm: Ten color-coded groups containing fifty agents allocate the different activities in the grid using adversarial learning. During the simulation, each agent perceives four features in its cone of vision and can take four actions. The features are: (1) the agent is on the ground level; (2) there is a teammate nearby; (3) there is an opponent nearby; and (4) there is a building block nearby. The actions are: (1) move forward; (2) flock, to align its heading with its teammates; (3) attack, which converts opponents to teammates or eliminates teammates; and (4) build, which places a building block of the team's color in the cell. Each group uses a neural network with shared weights as its policy. At each time step, the agent uses the four features detected as the input of the network, which outputs the probability of executing the four actions. Then, the agent selects the most probable one. At the end of the simulation, the performance of each team is measured by the number of blocks produced multiplied by the number of surviving team members. The four elite networks remain for the next episode, while the other six are defined by mutation and combination. This experiment is repeated, resulting in different arrangements.

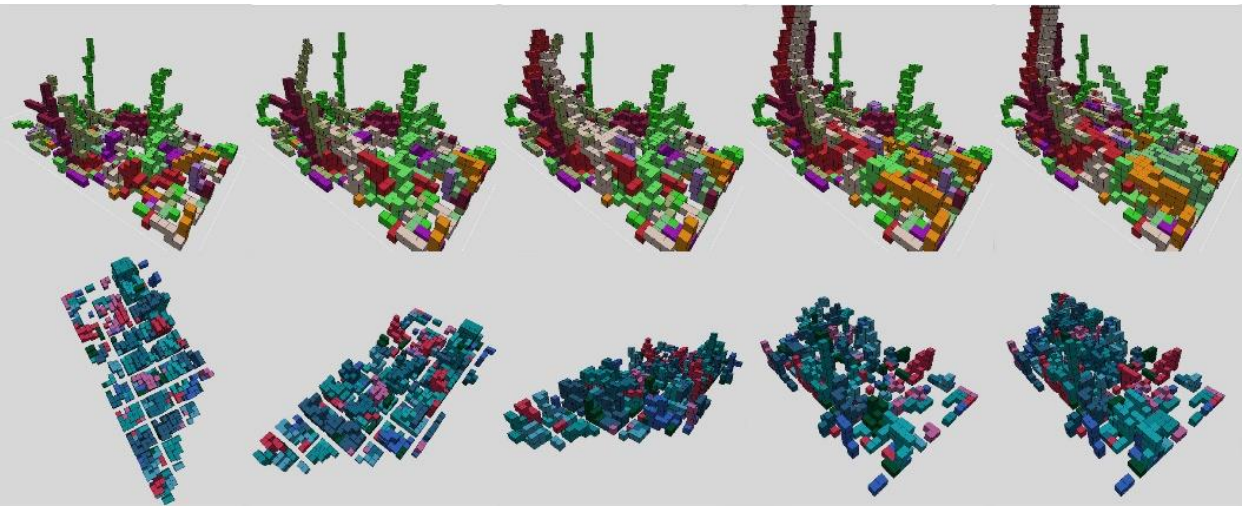


Figure 126 Collective construction. Results of the experiments (Narahara 2017, 347). Reproduced with permission of the author.

7.5.2.4. Artificial Life Space Planning (ALSP)

System description: ALSP (Fernando 2014) is hybrid system of type 1 and 2 that combines activity allocation in a two-dimensional grid with a bubble diagram.

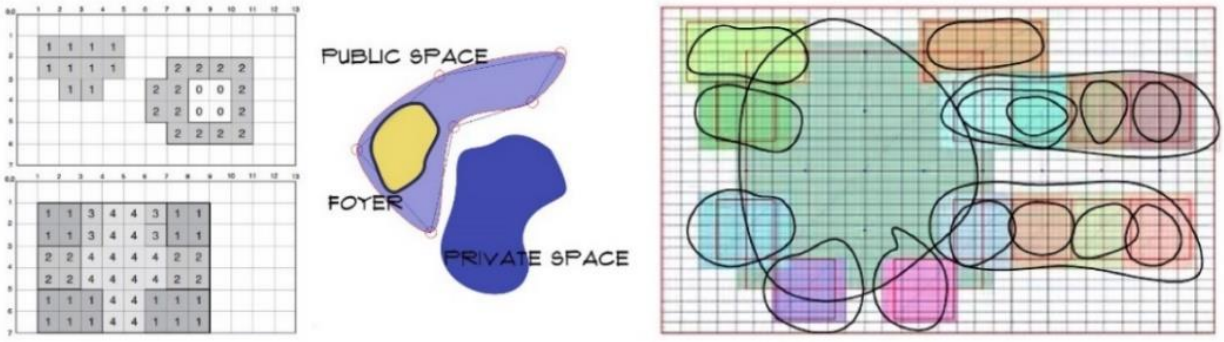


Figure 127 Artificial Life Space Planning. Left: grid-planning system; center: bubble diagram; right; integration of grid-planning system and bubble diagram (Fernando 2014, 661, 662, 665). Reproduced with permission of the author.

System algorithm: Each bubble is a soft body defined by a curve interpolated over a series of control points. The user can create, edit, drag, and connect these bubbles. Then, a genetic algorithm (GA) specifies rectangles on a grid. The genome is based on five descriptors: row, col, width, length and state. A fitness function evaluates the area and adjacency of a grid allocation based on the specifications of the bubble diagram (area of the bubbles and connections). Finally, the bubbles will move to the solution specified on the grid, stablishing a design cycle.

7.5.3. Type 3: Agents that partition space

In contrast to agents that have an attached spatial representation or agents that occupy a discrete set of cells, type 3 agents navigate in the territory and parametrize the partition of the space.

7.5.3.1. YONA (Y)

System description: YONA (Weinzapfel and Negroponte 1976) is a graph-based system to generate

residential layout. It serves as an historically important precedent for type 3,

System algorithm: It is based on a three-stage representation with incremental design information: graph embedding, bubble-diagram, and schematic plan. YONA tests the adjacency graph specified by the user to ensure its planarity and then uses a top-down heuristic to generate a graph embedding. The user can rearrange the embedding by moving each node to a new position. After a satisfactory arrangement, the program draws an offset boundary and a dual graph, creating multiple polygonal partitions around the original nodes. Then, it generates b-spline curves inside these partitions, forming a bubble-diagram over which the user can sketch the shape of the rooms.

7.5.3.2. Associative Spatial Networks (ASN)

System description: ASN (Harding and Derix 2011) is a system that generates an exhibition hall layout according to the topology of the exhibition and potentially to the users' feedback.

System algorithm: The system is divided in 3 parts. Firstly, adjacency graphs are generated for the exhibitions. Each individual exhibit of an exhibition is described by multiple features and projected to a point in $\mathbb{R}^2$ by a self-organizing map (SOM). A planar graph is created by connecting each point to the neighbors with the closest distance and by storing these values in the edges. The second stage involves clustering exhibitions that can share similar spatial configuration of the layout. This is solved by a growing neural network that adapts to the varying dimension of the input spectrum for each adjacency graph, clustering them according to similar topologies. The graph with the spectrum closest to the average in each cluster is selected to define the floor plan. In the third stage, the selected average graphs are embedded on a plane and their nodes are distributed in the exhibition hall layout by a repulsion algorithm. Using a Voronoi diagram, each node becomes a polygonal exhibition cell, and the permeability of the wall is defined by the connection between the neighboring nodes (wall for no connection, permeable wall for high edge values and no wall for low edge values).

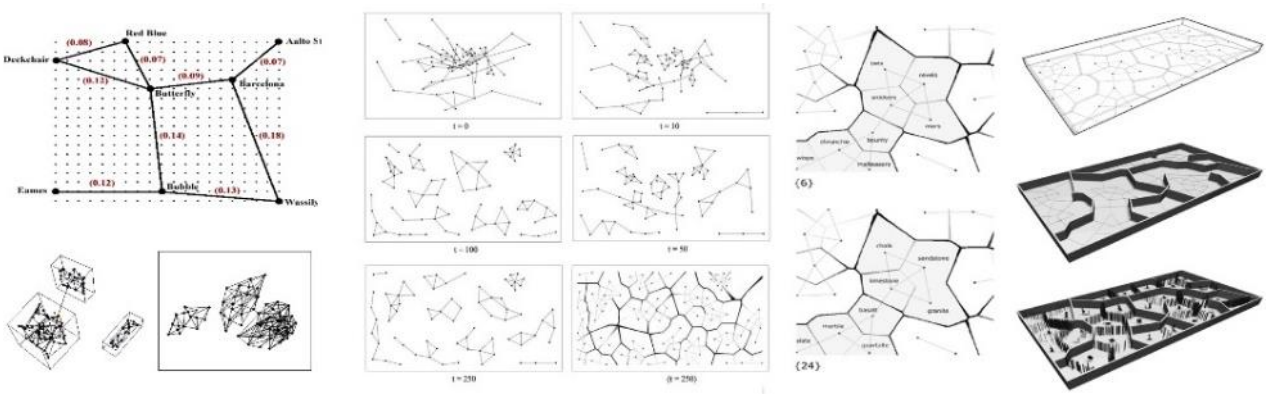


Figure 128 Associative Spatial Networks. Top-left: adjacency graph for exhibition; bottom-left: growing neural gas mapping; middle: repulsion algorithm distributes graph nodes over the space; right: translating a graph into the layout of the exhibition with a 2d Voronoi diagram (Harding and Derix 2011, 310, 312, 316, 317). Reproduced with permission of the authors.

7.5.3.3. Dwelling Agglomerator (DA)

System description: DA (Puusepp 2014a; 2014b) is a self-organizing settlement of dwelling units

connected by public spaces.

System algorithm: It associates a custom flocking model with Voronoi partitioning and an evaluation function to generate settlements in real-time. A flocking simulation moves the agents to form a network with approximated orthogonal connections, following five basic navigation rules. The first two rules are movements that preserve a specific distance range between the agent and its closest neighbors. With this range satisfied, other three rules specify a rotation around the closest neighbor, moving the agent towards a position aligned or perpendicular to the edge connecting the two closest neighbors. The agents are used to partition the space with a Voronoi Diagram, forming the dwelling units and the open space. An evaluation function calculates a score for each unit based on criteria such as area, direct sunlight, or accessibility. It defines whether the agent should stop or keep flocking to look for a better position.

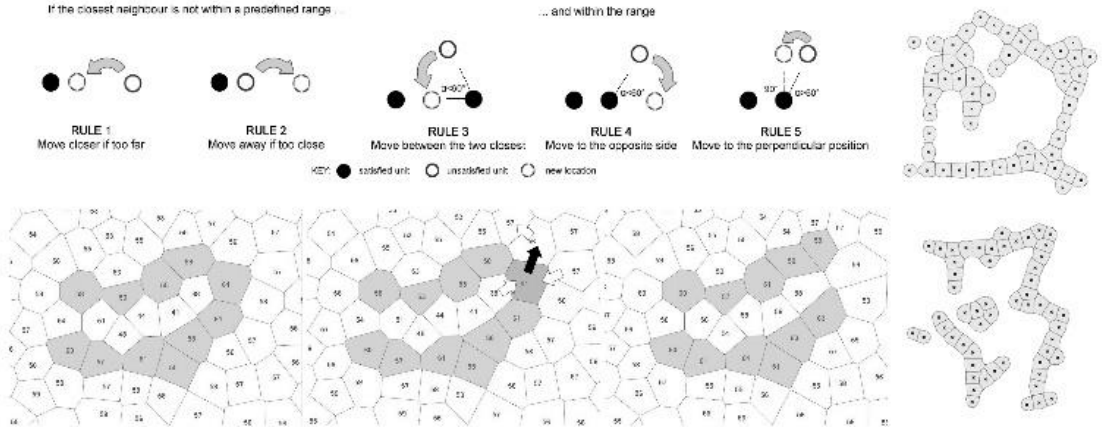


Figure 129 Dwelling Agglomerator. Top-left: flocking rules; Bottom-left: creation of access routes; right: two layouts (Puusepp 2014a, 49–50). Reproduced with permission of the author.

7.5.3.4. Responsive Algorithms (RA)

System description: RA(Bazalo and Moleta 2015) is a MASS of type 1 and 3 that generates spatial partitions based on an architectural brief.

System algorithm: RA combines bubble diagrams, graph embeddings and Voronoi partition inside a parametric modelling editor. The user defines a bubble diagram, with disks representing the spatial units with their required area, and edges representing the adjacency. The bubble diagram is treated as a circle packing problem and solved by physics simulation: circular discs are rigid bodies, and the edges are springs representing adjacencies. A Voronoi diagram generates a spatial partition using the centers of the disks. This workflow is extensible to the three-dimensional space, where the nodes of the adjacency graph are the centers of packed spheres and a Voronoi partition results in a packing of polyhedral cells.

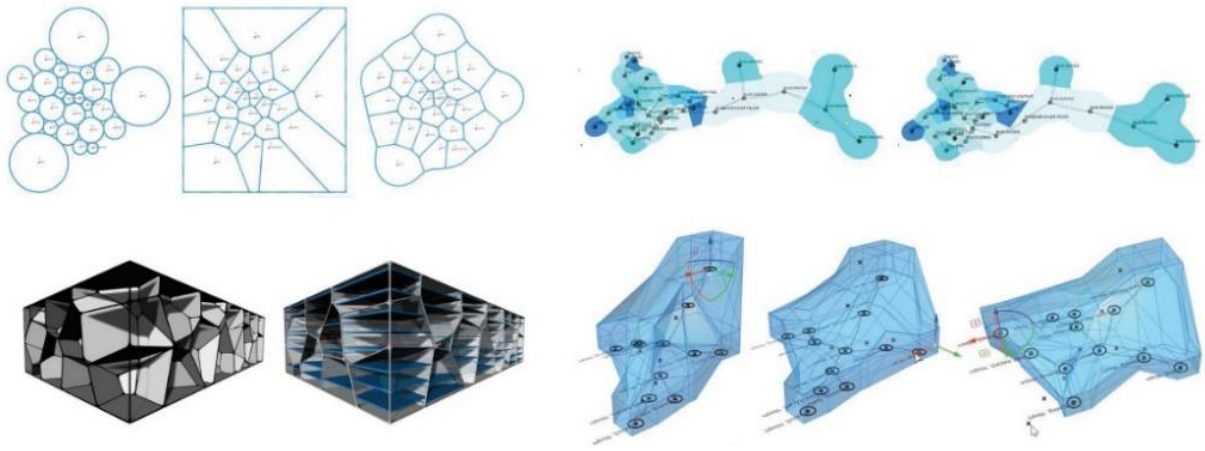


Figure 130 Responsive Algorithms. Top-left: circle packing, Voronoi diagram and Voronoi diagram with radii; top-right: force-directed graph over Voronoi diagram with radii; bottom-left: 3D Voronoi cells; bottom-right: three-dimensional packing with a skin (Bazalo and Moleta 2015, 212–14, 217). Reproduced with permission of the authors.

7.5.3.5. Evolving Floorplans (EF)

System description: EF (J. Simon 2017; Carta et al. 2020) is MASS of type 1 and 3 that generates floorplans based on topological and flow objectives.

System algorithm: EF uses a general version of Neuroevolution of Augmenting Topologies (NEAT) to evolves a weighted, connected, and undirected graph representing a floorplan. Chronological markers enable crossover and mutation of solutions with different topology and numbers of rooms. The floorplan genome of NEAT is composed of node genes and connection genes. The former store information such as room's size, while the latter assign weights to the connections between pairs of node genes and are added randomly until the graph is connected.

The phenotype (i.e., the floorplan) is generated in four steps. (1) A spectral layout embeds the graph in space, which is the input for (2) a physics simulation that converts the nodes and edges into discs and springs. After the resulting packing, the centers of the discs are extracted and combined with the vertices of an offset concave hull for (3) the Voronoi tessellation. With this combination, the internal Voronoi cells have a boundary like the original circle packing boundary.

Finally, (4) an ant-colony algorithm is applied to the interior edges of the Voronoi mesh to generate a weighted network to satisfy the connection requirements. The edges of the resulting hallways are smoothed and dimensioned according to their flow. The phenotype is evaluated by the NEAT algorithm, enabling the use of objectives such as minimization of traffic between classes, material usage, fire escape paths, and windows.

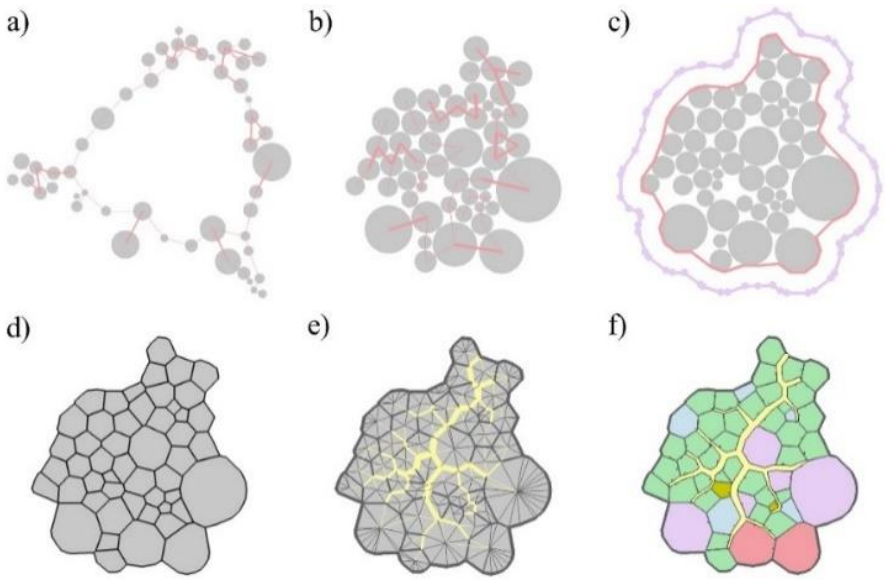


Figure 131 a) spectral layout; b) physics simulation; c) offset of concave hull; d) Voronoi tessellation; e) generation of circulation using an ant-colony algorithm; f) final floorplan (J. Simon 2017). Reproduced with permission of the author.

7.5.4. Analysis of features

In this section, I analyze the different MASS prototypes to identify the occurrence of similar features of representation, objectives, and control procedures. Table 16 synthesizes the general categorization of types.

In terms of representation, features used in spatial synthesis for more than four decades (W. Mitchell 1977), such as grids, polygonal representations, and graphs, are still pervasive in MASS. The prototypes also incorporate new features, such as soft shapes (bubbles) and the Voronoi diagram for spatial partition. As shown in Table 17, bubbles and polygons are dominant in type 1. Grids are mostly used in type 2. Voronoi diagrams are dominant in type 3. Graphs are used predominantly in type 1 and 3 to store the connections between spaces. In some prototypes, the diagrammatic solutions were also converted to richer architectural representations with constructive elements (FB and MAE).

Table 16. MASS Types.

	IMG	FLP	SS1	ACT	FB	RTF	IB	NL	MAE	SS2	SSAS	CC	ALSP	Y	ASN	DA	RA	EF
Type 1	■	■	■	■	■	■	■	■	■				■				■	■
Type 2			■						■	■	■	■	■					
Type 3														■	■	■	■	■

Table 17. Representation of the agents.

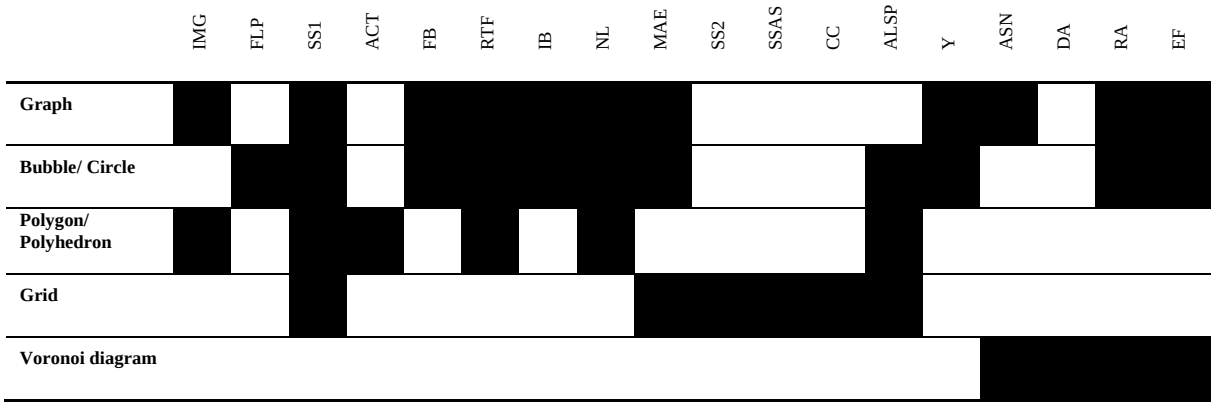
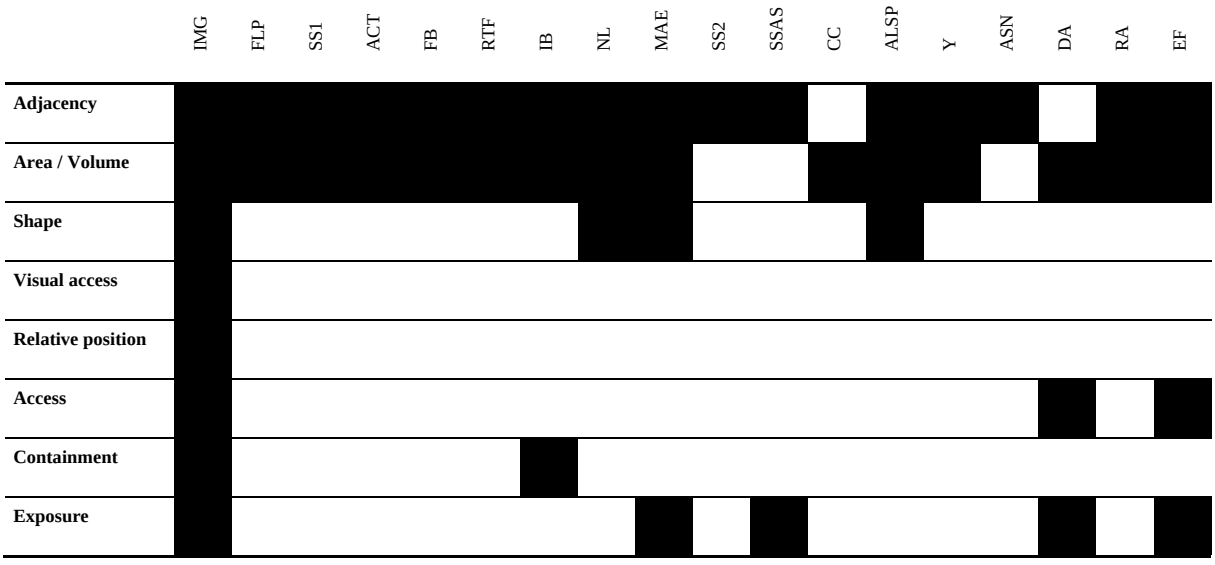


Table 18 shows a contrast between the number of objectives in IMG and in new prototypes. IMG has a list of seventeen objectives, which I simplified to eight features for analysis. Current systems are more abstract and focus on satisfying basic objectives such as adjacency and area/volume. In some of them, relations such as exposure or visual access can be customized by using the tools for adjacency. For example, SSAS uses pheromone markers to promote the exposure or privacy of a part of the activities to the exterior of the building and MAE uses a form of attraction to reduce the external surface of the building. Some systems have an explicit implementation of different objectives, such as the containment of the agents in a custom polygon (IB) or an objective function to evaluate accessibility and exposure (DA).

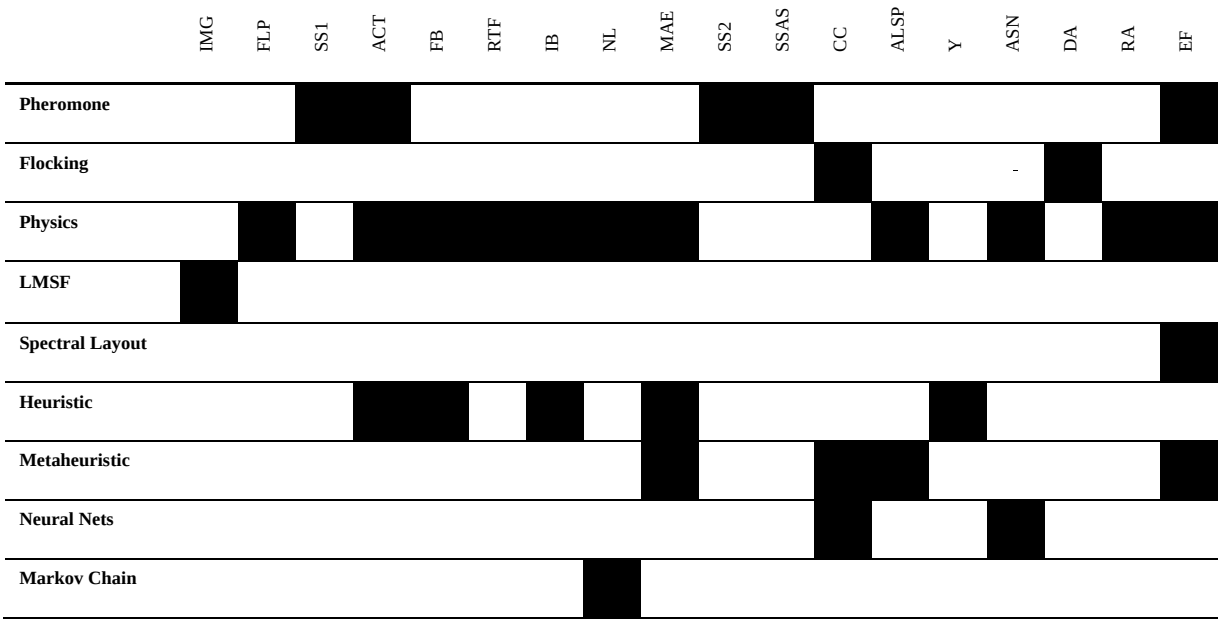
Table 18. Spatial objectives.



In terms of control procedures, most papers reference general ideas that inspired their approach – such as Cybernetics, Semiotics, Biology, etc. – and present a high-level description of the

algorithms, but they do not provide a formal description of the control procedures or share the code. Table 19 displays the large diversity of control procedures in the prototypes.

Table 19. Control procedures



Different types of physics simulations are present in most prototypes. They implement a custom version of attraction/repulsion forces (FB, IB and MAE), rigid body simulation (RTF, RA, and EF), soft body simulation (ALSP, NL, and ACT), or another unspecified implementation (ASN and FLP).

The second most frequent occurrence is pheromone routing. The algorithms are used to guide large spatial clusters (SS1 and SSAS), to control the vertices of polygonal entities (ACT), to guide agents that carry and drop spatial units (SS2), or to generate a weighted circulation network between the consolidated spatial units (EF). Flocking has only two occurrences. In DA, the system is a customized Boid model that generates semi-orthogonal networks with spatial agents. In CC, flocking is one of the actions in the policy of agents in a three-dimensional grid. Hybrid methods are used for all MASS types. They incorporate techniques outside of the scope of agent-based modeling, such as LMSF to guide the agent (IMG), metaheuristics to initialize or refine the solutions (MAE, ALSP and EF), neural networks to generate the adjacency between agents (ASN) or to define the policy of the agents (CC), and Markov Chain to generate new agents (NL).

In terms of evaluation, few papers analyze the performance or expression of the prototypes. One paper provides a brief convergence analysis (FB), some papers present a concrete architectural application (IMG and FB) and one prototype was applied to a real project and used in a workshop (SSAS).

Using correlations between features (Table 20), correlations between types and features (

Table 21) and the radar charts for the occurrence of features in the types (Figure 132), it is possible to visualize certain patterns. The occurrences of representational and control features present clear and distinct patterns for each MASS type. In the occurrence of objectives, all types overlap on adjacency and area.

Table 20. Correlation matrix for occurrence of features. For hybrid types, we divided the occurrence uniformly.

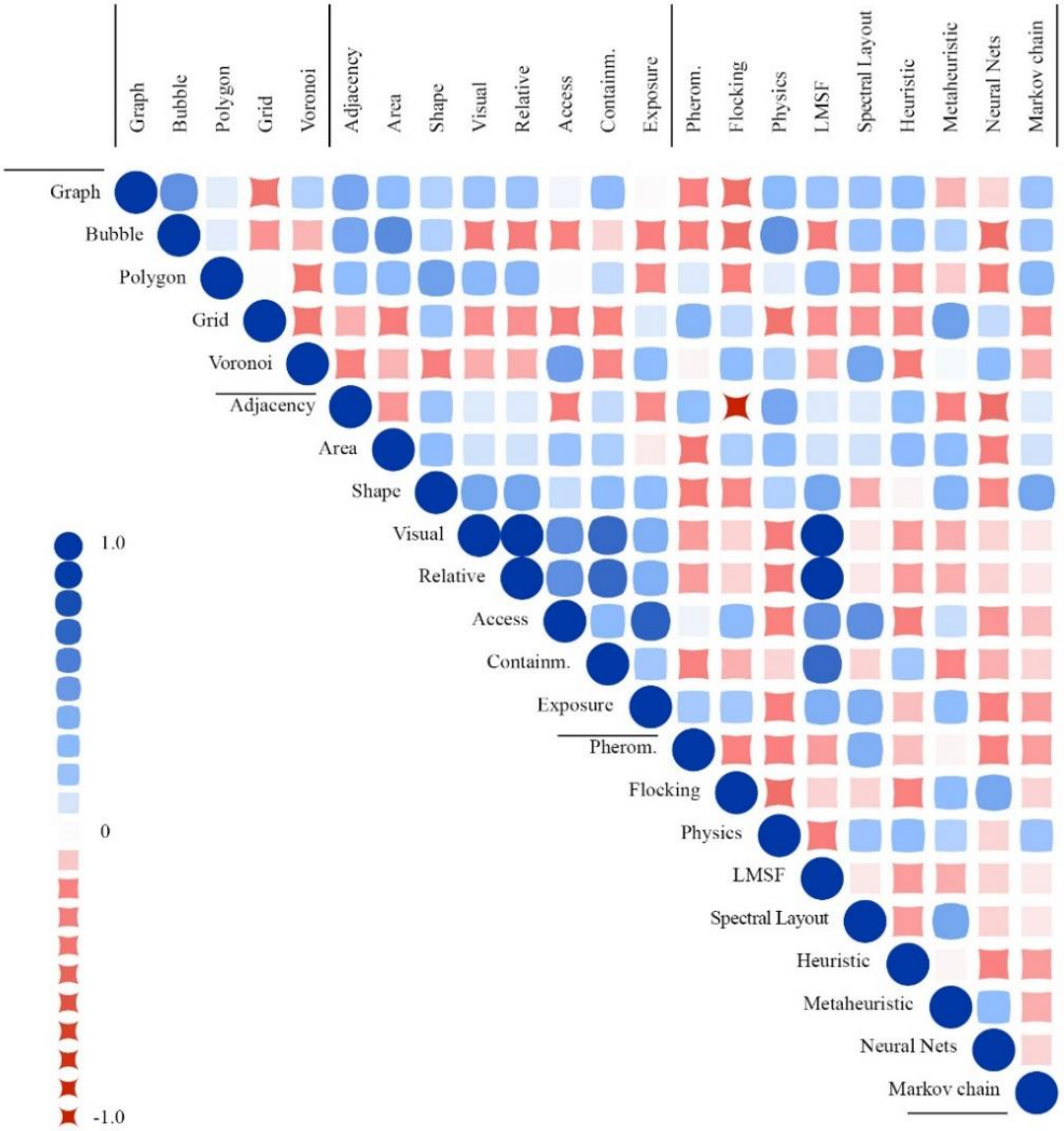


Table 21. Correlation matrix for relations between types and features. Blue is positive and red is negative. For hybrid types, we divided the occurrence uniformly.

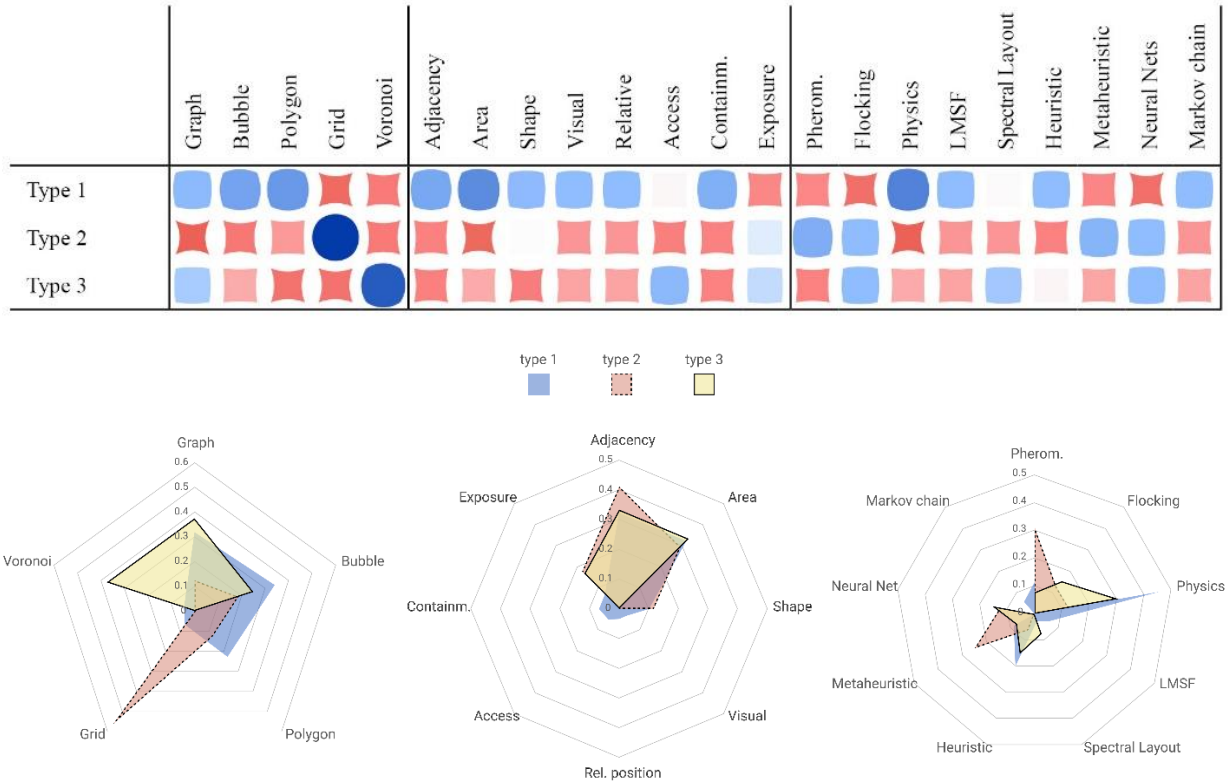


Figure 132 Radar charts with the relative occurrences of the features for each type. By Author and Jinmo Rhee.

Type 1 has strong occurrence and positive correlation with graphs, bubbles, polygons, and physics simulation. Bubbles are used to represent fluid moving shapes, while polygons mostly represent rigid shapes for packing (ACT is an exception and uses polygons for fluid shapes). Other than access and exposure, Type 1 has the strongest positive correlation with all features, even with the two objectives that occur in all types (adjacency and area), which are also strongly correlated to the main representational features of type 1 (graphs, bubbles, and polygons). The occurrence of adjacency and area in other types is mostly related to hybridization with type 1.

Type 2 has high occurrence and positive correlation with the grids, which are the representational support for correlated control strategies, such as pheromone routing (SS1, SS2, and SSAS), metaheuristic optimization (MAE and ALSP) or a neural network for multi-agent flocking (CC). Type 2 is slightly positively correlated with exposure, not correlated with shape, and negatively correlated with all other objectives.

Type 3 has a strong correlation with Voronoi diagrams, which translate a population of points to polygonal or polyhedral partitions. It also presents occurrences of graphs and bubbles due to the hybrids, but these correlations are respectively weak positive and negative. In terms of control, the significant positive correlations are related to flocking, spectral layout, and neural networks. In terms of objectives, it has the largest correlations with access and exposure among all types.

7.6. Appendix F: Deep Learning

7.6.1. Origins of Deep Learning

Deep learning is the subset of ML that relies on models that can represent and learn complex concepts by modeling them as a hierarchical representation of simpler concepts (Goodfellow, Yoshua, and Aaron 2016, 1–2), such as neural networks. Computational learning with neural networks has been an important area of research since the early days of computers, in Cybernetics and in the pioneering research on pattern recognition (Goodfellow, Yoshua, and Aaron 2016, 12–18; Nilsson 2009, 60–62; Cardon, Cointet, and Maziéres 2018b, 4–10). Overall, the connectionist AI tradition that neural networks belong to has been in constant tension and often in conflict with the cognitivist and symbolic tradition of AI, which largely relied on the pioneering work of Herbert Simon and Allen Newell.

Cardon, Cointet, and Maziéres (2018a) describes the emergence of deep learning in this tension between the symbolic and the connectionist traditions, which has largely defined the history of artificial intelligence. They provide a timeline and graphs showing the evolution and alternating influence of both paradigms in academia since the mid-1930s intermediated by two major AI winters (Figure 133). This timeline depicts a common historical interpretation of AI history from where it is possible to trace the recent advancements of Deep Learning.

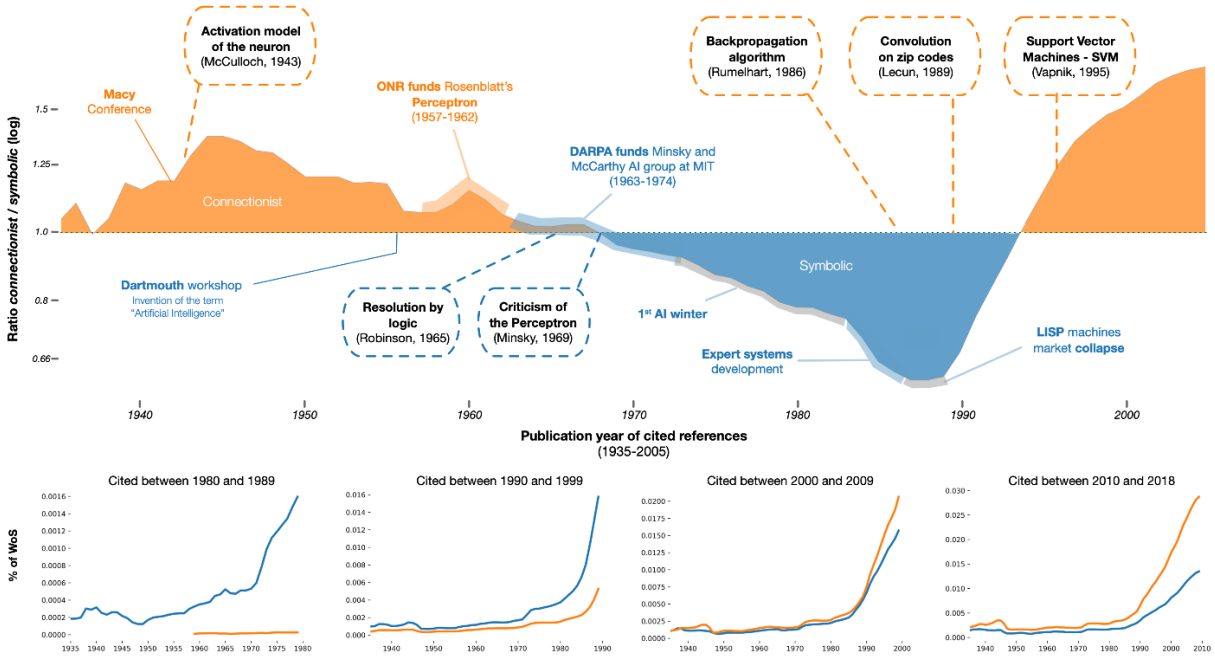


Figure 133 The evolution of the academic influence of the connectionist and symbolic approaches. "The main plot (top) represent[s] the evolution of the ratio between the number of cited publications in the "connectionist" corpus2 (orange) and the corresponding number in the "symbolic" corpus3 (blue), both normalised by the total number of publication indexed by Web of Science. The plots at the bottom represent for each corpus, the number of cited publications cited during the given period" (Cardon, Cointet, and Maziéres 2018a). Images licensed under CC BY-ND 4.0, https://creativecommons.org/licenses/by-nd/4.0/.

Russel and Norvig consider that Deep Learning emerged during and after the second AI winter (1987-1993), a period where the previous dominant paradigm – expert systems – reached a limit

and the research interest and funding radically shrank (2021, pp. 24–27). For example, they mention that AI researchers re-invented the back-propagation algorithm in the late 1980s, which revived the interest in the connectionist models of AI. Around the same time, there was an interest in establishing more scientific approaches to AI research. Many subfields, such as computer vision, robotics, speech recognition, multiagent systems, and natural language processing were gradually reunified with the "newfound appreciation for data, statistical model, optimization, and machine learning" (Russell & Norvig, 2021, p. 25).

Later, in the 2000s, advances in computing power and the creation of the world-wide web supported the availability of big data and the shift towards deep learning. Overall, deep learning models enable the identifying and extracting patterns from large datasets that accurately map from sets of complex inputs to good decision outcomes.

In the last decade, a new wave of AI based on DL has emerged with growing success of the application of machine learning with models that contain multiple layers to problems from different domains, such as neural networks. DL succeeded in a variety of tasks, outperforming experts in diverse areas. Some examples of these breakthroughs are: AlexNet (Krizhevsky, Sutskever, and Hinton 2012), the learning system for image classification proposed by Geoffrey Hinton and his team to the 2012 ImageNet competition; AlphaGo (Silver et al. 2017), a reinforcement learning model that was able to defeat human champions in the game of Go; and recently AlphaFold (Jumper et al. 2021), a model that predicts the 3-D structure of a protein based on its amino acid sequence.

7.6.2. From symbolic to learning models

Cardon et al. (2018a) coined the terms Hypothetical-deductive machines (1) and inductive machines (2) to refer to the two approaches paradigms of AI (Figure 134).

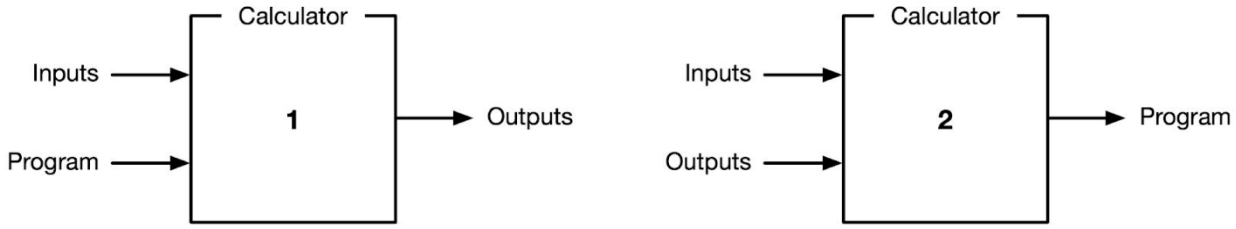


Figure 134 Hypothetical-deductive machines (1) and inductive machines (2) (Cardon, Cointet, and Maziéres 2018b, 3). Images licensed under CC BY-ND 4.0, https://creativecommons.org/licenses/by-nd/4.0/.

Hypothetical-deductive machines refer to the symbolic tradition of AI, where information processing is a fundamental model both for cognition and for problem solving methods. Typically, the model for this symbolic decision maker is formulated as a heuristic search used for the progressive transformation of symbol structures. Given a hypothesis about a problem, the designer can create a program with a systematic sequence of steps to map from certain inputs from that problem to solutions in the output. This is equivalent to the classic generative models in architectural design.

In contrast, the term inductive machine refers to the connectionist tradition of AI. Deep learning is a subset of this tradition, as it typically relies on the distributed computation of neural networks, which are loosely inspired by the distributed computation of the human brain, for creating a model based on experience. In this case, the designer does not create the program directly, but induces it from data related to the problem. More specifically, deep learning is part of machine learning (ML), a multidisciplinary field concerned with the design of computer programs that, instead of relying on an explicit implementation of its designer to address a task, learn it from experience. More specifically, Mitchell (1997, 2) considers that in ML "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, it its performance in all tasks in T, as measured by P, improves with experience E".

After defining a task T, a performance measure P, and some form of training experience E, it is necessary to define a target function to be learned. By mapping the input of the task to some form of solution in its output, this target function represents the knowledge that is being learned. Target functions can comprehend many tasks, from a chess-player program that uses the input value x of the board to generate a play as a value y to a generative model that receives a site as the input x and generate a spatial configuration of a house represented by y. In the case of deep learning, hierarchical models such as neural networks are used to represent the target function and specific algorithms are developed to train them.

In Russel and Norvig's map of artificial intelligence, learning is represented by an additional category of the structure of a learning agent. This category is not presented as an opposition to the previous four standard AI agents (Figure 110 and Table 13), but as an additional layer that enables the computer to induce some of the agent components from experience. Besides the basic architecture of agents with sensors and actuators, the general learning agent has four basic components (Figure 135):

1. The **performance element** corresponds to the internal structure of the agent, which reads the percepts and uses its program to select actions.
2. The **critic** receives an external performance standard as the input and provides a feedback signal on the performance of the agent in the current task
3. The **learning element** is responsible for making improvements in the performance element; it receives the feedback both from the performance element and from the critic, and it can make changes in any of the knowledge components in the structure of the previous agent models.
4. The **problem generator** receives the learning goals from the learning element and suggests exploratory actions to the performance element that can lead to a better performance in the long run.

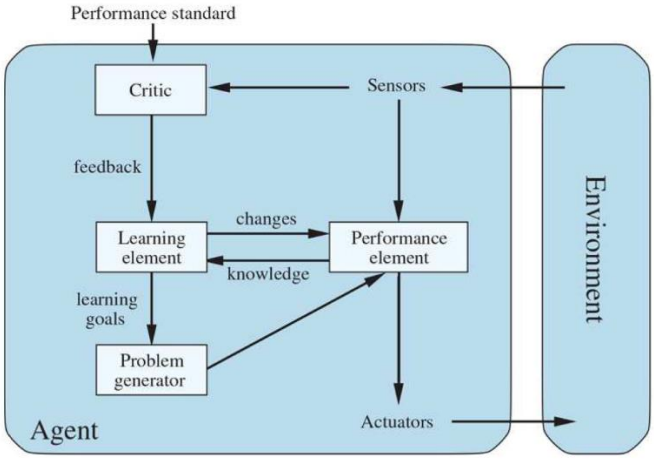


Figure 135 Structure of a general learning agent (Russell and Norvig 2021, 56). Reproduced by permission of Pearson Education, Inc.

7.6.3. Neural Networks for inference

A neural network is a parametric model vaguely inspired by the human brain and by its capacity to change its own pattern according to different experiences from the world. It is an adaptive function approximator that relies on a hierarchical representation for learning features without the need of a human to hardcode it. In contrast to parametric models used in architecture, which largely rely on a graph with geometric operations, neural networks are composed of a graph with differentiable mathematical parameters, operations, and functions.

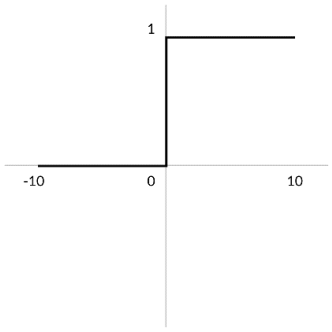
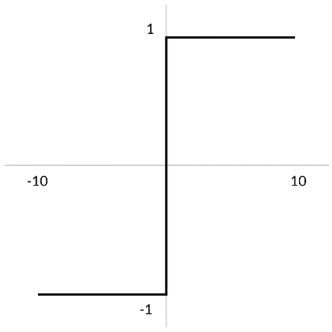
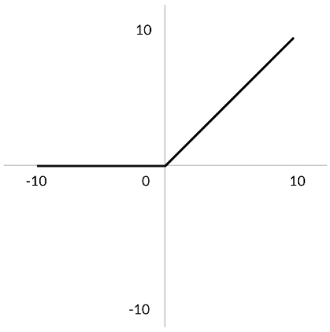
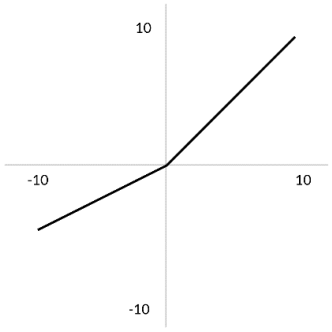
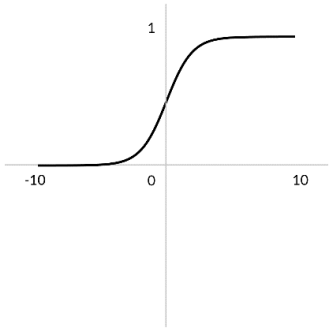
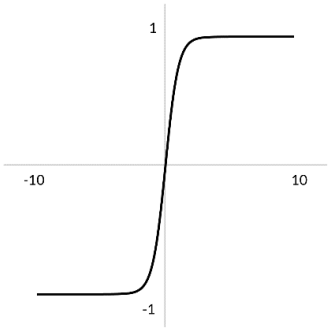
These operations are commonly referred to as nodes and organized in layers, which are connected to the units of the following layer. The first is the input of the neural network, the last is the output, and any intermediate layer is referred to as a hidden layer. The sequence of layer enables the creation of deep neural networks and the hierarchical construction of internal features. To use the neural network for mapping a given input to an output, the variables and operations will be processed according to the order and the dependencies of operators in the graph, which is called a forward propagation. As the parameters, such as weights and bias, are adjusted based on the data, the neural network can adapt and approximate the desired target function.

There are many neural network architectures with custom components, layers, and patterns of connections to address different data and operations.

A typical neural network is organized as a feed-forward sequence of fully connected layers derived from earlier research on perceptron. Except for the input, each layer l contains a matrix-vector multiplication $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$. $\boldsymbol{W}^l$ is a matrix with the weights of the linear equation for the current layer, $\boldsymbol{a}^{l-1}$ is the output of the previous layer, and $\boldsymbol{b}^l$ is a vector with the bias terms for the linear equation. The linear combination is followed by a nonlinear activation function g, which can include functions such as a step function, rectified linear unit (ReLU), leaky rectified linear unit (leaky-ReLU), Sigmoid function, Hyperbolic tangent function (Tanh), etc. It results in the layer output $\boldsymbol{a}^l = g^l(\boldsymbol{z}^l)$, which enables the network to approximate non-linear functions. See

examples of activation functions in Table 22.

Table 22 Activation functions

Step Function	Sign Function	ReLU
		
$g(z) = \begin{cases} 0 \text{ if } z < 0 \\ 1 \; if \; z \geq 0 \end{cases}$	$g(z) = \begin{cases} -1 \text{ if } z < 0 \\ 0 \; if \; z = 0 \\ 1 \text{ if } z > 0 \end{cases}$	$g(z) = \begin{cases} 0 \text{ if } z < 0 \\ z \; if \; z \geq 0 \end{cases}$
Leaky-ReLU	**Sigmoid**	**Tanh**
		
$g(z) = \begin{cases} cz \text{ if } z < 0 \\ z \; if \; z \geq 0 \end{cases}$, with $c \in (0,1)$	$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

Other types of networks can use other patterns of connection between parts and other types of operations to address geometric or spatial information. For example, convolutional neural networks (CNN) replace the fully connected linear combinations of the standard neural network by convolution operators[23] for tasks involving images or any grid-based data. This operator restricts it to the Moore neighborhood of each position in the grid, over all channels, taking advantage of the pre-defined structure of the data. Therefore, instead of a large matrix of weights plus a bias vector, it uses an arbitrary number of kernels with $w \times h \times d$ weight parameters, each with its own bias term. Each kernel is convoluted with every $w \times h \times d$ sub-region of the input data, resulting in a new 2-D channel of the output that captures local features of the input. With n kernels, the output of a CNN layer is a feature map composed of 2-D data with n channels. The

[23] While the official name of the operation is a convolution, it is more accurately described as a cross-correlation (Zhang et al. 2021). For convenience, I will still use convolution when I refer to it.

2-D dimensions of the output depend on the parameters of the convolutions, such as the size of the filter, stride, or padding.

Convolutional layers typically combine the convolution with operators such as max pooling, which only preserves the maximum value of each $w \, x \, h$ region of each input channel, reducing the resolution of the output data. Over multiple layers, the convolutional network can hierarchically build more and more complex spatial feature extractors that depend on the ones in the previous layers. This enables the CNN to capture different 2-D spatial features of the input and can even be extended to larger dimensions ().

There is a large variety of operations that can be added to the neural networks and many other network architectures that can address other types of data. For example, Recurrent Neural Networks (RNN) have internal memory and operators that can handle sequential data with variable length. Graph neural networks generalize the convolutions layers from structured representations such as images to non-Euclidean domains such as graphs and manifolds networks and meshes.

7.6.4. Training neural networks

To train a neural network to approximate the target function, deep learning usually relies on a loss function, and the combination of a backpropagation algorithm with an optimization algorithm.

The loss function measures how well the neural network approximation is performing according to the task and the data. It is typically a differentiable function that maps the input and output of the neural network to a single value indicating error. If many examples in the dataset are used as the input, this function usually return the average error.

The gradient information of the loss function with respect to each trainable parameter of the network to can be used adjust them to reduce the error represented by the cost function conditioned on the current input and output data of the neural network.

It is possible to use black-box optimization methods, such as genetic algorithms, to search for neural network parameters that result in better approximations of the target function. However, in deep learning, the optimization algorithm is commonly a variation of the gradient-descent algorithm, which uses the gradient information of the loss function with respect to each trainable parameter of the network.

It iteratively adjusts the neural network parameters toward the opposite direction of the gradient, which aims at locally reducing the error represented by the cost function based on the current input and output data of the neural network. As this process is repeated using samples from a dataset, the optimization algorithm will iteratively fine tune the neural network in the steps toward the opposite direction of the gradient. Moving the parameters towards the descent of the gradient is a local strategy to minimize the loss function, which is equivalent to improving the performance of the network in approximating the target function.

In one extreme, the vanilla gradient descent algorithm computes the average error using all

examples from the batch at every iteration, what takes advantage of vectorization but has a theoretical cost of $O(n)$ and is not resilient to redundant data. In another extreme, stochastic gradient descent samples a single example from the dataset at every iteration, which reduces the computation to $O(1)$ but does not use the full power of vectorization. In between these extremes, there is minibatch stochastic gradient descent, which accelerates computation by sampling minibatches from the dataset at every iteration. Other variations, such as Momentum, Adagrad, or Adam, use the history of the previous updates and/or custom per-parameter and per-timestep updates, to adapt the size of the learning steps to ill-conditioned loss landscapes.

Gradient descent and its variations require a form of automatic differentiation to calculate the gradient of loss function with the respect to the neural network parameters. Once the neural network and the cost function are differentiable, a backpropagation algorithm traverses the neural network in the reverse order of the computational graph, using the chain rule of calculus to incrementally calculate the gradients of each intermediate parameters.

With the loss function and an optimization algorithm, the neural network can be systematically trained with batches of the data and approximate the desired target function, which can then be used with forward propagation.

7.6.5. Types of Machine Learning

Research in ML and DL comprehends a vast area of applications in different fields. These tasks can be conceptually divided in three main branches: supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning the experience is represented by a fixed dataset with labeled input-output pairs (x_i-y_i) that are generated by un unknown target function f(x) to be approximated by the learning model. It includes tasks such as classification and regression. The labeled dataset with pairs of input and output elements works as an instructor to tell the model what function to approximate (Goodfellow, Yoshua, and Aaron 2016, 105). Part of the data is removed from the training and used to test if the function can generalize to new data. The training relies on an error function that measures the difference between the known output (target) and the output predicted by the neural network, given the input, such as mean squared error (MSE). Then, an optimization algorithm changes the internal parameters of the neural network to minimize this error and, consequently, the network approximates the target function.

For example, in VGG16 (Simonyan and Zisserman 2015) (Figure 136), a convolutional neural network was trained for classifying an input image in 1000 classes of objects, such as teapots, Siberian huskies, and churches. After successful training, the network classifies the input image by using a series of layers with using convolutions and other operations, which result in increasingly complex internal representations. These representations are called feature maps and they are the knowledge used by the model to classify the input. In the first layers, these feature maps might capture abstract patterns such as the occurrence of horizontal lines. Closer to the end, they can capture more sophisticated shapes of the objects to be classified.

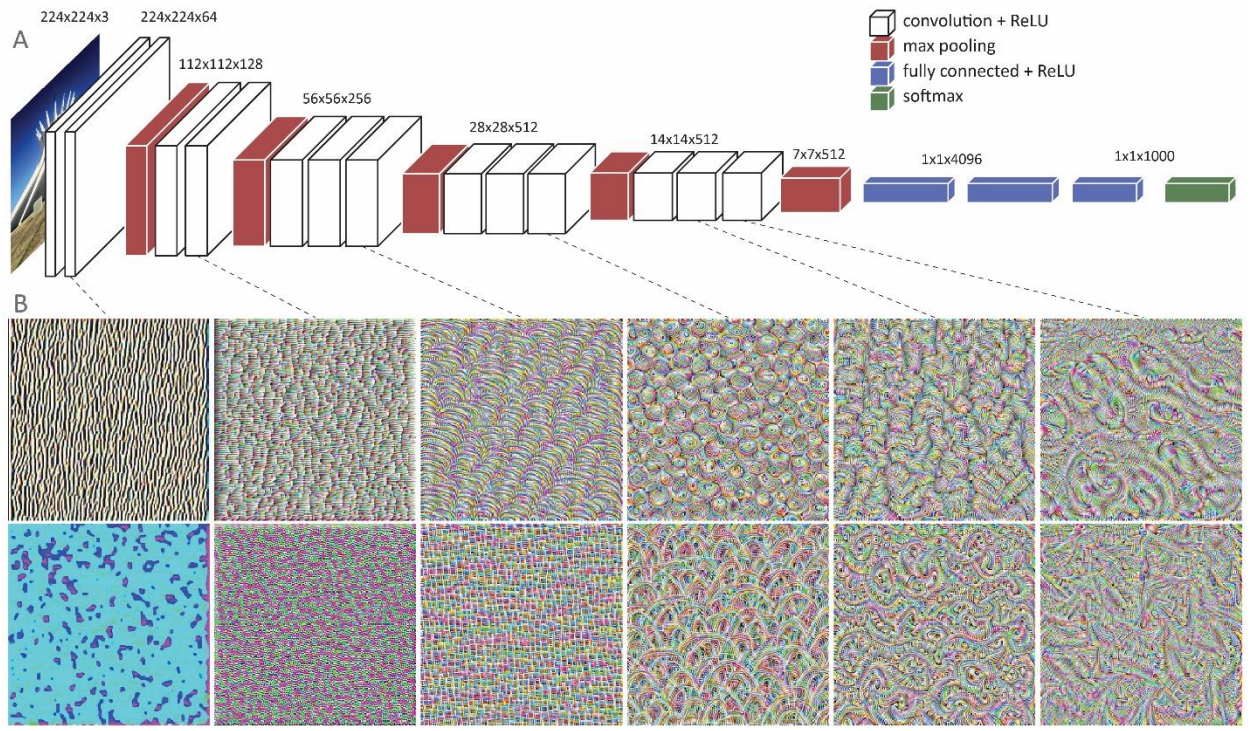


Figure 136 A: architecture of VGG-16; B: every column contains examples of input images that activate two features of different layers of the network (indicated by the dotted line). By author, with code based on (Ozbulak 2017).

In unsupervised learning the agent learns patterns from the input data without any external feedback or label. This Includes tasks such as clustering, dimensional reduction, and generative modeling. The latter will be described in the following section.

In Reinforcement learning an agent learns how to take sequential decisions in a task environment based on reward signals. This branch will be explored in detail in section later in this chapter.

7.6.6. Generative Learning

Generative learning is a new category in the taxonomy of generative models. It comprehends techniques where the generative model is directly learned from data. This is an open area of research in design and has been an important trend in the computational design community (Joyce and Nazim 2021; Leach 2021; As and Basu 2021). The most notable examples of techniques for generative learning, such as deep dream, neural style transfer, autoencoder, and GANs.

Deep dream (Mordvintsev, Olah, and Tyka 2015) was originally invented to investigate what the neural network "sees" when it receives a given input image. This is a hybrid model, as it uses an optimization for synthesis. It incrementally changes the pixels of an input image to maximize the activations of one or multiple feature maps, so the patterns of the selected feature maps start to emerge like a hallucination. While this process typically starts with an image with random noise, a real photo and drawing can be used and transformed. For example, I used this algorithm to change an image of the Cathedral of Brasilia, designed by Oscar Niemeyer (Figure 137: A). I selected a feature map in the last convolutional layers to generate a more complex pattern of hallucination.

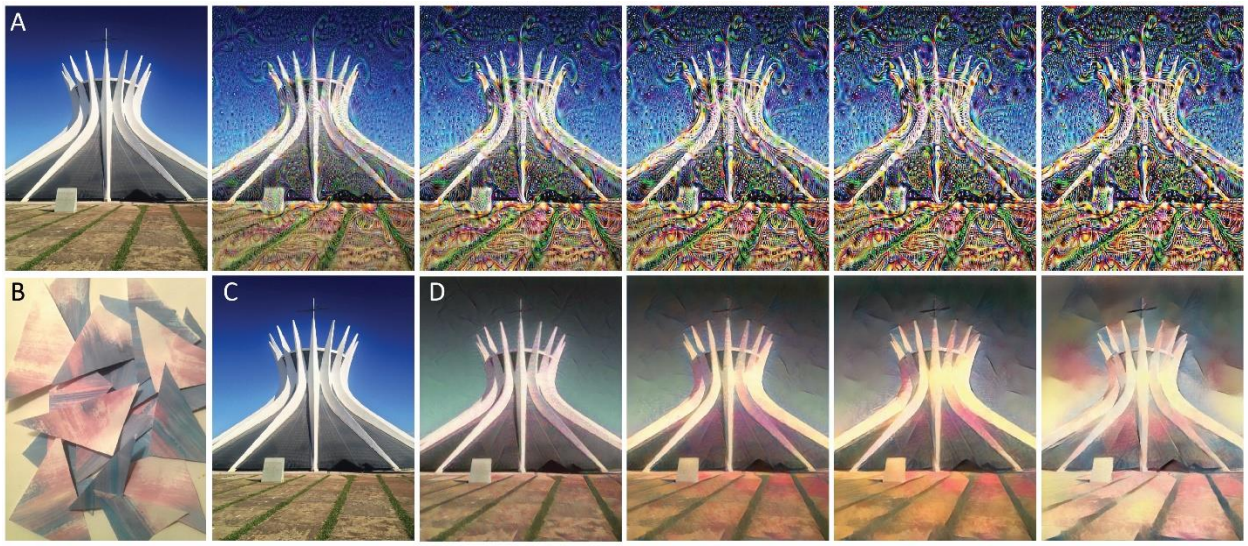


Figure 137 A: applying deep dream to an image of the Cathedral of Brasilia. Image by Author, with code based on (Ozbulak 2017). B-D: Custom style transfer (based on the code from Jacq n.d.). B: Style image used and reproduced with permission of Kate McLean. C: content image by Author. D: content image incorporating style. By Author.

In the case of neural style transfer (Gatys, Ecker, and Bethge 2016), optimization is associated with the trained neural network to create and image that has abstract qualities of a style image and the concrete qualities of a content image. In this case, different feature maps are used as a mechanism to measure the similarity between the synthesized image with the original content image and the style of the original style image. Optimization is used to change the pixels and minimize these differences, which results in a stylized image (Figure 137: B, C, and D).

While these examples are interesting, they are still only hybrid optimization models that rely on the knowledge embedded in a model trained for classification. On the other hand, there is a subset of unsupervised learning that addresses the synthesis of data: generative modeling[24]. Generative modeling in machine learning consists of explicitly or implicitly learning the distribution of a certain dataset, which enables sampling synthetic examples from it (Bishop 2006, 43; Foster 2019). Generative modeling is one of the current frontiers in machine learning and has been responsible for many of the advances in art, image, and video synthesis. There are several generative models in ML, including but not limited to: Autoencoder and Generative Adversarial Networks.

An Autoencoder (AE) is a model composed of two parts: an encoder and a decoder (Foster 2019) (Figure 138: A). The encoder learns how to compress the samples of a data distribution into points in a lower-dimensional representation (latent space) or into a normal distribution around this point[25] in the case of a Variational Autoencoder (Goodfellow, Yoshua, and Aaron 2016, 696–99;

[24] While there is an overlapping, it is important to distinguish the idea of generative models for design from generative models in machine learning.

[25] Usually, it is assumed that there is no correlation between the dimensions, so the encoder can output the vectors with the mean and the variance (or the logarithm of the variance).

Foster 2019). The decoder learns how to reconstruct the original input based on the observed values in this compressed representation. They are trained simultaneously to compress and reconstruct each of the samples in a dataset with a consistent representation, such as images or point-clouds. After training, the decoder can be decoupled and used to synthesize new output data based, using the compressed vector as the input parameters.

In the context of generative models, autoencoders can be trained to reconstruct a set of design examples with certain characteristics. For example, in DeepCloud (Bidgoli and Veloso 2018) (Figure 138), an Autoencoder reconstructs point clouds from datasets of chairs, cars, hats, and tables. After training, DeepCloud captures structural aspects of the dataset in the space of the compressed vector. Therefore, the user can manipulate the parameters of the decoder to reconstruct the design examples in the dataset or to create novel or hybrid designs based on different parameter values.

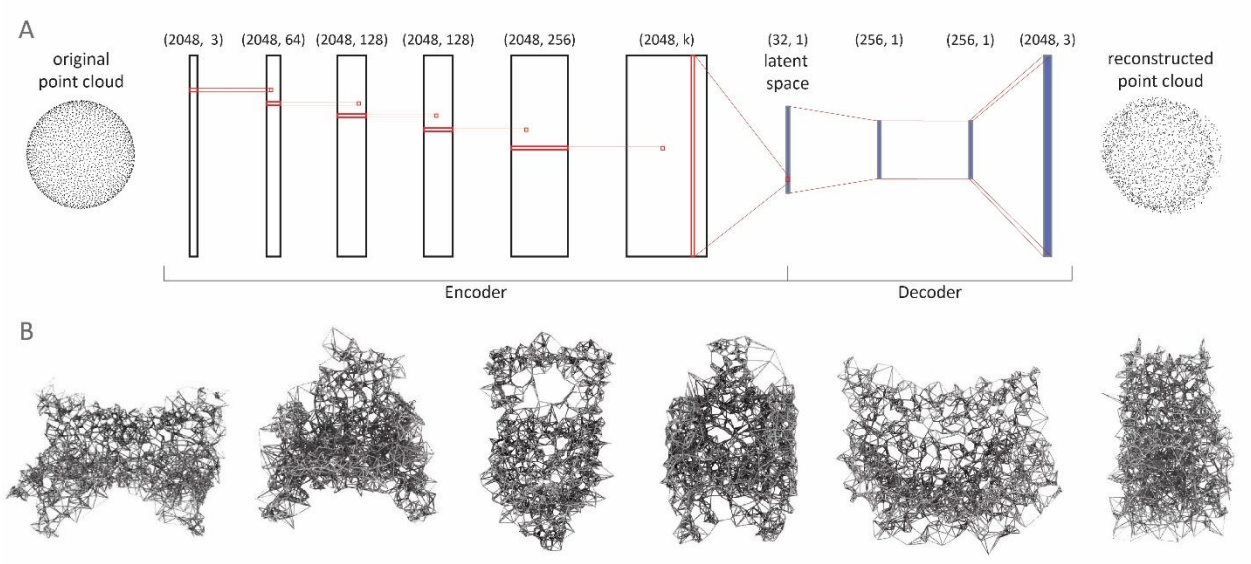


Figure 138 Deepcloud. A: architecture of autoencoder trained on a dataset of point clouds of chairs. B: monstrous chair series generated by exploring the parameters of the latent space as input. By Author.

In contrast to autoencoders, which learn a lower-dimensional representation, Generative Adversarial Networks are generative models created with the primary purpose of synthesizing new data that look like sampled from a dataset (Goodfellow et al. 2014; Foster 2019; Goodfellow, Yoshua, and Aaron 2016, 699–703). GAN architecture leverages two adversarial neural networks, a generator, and a discriminator. The discriminator is trained with the samples from a dataset and learns to test if a given input belongs to it. The generator has no access to the dataset and should learn how to generate samples to trick the discriminator. As a result, the generator masters the task of synthesizing data that fits in the distribution of the database.

After training a GAN with a certain dataset of design alternatives, the generator can synthesize new design alternatives with characteristics like the ones in the dataset (Figure 139). The input parameters of the generator are typically a random vector used to promote variations on the

synthesized data, but with the extensions of the model (Hindupur n.d.), conditional information or even user input can be used for additional control. Recently, GANs have been widely applied to architectural design to analyze and generate building representations, such as floorplan layouts (Zheng 2018; W. Huang and Zheng 2018; Chaillou 2019; Newton 2019; Nauata et al. 2020; Carta 2021).

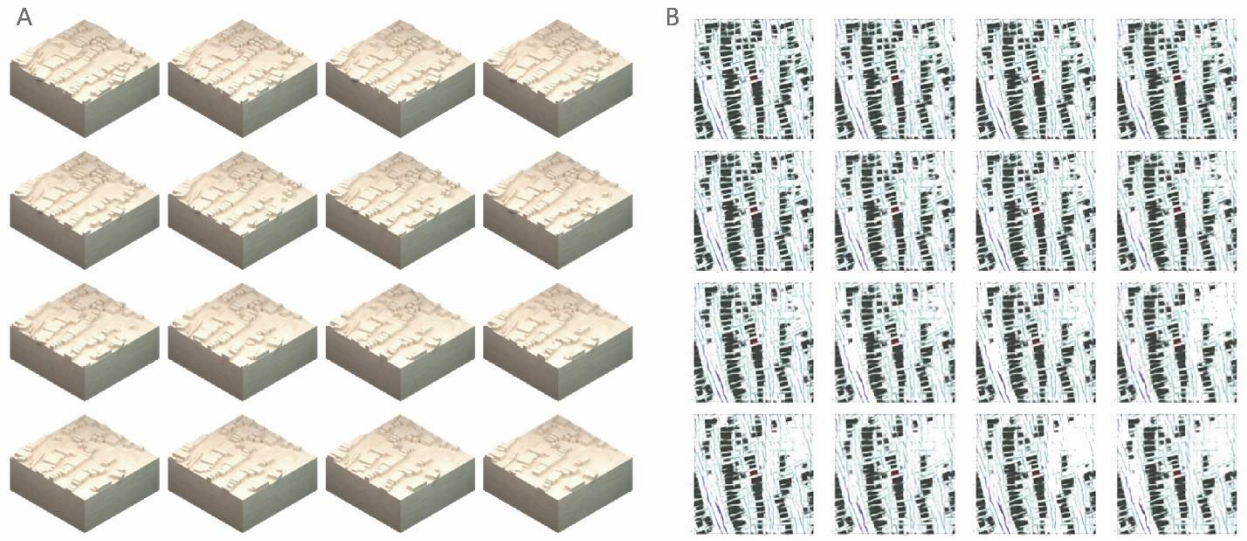


Figure 139 Urban Structure Synthesizer: 3D models (A) and 2D diagrams (B) with incremental variations of a urban fabric. The diagrams were generated with a model (WGAN-GP) trained on a dataset of images from Pittsburgh, USA. Reproduced with permission of Jinmo Rhee. Work produced for Generative Models for Design (CMU, Fall 2019) and published (Rhee and Veloso 2021).

Generative learning emphasizes the capacity of computational models to use datasets to capture design patterns and qualities that are often hard to formalize, which are then used to map abstractions, such as categorical images, graphs, textual description, to architectural forms, such as floorplan layouts. While this interaction typically relies on a single mapping, it is possible to train multiple generative models to create a multi-step pipeline (Chaillou 2019). Nevertheless, even in the case of the multi-step pipeline, the design process is still structured by a pre-defined and coarse decision structure based on representation mapping with the designer outside of the generative loop.

7.7. Appendix G: Reinforcement learning

7.7.1. The goal of a RL agent

RL addresses sequential decision problems that can be formalized as a MDP. This includes a variety of tasks, such as board games, video games, autonomous driving, robotic control, HVAC control, recommendation systems, trading stocks, portfolio management, etc. The overall goal is to teach a RL agent how to perform well in any of these tasks with methods that commonly rely on interacting with the environment and trial and error.

The agent starts at certain initial state of the environment. It uses its current policy to select the action, which triggers an update in the environment by the transition function and in the reward function. Then, the environment sends an observation and a reward signal to the agent. This process continues over time, until the episode is ended, generating a trajectory τ in the form of a sequence of tuples

$$\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), (s_2, a_2, r_1) \ \dots \ (s_T, a_T, r_T)$$

The cumulative reward over an episode is captured by the return function

$$R(\tau) = r_0 + r_1 + \ \dots \ + r_T = \sum_{t=0}^{T} r_t$$

While some tasks in RL are episodic and have a clear end, others do not break into episodes and do not have time limits, so they are called continuing tasks (Sutton and Barto 2018a, 54). In the latter case, the rewards are potentially infinite. To restrict this infinite horizon, there is a discount factor $\gamma \in [0,1]$ that changes how future rewards are valued. It is used as a weight for the reward for the cumulative rewards from the next step on, what recursively results in a weight with the delta time as the exponent. On one extreme, if $\gamma = 0$, the agent is completely shortsighted, and the return only considers the current reward signal. If $\gamma = 1$, the return function is equivalent to its formulation without γ. Typically, this $\gamma \in [0.8, 1.0)$.

$$R(\tau) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_1 \dots + \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_1 \dots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$$

So, more formally, the goal of the RL agent is to learn a policy that maximizes the expected cumulative reward by interacting with the environment.

$$J(\tau) = \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

This goal relies on the hypothesis that any purpose in the interaction of the agent with the environment over time can be "thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)" (Sutton and Barto 2018a, 53). This reward hypothesis is often used to support the argument that RL constitutes a unified framework for artificial general intelligence. The idea is that the maximization of simple reward signals is the

enough to drive behaviors in different tasks related to intelligence, such as knowledge, learning, perception, social intelligence, language, generalization, and imitation (Silver et al. 2021).

While the RL agent is usually trained in a simulation to maximize the expected cumulative reward of the MDP, depending on the problem it can be deployed to the real world, using the learned policy with real sensors and real actuators, such as in robotics.

7.7.2. Challenges in reward functions

The agent learns from the dynamics of the environment, with its two functions. While the transition function $P(s_{t+1}|s_t, a_t)$, describes the probabilities that underly the model of the world, the reward function $\mathcal{R}(r_t|s_t, a_t, s_{t+1})$ provides the feedback signal that the agent is trying to maximize over a time horizon. The design of good reward functions is an open problem in RL and it includes many challenges.

In the canonical approach to RL, the signals produced by the reward function should not communicate how designers expect the agent to behave or with subgoals designed to facilitate learning but only an assessment of the agent's progress towards the real goal that they expect it to achieve (2018a, 53–54, 469). Without behavioral bias from the designer, the agent can potentially discover optimal behaviors for the task at hand. However, while the interpretation of goals as reward signals might be easier in simple and well-defined tasks, goals involve design preferences, different interpretations, multiple criteria, or even undefined components, which make the design of the reward function challenging.

Besides, even if it is straightforward to associate the goals with the rewards, identifying the state-action pairs that are tightly related to the success or failure with the respect to the goals and assigning an informative reward signal can be hard, especially for complex tasks and ill-defined problems. Besides, even it is possible to systematically define proper values for the reward function and to identify the key state-action pairs to trigger its signals, these pairs may be few, so the information provided to the agent toward a goal can be infrequent; this is the problem of reward sparsity.

Let's consider some common reward function types derived from basic RL problems.

Board games such as tic-tac toe, chess, or go are two-player games that can be modelled as zero-sum games (Table 23). This means that they are purely competitive games where the gained resources of the winner are equal the resources losses by the other player i.e., they sum to zero. Typically, in RL, the reward function for these games assigns a constant value to the winner and the negative of that value to the loser at the end of the game.

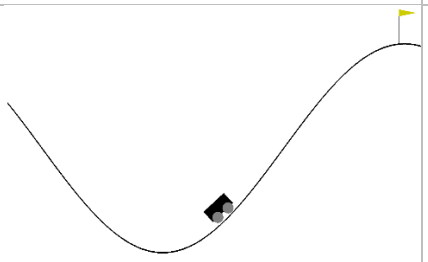
In other toy problems from RL, such as mountain car or cartpole, there is a task that should be solved by a single agent in an episode with a fixed number of steps.

In mountain car (Table 23) there is a car with a weak engine, which can move it to the left or right on the screen. It starts at the bottom of valley, and it should be able to move to the top of the hill on the right, where there is a target represented by a flag. To reach the peak with the target, the

agent should learn to move to combine continuous movements to the left and right with the gravity. To represent the urgency of the task, the reward function assigns a -1 value to the agent to every step. If the agent reaches the target before the maximum number of steps, the reward function returns 0. Therefore, the longer the agent takes to reach the target or, in the extreme, if it does not reach the target until the maximum number of episodes, it will accumulate more negative reward signals.

In cartpole (see Table 23) a cart can move along a flat line representing a frictionless track and it has an un-actuated pole attached to its body by a pin joint. The agent can control the cart by applying a force of +1 or -1. At the beginning, the pendulum starts upright, and if the falls over an angle that is more than 15° from vertical the episode is over. The cart should try to be close to the center of the track, so if it is further than 2.4 units from the center, the episode is also over. In contrast to mountain car, where the agent tries to end the episode as far as possible, the goal here is to keep the pole balanced and the cart close to the center as long as possible – i.e., until the max number of steps is reached.

Table 23 Examples of reward functions

Task	Image	Reward functions
games such as tic-tac toe, chess, or go.		$r = \begin{cases} +c \; if \; agent \; wins \\ -c \; if \; agent \; loses \\ 0 \; elsewhere \end{cases}$
mountain car.		$r = \begin{cases} 0 \; if \; agent \; wins \\ -c \; elsewhere \end{cases}$
cartpole.		$r = \begin{cases} 0 \; agent \; loses \\ +c \; elsewhere \end{cases}$

RL typically learns by interacting with the environment and trying different actions, so the intermediate rewards signals of these examples do not provide valuable indication of partial achievements that can lead to the final goal. In the first example, the only informative reward

signals happen at the end of the episode, indicating victory or loss. In the other cases the reward signal is not informative during the episode because all state-action pairs result in the same value. However, the number of timesteps as the episode is over will determine how much positive or negative reward the agent accumulated.

To successfully learn in these environments, the agent can wander for longer periods over multiple episodes without any indication of progress. If the agents cannot succeed it won't propagate any informative reward signal to the earlier stages of the simulation to create a proper policy.

Besides, associated with the sparse rewards is the problem of credit assignment: how to distribute credit for every action in the sequence of decisions that resulted in the success or failure of the agent when it receives an informative reward?

7.7.3. Challenges in deep reinforcement learning

The structure of RL, based on the interaction of the agent with the environment imposes in many challenges for deep learning that are non-existent in other branches, such as supervised learning. The table below synthesizes the differences, some of which were based on ideas from (Sutton and Barto 2018a, 195–96; Graesser and Keng 2019, 18–21).

Table 24 Supervised Learning x Reinforcement Learning

	SL	RL
Data and algorithm coupling	Dataset generated independently from algorithm	Dataset is generated by agent during training
	Static dataset	Agent changes the environment, which affects future decisions
Access to information	Access to labels (ground truth) for every sample	Access to reward signal during interaction
		Reward signal is typically sparse
	All the training data is used for training	Agent only access part of the environment states during training
Stationarity	Target function is stationarity	Target function is nonstationary
Credit assignment	Success/error refers to decision over current data	Delayed reinforcement of decisions
Loss functions	Loss function is directly related to the set of labeled examples	Reward signals are used to construct a long-term objective
Parallelization	Learning is parallelizable	Sequential nature of agent-environment interaction restricts parallelization

These challenges show that RL is still a challenging field of research compared to other domains such as SL, so its applications tend to be restricted to domains where it is easy to generate huge amount of simulated data, such as board games, video games, and robotics.

7.7.4. Reward Engineering and Shaping

The design of the reward function in RL is itself an active area of research.

One common approach is to break the assumption that the reward function should not have components that add behavioral bias or incentivize agents to move towards success faster. This is a common practice in RL that is called reward engineering or reward shaping (Russell and Norvig 2021, 807). It consists of augmenting the reward function with informative non-sparse reward

functions for RL based on domain knowledge and qualitative goals. This augmented reward function $\mathcal{R}'$ typically relies on an original function $\mathcal{R}$ and additional shaping reward function F with signals that make the original function smoother and help guiding the agent towards success (Equation 2).

$$\mathcal{R}'(s_t, a_t, s_{t+1}) = \mathcal{R}(s_t, a_t, s_{t+1}) + F(s_t, a_t, s_{t+1}) \quad \text{Equation 2}$$

In the simple case, F contains signals in additional transitions to indicate sub-goals and reinforce the path towards the original goal. However, in some cases, this can also distract the agents from the original goal, leading to trajectories in the new MDP that result in more cumulative rewards than achieving the original one.

Ideally, in reward shaping the decision policy should be invariant under the modifications of the reward function, so that optimal policy the new MDP $M' = (S, A, T, \mathcal{R}')$ corresponds to the optimal policy in the original $M = (S, A, T, \mathcal{R})$. If the policy invariance can be ensured, an agent with an optimal policy in M' will achieve the goal represented by $\mathcal{R}$ in the original M. To guarantee that the F does not create cycles that move the agent away from the optimal paths to the goal and preserve policy invariance, it can be defined as a potential-shaping reward function. In this case F is the delta of a real-valued function Φ of the states in the MDP, modified by the discount factor (Equation 3).

$$F(s_t, a_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t) \quad \text{Equation 3}$$

Other guidelines for reward shaping (Bonsai 2017; Bilgin 2020, 315–18) include keeping the reward in a fixed range, such as [-1, +1], creating a smooth reward function with gradual feedback, correlating the rewards with the observation, expressing the reward and state with terms that are generalizable to variations of the problem, ensuring that reward incentives do not outweigh the original reward signal, ensuring that reward and terminal condition do not conflict, and using time-step penalties if behavior should incorporate urgency. Also, it is possible to start with the reward function with augmented signals to accelerate learning and gradually remove the augmentation over training to ensure that the policy is compatible with the original MDP (Sutton and Barto 2018a, 470).

In the extreme case, the reward signal is designed based solely on the state information, so all the agent needs to do is to find the actions to get to and eventually stay at a better state over the time horizon. Heess et al (2017, 14) uses a set of reward functions based on state information for the locomotion of three distinct rigid-body creatures in a simulated environment with diverse terrains: a Planar walker, a Quadruped, and a Humanoid. These reward functions encode the idea that the agent should move along the x-axis, so they are mainly associated with the velocity of the body on this axis, but they also contain terms in the reward related to the specific characteristics of the bodies and the control applied to the joints. The main motivation of the paper is to show that combining a generic reward signal with a curriculum of increasingly challenging environments

enables the agent to learn complex behaviors, such as jumping.

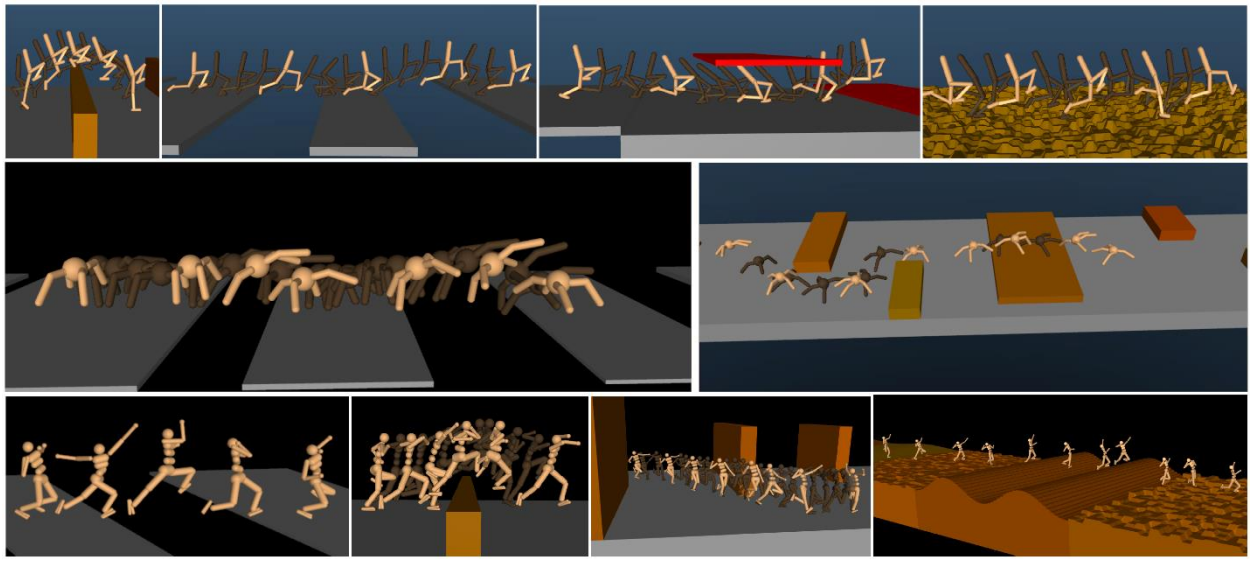


Figure 140 Time-lapse sequences of Planar Walker, Quadruped, and Humanoid. Source: Figures 4, 5, and 7 in (Heess et al. 2017, 6–8). Reproduced with permission of DeepMind Technologies Limited.

Table 25 Reward functions for rigid body creatures

Agent	Reward functions
Planar walker	$r = 10.0v_x + 0.5n_z - \lvert\Delta_h - 1.2\rvert - 10.0I[\Delta_h < 0.3] - 0.1\lVert u\rVert^2$
Quadruped	$r = v_x + 0.05n_z - 0.01\lVert u\rVert^2$
Humanoid	$r = min(v_x, v_{max}) - 0.005\left({v_x}^2 + {v_y}^2\right) - 0.05y^2 - 0.02\lVert u\rVert^2 + 0.02$
	v_x: velocity along the x-axis v_y: velocity along the y-axis v_{max}: cutoff for the velocity set to 4m/s y: position on the y-axis n_z: projection of the torso coordinate frame onto the z-axis ∈ [-1.0, 1.0] u: control applied to the joints Δ_h: height of the torso above the feet

It is also the case for the two-step deep reinforcement learning (RL) framework to learn 3D modeling policies developed by (Lin et al. 2020). This framework trains two agents that use primitive and mesh modeling as a sequential and hierarchical strategy to approximate a target shape O. To measure the quality of the actions of the agent that models with primitives, it relies on a reward function that combines the ratio between the intersection and the union between the modeled and the target shape with a term that counts the removed primitives.

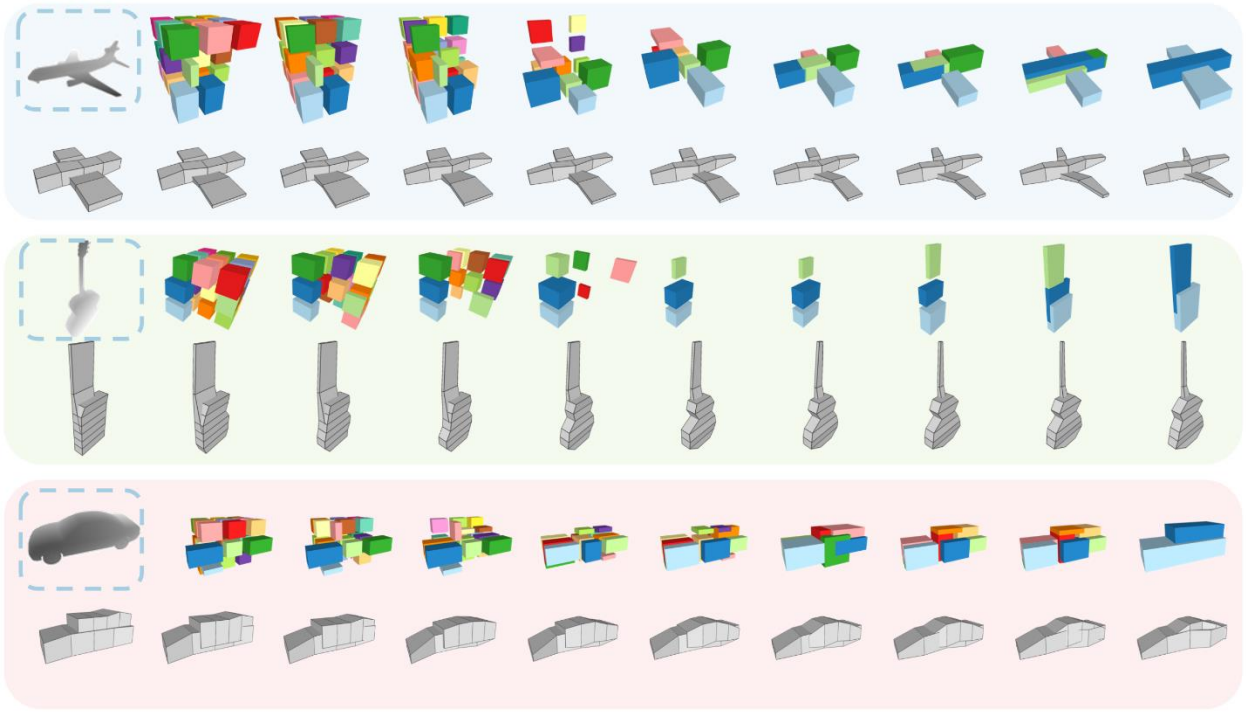


Figure 141 The step-by-step procedure of 3D shape modeling. The first row of each sub-figure shows the Prim-Agent approximating the target shape by operating the primitives (steps 5, 10, 20, 40, 60, 80, 100, 200, and 300). The second row shows the Mesh-Agent approximating the target by mesh editing (steps 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100). Source: Figure 7 in (Lin et al. 2020, 11). Reproduced by permission from Springer Nature: Springer Computer Vision – ECCV 2020 Modeling 3D Shapes by Reinforcement Learning, Lin et al. © 2020 Springer Nature Switzerland AG (2020).

Table 26 Reward function for agents that model 3-D shapes with primitives

Reward function at k^{th} step	$R_k = \left(I_1^k - I_1^{k-1}\right) + \alpha_1\left(I_2^k - I_2^{k-1}\right) + \alpha_2(N^k - N^{k-1})$
Global IoU	$I_1 = IoU\left(\cup_i P_i, O\right)$
Local IoU	$I_2 = \frac{1}{K}\sum_{P_i \in \bar{P}} IoU\left(P_i, O\right)$
	N^K: number of deleted primitives at k^{th} step IoU: intersect over union O: target model P_i: one of the primitive shapes in the model $\cup_i P_i$: union of the primitive shapes in the model $\bar{P}$: set of primitives not deleted yet K: number of primitives not deleted yet

7.7.5. Single-agent RL Algorithms

In this section I will refer to a subset of RL methods, which will support the construction of the research framework. I use and cite multiple sources to support my description, but I mostly rely on the taxonomies in (Sutton and Barto 2018a; Graesser and Keng 2019; Morales 2020).

There are different RL methods to train an agent to maximize the expected cumulative reward in the task environment. Some of these methods rely on basic concepts and ideas developed for the tabular case in RL, while others transpose these ideas to the deep learning setting. Here, I organize these methods around the three main functions that are learned or used by RL agents: value functions, policies, and a dynamics model of the environment (Figure 142). Each of these functions and their combinations support different classes of methods for learning the agent policy. Besides, in the extremities of the diagram, they also intersect with adjacent methods that are often not considered as part of RL, such as policy optimization and planning.

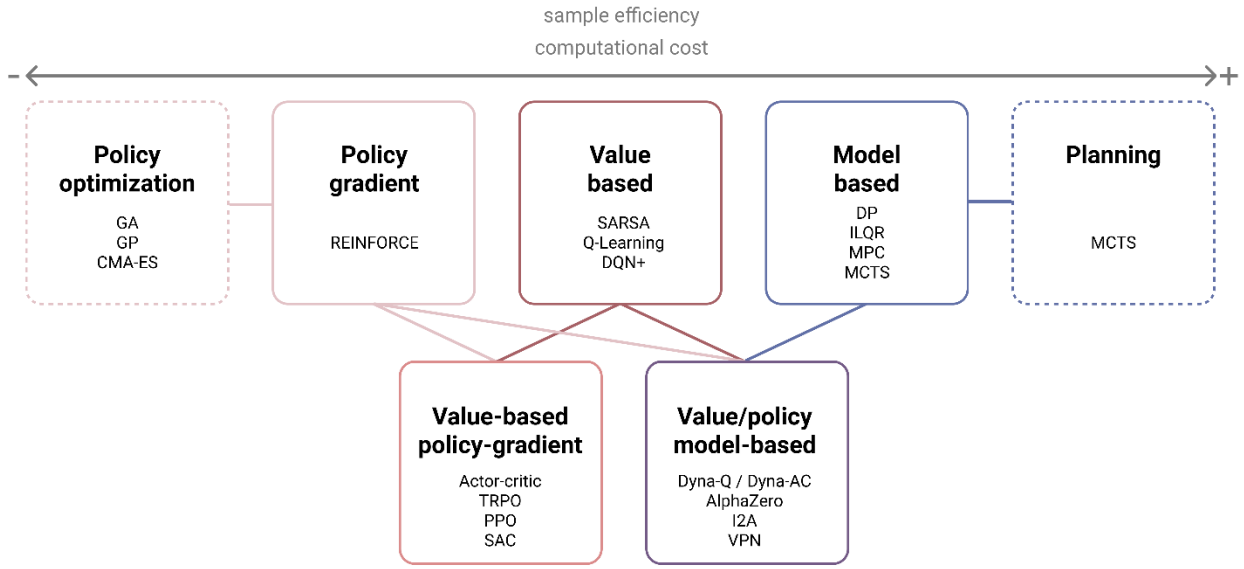


Figure 142 Classification of RL, DRL, and adjacent methods based on the function learned/used. By Author, inspired by (AI 2018; Graesser and Keng 2019, 11; Causevic 2020; Morales 2020, 26).

The following sections present a set of algorithms based on this taxonomy. Each of these algorithms will be briefly described and analyzed. In order to provide a high-level and unified language to compare the algorithms, I use a diagram with the main components of the algorithms connected according to their specification — Figure 143. The diagram contains four nodes indicating the different parts of the algorithm. There is a node for a value function (V), which can be either a V or Q-value. There is a node for the policy function (π). There is a node for the trajectory (τ) and a node for the dynamics model (M). The existence of the nodes and the potential connection between them depend on the configuration of the specific algorithm. Overall, this diagram is an abstraction, so there is some flexibility in the meaning of the nodes and edges, which will be specified over the next sections.

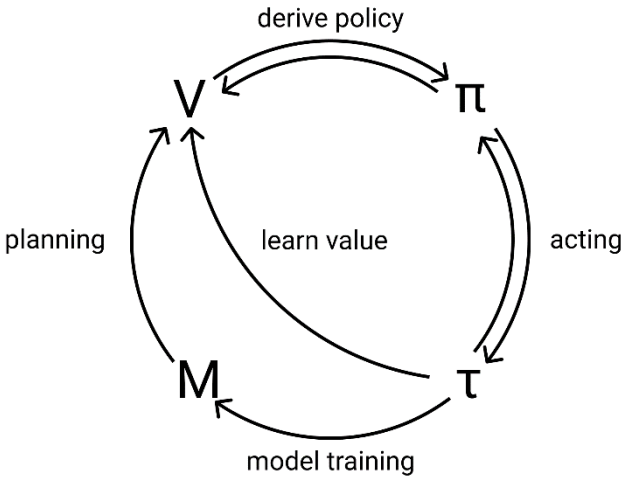


Figure 143 Template diagram with the anatomy of RL algorithms. Developed by the author based on (inspired by (Sutton and Barto 2018b, 162). By Author.

7.7.5.1. Value-based methods

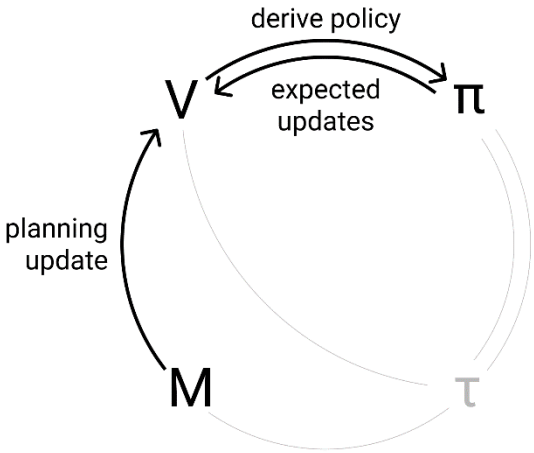


Figure 144 Diagram of Dynamic Programming. By Author.

Dynamic programming (DP) is one of the theoretical foundation of value-based methods for reinforcement learning (Sutton and Barto 2018a, 73–74). DP was developed by Richard Bellman as a mathematical tool for multi-stage decision processes based on their decomposition into a sequence of smaller problems (Bellman 1957). In RL, DP comprehends a collection of algorithms that compute approximations of a value function – i.e., a function that estimates $J(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)]$ – to support a search for good policies.

There are two variations of the value function, which are dependent on a given policy π. The first one is the state-value function $V^{\pi}(s)$. It maps a given state of the MDP to the expected cumulative reward considering that the agent follows a certain policy π. The second is $Q^{\pi}(s, a)$ and it maps a state action pair – i.e., the situation where the agent is at a state s and takes the action a – to the

expected cumulative reward as the agent follows a certain policy π.

$$V^{\pi}(s) = \mathbb{E}_{s_0=s,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \quad \text{Equation 4}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0=s,a_0=a,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \quad \text{Equation 5}$$

A fundamental property of the return function in a MDPs is that, for a given policy, it satisfies a recursive relationship with the returns of the following steps. The Bellman equations express this recursive relationship for both value functions. In the case of the state-value function, the relationship between value of a certain state and the values of the successor states. In the case of the action-value function, the relationship between the Q-value of an action-value pair and the Q-values of the successor pairs.

$$\begin{aligned} V^{\pi}(s) &= \sum_{a}\pi(a|s)\sum_{s',r}P(s',r|s,a)\left[r+\gamma V^{\pi}(s')\right] \\ &= \sum_{a}\pi(a|s)\sum_{s',r}P(s',r|s,a)\left[r+\gamma\sum_{a'}\pi(a'|s')Q^{\pi}(s',a')\right] \end{aligned} \quad \text{Equation 6}$$

$$\begin{aligned} Q^{\pi}(s,a) &= \sum_{s',r}P(s',r|s,a)\left[r+\gamma\sum_{a'}\pi(a'|s')Q^{\pi}(s',a')\right] \\ &= \sum_{s',r}P(s',r|s,a)\left[r+\gamma V^{\pi}(s')\right] \end{aligned} \quad \text{Equation 7}$$

DP relies on two steps: the computation of the value function for a given policy (policy evaluation) and the computation of an improved policy given the value function of that policy (policy improvement).

The algorithms based on DP build an approximation of the desired value function, which is initialized arbitrarily (except for the terminal states, which should be 0). For the policy evaluation step, the Bellman equation is used as an update rule that will iteratively update the estimates for the value-function of a state by bootstrapping from the estimate values of a successor state combined with the information from the dynamics. While it is possible to use asynchronous strategies in DP, bootstrapping is typically computed in systematic sweeps through the state space, so the value-function approximation at timestep t is used to build the values for the approximation at timestep $t+1$.

For policy improvement, DP algorithms updates the previous policy of the agent by selecting the best performing action for every state with the updated value function approximation – i.e., by deriving a greedy policy. According to the policy improvement theorem, the new policy is as good or better than the original one, except for the case where the policy is already optimal (Sutton and Barto 2018a, 76–80). With the value function V the agent only has direct access to the value of

states, so it must consider the next state s from every available move and the transition probability of stochastic environments. In contrast, the state-action function Q learns from the larger domain of state-action pairs (s, a), but it enables the selection of the best expected action at a certain state with a query over the argmax among the available actions.

$$\pi'(s) = argmax_a Q^{\pi}(s,a)$$
$$= argmax_a \sum_{s',r} P(s', r|s, a)\,[r + \gamma V^{\pi}(s')] \qquad \text{Equation 8}$$

The interaction between these two processes is called generalized policy iteration (GPI) and is the base for many RL methods (Sutton and Barto 2018a, 86–87). When the policy iteration stabilizes, the value function is consistent with the current policy. When the policy improvement stabilizes, the policy is greedy with the respect to the current value function. When both processes stabilize, the policy should be optimal for the MDP

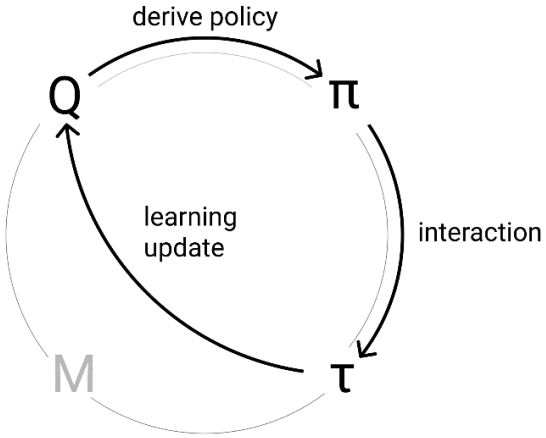


Figure 145 Diagram of model-free and value-based RL. By Author.

In contrast to DP, which relies on the dynamics of the environment for computing one-step expected updates of the value function, model-free methods perform the general steps of policy evaluation and policy improvement without access to the model.

In terms of policy improvement, model-free algorithms must adopt a policy that balances exploitation of the current estimates with continual exploration of the state space. To address this exploration-exploitation tradeoff, they use strategies that combine policy greedification with the respect to the current value estimates and randomness. For example, the ε-greedy policy selects random actions with probability $\varepsilon \in [0,1]$ or the greedy action according to the current estimates with probability $1 - \varepsilon$. The greedy step exploits the current best estimates, and the random step ignores the estimates and explores new trajectories, which can potentially lead to better returns. At the beginning of a training session, ε is a large value for exploration, then it is reduced over training, so the agent switches to the exploitation of the best estimates.

The policy evaluation step of model-free and value-based algorithms relies on learning the value

function from interaction with the environment. Usually, they update the estimates for the value-function by bootstrapping from the estimate values of a successor state as they interact with the environment, which referred to as Temporal Difference Learning (TD-Learning) (Sutton and Barto 2018a, 119–40). Variations of TD-learning also allow the use n-steps bootstrapping.

The Q-function is favored over the V-function for model-free algorithms because the argmax query eliminates the need of the knowing or the dynamics or estimating it by one-step look ahead computation during execution. Based on this recursive relationship of the Bellman equation, the Q-values of non-terminal states can be initialized with a constant or a random value, and as the agent interacts with the environment and generates trajectories, it can bootstrap the Q-values with the reward received during the transition (s, a, s') and the biased estimate of the $Q^{\pi}(s', a')$ discounted by γ.

The two expectations in the Bellman equation should be considered for the TD-updates of the Q-values.

If the environment is deterministic, this single update will always use the same transition, but if it is stochastic, the outer expectation and the transitions of the agent depend on the probability of $T(s'|s, a)$. The Q-values represent expectations that are built with updates from multiple trajectories as the agent interact with the environment, so the outer expectation can be replaced by real transitions, which can approximate the transition distribution over time.

In the case of the inner expectation, the update relies on the distribution of the policy π and the probability of selecting a specific action a' at state s'. To address this expectation, the different value-based algorithms use different formulations of the TD-target for the update of the estimations (

Table 27).

Table 27 Update rules for algorithms based on the Q-function

Sarsa	$\boldsymbol{Q^{\pi}(s, a) \approx r + \gamma\, Q^{\pi}(s', a')}$	Equation 9
Expected Sarsa	$Q^{\pi}(s, a) \approx r + \gamma \sum_{a_i'} \pi(a_i'\|s')\, Q^{\pi}(s', a_i')$	Equation 10
Q-learning	$Q^{\pi}(s, a) \approx r + \gamma\, max_{a_i'} Q^{\pi}(s', a_i')$	Equation 11
Deep Q-Networks (DQN)	$Q^{\theta}(s, a) \approx r + \gamma\, max_{a'} Q^{\theta^-}(s', a')$	Equation 12
Double Deep Q-Networks (DDQN)	$Q^{\theta}(s, a) \approx r + \gamma\, Q^{\theta^-}\left(s', argmax_{a'} Q^{\theta}(s', a')\right)$	Equation 13

Sarsa uses the next action that was selected by the agent in the production of the trajectory for

the update. This is the reason that it is called an on-policy method. In contrast, Expected Sarsa and Q-learning are off-policy methods. Expected Sarsa derives the probability distribution over action from the policy to explicitly calculate the expectation. Q-learning relies on the Bellman optimality equation, so it uses the maximum Q-value in the next state for the update, which is equivalent of assuming that a greedy policy was followed on that step.

Algorithm 6 Q-learning based on (Sutton and Barto 2018a, 131)

1: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
2: Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
3: Loop for each episode:
4: Initialize s
5: Loop for each step of episode:
6: Choose a from s using policy derived from Q (e.g., ε-greedy)
7: Take action a, observe r, s'
8: $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \, max_{a_i'} \, Q(s', a_i') - Q(s,a)\right]$
9: $s \leftarrow s'$
10: until s is terminal

Sarsa and Q-learning are originally tabular methods, which store the estimated value of every state-action pair (Q-values) explicitly in a table. However, it is possible to extend them to the DRL setting by using neural networks to approximate the Q-functions by using the TD-target as a loss function.

While deep learning setting removes some theoretical guarantees of convergence from the tabular setting, it enables the application to more complex problems in larger MDPs. For example, DQN was developed by DeepMind based on tabular Q-learning and deployed to train neural networks to play Atari games based on the high-dimensional sensory input (Mnih et al. 2013; 2015). The same algorithm was trained with raw images of the games, surpassing the performance of previous algorithms, and achieving human performance across a set of 49 games.

In Deep Q-Networks (DQN), neural networks represent the Q-function, so the change in the estimates can be achieved by changing their parameters. More specifically, DQN uses two deep neural networks with the same architecture (the on-line network Q^{θ} and the target network Q^{θ^-}) to estimate the Q-values. The target network Q^{θ^-} is basically a version of Q^{θ} that is only updated periodically for the stability of the training.

In the interaction step, the agent uses an ε-greedy policy and the on-line network Q^{θ} to generate observations (s, a, r, s') in the environment. The observations are added to a large memory buffer D and later retrieved for training. The buffer randomizes the data, removing the correlation between observation samples and reducing the variance for the updates.

For the policy evaluation, the algorithm updates the current estimate of Q^{θ} with a batch of observations (S, A, R, S') retrieved from the replay buffer D, which increases sample efficiency and enables the use of parallelization. The target network Q^{θ^-} is used for the estimation of the max

Q-value of the state $s'.Q^{\theta}$ is trained with a gradient descent method using the mean squared error (MSE) between its estimate and the TD-target based on Q^{θ^-}. The TD-step in DQN treats the target as used as a temporary ground truth for the policy estimation, which results in an update like in regression, but with a target that is nonstationary and that depend on the parameters of the neural network.

Algorithm 7 Deep Q-learning with Experience Replay - based on (Mnih et al. 2013, 5)

1: Initialize replay memory D to capacity N
2: Initialize action-value function Q with random parameters θ
3: for episode = 1, M do
4: Initialize sequence s_1
5: for t = 1, T do
6: With probability ϵ select a random action a_t
7: otherwise select $a_t = max_a Q^{\theta}(\varphi(s_t), a)$
8: Execute action a_t in simulation and observe reward r_t and s_{t+1}
9: Store transition (s_t, a_t, r_t, s_{t+1}) in D
10: Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
11: Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q^{\theta^-}.(s_{j+1}, a') & \text{for nonterminal } s_{j+1} \end{cases}$
12: Perform a gradient descent step on $(y_j - Q^{\theta}(s_j, a_j))^2$
13: Every c time steps $\theta^- \leftarrow \theta$
14: end for
15: end for

There are many variations of DQN that attempt to improve its different components. For example, in Double DQN (DDQN) (van Hassel, Guez, and Silver 2016) the max Q-value estimated by Q_{θ}- in the TD-target is replaced by the estimate of Q_{θ}-of the argmax of the online network for the same state, to prevent over-estimation. Other variations are synthesized and analyzed in (Hessel et al. 2018).

7.7.5.2. Policy-gradient

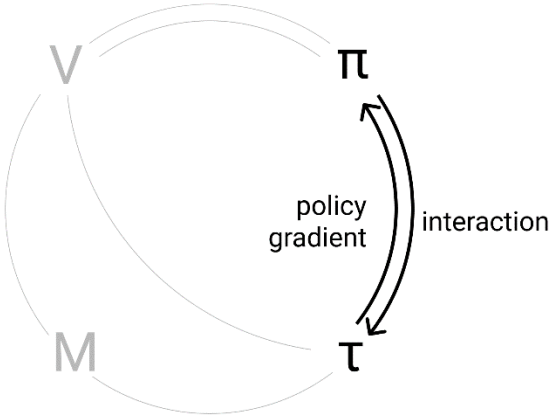


Figure 146 Diagram of Vanilla Policy Gradient. By Author.

Policy-gradient methods rely on learning the policy π, which allows an agent to take the decisions over time by directly reacting to the observations. A policy can be either deterministic or stochastic. In the former case, the function maps a state to a single action. In the latter case, the function maps a state to a probability distribution from which the selected action should be sampled. There are many advantages in learning a parameterized model for the policy instead of the value-function or the transition. First, discovering a good policy is the final goal of RL, so policy-based methods avoid additional steps, such as learning other aspects of the environment. It can address continuous action spaces and it enables the injection of prior knowledge in the form of the policy. Besides, the algorithm based on policy gradient was originally developed for parameterized models, such as neural networks, and not to the tabular setting, so it is a natural choice for DRL.

The foundation for policy-based methods is the algorithm REINFORCE (Williams 1992), which learns a parameterized policy $\pi_\theta(s)$ that is represented by a neural network. With different parameters, a neural network with a fixed architecture can represent different policies. Good policies are represented by neural networks with parameters that produce trajectories that increase the expected cumulative return $J(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$. Potentially, having the gradient of this function J with the respect to the network parameters ($\nabla_\theta J(\pi^\theta)$) would enable improving the performance of the policies by changing the neural network parameters towards the gradient ascent. However, it is not possible to differentiate J because it depends on the effects of the policy on the state distribution, which include parts of the environment, such as the transition T or the reward function $\mathcal{R}$, that may be unknown or may not be differentiable. To address this problem, the policy gradient theorem derives an identity of that gradient that is differentiable (Williams 1992; Sutton and Barto 2018a, 324–26; Graesser and Keng 2019, 26–28).

$$\nabla_\theta J(\pi^\theta) = \mathbb{E}_{\tau\sim\pi^\theta}\left[\sum_{t=0}^{T} R_t(\tau)\nabla_\theta log\pi^\theta(a_t|s_t)\right] \qquad \text{Equation 14}$$

This identity is referred to as the policy gradient and it shows that the expected sum of the log gradients of the actions a_t from the trajectory τ sampled by the neural network π^θ multiplied by the corresponding return of the trajectory $R_t(\tau)$ is equivalent to the gradient of the original function. With this policy gradient estimate, it is possible to update the parameters of the neural networks to increase the probability of actions that are associated with good returns and decrease the probability of the bad ones.

In contrast to value-based methods, which typically rely on bootstrapping, REINFORCE uses Monte Carlo to iteratively sample trajectories using the current policy (Algorithm 8- line 4). Then, for each step of the current trajectory, it calculates the return (line 7) and computes the policy gradient (line 8). Finally, the algorithm updates the policy network parameters using a step weighted by a parameter α towards the sum off all the policy gradients calculated in the trajectory (line 10).

Algorithm 8 REINFORCE – based on (Sutton and Barto 2018a, 328; Graesser and Keng 2019, 31)

1: Initialize learning rate $\alpha > 0$
2: Initialize the parameters $\theta \in \mathbb{R}^d$ of a policy network π^θ
3: for each episode in a max number of episodes:
4: Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_t, a_t, r_t$, following $\pi_\theta(\cdot \mid \cdot)$
5: Set gradient $\nabla_\theta J(\pi^\theta) \leftarrow 0$
6: Loop for each step of the episode $t = 0, 1, \ldots, T$:
7: define return $R_t(\tau) \leftarrow \sum_{t'=t}^{T} \gamma^{t'-t} r_t$
8: accumulate gradient $\nabla_\theta J(\pi^\theta) \leftarrow \nabla_\theta J(\pi^\theta) + R_t(\tau) \nabla_\theta \log \pi^\theta(a_t|s_t)$
9: end for
9: Perform a gradient step on $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi^\theta)$
10: end for

REINFORCE is an on-policy algorithm because every iteration relies on a single trajectory generated by Monte Carlo sampling with the current policy for updating the neural network, which is an estimate without bias – i.e., it uses the actual return from the interaction. However, it results in high-variance, because of the potential differences in the episodes and of the stochasticity in the production of the trajectory, due to the policy and transition functions (Graesser and Keng 2019, 32). Also, it is an on-policy algorithm, so it is sample inefficient, because the experience data generated with a policy must be discarded after updating that policy. As a result, policy gradient algorithms need a lot of data to converge.

One way to reduce the variance in policy gradient algorithms is to replace absolute return values by values relative to an action-independent baseline (Graesser and Keng 2019, 32–33). This can be achieved by subtracting the return of the trajectory by different baselines that do not vary with the action selection, so the new term theoretically collapses to zero in the gradient. Examples of the baselines include a, the mean reward, or even the state value of initial state. Additionally, the term can also be replaced by an advantage function (Graesser and Keng 2019, 135), which is defined by the difference between the action-value and state value functions: $A_\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

7.7.5.3. Actor-critic algorithms

To improve policy gradient methods and reduce its variance, a common approach is to combine it with value-based methods. In this context, a natural candidate for a baseline for policy gradient methods is to use a parameterized state value function V. The policy network is called the actor because it selects the action, while the value-function is called the critic, because it critiques the actions – i.e., it provides additional feedback for training the actor as a baseline. In comparison to the return estimated by Monte Carlo sampling in REINFORCE, the critic introduces bias has a low variance. Besides, the V function can bootstrap reward information and create a learned informative signal that can reduce uncertainty, reduce sparsity of rewards, and accelerate learning (Graesser and Keng 2019, 135).

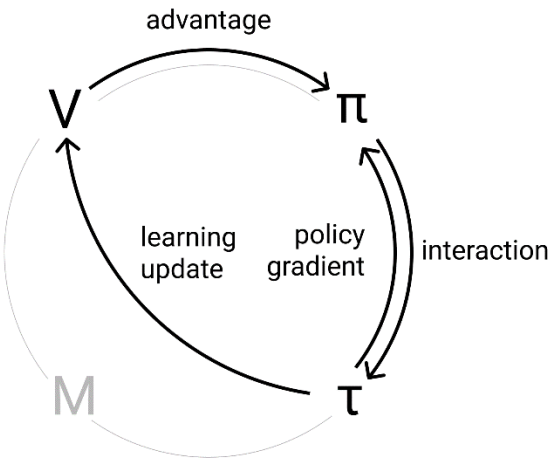


Figure 147 Diagram of Advantage Actor-critic algorithms. By Author.

Notice that the return function $R_t(\tau)$ in the policy gradient is based on the trajectory rewards and is an estimate of $Q^{\pi}(s,a)$. Both value functions compose an advantage function in the form of Equation 15, which indicates the difference between taking the action a in the state s and of following the policy's default action (Sutton et al. 2000; Mnih et al. 2016). With an advantage function, the policy gradient penalizes or credits actions relatively to the state value. Therefore, it captures the long-term effects of the action from the current time step on and in relation to all the available actions.

In a typical estimation of the advantage function, the value function V is learned by the critic while the action-value function Q is estimated with a combination of the return function in a certain horizon and the estimate of V for a future state (Equation 16). This combination establishes a trade-off between the high variance no bias of the cumulative and discounted rewards with the low variance and high bias of the estimate of the V value. As an alternative to estimate with a fixed n of steps, the Generalized Advantage Estimation (GAE, Equation 17) (Sutton et al. 2000; Schulman et al. 2017) uses a exponentially weighted average of advantages with multiple values of n. GAE uses a decay parameter λ, which controls the bias-variance tradeoff. With $\lambda = 0$, GAE is equivalent to the TD-estimate (high bias, low variance), and with $\lambda = 1$, it is a Monte Carlo estimate of returns (low bias, high variance).

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \qquad \text{Equation 15}$$

$$A^{\pi}_{NSTEP}(s_t,\ a_t) \approx \left[\sum_{n=0}^{N} \gamma^n r_{t+\mathrm{n}}\right] +\ \gamma^{n+1} V^{\pi}(s_{t+n+1}) - V^{\pi}(s_t) \qquad \text{Equation 16}$$

$$A^{\pi}_{GAE}(s_t,\ a_t) \approx \sum_{l=0}^{\infty} (\gamma\lambda)^l\ (r_{t+l} + \gamma V^{\pi}(s_{t+l+1}) - V^{\pi}(s_{t+l})) \qquad \text{Equation 17}$$

$$\nabla_\theta J(\pi^\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} A^\pi(s_t, a_t)\nabla_\theta log\pi^\theta(a_t|s_t)\right]$$ Equation 18

The actor-critic algorithms rely on separate function approximators for the actor and the critic. However, they are often implemented as a neural network with a shared body (initial layers) and separate network heads (the final layers) for the different functions. This allows both to build a common representation based on the state information, then establish custom features for their distinct tasks in the independent portion of the neural network.

There are many alternatives to train actor-critic algorithms. For example, in Advantage Actor-Critic (A2C, see

Algorithm 9), the actor can be trained with the policy gradient function updated with one of the advantage functions (see Equation 18). Optionally, an entropy regularization term H can be subtracted from the loss function. The term encourages exploration by incentivizing policies with more uniform distributions.

Part of the computation to estimate the advantage, can also be used to estimate the V target to train the critic (Graesser and Keng 2019, 140–41). If the actor-critic algorithm uses the A^π_{NSTEP}, the n-step estimation of the Q-value can be used as a V target value. If the A^π_{GAE} is computed, the V target can be derived by subtracting the current V estimate from A^π_{GAE}.

Algorithm 9 Advantage Actor-Critic (A2C) – based on (Graesser and Keng 2019, 141–42)

1: Set parameters $\beta \geq 0$ (entropy weight), $\alpha_A \geq 0$ (actor learning rate), $\alpha_C \geq 0$ (critic learning rate)
2: Randomly initialize the actor and critic parameters θ_A, θ_C
3: for episode = 0 . . . MAX_EPISODE do
4: Gather and store data (s_t, a_t, r_t, s_{t+1}) by acting in the environment with policy π^θ
5: for t = 0 . . . T do
6: Calculate predicted $\hat{V}^\pi(s_t)$ using the critic network θ_C
7: Calculate the advantage $A^\pi(s_t, a_t)$ using the critic network θ_C
8: Calculate $V^\pi_{target}(s_t)$ using the critic network θ_C and/or trajectory data
9: Optionally, calculate entropy H_t of the policy distribution, using θ_C. Otherwise, set $\beta = 0$
10: end for
11: Calculate value loss using MSE: $L_{val}(\theta_C) = \frac{1}{T}\sum_{t=0}^{T}(\hat{V}^\pi(s_t) - V^\pi_{target}(s_t))^2$
12: Calculate policy loss: $L_{pol}(\theta_C) = \frac{1}{T}\sum_{t=0}^{T}(-A^\pi(s_t, a_t)\nabla_\theta log\pi^\theta(a_t|s_t) - \beta H_t)$
13: Perform a gradient step on critic: $\theta_C \leftarrow \theta_C - \alpha_C \nabla \theta_C L_{val}(\theta_C)$
14: Perform a gradient step on actor: $\theta_A \leftarrow \theta_A - \alpha_A \nabla \theta_A L_{pol}(\theta_A)$
15: end for

One of the challenges of policy gradient algorithms such as REINFORCE and A2C algorithms is that they rely on an indirect optimization. They update the actor in the parameter $\theta \in \mathbb{R}^d$ but the performance is evaluated by the trajectories generated by the policies $\pi^\theta \in \Pi$ in the

environment. As the distances in these spaces do not necessarily correspond, it is hard to define the proper step size α for the gradient update. Besides, as the mapping between the two spaces is not isometric, so a small update in θ can potentially lead to a large update in π^{θ}. This can not only hurt the sample efficiency of the algorithm but also lead to performance collapse (Graesser and Keng 2019, 165–67). To solve this problem, algorithms such as Natural Policy Gradient (NPG), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO) change the update mechanism to guarantee the improvement of the policy during training.

7.7.5.4. Black-box policy optimization

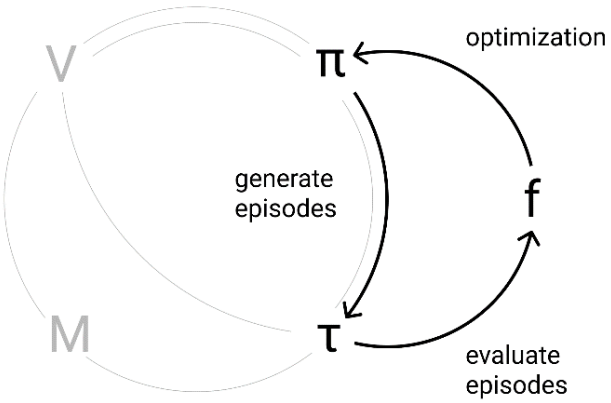


Figure 148 Diagram of Black-box Policy Optimization. By Author.

Direct policy search with black-box algorithms is typically not included in the field RL because its formulation does not exploit the structure of the MDP. However, it performs a task that is similar to the policy-gradient algorithm – i.e., it searches for good policies in a certain task environment –, using an optimization algorithm.

There are many optimization algorithms under the umbrella of black-box policy optimization or direct policy search, but the most common are in the class of evolutionary algorithms, which rely on computational methods inspired by mechanisms of biological evolution (Brownlee 2012, 91). In policy optimization, these algorithms are used to evolve agent programs in different forms of representation, from string representations to deep neural networks. In some cases, such as in genetic algorithm or genetic programming, the optimization uses strategies inspired by population evolution and genetic mechanisms to explore candidates for the policy. In other cases, such as in Covariance matrix adaptation evolution strategy (CMA-ES), the algorithm explores the parameters of a distribution from which it samples policy candidates. Besides, while policy gradient algorithms rely on the sequential information provided by the MDP for learning, black-box optimization typically uses a fitness value, such as the final return of one or more episodes, for the performance evaluation of the policy candidates.

As this category is a large umbrella for different approaches, I will briefly mention notorious examples and discuss its relationship with policy gradient and DRL.

Holland's framework for complex adaptive systems (cas) (Holland 1995; 2012) included the capacity to adapt or learn by changing its program so that it performs better in its environment. Each agent program is represented internally as a string that encodes its rule set. To evolve the agents, Holland combines rule-discovery by genetic algorithms and an evaluation by credit assignment. The genetic algorithms generates new internal representations with new rules for the agents (Holland 1995, 60–80). Then, a custom credit assignment approach called the bucket brigade algorithm (Holland 1995, 53–56) is used to evaluate what rules perform better in a specific task environment.

Koza (1992) addressed the problem of program induction by developing a variation of evolutionary algorithms called genetic programming (GP). In contrast to genetic algorithms, which typically rely on a representation in the form of a string or a list, GP addresses the optimization of hierarchical computer programs of dynamically varying size and shape, represented as tree data structures by symbolic expressions. These trees contain functions in its inner nodes and atoms (or input for the programs) in the terminal nodes. GP relies on many properties for the representation, such as closure and sufficiency. The former indicates that each function in the function set available for the nodes should accept as its argument any value or data type returned by any function in the set or in the terminal set. The latter indicates that the terminal and function set should be able to express a solution to the problem.

A population of candidates form a generation, which is tested in the task environment with pre-defined fitness cases. The fitness evaluation over the different cases is used for ranking each candidate. Then, the algorithm selects genetic operations probabilistically. In some cases, a candidate with a good fitness is reproduced in the next generation, in other cases two strong candidates are selected for crossover, generating two new candidates for the next generation. Additionally, operations such as mutation and permutation can also be incorporated to increase the diversity of the population. From an initially generated population, this process repeats over multiple iterations to generate programs with better performance.

Sims (1994a; 1994b) created a methodology similar to genetic programming to evolve artificial creatures that can perform specific tasks in a simulation. Each creature is composed of connected rigid bodies, sensors, effectors, and brain. The body morphology is represented by a directed graph, with developmental instructions for generating a creature. The brain is a directed graph of neurons with mathematical operations conditioned on the configuration of the morphology graph with additional external neurons and connections for centralized control. These graph genotypes for body and brain result in a creature phenotype that is evaluated in a task in a physics simulation. The individuals with larger fitness survive and offspring creatures are generated by mutations, crossovers, and grafting operations on the graphs.

Like policy gradient, black-box policy optimization can also address parameterized policies. It performs well in small and medium parameter spaces. In more specific video game domains, such as in Tetris, genetic algorithms, cross entropy, and CMA-ES have often surpassed many of the conventional RL approaches or have been integrated to them for better performance (Algorta and

Şimşek 2019).

When policy optimization is combined with neural networks, it is commonly referred to as neuro-evolution. Instead of estimating gradient of the model parameters with the respect to the expected return, it uses evolutionary algorithms to explore model parameters. It updates the parameter $\theta \in \mathbb{R}^d$ for the actor but the performance is evaluated by the return of trajectories generated by the policies $\pi^\theta \in \Pi$. It is a straightforward approach that does not access the MDP or any gradient of the policy model. This simplifies the computation, because there is no need to consider a time horizon, reward distribution, or propagating gradients. Salimans et al. (2017) showed that evolutionary algorithms, such as evolution strategies, are highly parallelizable, because they operate on complete episodes and consider a scalar return as the fitness value, so they require less communications between workers. Besides, they do not require value-function approximations. Overall, they demonstrate that evolutionary algorithms can be a competitive alternative to learn policies on large domains typically addressed with DRL algorithms, such as Atari games and robotic control.

Algorithm 10 Parallelized Evolution Strategies (Salimans et al. 2017, 3)

1: Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: Initialize: n workers with known random seeds, and initial parameters θ_0
3: for $t = 0, 1, 2, \ldots$ do
4: for each worker $i = 1, \ldots, n$ do
5: Sample $\epsilon_i \sim N(0, I)$
6: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
7: end for
8: Send all scalar returns F_i from each worker to every other worker
9: for each worker $i = 1, \ldots, n$ do
10: Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
11: Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
12: end for
13: end for

7.7.5.5. Model-based algorithms and hybrids

The dynamics model of an environment describes the consequences of the actions of the agents at the different states. It is a useful resource that enables the agent to foresee the potential next states and rewards for its different courses of action. However, RL algorithms typically assume that the information about the dynamics of the environment is not available for training, so the agent relies on the real experience generated by the environment, and tables or approximations of values or policy functions, to derive good courses of action. These algorithms are model-free, and they focus direct reinforcement learning by trial and error.

RL also comprehend model-based methods, which use simulated experience from the model of the environment to discover good policies - i.e., they rely on planning. Sutton and Barto provide a unified view of state-space planning methods for model-based RL that comprehends background

planning and decision-time planning (Sutton and Barto 2018a, 159–93). According to them, all these methods estimate value functions by backing-up updates applied to simulated experience and they use that estimate to improve a policy or an optimal path to a goal (2018b, 182).

Background planning is used to gradually improve the value function with simulated experiences from the model. Before an action is selected, planning is used to update the table or function approximator, which then can be used for action selection. For example, dynamic programming (DP) methods introduced in 7.7.5.1 assume that the dynamics function is given, so they compute sweeps of one-step expected updates of the value function. After DP computes the value-function off-line, it is possible to derive a good policy for any state of the MDP. Even if the model is not given or if the agent must operate in real-time, there are algorithms that combine learning from direct reinforcement learning and background planning. For example, Dyna-Q uses the real experiences in the environment both for Q-learning and for learning the dynamics. Inside the learning loop, the learned dynamics model is used for a Q-planning update – i.e., the model provides the transition and reward for a sampled state, action pair, which are used to improve the estimates of the Q-values.

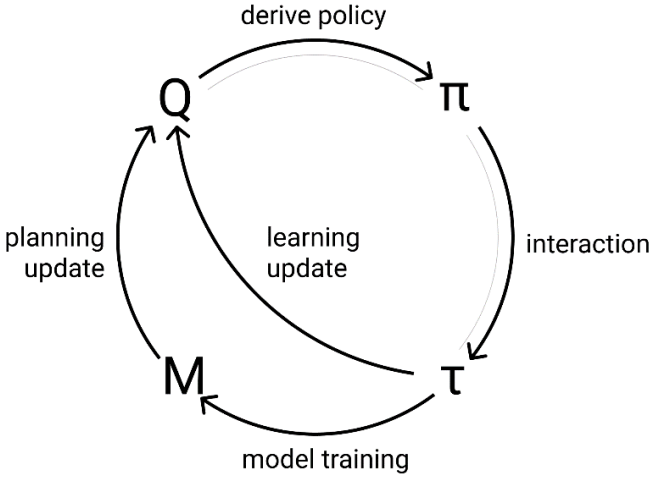


Figure 149 Diagram of Dyna-Q. By Author.

Algorithm 11 Tabular Dyna-Q – based on (Sutton and Barto 2018b, 164)

1; Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A(s)$
2: Loop forever:
3: (a) $s \leftarrow$ current (nonterminal) state
4: (b) $a \leftarrow$ ε-greedy(s,Q)
5: (c) Take action a; observe resultant reward, r, and state, s'
6: (d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, max_a, Q(s',a') - Q(s,a)]$
7: (e) $Model(s,a) \leftarrow r, s'$ (assuming deterministic environment)
8: (f) Loop repeat n times:
9: $s \leftarrow$ random previously observed state
10: $a \leftarrow$ random action previously taken in s
11: $r, s' \leftarrow Model(s,a)$
12: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, max_a, Q(s',a') - Q(s,a)]$

According to Sutton and Barto's unified view of model-based RL, another alternative for model-based algorithms is decision-time planning, which includes conventional heuristic search in artificial intelligence and rollout algorithms (2018b, 181–85). In contrast to other RL methods, where the output is a value or policy function that can be used to select actions at different states, decision-time planning uses the memory and computational resources to discover the best course of action at a specific s_t.

Search algorithms simulate sequences of action from the initial state in the model to find the ones that lead to a goal state (Russell and Norvig 2021, 64). In the off-line setting, planning by search results in a trajectory, which can be executed in the real environment. In the taxonomy developed by this research, these techniques are typically used to synthesize "satisficing" design alternatives based on designer aspirations and criteria (see composition in 1.3.4). In the case of decision-time planning for RL, the heuristic search is performed at a time step t and results in a single action a_t. In this sense, a heuristic search is considered an extension of an action-selected method based on a greedy policy. This extension relies on a value function designed by humans and builds a search tree with simulated trajectories from s over multiple steps.

The rollout algorithms are decision-time planning algorithms that use Monte Carlo sampling to look for the best action at a state s_t. They rely on a rollout policy to generate a collection of simulated trajectories and averaging their return to produce estimates of the Q-values of all the available actions at s_t.

In-between heuristic search, rollout algorithms, and heuristic search, there is a decision-time planning algorithm called Monte Carlo Tree Search (MCTS). The main ideas of MCTS were developed by Coulom (2006) and by Kocsis and Szepesvári (2006). Over the last decades, MCTS had great success in the longtime challenge of learning how to play the game of Go. MCTS It is one of the main components of the algorithms AlphaGo and AlphaGo Zero , which were developed by DeepMind to master the game of Go with (Silver et al. 2017).

Overall, MCTS uses the rollout policy $\pi^{roll}$to produce multiple simulated trajectories for incrementally constructing and extending a statistics tree rooted at the current state s_t, which in its turn supports the use of a planning tree policy π^{tree} that operates in the domain of the tree. This process is developed iteratively to guide the action selection for state s_t towards trajectories with higher expected rewards. The statistical tree can be either discarded after the selection of action a_t, so a new search is executed to find the best action at state s_{t+1}, or the relevant estimates can be re-used for that search.

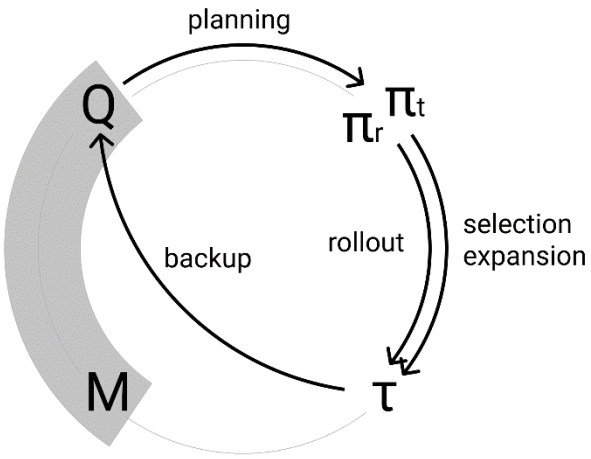


Figure 150 Diagram of MCTS. By Author.

The MCTS algorithm consists of four steps inside a loop (Sutton and Barto 2018b, 186–87):

1. Selection. Starting at s_t, a tree policy π^{tree} that balances exploration and exploitation, such as ε-greedy or UCT (Upper Confidence Bound), uses the estimates of the Q-values of the current tree to reach a leaf state node with unexplored actions.

2. Expansion. If this leaf node was not visited before, it is selected for the simulation; otherwise, one or more unexplored actions are used to expand children state nodes – one of which is selected for the simulation.

3. Simulation. The rollout policy π^{roll} is used to generate a collection of simulated episodes from an action at the selected node. Typically, MCTS is used in episodic tasks such as games, so each trajectory is over with the end of the game. However, it can also be limited by a time horizon for deep search trees or continuing tasks.

4. Backup. Outside of the tree – i.e., over the state action values of the Monte Carlo trajectories – no value is updated. Inside the statistical tree, from the node and action selected to initialize the simulation step towards the root s_t, the return of trajectory is backpropagated to the edges. A backup operator such as mean, max, max with a confidence bound action-value of move with maximum count (robust max), or a mix between them is used to update the Q-values (Coulom 2006; Silver et al. 2017). The number of counts at each edge is also updated to support the backup operators and π^{tree}.

Model-based methods are good for situations where it is time consuming to gather real experience or when there is time available to use some form of planning. They tend to be more sample efficient because they can plan with simulated experiences. Besides, transition models are re-usable for different tasks with different reward functions, so they can potentially support learning different tasks in the same environment. However, model-based methods are more computationally demanding. Background planning requires learning additional functions, such as the transition and the reward, and decision-time planning allocates all the computational

resources to select a single action.

7.7.6. Multi-agent RL algorithms and methods

There are different methods to train multiple agents in a shared environment, which vary in the way that they address the learning process and the execution of the policy.

In one extreme, using centralized training and execution allows the direct application of single-agent and model-free algorithms with a centralized neural network. It assumes a model that maps the joint observation to the joint action of the agents, what in practice can be factored in the output of the neural networks (Gupta, Egorov, and Kochenderfer 2017). The advantage of centralized learning is that by learning in the joint action space, it can explicitly handle coordination of the agents. However, this approach leads to the exponential growth in the observation and actions space with the number of agents. Besides, a centralized neural network for training will learn the action space of several agents limited by its architecture.

In the other extreme of the spectrum of MADRL methods, there is the concurrent or independent learning, which also enable the use of the standard RL algorithms. The neural network is used to train each heterogeneous agent, so the individual observations and rewards are used for training, while the other agents are treated as part of the environment. The advantage is that policy only learns to map a single agent's observation to its action. However, independent learning is unstable due to the non-stationarity of all other agents adjusting their policies simultaneously throughout training, so it generally requires some form of communication to alleviate it. Besides, the agents do not share experiences, it adds sample complexity to the training, and it does not scale well to large number of agents (Gupta, Egorov, and Kochenderfer 2017).

In between these extremes, there are other methods that try to conciliate the potential for coordination of centralized learning with the scalability of decentralized execution.

For example, in the last years many specialized algorithms were developed under the paradigm of centralized training and decentralized execution (CTDE), which relies on centralization techniques that are used during training to support individual agents to develop cooperative policies, improve collective rewards, and stabilize training. COMA (Foerster et al. 2017) and MADDPG (Lowe et al. 2017) use centralized critic networks to estimate a joint Q-function that relies on all available state and action information, which is shared with the decentralized actor networks representing the different agents. QMIX (Rashid et al. 2018) is a value-based methods that employs a network to estimate the joint Q-value as a non-linear combination of the Q-values of the individual agents derived from local observations. After training, only the local agents are used at the execution stage.

Another paradigm that became widespread in MADRL is parameter sharing (Gupta, Egorov, and Kochenderfer 2017; Justin K. Terry et al. 2021). In the parameter sharing approach, all the agents share the same individual policy, so the other agents are not interpreted merely as part of the environment. To ensure that a single policy can express a diversity of actions and agent types, the information in the observation is customized for the agent, which can include part of its own

internal state or its id. In the case of the value-function network, the observation can even be augmented with global information that will not be available during the execution. Besides, padding can be used for the cases where there are different observation or action sizes (Justin K. Terry et al. 2021, 8). In contrast to CTDE, parameter sharing is a simple approach that enables the use of the original RL algorithms with minimal modifications. It also works in the simpler observation and action spaces of the individual agents. The use of the shared policy enables training it with the data from all the agents, provides some level of centralization in the learning that can potentially mitigate non-stationarity, accelerate training (Justin K. Terry et al. 2021), and lead to emergent cooperative behavior (Gupta, Egorov, and Kochenderfer 2017). The disadvantage is that it relies on cases where a single policy is enough to model the range of behaviors in the system.

Table 28 Synthesis of multi-agent approaches for RL

	Multi-agent RL approaches			
	Centralized	**Concurrent/Independent**	**CTDE**	**Parameter Sharing**
Training	Centralized	Concurrent	Centralized	Parameter sharing
Execution	Centralized	Decentralized	Decentralized	Decentralized
Enables coordination	Yes	No	Yes	Potentially
Modification to simple RL algorithms	Minimal	Minimal	Large	Minimal

7.7.7. Multi-agent RL environments

There are many models to describe decision processes for multi-agent RL.

In Extensive Form Games (J. K. Terry et al. 2021, 17–19), two agents choose actions alternately and receive individual reward after the game. If this game is competitive, the reward function will typically output inverse reward signal at the end. This is typically the model for 2-player and zero-sum board games such as chess, checkers, and go.

In contrast, a Decentralized Partially Observable MDP (Dec-POMDP) describes a fully cooperative task (Gupta, Egorov, and Kochenderfer 2017, 1–2). In Dec-POMDP there is only one reward function $\mathcal{R}$ that relies on the joint actions and cooperation of the agents. Each agent only has access to partial information, so they must collaborate to accomplish a shared task. The Dec-POMDP[26] can be defined by the tuple $(S, N, \{A_i\}, T, \{\Omega_i\}, \{O_i\}, \mathcal{R})$, where:

- S is the state of the environment
- N is the number of agents

[26] In this definition, I adapted the notation to the one used for the definition of the POSG used in the next paragraph and based on (J. K. Terry et al. 2021, 17). Besides, for generality, instead of using as single observation model O, I opted for individual models O_i, which can capture cases where the derivation for the observations for each agent is different.

- A_i is the action space for the agent i
- T is the transition function
- Ω_i is the observation space for agent i
- O_i is the observation model for agent i
- $\mathcal{R}$ is the reward function shared by all the agents

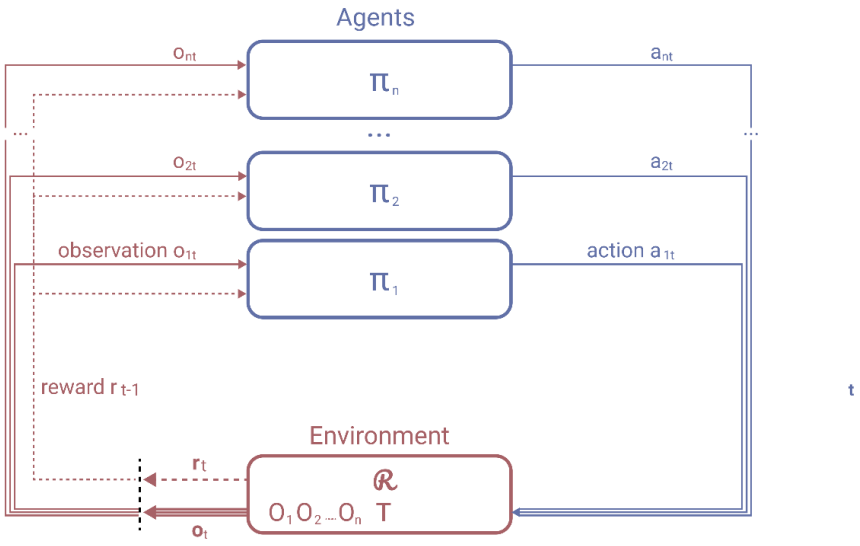


Figure 151 The agent-environment interaction in the case of a Dec-POMDP. By Author.

In contrast, a Partially Observable Stochastic Game (POSG) is a self-interested version with a separate reward function for each agent, so it can be defined by the tuple $(S, N, \{A_i\}, T, \{\Omega_i\}, \{O_i\}, \{\mathcal{R}_i\})$ (J. K. Terry et al. 2021, 17).

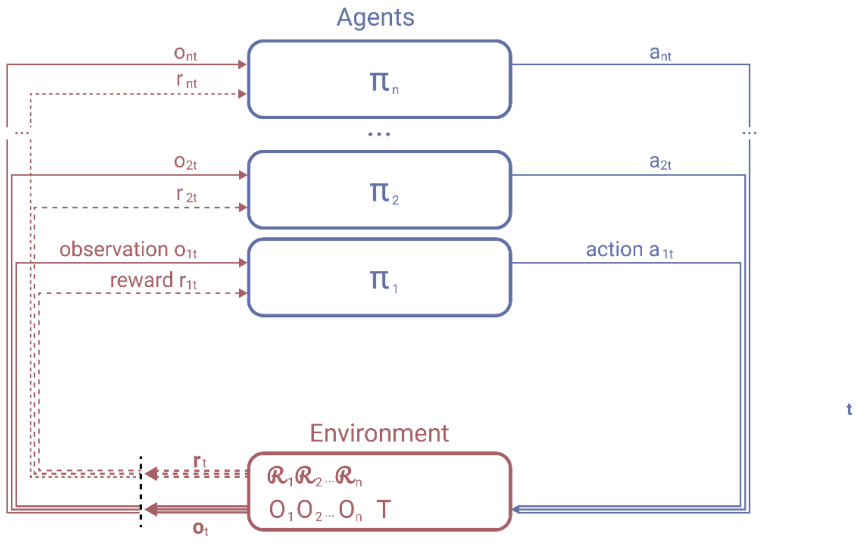


Figure 152 The agent-environment interaction in reinforcement learning in the case of a POSG. By Author.

Multiple interacting agents with inverse reward signals must compete with one another to win in the case of competitive games such as team sports. Else, independent reward signals indicate a non-cooperative game, such as in an ecosystem of agents sharing the same environment.

7.7.8. A brief overview of reinforcement learning in architectural design

During the development of my research artifact, a considerable number of research projects that address RL for architectural design were published. I opted for including both the precedent works (before 2020) and the new ones (2020-2022) in this section.

The recent work in reinforcement learning for architecture explores a variety of problem domains, such as intelligent adaptive building control (Smith and Lasch 2016), autonomous robotic tensegrity (Hosmer and Tigas, 2019), adaptive and distributed drone-assisted construction (Fang et al. 2020), fire egress evaluation (Jabi et al., 2019), and machine feedback (Xu et al., 2018).

Particularly, the use of RL to generate in speculative forms, fabrication assemblies, architectural, and urban assemblies, is an emerging trend in computational design. As mentioned in chapter 3, Narahara (2017) uses a RL-related techniques (policy search) for fine-tuning neural networks that guide competing teams of agents that create clusters of building blocks on a voxel. The recent trend but focuses on the single-agent case and emphasizes the use of spatial and constructive performance as a source of reward.

Wang and Snooks explored self-organizing speculative designs based on a reinforcement learning. They used PPO to train a mesh graph with a pre-defined topology to form a pavilion form in a space with obstacles (Wang and Snooks 2021b). The agent and environment are encoded in a voxel grid and the goal is to build a pavilion that achieve a series of spatial and typological goals. They also combined Deep Q-Learning with a random walk to train an agent that moves on a voxel space, building a speculative assembly based on spatial and typological rewards (Wang and Snooks 2021a). Huang (2021) use a Deep Q-Learning algorithm to train an agent that use pre-existing fragments to assemble a structural shell that address multiple objectives. In a more abstract level, Kotov and Vukorep (2021) proposed a testbed to test and solve abstract architecture problems related to basic spatial concepts, such as borders, connections, and zoning, in a 2D virtual environments. The idea is that the testbed supports the development of games in the minimal form of the gridworld that can support computational agent to gain knowledge about different spatial skills.

In the case of fabrication assemblies, Akizuki et al (2020) used DQN to train an agent that builds trails and assemblies based on pre-existing forms encoded in images or voxels. The agent was also trained to use LEGO blocks to build an assembly that approximates a pre-existing form. Wibranek et al. (2021) addresses the problem of sequential assembly of SL-blocks with reinforcement learning to produce physically constructible and stable structures. After comparing the performance of PPO and DQN agents with evolutionary and greedy algorithms, they discovered that RL performs better in 3D curves and is robust to the presence of obstacles.

In the case of architectural assemblies, Hosmer et al. (2020) use PPO to train an agent to create

constraint-based assemblies that satisfy multiple objectives. The agent learns how to build the assembly by iteratively selecting an available location and one of parts in a pre-defined catalogue to add at this location. In a similar fashion, Mintrone and Erioli (2021) integrates Reinforcement Learning to the Wave Function Collapse (WFC) algorithm to generate constraint-based assemblies that address spatial and structural objectives. However, in this case, during the iterative generation of the assembly, WFC is responsible for choosing the location and the RL agent the spatial part.

Song and Yuan (2021) use deep reinforcement learning for urban cluster generation. A single PPO agent is used to control the height of the buildings in an urban grid representation to improve the environmental performance with the respect to wind. C. Huang, Zhang, and Yao (2022) use Deep Q-Network as a replacement for an optimization algorithm to control the parameters of the parametric and rule-based urban model. The goal is to improve the performance of the urban blocks with the respect to real-time energy performance predictions (minimizing energy and the maximizing the potential for rooftop photovoltaic power).

In the case of specific works related to RL and spatial synthesis, Ruiz-Montiel et al. (2013) combined RL and shape grammars to generate 2D layouts of single-family housing units that satisfy programmatic and spatial requirements. Specifically, they used Q-learning with n-step Bootstrapping to train an agent to guide the rule application of the grammar. Recently, Saha, Haymaker, and Shelden (2020) used Deep Q-Network to train an agent that allocates spaces to a pre-defined topological model with locations to satisfy user-defined attributes of the spatial layout (adjacencies, proximity, access to views, and orientation).

7.8. Appendix H: Additional data for the design artifact evaluation

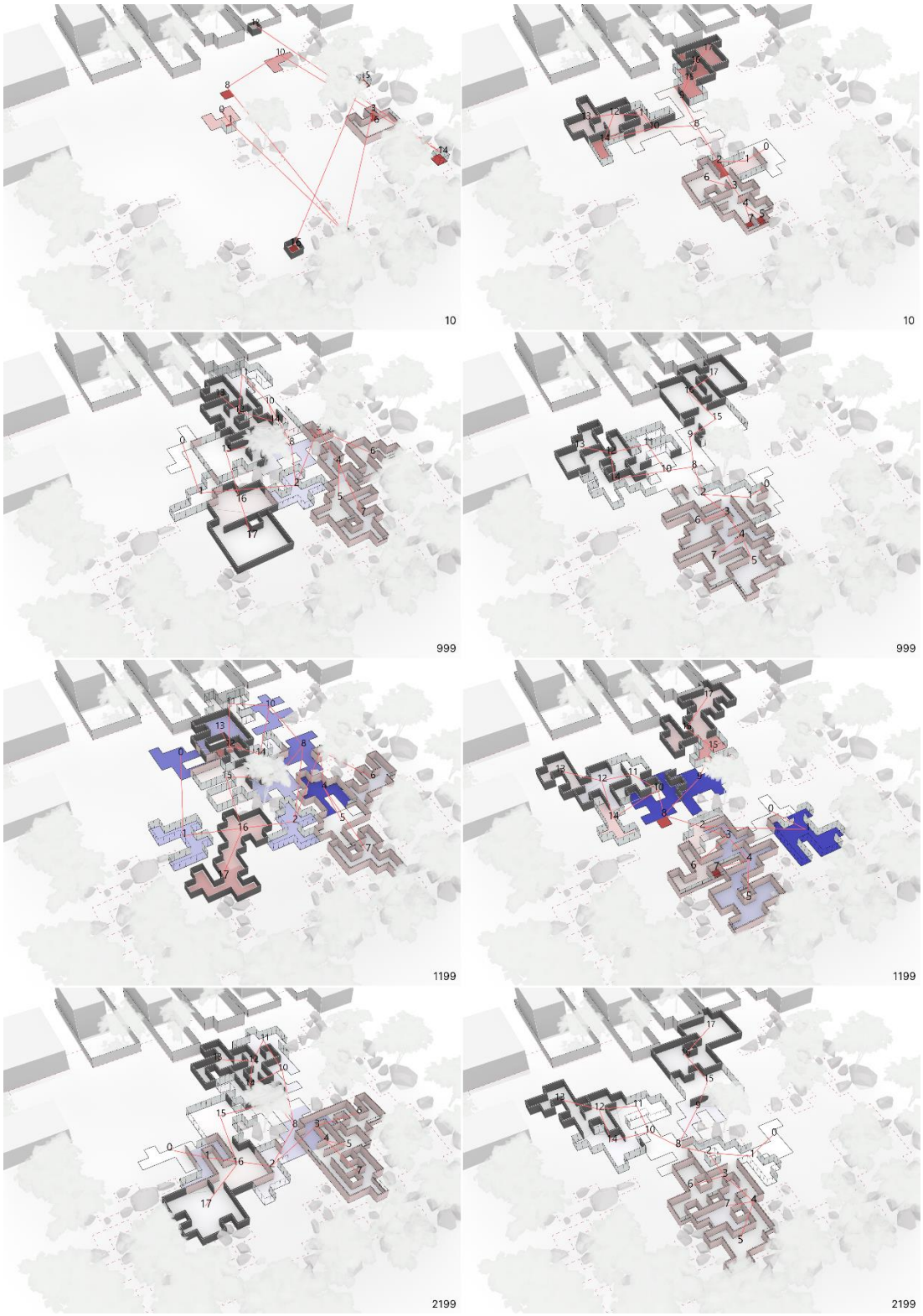


Figure 153 Agents trained for adjacency and area goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

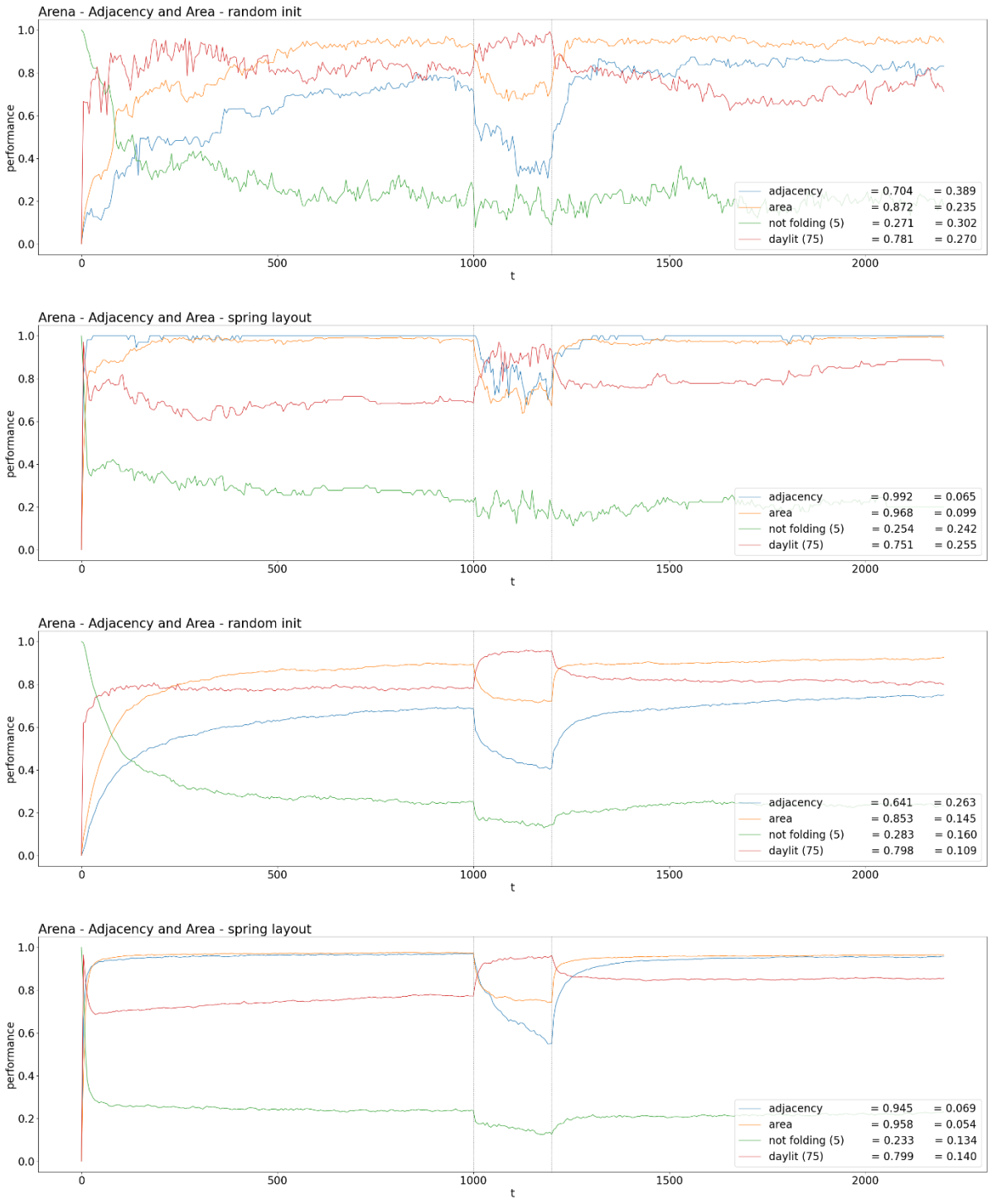


Figure 154 Performance of P1 with policy Adjacency and Area. Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

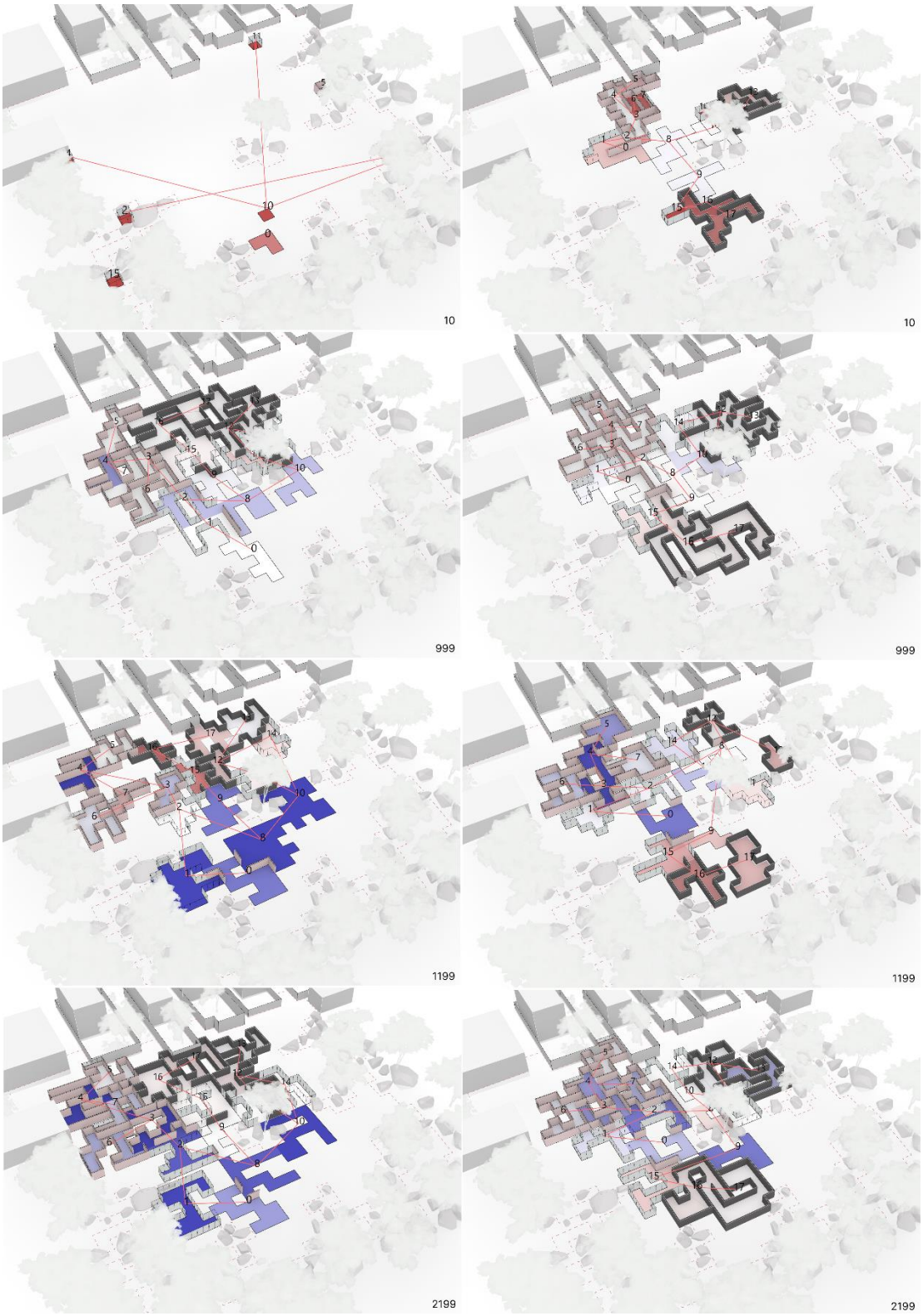


Figure 155 Agents trained for adjacency, area, and daylit (75%) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

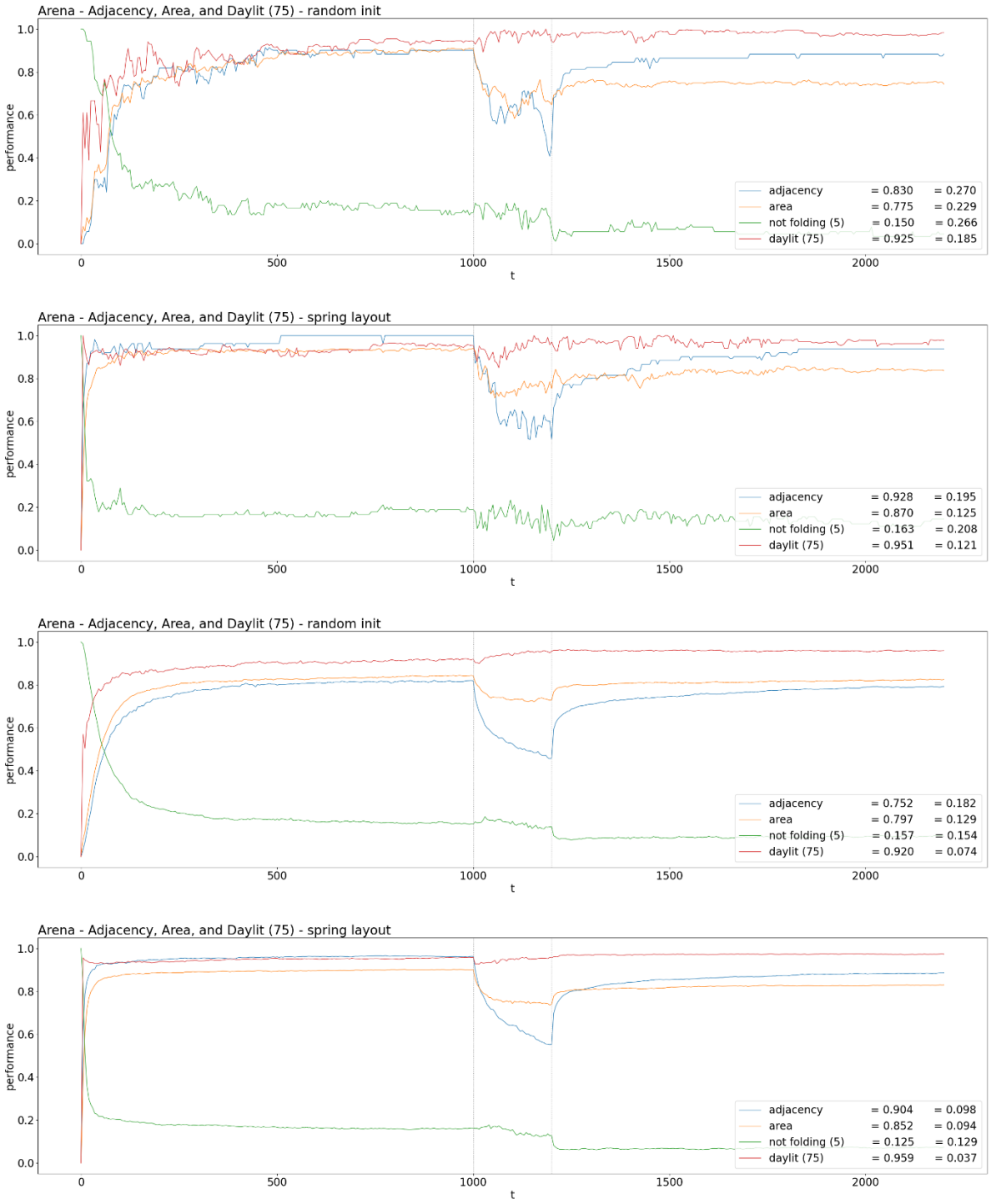


Figure 156 Performance of P1 with policy for adjacency, area, and daylit (75%). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.

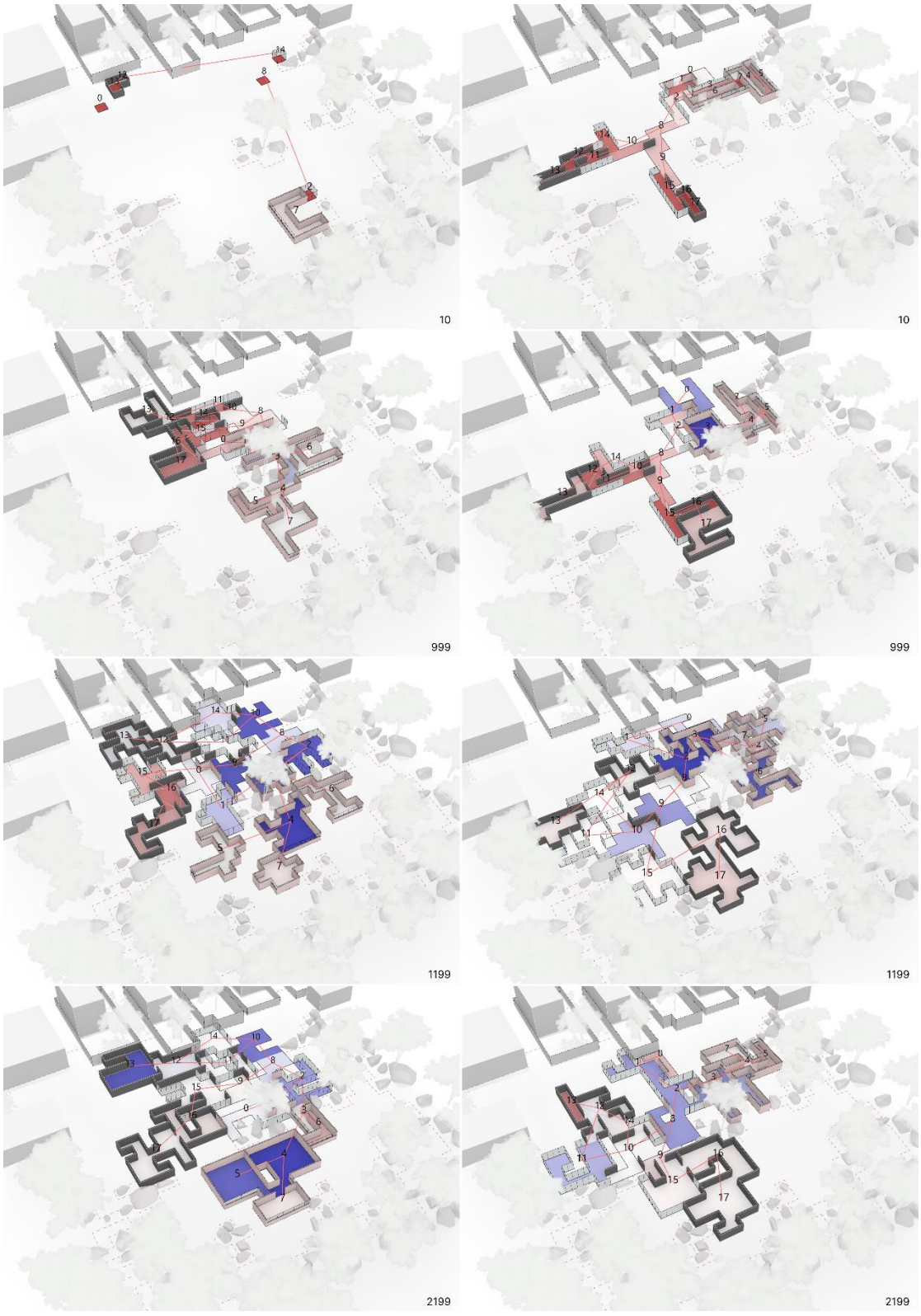


Figure 157 Agents trained for adjacency, area, not folding (5), and daylit (75%) goals. Col 1: simulation of 18 agents with random initialization in a large terrain with obstacles Col 2: simulation of 18 agents with spring layout initialization in a large terrain with obstacles. By Author.

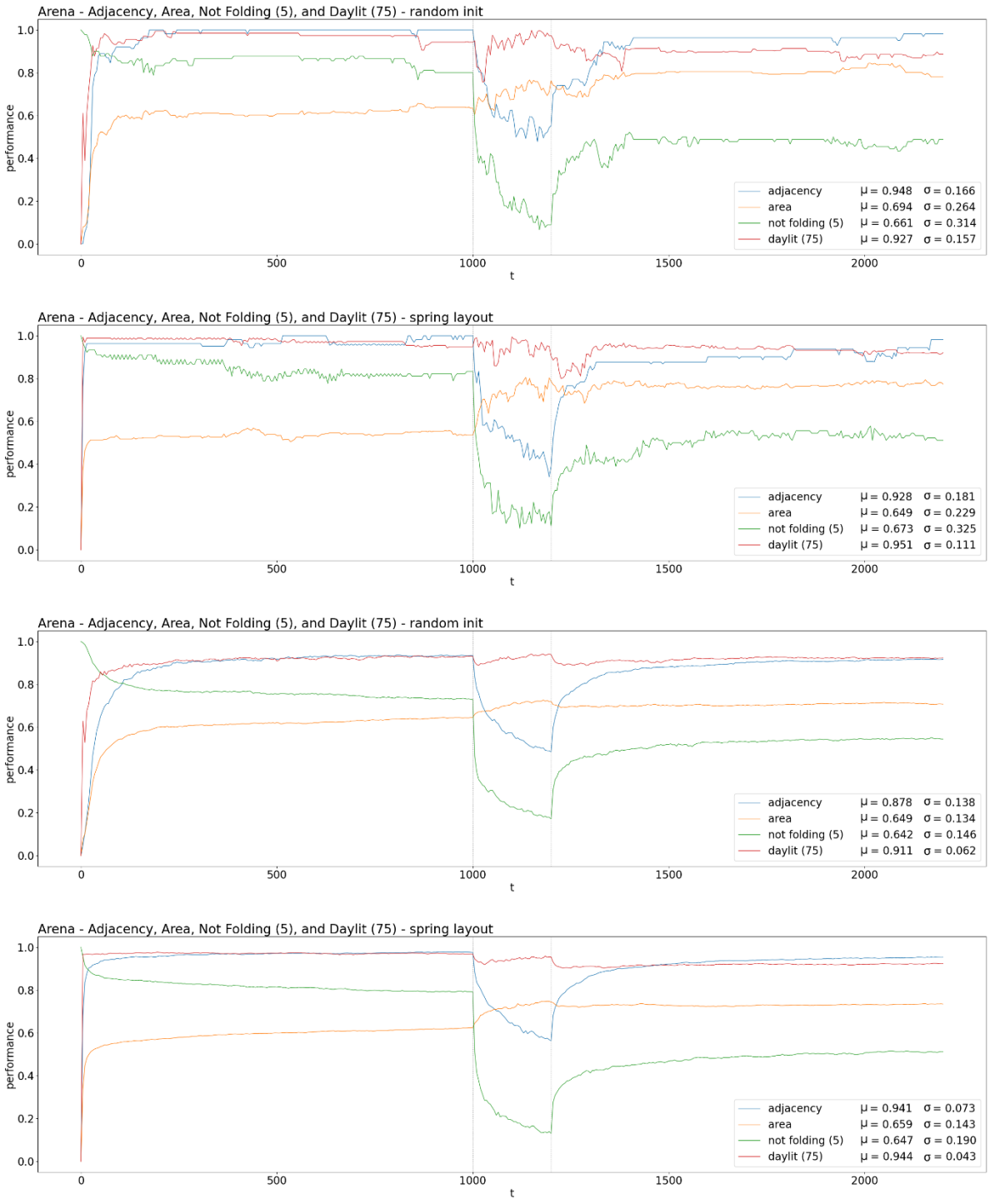


Figure 158 Performance of P1 with policy for adjacency, area, not folding (5), and daylit (75%). Row 1: single episode and random initialization. Row 2: single episode and spring layout initialization. Row 3: average of 50 episodes with random initialization. Row 4 average of 50 episodes with spring layout initialization. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy. By Author.