CARNEGIE MELLON UNIVERSITY

School of Architecture

Thesis

Submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE :

3D Design through Interactive Training
of a Generative Reinforcement Learning Agent

AUTHOR :

Chloe (Soo hwa) Hong

ACCEPTED BY ADVISORY COMMITTEE:

Professor Daniel Cardoso Llach	Principal Advisor	May 10, 2023 DATE
Professor Chris McComb	Advisor	May 10, 2023 DATE

ABSTRACT

Designing is an open-ended process that involves multiple iterations of creating artifacts and evaluating them with multifaceted criteria. The challenges come from the fact that not only the artifact, but the criteria for design are gradually articulated and that they are unique to the domain, context, and designer's intentions. Recent advancements in generative design have sought to aid the 3D design process by delegating the task of auto generating and visualizing multiple artifacts. However, these frameworks still come short by enforcing the designers to prematurely define constraints and prompting them to translate their intentions into a limited set of parametric or operational inputs which is non-trivial and confines the design to certain forms.

This thesis introduces an alternative framework of 3D designing where the designer interactively trains a reinforcement learning (RL) based generative agent. It first aims to formulate a RL agent that can be trained to procedurally generate 3D forms given partial and evolving constraints. The proposed method leverages the capabilities of RL agents to procedurally generate feasible 3D forms while adapting to user-provided constraints. It also presents and experiments with an interactive framework between the designer and generative RL agent, that would allow simultaneous articulation of design criteria and artifacts, which is critical in early design development. The interactive framework allows designers to iteratively evaluate auto-generated artifacts and provide intermittent feedback that serves to articulate the implicit design criteria. As a result, this work sheds light on ways to integrate learning-based computational frameworks for idiosyncratic design tasks as well as presenting a novel mode of designing through training generative learning-based agents.

ACKNOWLEDGMENTS

I am deeply grateful to a number of individuals whose unwavering support made this thesis possible. Without them, this work would not have been accomplished.

First, I would like to express my appreciation to my advising committee. Daniel Cardoso Llach was a patient teacher who encouraged me to think with a broader lens and to place technical artifacts in the context of human society. With his guidance, this thesis was able to gain its richness and rigor. Chris McComb was an essential force in shaping this thesis throughout its various iterations. He was a thoughtful mentor who encouraged me to expand and clarify my ideas and most importantly, urged me to put my work out there.

I owe a debt of gratitude to my MSCD cohort and peers in CodeLab, whose insightful conversations and exceptional works provided me with inspiration and encouragement. I express my appreciation for Mitchell Foo whose enthusiasm and energy helped me maintain the momentum I needed throughout.

Last but not least, I am grateful to my family, whose unconditional support has been an invaluable source of strength and has helped shape the person I am today.

TABLE OF CONTENTS

ABSTRACT **2**

ACKNOWLEDGMENTS **3**

TABLE OF CONTENTS **4**

1 Introduction **6**

2 Background **9**

2.1 Design Exploration 9

Design exploration is iterative and evaluative 9

Design exploration within the design space 11

2.2 Generative Design 13

Generative Design 13

Interactive Generative Design Frameworks 17

2.3 Reinforcement Learning 19

Why use Reinforcement Learning (RL) for generative design? 19

Challenges in Designing an RL Agent for a Design Task 20

Reinforcement Learning for 3D Design Tasks 21

3 Methods **26**

3.1 Reinforcement Learning Overview 26

3.2 Unity Engine and Unity Machine Learning Agents Toolkit 27

3.3 3D Form Finding Task, Voxel Grid Environment, and Builder Agent 29

3D Form Finding Task 29

Voxel grid Environment 30

Builder Agent State and Actions 34

3.4 Builder Agent Observations and Rewards 38

Observations 38

Reward function 40

Training Hyperparameters..... 42
3.5 Interactive Training Framework..... 43
4 Results..... 47
4.1 Evaluation of the Builder Agent..... 47
4.2 Evaluation of Interactive Framework..... 58
5 Conclusion..... 64
5.1 Summary..... 64
5.2 Limitations and Future works..... 64
5.3 Contributions and Discussion..... 66
How can designers have agency in formulating learning-based design tools?.....67
How can computational design tools adapt to domain, context, problem specific design processes?..... 68
How can learning-based methods be used for design tasks with ambiguous goals and changing criteria?.....68
REFERENCES.....70
LIST OF FIGURES AND TABLES.....75

1 Introduction

This thesis is built on the claim that current generative design systems fail to support open-ended and idiosyncratic design processes. I attribute this to the commonly used generators, the core mechanisms of synthesis in these systems, which are feed-forward, one-shot processes. For instance, knowledge-driven generators prompt the designer to define the goal or criteria upfront in an explicit format such as parametric or operational inputs. This enforces the non-trivial task to the designers to translate rich information such as intentions, domain knowledge, or context into and prematurely define constraints even when they are unknown or ambiguous. As a result, designers end up limiting their criteria to that that can be easily expressed in this explicit format. There are also data-driven generators that prompt the designer to provide a corpus of instances or a dataset that is used to define the design space. Creating large datasets, which involves gathering, cleaning, and formatting data, especially for a very specific purpose and context is non-trivial.

With these characteristics, current generative design frameworks hinder design exploration in three ways. First, the design criteria involved in the generation are limited to those that are defined explicitly and easily expressed with the format the generator requires. Second, the synthesized designs are limited to the fixed design space that was defined solely by the inputs that the designer provided. Third, it does not allow the designer to discover new criteria and redefine the problem during the process which is critical in design exploration.

The objective of this thesis is to expand the capabilities of generative design systems to allow the gradual articulation of design criteria and artifacts in the midst of emergent discovery which is critical to design exploration. Furthermore, this thesis seeks ways to enable the customization of generative design systems for idiosyncratic design processes and make it accessible to the designer with limited resources and knowledge about the system's backend.

The research questions that lead this thesis are

- ➔ How can a generative design system support the gradual articulation of design criteria and artifacts essential to early design development?

- ➔ How can designers be involved in determining the affordances of generative design systems?
- ➔ How can a generative design system be formulated - representations, interfaces, training mechanisms - so that it could dynamically adapt to idiosyncratic design processes specific to the domain, context, problem, and user?

This thesis introduces and explores a novel generative design system for 3D design that is driven by the interactive training of a reinforcement learning (RL) agent. My interest in employing agent-driven methods and, in particular, RL as the generator stems from two possibilities. First from its capabilities to learn complex behaviors and take into account future states from a simple set of rules, I saw the potential for RL methods to generate complete 3D forms from the partial information that the designer has provided. At the same time, its emergent qualities - where there is a coherent rise of patterns at the macro level (results ensuing agents' actions) from the interactions between parts at the micro level (agent and environment), could present forms that are unimaginable by the designer. This balance of exploration and exploitation is key to design exploration, which was not possible with other methods. Furthermore, the process in which agent-driven methods learn by trial and error in a feedback loop is similar to the iterative and evaluative design process that human designers are capable of, opening the possibility of an interactive dynamic in which the designer can be involved in the training of the generator.

The hypothesis is that

1. The proposed generative RL agent is capable of the synthesis of 3D forms under ambiguous and evolving constraints
2. The proposed framework of interactively training the generative RL agent allows designers to customize the generative system to their specific needs by providing intermittent feedback to the RL agent.

This thesis is divided into multiple chapters. The Introduction chapter expands on the questions that motivate and drive the exploration of this work. The Background chapter covers topics in design exploration, generative design, and reinforcement learning that are

relevant to this thesis. This chapter relies on previous research to provide the terminology and ideas foundational to this work and explains how this thesis relates to these bodies of research. The following Methods chapter expounds on the proof-of-concept in which I first formulate an RL agent to procedurally build 3D forms given ambiguous and evolving constraints and then provide an interactive training framework involving the RL agent and designer that allows intermittent feedback in the training of the learning-based generator. The Results chapter provides an evaluation of chosen instances of the framework in relation to how it can be used for design exploration. Lastly, the conclusion chapter provides a summary of the thesis including its contributions, limitations, and possible future works. The accomplishments of this thesis are posed to start a conversation around questions about the future of generative design, which is one of the main motivations for writing this thesis.

2 Background

This chapter provides definitions and preceding works related to design exploration, generative design, and reinforcement learning that are relevant to this thesis.

2.1 Design Exploration

Design exploration is iterative and evaluative

Design exploration is an iterative process that involves the evaluation and creation of an artifact. Reflecting on prototyping processes such as sculpting with clay, sketching, and mass modeling one can observe a common process of creating an artifact, evaluating the artifact, and making adjustments or creating a new artifact based on the evaluation which is reiterated. This process has been quoted as a recursive process of making and seeing (Schon and Wiggins 1992, 135–56; Liu, Gao, and Liu 2005, 1) which involves critical and cyclic engagement supported by direct interaction with the design media, is central to design ideation (Schon and Wiggins 1992; Corona-Martínez and Martinez 2003; Lawson 2005; Gänshirt 2012; Veloso and Krishnamurti 2021a, 1). Such an iterative and holistic revision process is also highlighted by the concurrent design framework (Ma, Chen, and Thimm 2008) used in product development, known to be effective in integrating multifaceted requirements in design and manufacturing engineering. As opposed to the more traditional linear design flow, or the waterfall model (Barkstrom, B. R. 2004), the iterative process encourages prompt changes of tack and discoveries of problems within the development of solutions.



Figure 2-1. Artist sculpting an animal body as an example of the iterative and evaluative process of design. Screen shots taken from (Erik Durant 2021) and annotated by by author.

What is unique in this process is that the articulation of the design artifact and related criteria are gradual and occur at the same time. A design artifact indicates a partial representation of the yet-to-be-articulated result design, which serves as material and stimuli for design interaction. When creating these artifacts, designers are conscious of certain criteria, and they are explicitly or implicitly reflected in the artifacts. Also, domain, user, or context-specific heuristics are typically used to control the process of modifying the artifact (Aelion et al 1992). At the same time, while examining and evaluating these artifacts, designers acquire knowledge that can lead to discoveries and changes in criteria. Previous works have quoted this as "creating a frame of reference, and reformulating the problem and solution in parallel (Cross 2006, 304–325; Simon 1973; Akin et al. 1987; Veloso and Krishnamurti 2021b, 266). Other works have put it in terms of the designer performing actions to change the environment, where they use interpretations of its current environment to produce a response as opposed to simply reacting reflexively (Clancey 1997; Liu, Gao, and Liu 2005, 106).

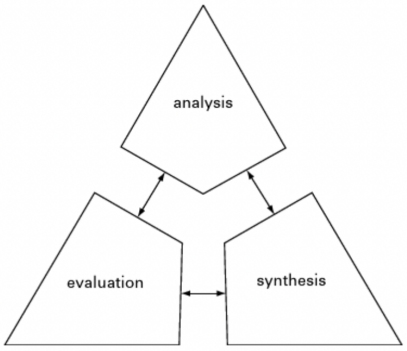


Figure 2-2. Generalized map of the feedback loop within the design process borrowing from (Markus 1969; Maver 1970) taken from (Lawson 2005, 40). *Analysis is the ordering and structuring of the problem. Synthesis is .. the generation of solutions. [Evaluation] (Appraisal) involves the critical evaluation of suggested solutions against the objectives identified in the analysis phase.*

Design exploration within the design space

The process of designing is often put in terms of exploring the design space. One of the challenges in design exploration is that the design space is immensely large. Design space is the network of the possible states a design can take (Toulkeridou 2019, 3) or the space of all potential design candidates (W. Chen 2019, 1). Each artifact that is created with any kind of modification can be represented as a point in the infinite design space. In this sense, the aforementioned iterative process of creating design artifacts can be considered sequentially traversing the design space to find a desired solution. Previous works have closely knit the terms synthesizing designs, design generation, exploring design options, and traversing the design space (W. Chen 2019; Toulkeridou 2019).

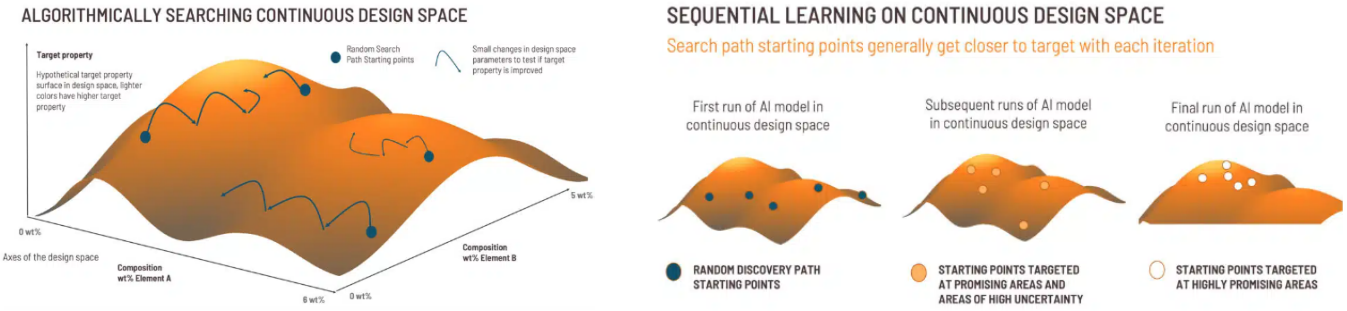


Figure 2-3. Illustration of algorithmically searching a continuous design space from (Green 2021)

To search within the design space, the design space and its bounds have to first be defined, and this is often based on the design criteria. For instance, for a parametric model schema, the geometry and numerical primitive units, the topological relationships and constraints, the manipulative functions, as well as the parameter ranges, formulate the description of the design space (Toulkeridou 2019, 3). In this sense, the designer is articulating both the design space and searching within that design space. Schön has put this as the designer construct[ing] the design world within which he/she sets the dimensions of his/her problem space and invent[ing] the moves by which he/she attempts to find solutions (Schon and Wiggins 1992, 11; Veloso and Krishnamurti 2021a, 30). Some difficulties come from implicit boundaries as it is not feasible to express all criteria explicitly, and added dimensionality of the design space, which adds complexity to navigation (W. Chen 2019, 10).

An additional challenge arises from the fact that design criteria change. Changing requirements and goals may indicate a different design space to be searched for (Toulkeridou 2019, 3). As redefining the design space can be critical to discovering innovative solutions, past works have presented design space expansion techniques to modify the design topology (Cagan, Mitchell, and Center. 1991). If the design space is not adjusted to the evolving criteria, there is a chance that solutions may be overlooked because they are not included in the search space. Another difficulty that comes with design exploration is that there is a range of acceptable solutions as opposed to one optimal solution. Design solutions are often holistic responses (Lawson 2005). A set of designs can be considered optimal but each of them addresses the criteria in different ways to a different degree. There are tradeoffs between different criteria and the weight among them are often undecided (Schulz et al. 2018, 11). As a result, design decisions and identifying the end of the design process remain a matter of judgment and experience of the designer.

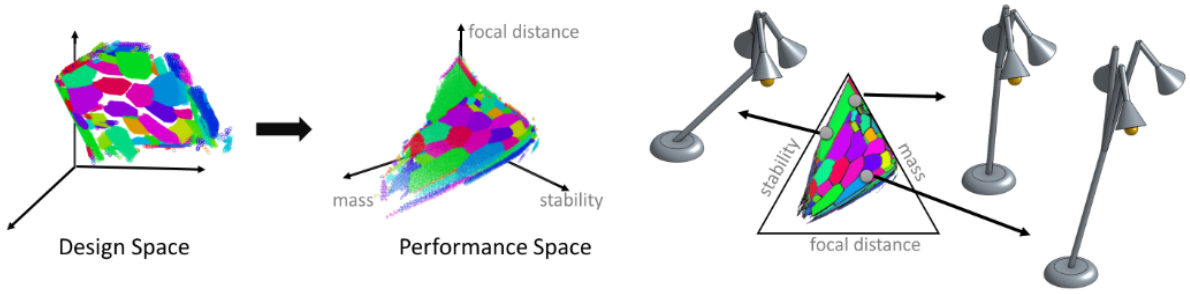


Figure 2-4. The design space is used to visualize and find design solutions. Illustration from Schulz et al. 2018 where the mapping from manifolds in performance space back to design space allows designers to explore performance trade-offs interactively while visualizing the corresponding geometry and gaining an understanding of a model's underlying properties.

While the design space can be of any kind, this work focuses on the 3D geometric design space. As one can imagine, the design space of 3D geometry design is generally high-dimensional and complex which complicates both human exploration and computational optimization (W. Chen 2019, 10).

2.2 Generative Design

Generative Design

Computational generative systems are constructs intended to navigate the design space and automatically synthesize design alternatives, given proper input of information (Veloso and Krishnamurti 2021b, 265). The main component, the generator, typically takes changes in parameters to propagate through all dependencies to update the artifact. With conventional design ideation processes, the designer is navigating the design space by interacting with design media, whereas with computational generative design systems, the designer is formulating or using the results of a generative model that induces the design space (Veloso and Krishnamurti 2021b, 30; Bidgoli and Veloso 2019, 3). The designer changes the input information and observes the set of design alternatives. They can either exploit a subspace of the solutions or explore alternatives. The designer can choose to reformulate the model based on their reflections (Veloso and Krishnamurti 2021b, 30). Previous research has categorized computational generative systems based on how the

generator is constructed into knowledge-driven methods and data-driven methods (Xu et al. 2015). Employing agent-based models in a generative system for design has been unexplored, with the few prior works demonstrating its potential and challenges.

- **Knowledge-driven methods**

Knowledge-driven methods employ explicit rules such as topological or parametric rules or constraints. With these methods, the designer develops the representations for the input, the output, and the algorithm to be encoded in the generator (Veloso and Krishnamurti 2021b, 30). However, such explicit rules are hard to specify or generalize to diverse design objectives (W. Chen 2019, 28). Parametric models are a representative example of knowledge-driven methods. These methods enforce early commitment to a parametric structure, which requires significant frontloading. The designer should anticipate and accommodate the intended interdependencies, changes, and future explorations. The design space is predefined by this set of parameters, creating a closed system that would have to be reconstructed to adapt to changing requirements (Davis 2013, 37–47; Toulkeridou 2019, 3). The cognitive load for making changes to the topological structure of the parametric model schema is big (Holzer, Hough, and Burry 2007; Turrin, von Buelow, and Stouffs 2011). Therefore designers can easily become locked in (Harding and Shepherd 2017) working on a specific parametric model schema and not attempt to explore different conceptual design alternatives (Toulkeridou 2019, 3').

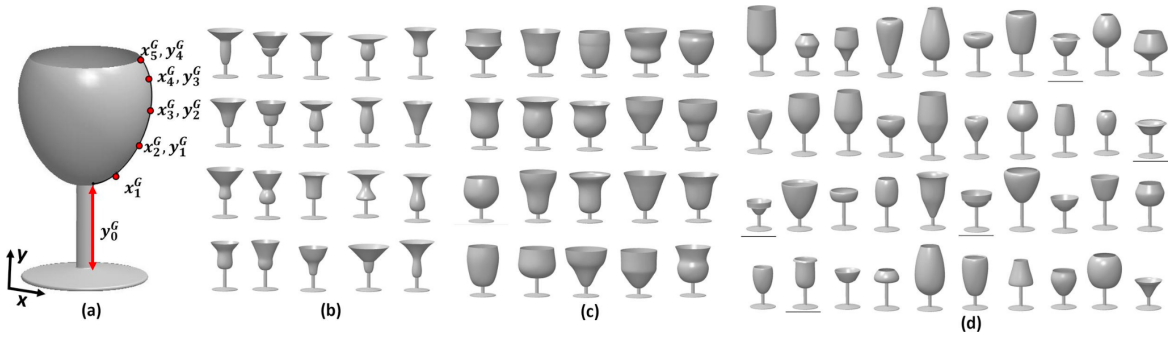


Figure 2-5. A parametric representation of a wine glass model by (Khan and Awan 2018). Design alternatives of the glass model are generated by a space-filling-generative design technique which generates design alternatives that cover the entire design space to give a designer a better understanding of possible design options

- **Data-driven methods**

Data-driven methods, by contrast, learn the representation of geometric structure from examples, adjusting the model to match provided design data (W. Chen 2019, 28). Data-driven generative systems enable designers to build components of a generative model without explicitly describing its logic, which is an important step in bridging computation and intuitive design (Veloso and Krishnamurti 2021a, 47). There are several generative models in ML, including but not limited to Principal Component Analysis, Autoencoder, Variational Autoencoder, and Generative Adversarial Networks (Bidgoli and Veloso 2019, 4). The caveat is that these methods require a corpus of data, the curation of data implies strong assumptions about the problem, and thus the design space is defined by the dataset. For instance, VAEs learn a continuous function to interpolate samples within the dataset and GANs are limited to learning a function that can generate solutions similar to those existing in the dataset. In addition, multiple works have pointed out the absence of domain-specific datasets (Koh 2017). Because design is contextual, it is often the case that datasets are built for a particular design problem but would not apply to design problems in a different context.

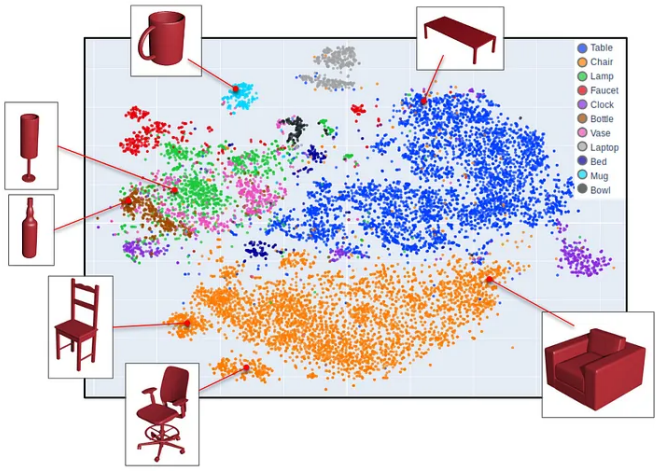


Figure 2-6. (Habowski 2020) presents an example of a data-driven generative design system. The TSNE[1] map of the latent space vectors is colored according to the object category.

[1] T-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map.

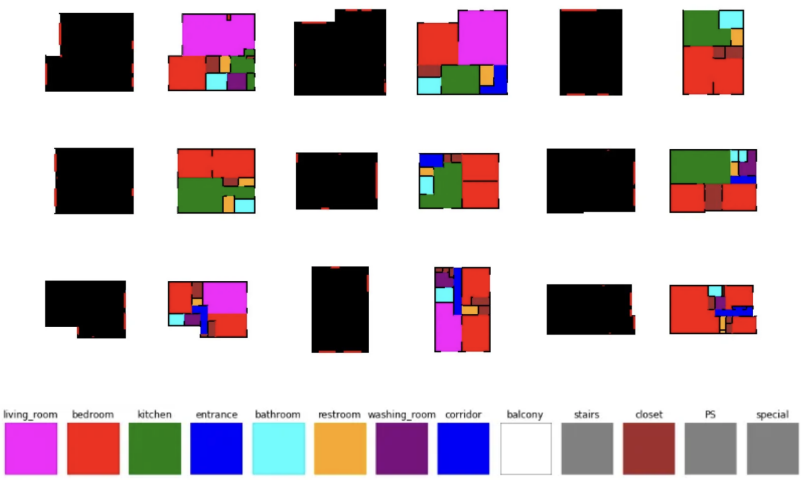


Figure 2-7. Example of a custom-made dataset for data-driven floor plan generation from (Chaillou 2020). The left side encodes the footprint of the apartment unit in black and the position of the facade openings in red. On the right, the pair displays the corresponding room layout with their programs encoded with the color scheme below.

- **Agent-driven methods**

Agent-driven methods have been less explored in the context of generative design compared to knowledge- and data-driven methods, with the few prior works demonstrating their potential and challenges. This category is more commonly known as agent-based models (ABMs), an analytical approach that focuses on the study of complex systems, via computational models that represent systems through their entities and their interactions (Cheliotis 2021, 1). The term agent refers to a construct immersed in an environment that uses its program to map percept sequences to actions, to solve a certain task (Russell and Norvig 2009). As opposed to data-driven methods that learn a set of results, agent-based models are experience-driven methods that learn certain behaviors. The policy of the agent defines what action to take based on the current and previous observations of the environment. This policy can be a simple reactive strategy, a set of if-then rules that looks for preconditions in the input to select the action, a set of cumulative behaviors, or a more sophisticated function that chooses an action, based on an estimate of the future performance (Veloso and Krishnamurti 2021a, 37). Reinforcement learning agent belongs to the last case, where the policy is a function in the form of a neural network that is trained to maximize the

estimated reward through trial-and-errors of the agent interacting with the environment. It is distinguishable from other agent-based methods because of its trial-and-error search and delayed rewards, which allows it to learn complex emergent behaviors.

When used as part of a generative system, the agent, by learning behaviors, embeds some form of decision-making to generate design alternatives. At any point in training, the agent performs a sequence of actions according to its learned policy that changes the design artifact. The behavioral patterns that the agent displays are directed by the configuration of the policy, characteristics of the environment, or initial state, all of which the designer can craft to their intentions and objectives. Like data-driven generative models, the system does not require an explicit description of the generative logic, but a simpler derivation that can govern interactions at the micro level - rewards for certain actions or states. It creates a phenomenon referred to as emergence in design where there is a coherent rise of patterns at the macro level (generation of designs) from the interactions between parts at the micro level (agent and environment) (Simon 2019, 169–172; Holland 1999, 115–124; De Wolf and Holvoet 2005). The agent can display unexpected behaviors or formulations that are only revealed during the execution of the generative model. Such methods can create flexible design spaces but come at the cost of lack of control, which is a trade-off that should be considered during the development of the model (Veloso and Krishnamurti 2021a, 38). Unlike data-driven methods where the generative process is encapsulated, techniques based on agents provide information during the training process and procedural information during the generation of artifacts which opens the possibility for more fine-grained interaction.

Interactive Generative Design Frameworks

The majority of generative frameworks involve interactions that are one-shot and unidirectional. The designer inputs an explicit set of criteria or corpus of data, and the generator synthesizes a possible set of designs. It is often the case that there is a large number of synthesized designs which the designer needs to scrutinize to make holistic evaluations. Previous works have sought to alleviate this process by proposing interactions where the user can navigate the generated design space with intuitive gestures such as drawing on an instance of the generated(Liu, Gao, and Liu 2005, 1; Matejka et al. 2018) or using sliders to

navigate the latent space and interpolate instances (Bidgoli and Veloso 2019, 7). However, these frameworks are still incapable of taking into account the ill-defined and changing criteria that are common in the design process. If the designer discovered new criteria or would like to change features, the model or dataset would have to be reformulated due to the one-shot nature of the neural networks involved.

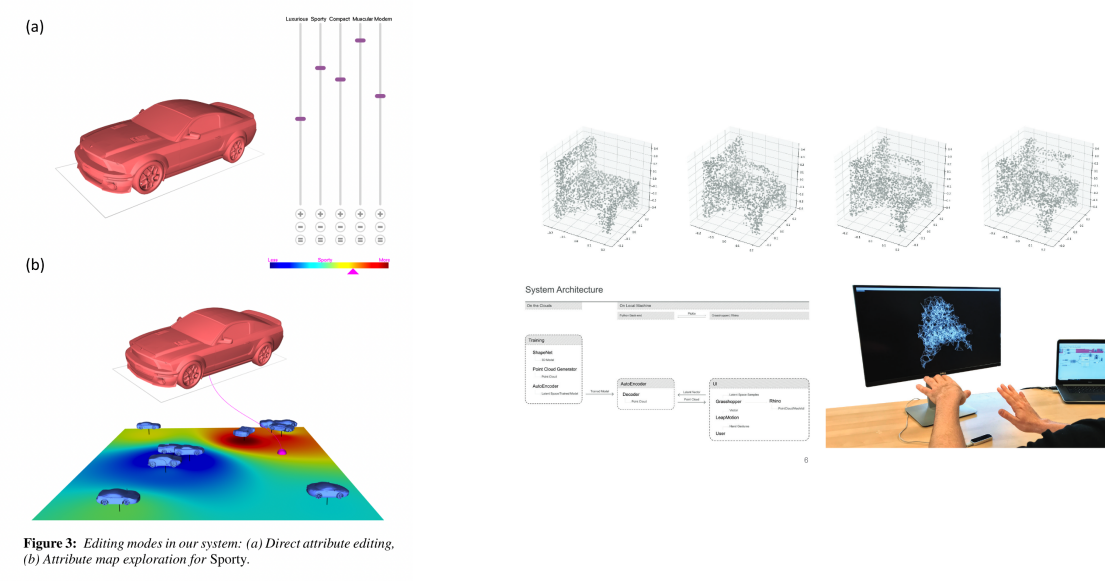


Figure 3: *Editing modes in our system: (a) Direct attribute editing, (b) Attribute map exploration for* Sporty.

Figure 2-8. Examples of interactive generative design frameworks. (Yumer et al. 2015) maps semantic attributes to geometry for smooth shape editing using deformation handles (left). (Bidgoli and Veloso 2019) employs Leap Motion for users to manipulate the latent space representation of a point cloud design space (right).

This brings attention to employing agent-based methods that are adaptive to changing conditions and feedback in generative design systems. (Groenewolt et al. 2018, 155) proposes an interactive computational design process for materialization-informed architectural design in 3D space where the designer interacts with individual swarm agents by modifying behavior parameters, behavior weighting, and the environment in real time. They propose the agent-based design approach not only as a method to generate a variation of design options and differentiation of building but also as a method for accommodating changes and for reformulating the solution space during the sequential development process of architectural design, which potentially reduces the complexity and cost of design changes (Groenewolt et al. 2018, 157).

2.3 Reinforcement Learning

Why use Reinforcement Learning (RL) for generative design?

This section introduces reinforcement learning as a means to enable an adaptive and interactive generative design framework. Reinforcement learning is a distinctive branch of machine learning with its feedback loop structure consisting of state, action, and reward information (figure) which allows it to learn from trial and error. This differs from the linear structure of other data-driven machine learning methods commonly used in generative systems such as GANs or VAEs. Such branches of machine learning algorithms have separate phases of training and testing, and trained models tend to perform poorly on instances that vary from the dataset it was trained on. On the other hand, RL models are continuously training, adjusting their policy to previous experiences as well as estimates of future performance. What distinguishes RL from other agent-based methods is that its policy is a neural network that is trained through trial and error while estimating future performance which allows it to learn relatively complex behaviors.

The proposed framework leverages the capabilities of RL agents to procedurally generate feasible 3D forms while adapting to evolving constraints. It seeks to enable the simultaneous articulation of design criteria and design artifacts which was not possible with previous generative design frameworks. I focus on the similarities in the feedback loop structure of RL algorithms and the iterative and evaluative process central to the design and how this opens possibilities for the generative design framework to be a shared platform where the computational algorithm and designer can interact i.e., observe and evaluate and make changes to the same design artifact. Previous works have shown that RL methods are effective in learning strategies for the sequential decision-making task of exploring the design space (Dworschak et al. 2022; Ororbia and Warn 2021; Agrawal and McComb 2023, 2). They have brought attention to RL as a viable alternative to support domain-specific agents in design (Veloso and Krishnamurti 2021b; Agrawal and McComb 2023) and integrate fine-grained interaction in the synthesis process in varied states (Groenewolt et al. 2018). Further, they highlight the generalization and transfer capabilities of RL-based methods as well as how they can accommodate non-differentiable objective formulations, not limited by

the data input by the designer (Regenwetter, Nobari, and Ahmed 2022; Agrawal and McComb 2023, 2).

Challenges in Designing an RL Agent for a Design Task

Reinforcement Learning is known to perform tasks for control, from video games (Mnih et al. 2013; Vinyals et al. 2019) to 3D locomotion ("DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning: ACM Transactions on Graphics: Vol 36, No 4" n.d.), to Go (Silver et al. 2018). The design of the environment and RL agent is characterized by the features of the task or its goal state. To an extent, the algorithm and its hyperparameters also is characterized by the task. Algorithms vary by the way data is collected, what data the agent learns from, to how it is synthesized, making certain algorithms better suited to perform certain tasks. All in all, training an RL agent to effectively perform a task, requires the design of the environment, agent, and algorithm to go hand in hand.

The designing of an RL agent itself is a process of trial and error; adjusting features of the environment, agent, and algorithm to test if it can train, i.e. display desired behaviors. To expedite the process, previous works using RL have relied on precedents involving similar tasks. One of the main challenges in this thesis is the lack of precedents with the similar task of creating a 3D form with an ambiguous goal. An ambiguous goal can mean that it is not certain to be feasible, bound to change, or gradually defined. The goal state of the task for traditional RL tasks is often explicit which is directly reflected in the agent's reward function. For instance, the agent aims to move to have the pendulum stay up as long as possible (Israilov et al. 2023) or reach the goal cell while collecting as many tokens (L. Chen et al. 2021). The goal state involved in this work is ambiguous because the designer is unable to provide the exact 3D form as they hope to find it through exploration.

This works experiments with a novel framework that sets partial information of the 3D form as pseudo goal states and articulates ambiguous goal states within an interactive training process. The designer provides a rough outline of the 3D form they would like to explore, i.e. a silhouette. Also, given a 3D form, they annotate or add on preferable features, which can serve to articulate the goal state. In the end, the RL agent and designer explore 3D

forms in collaboration - the agent suggests 3D forms according to the feedback that the designer provides.

Reinforcement Learning for 3D Design Tasks

3D shape modeling is a challenging and under-explored domain compared to 2D modeling. The existing agent-based modeling frameworks have limitations in developing fully 3D models, and even frameworks that support 3D are often restricted to 3D visualizations of 2D models (Cheliotis 2021). This thesis builds upon previous works that have employed reinforcement learning (RL) for design tasks or 3D shape modeling, which served as valuable references for developing a proof-of-concept.

Each of the previous studies delves into their approach in formulating the elements of RL for a design task and scopes the method's capabilities and limitations. For example, Veloso and Krishnamurti 2021b provide an example of spatial synthesis using multiple RL agents. These agents control spatial partitions represented as polyominoes with no holes and interact in an environment represented as a grid. They learn how to satisfy individual (shape, area, etc.) and collective goals (adjacency) provided by the user using multi-agent deep reinforcement learning (Veloso and Krishnamurti 2021b, 267) . Similarly, Narahara 2017 developed a multi-agent adversarial learning experiment that builds urban blocks in a three-dimensional grid. Ten competing color-coded groups of agents use a shallow neural network with custom weights to define their actions. Other works, including Lin et al. 2020, tackle the task of 3D shape modeling, where the depth map of a shape is given as a shape reference, and the goal is to 3D model this as a mesh. Two agents are trained - The Prim-Agent approximates the target shape using primitives, then the Mesh-Agent takes the primitives and edits the mesh of the primitives to produce more detailed geometry. “Onformative — AI Sculpting” n.d. presents a "sculpting agent" that learns a tool path to carve a known 3D form. Lastly, strategies for using RL in design search exploration have also been researched. Agrawal and McComb 2023 demonstrate how RL algorithms can autonomously learn to search a design space for high-performance solutions in engineered systems. This research informs how traditional RL frameworks can be altered to fit for design tasks, where the level of detail may vary alongside the development of the design.

The representations, use of agents, and definition of task of these works are reflective of the various facets the authors considered in formulating 3D synthesis or modeling as an RL task. First, the common representation of 3D forms in these works is through a grid format or set of predefined shapes. The agent takes actions of either deforming the units from the grid, adding or subtracting units from the grid, or selecting from the set of shapes. Such a format to represent 3D forms limits the size of the agent's action space which is integral in its learning. In terms of employing agents, Veloso and Krishnamurti 2021, Narahara 2017 use a multi agent approach where the agents are collectively taking actions and observing each other. They formulate 3D form synthesis as finding the form of individual parts so that they are compatible with each other and meet the constraints while creating a cohesive whole. Lin et al. 2020 have separate two agents that work consecutively - the output of the Prim-Agent is given to the Mesh-Agent as input. The 3D modeling task was divided into first finding the larger constitute volumes and then refining the shape by editing their edges. From this one can see that there are limitations in the complexity of the task that an RL agent can learn which should be considered in the formulation of the task. On the other hand, "Onformative — AI Sculpting" n.d. only employs one agent that learns a sequence of actions, i.e. a toolpath as a way of 3D form synthesis, but instead increases the resolution of the grid and uses post processing to produce a seemingly high resolution 3D form. One can also distinguish these works by whether the resulting 3D form was defined at the start of the task. Veloso and Krishnamurti 2021b and Narahara 2017 do not presume prior knowledge of the result shape and the objective is to find 3D forms that satisfy the given qualitative constraints. On the other hand, for Lin et al. 2020 and "Onformative — AI Sculpting" n.d. , the goal is a 3D form that is already known to exist, but only partial information of it in the format of a depth map is provided to the agent.

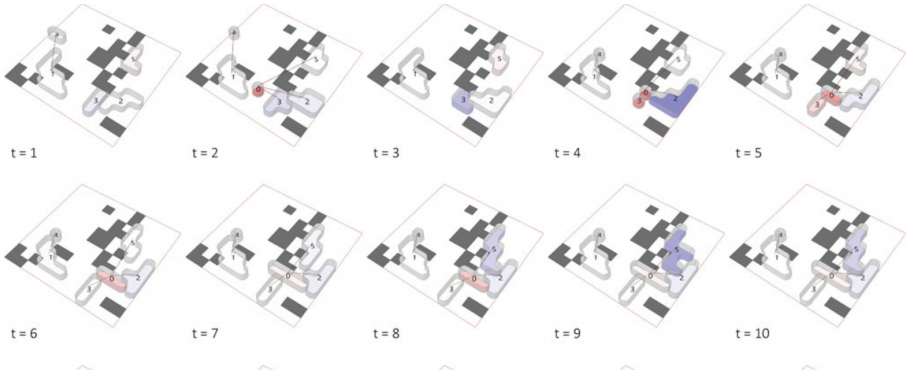


Figure 2-9. Veloso and Krishnamurti 2021b employ multiagent deep reinforcement for spatial synthesis. Simulation of six agents that are trained to satisfy shape, area individually, and adjacency collectively.

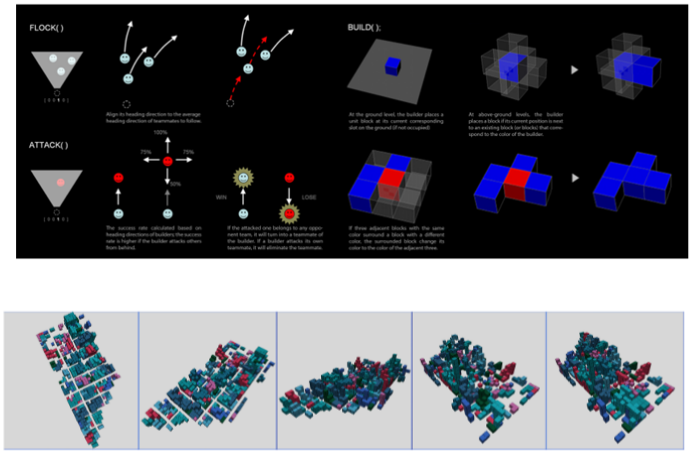


Figure 2-10. Diagram explaining builder's actions : step, flock, attach, and build (top) and 3D urban block results (bottom) from Narahara (2017).

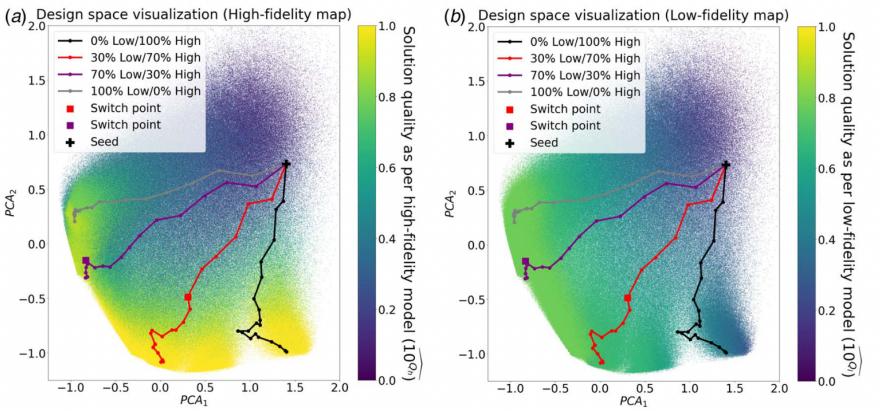


Figure 2-11. Exploration behavior of RL agents is visualized using PCA embeddings from Agrawal and McComb (2023).

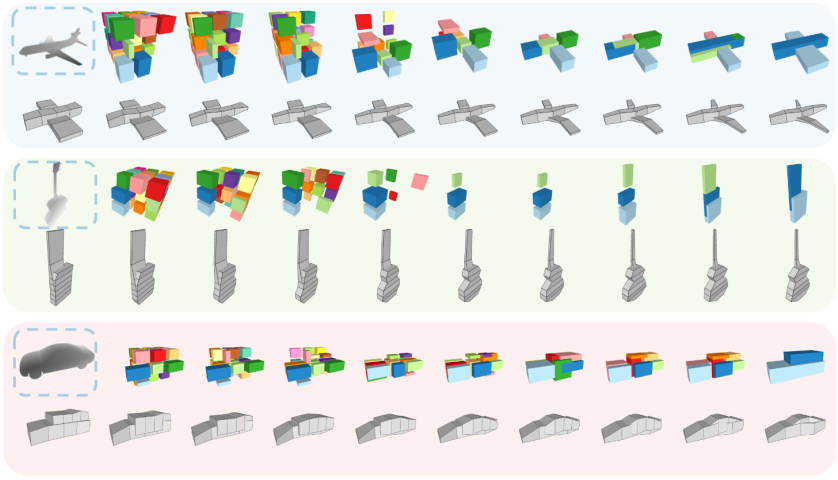


Figure 2-12. The step-by-step procedure of 3D shape modeling from Lin et al. 2020. The first row of each figure shows how the target shape is approximated with primitive shapes. The second row shows the process of mesh editing given the primitives.

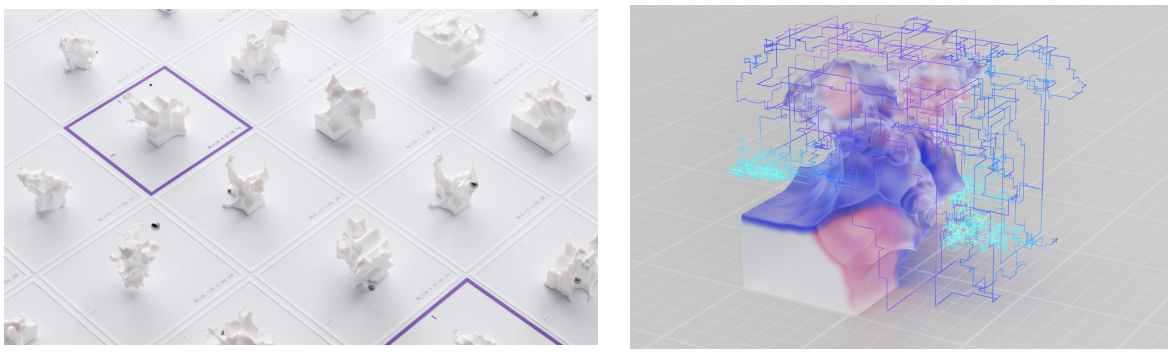


Figure 2-12. (“Onformative — AI Sculpting” n.d.) Presents an RL agent for 3D sculpting. *In the voxel-based environment infinite data and a clear reward structure was provided for the agent to move through. The starting state of the environment was one big cube. Out of this, the agent needed to remove mass to get closer to a predefined target state. With each step, the agent could decide where to go and if to remove a mass of voxels around itself and how.*

3 Methods

This chapter provides the methods involved in creating a proof-of-concept for this thesis. The first objective of this research is to formulate an unprecedented reinforcement learning (RL) environment and agent for 3D form finding. First, I provide an overview of the structure and elements of RL methods which will provide grounds to understand the challenges and considerations involved in designing an environment and designing and training agents for a novel task. Next, I present a 3D Form Finding Task, Voxel Grid Environment, and Builder Agent I have defined and employed for the proof-of-concept of this thesis including documentation of the trial and errors that I encountered to reach the results. The last section touches upon the methods involved in the second objective of this thesis, which is to explore an interactive training framework where the designer is involved in the feedback loop of the training of the generative agent to further articulate the design.

3.1 Reinforcement Learning Overview

In this section, I introduce the structure and elements of RL which will provide grounds to understand the challenges and considerations involved in designing an agent for the 3D form-finding task which is shown throughout this thesis. Reinforcement learning is about learning from interaction how to behave to achieve a goal. A RL task is typically formulated as a Markov Decision Process (MDP). An MDP has five components - agent, environment, state(observation), action, and reward that work together in a well-defined way. The *agent* interacts with its *environment* in discrete time steps. At each time *t*, the agent receives the current *state* S_t, or an *observation* which is partial information of the state, and *reward* r_t . It then chooses an *action* a_t from the set of available actions, which is set to the environment. The environment moves to a new state S_{t+1} and the reward r_{t+1} is associated with the transition (S_t , a_t , $S_{t+!}$) is determined (“Reinforcement Learning” 2023). The agent is taking a series of actions until it reaches the end goal(if there is any) or for a certain amount of steps which is indicated as one *episode* after which the environment is reset.

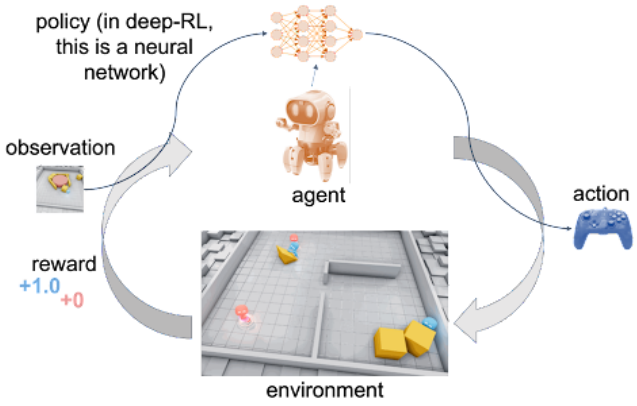


Figure 3-1. An abstraction of the interactions of the agent with the environment for reinforcement learning.[2]

The goal of the RL agent is to learn an optimal *policy* **π** which maximizes the *expected cumulative reward* or *return* from one episode. It is worth noting that the agent maximizes not just the immediate reward, but the cumulative rewards it receives over time which differentiates RL from other agent-based methods. The use of a reward signal to formulate the idea of a goal is one of the most distinctive features of RL (Sutton and Barto 1998). In the case of deep reinforcement learning, which is employed in this work, the policy is represented as a neural network and the specialized algorithms are used to train this policy.[3]

3.2 Unity Engine and Unity Machine Learning Agents Toolkit

I choose Unity as the development environment as it provides a graphical user interface (GUI) as well as an integrated machine learning toolkit. A GUI is essential to this study as one would like to observe the agent's behavioral patterns and the 3D form they create. Also, the interactive framework in this study requires an interface for the user to interact with the 3D form that the agent has created. The Unity Machine Learning Agents

[2] Image from ("An Introduction to Reinforcement Learning with OpenAI Gym, RLlib, and Google Colab | Anyscale" n.d.)
[3] In many practical decision-making problems, the states of the MDP are high-dimensional (e.g., images from a camera or the raw sensor stream from a robot) and cannot be solved by traditional RL algorithms. Deep reinforcement learning algorithms incorporate deep learning to solve such MDPs, often representing the policy or other learned functions as a neural network and developing specialized algorithms that perform well in this setting. ("Deep Reinforcement Learning" 2023)

Toolkit provides Python Trainers which contains RL algorithms, and Low-Level Python API which allows one to interact and manipulate a learning environment from simple commands in the terminal (Juliani et al. 2020). As a result, I choose Unity and Unity ML agents as it allows easy creation of custom environments, agents, and their training in a 3D environment especially compared to developing a GUI and implementing RL algorithms from scratch.

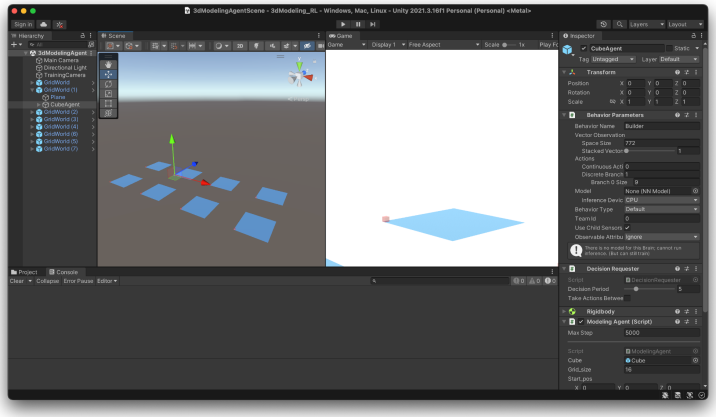


Figure 3-2. Screenshot of the voxel grid environment created within the Unity Game Engine. The environment is duplicated 8 times to accelerate the training process, as Unity ML-agent allows the training of multiple concurrent environment instances. Created by author.

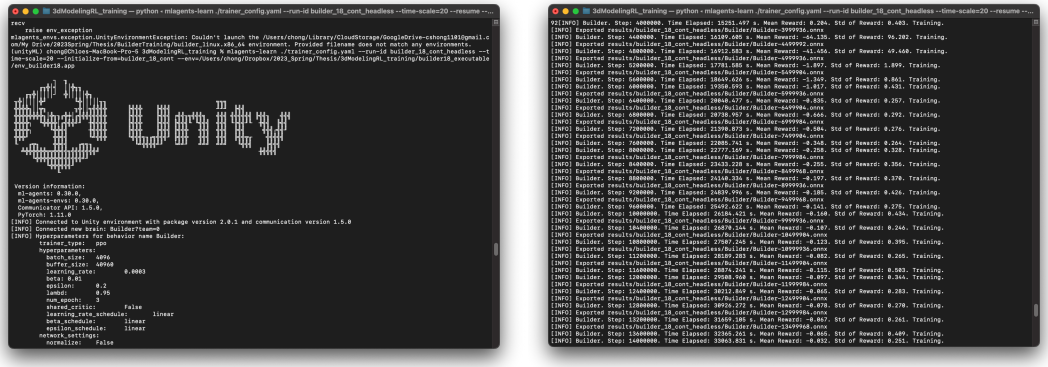


Figure 3-3. Screenshot of the terminal connecting the Python Trainer module to the Unity environment(left) and logging training results(right). Training the agent is possible through the Low-Level Python API provided by Unity ML-agent that allows communication between the Python Trainer modules and the Unity environment. Created by author.

3.3 3D Form Finding Task, Voxel Grid Environment, and Builder Agent

In this section I state how I define a 3D form finding task and formulate it into a Markov Decision Process (MDP) with the five components, agent, environment, state/observation, action and reward, that the RL methods require. The agent for this study is referred to as the *Builder Agent* as it is building cubes to generate a 3D form. The process of formulating each component was iterative and evaluative in itself. The details of each component were articulated through comparison or reasoning with relation to prior knowledge of training effectiveness and efficiency which is stated in this section.

3D Form Finding Task

I first define the 3D Form Finding Task as finding a sequence of building actions to create a 3D form given limited 2D information, or silhouettes. This resembles the process of translating sketches from multiple viewpoints in 2D into a cohesive 3D form. While it is relatively easy and effective in expressing the design criteria or distinctive features in 2D, the set of 2D illustrations does not necessarily form a cohesive 3D form. To generate one 3D form, that corresponds to the features intended in the 2D sketches, the designer must go through an additional process of iterating in 3D space while comparing with the initial 2D information. Likewise, the agent in this study is given 2D target silhouettes from three directions x,y, and z, and is expected to generate a 3D form that satisfies silhouettes simultaneously.

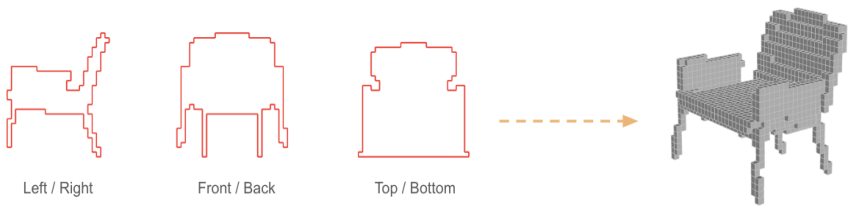


Figure 3-4. The 3D Form Finding Task is defined as finding a sequence of building actions to create a voxel 3D form given target silhouettes from three

sides. This example created by the author[4] illustrates the input and expected output.

Voxel grid Environment

I choose to implement a 3D grid environment where the 3D form is constructed by a combination of unit voxel cubes. Compared with other geometric representations, the voxel representation has three main advantages. One is that the transformation of a 3D form is a series of discrete actions of changing the value of the cell between 0 and 1, which is a differentiable function that a neural network can learn with backpropagation[5]. This can be compared with surface representations, where the components are interconnected and hierarchical, a change in one element propagates to adjacent elements, making its integration with learning-based methods complex. Another advantage is that with a voxel representation, any 3D form, regardless of its initial topology can be attained with the mentioned transformation. In contrast, when a 3D form is represented as a mesh, and transformed by moving vertices, the resulting geometry is limited to shapes with the same topology. Lastly, with voxels, the resolution of the result can be adjusted. While higher resolution allows expressing geometry in detail, it comes with the computational cost especially when used with learning-based methods. Thus the resolution can be adjusted according to the intended use.

In this study, a $16 \times 16 \times 16$ size voxel grid is used in consideration of the tradeoff between expressibility and computational cost. In terms of expressibility, the resulting 3D form must be expressive enough to be valid as a design tool. The higher the resolution, the resulting form can contain more details with smoother surfaces. However, the higher resolution demands more training time and computational power. It is important that the training time be kept within a reasonable range as multiple iterations of retraining is needed

[4] The example was created by taking an chair 3D mesh model, voxelizing it to a 32*32*32 resolution, and tracing the silhouettes from the three sides left, front and top.

[5] In machine learning, backpropagation is a widely used algorithm for training feedforward artificial neural networks or other parameterized networks with differentiable nodes. Backpropagation computes the gradient of a loss function with respect to the weights of the network for a single input–output example, and does so efficiently, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this can be derived through dynamic programming. Gradient descent, or variants such as stochastic gradient descent, are commonly used. The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network. This is the reason why backpropagation requires that the loss function be differentiable. (“Backpropagation” 2023)

especially in the interactive framework. The resolution of the voxel grid affects the hyperparameters of the RL agent training and thus training time.

The following factors were considered in deciding on the voxel resolution. First the expressibility of different resolutions was compared using the target silhouettes. An instance of a chair model from the ShapeNet dataset (Chang et al. 2015) was used to generate the set of target silhouettes which was used throughout the proof-of-concept. The dataset provides a voxelized version of the 3D model at the resolution of $32 \times 32 \times 32$, from which the silhouettes of size 32×32 from the three orthogonal x,y,z directions are taken. The set of silhouettes are downsized each to 16×16 and 8×8 to compare the expressiveness of the resolution. While the framework does not assume the target silhouette to be derived from a known 3D form, for the proof-of-concept this method was used as it would be enable the initial comparison of different resolutions to decide the voxel grid resolution and the original 3D form would serve as a reference to compare the generated results in the evaluation of the framework. In the comparison of target silhouettes of different resolutions, while the detail expressed in resolution 16×16 and 32×32 are similar, with 8×8 thinner features such as parts of the armrest and curvatures in the chair are lost.

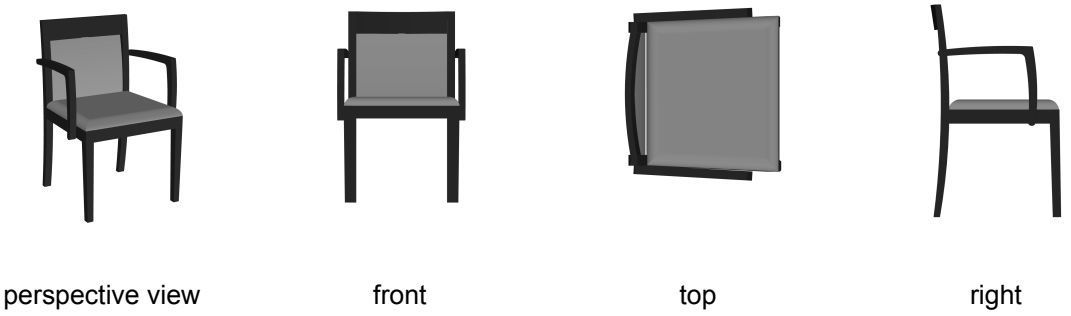


Figure 3-5. Original chair model from ShapeNet. Screenshot taken from top, front, right direction by author.



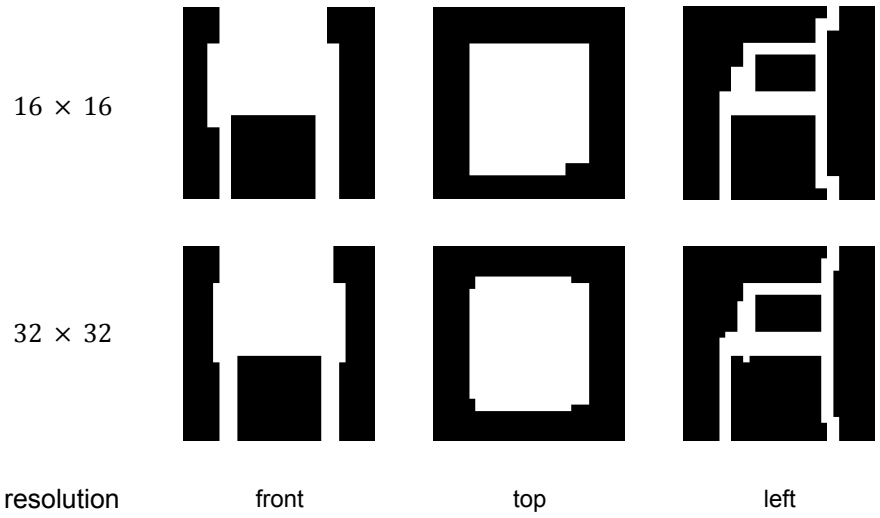


Figure 3-6. Comparison of expressibility of different resolutions in target silhouettes.

Secondly, the voxel resolution affects the time and computational power used to train the agent. The agent is taking multiple trials of performing the task to learn the optimal policy. Each trial is taken within an episode, where the agent takes a series of actions until the task is accomplished or up to the maximum number of steps in one episode that is decided beforehand. Because it is fixed that the agent takes one action in one time step, the longer the length of each episode, the more steps the agent has to take in total to learn the task, which prolongs the training time.

The maximum length of one episode should be at least as long as how much it would take for the agent to perform the task and this is contingent on how many voxels the target geometry is composed of. For instance, if the actions that the agent can take is move or create, and the target geometry is composed of 500 voxels, the maximum number of steps should exceed double the voxel count of the target geometry - 1000. This is because even when the agent has learned the most optimal policy, in other words, has learned to generate the form precisely without any unnecessary actions, it would require 2 actions (move and create) per voxel cell of the target geometry. However, considering that in the early stages of the training, the agent is not as efficient, it should be given a longer time to explore. Here, I make the assumption that 3~4 times the target geometry voxel count would be an ample length for the agent to learn. Also given the nature of the task where one only knows the 2D silhouettes, but not the exact 3D target geometry, I make the assumption that the target

geometry occupies approximately 50% of the entire voxel grid. As a result, in this comparison the episode length is 1.5~2 times the total voxel grid size.

Resolution	Maximum environment length
8x8x8	768~1024
16x16x16	6000~8000
32x32x32	50000~65000

Table 3-1. The maximum environment length is estimated as an indicator of training complexity for different voxel grid resolutions.

Other hyperparameters that are contingent on the voxel grid resolution and would likely increase the training time are buffer size and network size. These hyperparameters are decided based on the complexity of the task. Unlike the episode length, while it is difficult to calculate an approximate value based on the voxel resolution, it is intuitive that a higher resolution means that the task is more complex, thus requiring a higher value for each hyperparameter.

The buffer size corresponds to how many experiences (agent observations, actions and rewards obtained) should be collected before the update of the model and the batch size is the the number of experiences used for one iteration of a gradient descent update. The typical range for the buffer size is 2048 - 409600 and typically a larger buffer size corresponds to more stable training updates. For more complex tasks, for the agent to train, more information is required, asking for a larger buffer size. Depending on the size of the batch size[6], it would either require many gradient descent updates with small batches or few gradient descent updates with large batches to update the model. Either way, a large buffer size requires more computational power and the training time increases accordingly.

The network size consists of the number of hidden layers and the number of hidden units that are in each fully connected layer of the neural network. The typical range for the number of layers is 1-3. For simple problems, fewer layers are likely to train faster and more efficiently while more layers may be necessary for more complex control problems. Likewise for problems where the action is a very complex interaction between the observation

[6] Multiple iterations of gradient descent update are performed in one update of the model, making the buffer size a multiple of the batch size.

variables, the number of hidden units should be larger with the typical range being 32-512. It is known that a larger network (with more hidden units) will take longer to converge as the function being trained is more complex. Therefore, if the resolution of the voxel grid is increased, these hyperparameters in the reinforcement training algorithm must be increased to ensure that the agent learns the task, likely increasing the training time.

Builder Agent State and Actions

The agent's state and actions are the building block for its behavior. As the Builder agent's goal is to generate a 3D form within the voxel grid environment with a series of actions, the set of actions should enable behavior to reach that goal. As with the voxel grid resolution, the size of the action space has an impact on the training time and should be taken into consideration. Typically a larger action space requires more training because at every step the agent likely explores each action to find the one that maximizes the episode's reward. Therefore a smaller action space is preferable. In this study, I consider 3 different sets of actions - Move or Create, XYZ Create, and Concurrent Create, and perform training with 2 to compare the agent's behavior and training performance. After comparison, the Builder Agent is implemented with the Move or Create action set.

- **Move or Create**

With this set of actions, the Builder agent's state is a location within the voxel grid and it is able to choose to move orthogonally or create or destroy a voxel at each step. The action space is composed of one branch that has 8 indices, making the total size of the action space 8 regardless of the $grid\ size$. The actions were assigned as follows with an added "no action" action as a default not give bias toward any of the other valid actions.

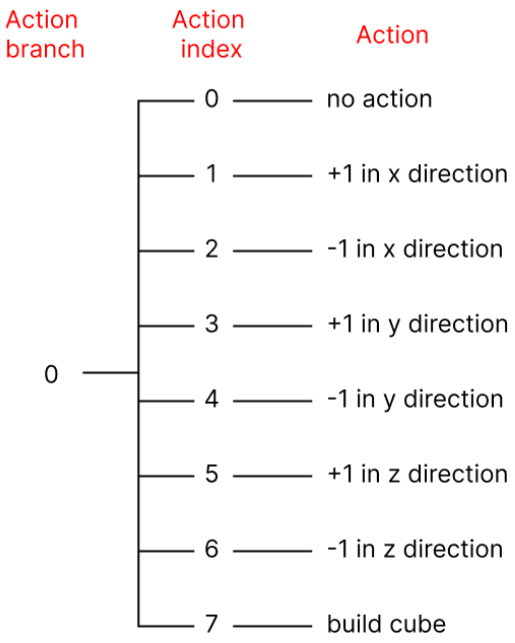


Figure 3-7. Move or Create action set mapping organized in action branch and index format used in Unity ML-agent.

- **XYZ Create**

With this set of actions, the Builder agent's state is a location within the voxel grid and is able to creates a voxel at any coordinate (x, y, z) within the voxel grid at each step. The sum of discrete branch sizes is $gridsize^3$ which increases exponentially with the grid size. Therefore I use a smaller size 8 grid to make the total action space size 512. However, an iteration of training with the Builder agent given this set of actions showed that it fails to train.

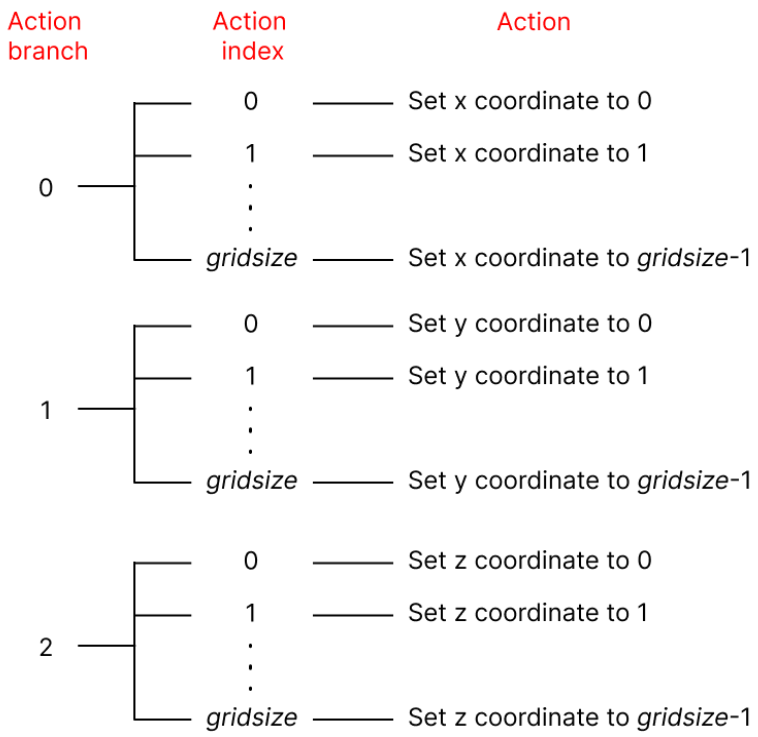


Figure 3-8. XYZ Create action set mapping organized in action branch and index format used in Unity ML-agent.

- **Concurrent Create**

With this set of actions, the Builder agent creates a set of voxels across the grid. Each index of the action space represents the occupancy of a voxel in the grid. The action branch represents the y coordinate, and the action indices each branch correspond to a (x,z) coordinate. The sum of discrete branch sizes is $gridsize^3$ which increases exponentially with the grid size. Even when the grid size is 8, the action space size would be $1.340781e154$ which is immensely large and would require extensive training time. Therefore I do not forgo this set of actions.

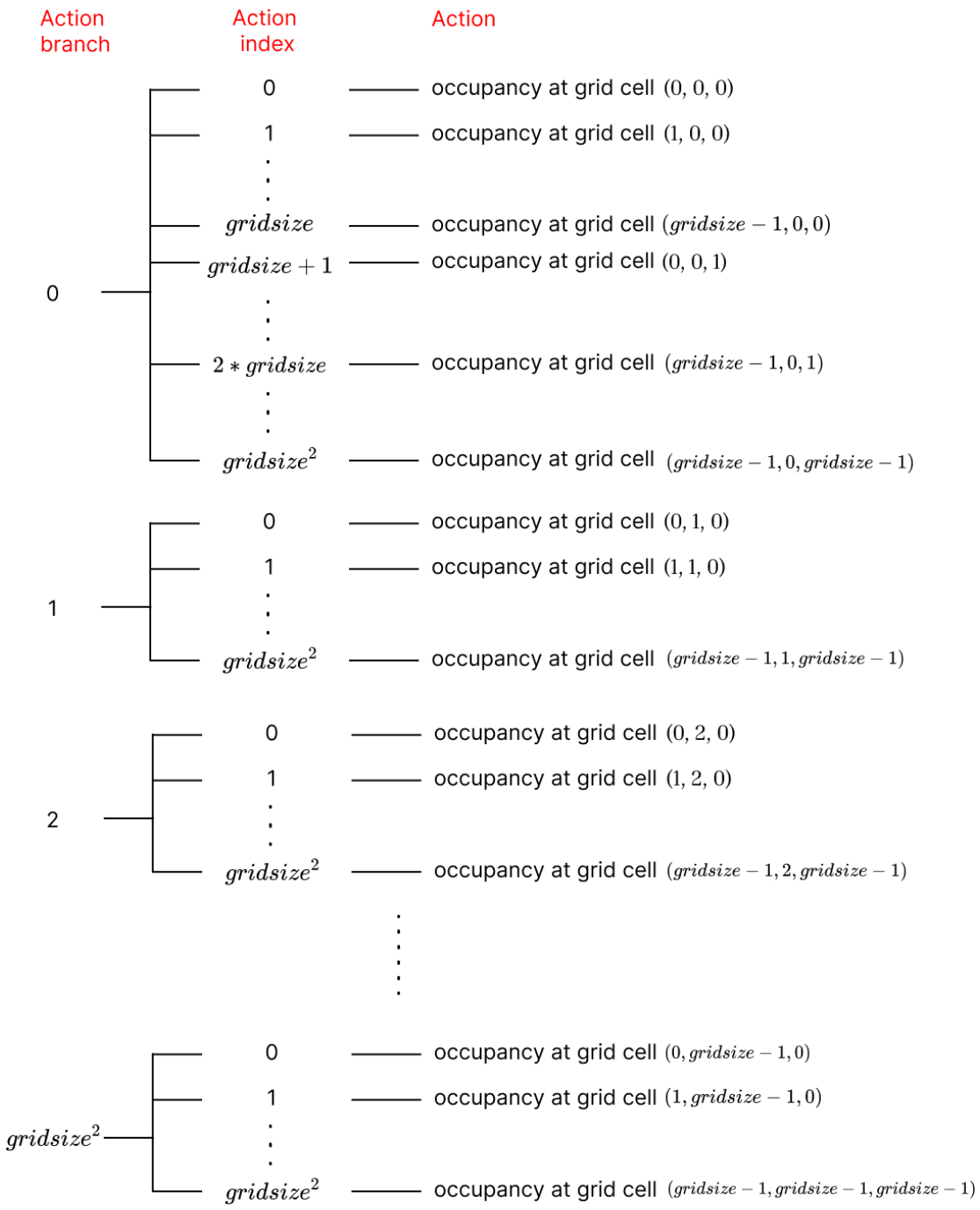


Figure 3-9. Concurrent Create action set mapping organized in action branch and index format used in Unity ML-agent.

3.4 Builder Agent Observations and Rewards

Observations

It is vital to choose a set of observations that provides the essential signal information to perform the task. If the observation is too sparse or has little correlation to the actions and goal state, the agent may fail to train. However, providing extensive information as observation is not ideal because excessive information acts as noise which makes it even difficult for the agent to train. Also it is worth noting that all the observations are stacked to a 1D vector. This shows that the agent does not know the semantic meanings of the observations but only finds patterns between them and the rewards to learn the policy.

The Builder agent's observations largely consist of the current location of the agent, current occupancy of the cell at the current location and information of the current geometry. With the move and create action set, at each step the agent is at a (x,y,z) coordinate within the voxel grid. The voxel in that location may be filled (in the case that the agent has built a cube there before) or empty, which is recorded with a boolean value. Lastly, three options were considered to represent the state of the current geometry – silhouettes, signed distance maps, and voxel map. Out of the three representations, the silhouette is directly related to our goal of generating a 3D form that matches the target silhouette. At the same time because the agent has to figure out how to build a 3D geometry, 3D information of the current state should be useful, which is why signed distance maps and the voxel map were also considered. While a map of the occupancy of the entire voxel grid would be the most direct signal of the 3D form, the large size (4096) would likely require a larger network and thus take a longer time to train. On the other hand, signed distance maps, with size 768 from 3 sides can be an alternative, it is an indirect indicator of the current silhouettes which the reward function is based on, also requiring a complex network and possibly more training time.

Furthermore, in order to create a closer correlation between the observation and rewards, I choose to express the state of the current silhouettes in relation to the target silhouettes. I speculate that this correlation would make it easier for the agent to discover a

pattern between actions and rewards, i.e. learn a policy. This information was encoded in a *sensor map* where each cell in the silhouette is labeled with the following indices.

Target state	Current state	Index
empty	empty	0
empty	filled	1
filled	empty	2
filled	filled	3

Table 3-2. Sensor map indices assigned by current state of silhouette cell in relation to target silhouette cell at the same location.

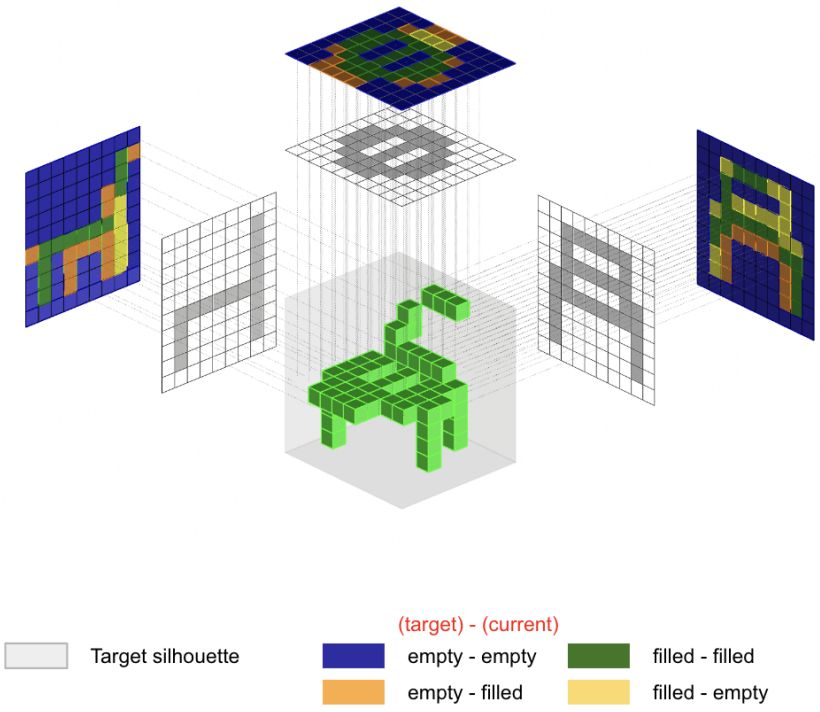


Figure 3-10. Illustration of sensor map where the current silhouette of the form is indexed in relation to the target silhouette.

Each cell in the silhouette can be filled or empty. If the target silhouette indicates that the cell should be empty, and the current silhouette is empty the cell is labeled with the index 0, but if it is filled, the cell is labeled with the index 1. Likewise, if the target silhouette indicates that the cell should be filled, and the current silhouette is empty the cell is labeled with the index 2, and if it is filled, it is labeled with index 3.

Reward function

The reward function has the greatest influence on the behavior, or patterns in the series of actions that the agent displays. The reward function is a combination of different signals. Typically the positive reward signals are given to encourage preferred behavior that would lead the agent to perform the task and a negative reward signals or penalty is given to discourage undesired behavior ("Define Reward Signals - MATLAB & Simulink - MathWorks España" n.d.). Reward functions are task-specific, thus was extensively tested and adjusted for in this study. I start out with an initial function based on my knowledge of the problem, observing how the agent performs in terms of both quantitative performance (changes in the cumulative reward increases) and qualitative performance (what behaviors the agent displays).

Some considerations when designing a reward function are the weights of each signal in the reward function and the frequency at which the rewards are given. First, the relative size of the signals and their corresponding contribution to the reward function must be scaled. Some basic rules I have applied were to assure that the combined reward signal is either an extreme positive or negative reward value as mixed signals which cancel out each other to be close to 0 does not give the agent useful information. Also it has been noted that the magnitude of the reward should be scaled to be within the range of [-1,1] to prevent vanishing or exploding gradients[7] in the training of the agent.

[7] There are a number of common ways for backpropagation to go wrong. The gradients for the lower layers (closer to the input) can become very small. In deep networks, computing these gradients can involve taking the product of many small terms. When the gradients vanish toward 0 for the lower layers, these layers train very slowly, or not at all. If the weights in a network are very large, then the gradients for the lower layers involve products of many large terms. In this case you can have exploding gradients: gradients that get too large to converge.

The final reward function that was effective in training a builder agent was as follows. Overall a strong positive reward is granted when the agent builds cubes in locations of the voxel grid that correspond to the target silhouette, and other smaller penalties for building in locations that do not correspond to the target silhouette. Also strong negative rewards are granted to prevent the agent from taking certain actions.

$$Reward = r_{sensormap} + r_{no\ action} + r_{not\ allowed} + r_{move}$$

$where$

$$r_{sensormap} = +0.25\ (at\ build\ cube\ if\ filled\_empty\ ->\ filled\_filled\ in\ each\ silhouette)$$
$$-0.025$$
$$(at\ build\ cube\ if\ empty\_empty\ ->\ empty\_filled\ in\ each\ silhouette)$$
$$-0.0125$$
$$(at\ build\ cube\ if\ empty\_filled\ ->\ empty\_filled\ in\ each\ silhouette)$$
$$r_{no\ action} = -0.125\ (when\ default\ action\ 0\ is\ taken)$$
$$r_{not\ allowed} = -0.25\ (when\ tries\ to\ build\ where\ cube\ already\ exists)$$
$$r_{move} = -0.05\ (when\ selects\ a\ moving\ action)$$

The sensor map which was used as part of the observation is also used to configure the reward signal. The reward signal $r_{sensormap}$ is intended to encourage the agent to build cubes in locations that would align to the target silhouette. A large positive reward (0.25) is given when the cube the agent builds fills in a cell in the silhouette that was previously empty, but based on the target silhouette should be filled. In terms of the sensor map this is when the index changes from index 2 to 3. A small penalty (-0.025) is given when the agent builds a cube to fill a silhouette cell that is supposed to be empty that was previously empty (index 0 to 1). A smaller penalty (-0.0125) is given when the agent additionally creates a cell in a silhouette cell that is supposed to be empty but is filled (index 1 to 1). The reason why these penalties are small is because the agent still should be encouraged to build. If the penalty of building in the wrong cell is too large, the agent would choose to avoid the risk of building and choose a "safer" action of moving. Other larger penalties were given to prevent certain actions. In the design of the agent's action set, a default action where the agent doesn't move nor build was added to prevent bias toward any of the other valid actions. To prevent the agent from taking this action, and to encourage it to move or build, $r_{no\ action}$ is a penalty (-0.125) given whenever the agent takes this default action. Moreover, a large penalty

$r_{not\ allowed}$ (-0.25) is given whenever the agent builds a cube in a location of the voxel grid that is already occupied. Otherwise, the agent would continue to build cubes in the same location that yields large $r_{sensormap}$. Lastly, a small penalty r_{move} is given for every time the action chooses a moving action as opposed to a building action. This is to encourage the agent to take the building action regardless of the risk of receiving a penalty for building in the wrong places.

Training Hyperparameters

For the training algorithm I chose to use a model free policy gradient method Proximal Policy Optimization (PPO) which has the advantages of ease of implementation, sample complexity and ease of tuning[8]. As the optimal hyperparameters for training can only be determined by trial and error, a recommended practice is to initialize hyperparameters from a similar task. Based on the similarity in the grid-like environment, agent observation space and actions, I take the hyperparameters used in the Pyramids task from Unity ML-Agents[9]. The batch size and buffer size were adjusted to a higher value of $4096,\ 40690$ considering that the ample amount of information needed to train the 3D form finding task. Compared to the Pyramids task, the expected time for the agent to perform the task is far longer, making the episode length longer (2500 time steps). Since it is ideal that the buffer includes information of multiple episodes before it is passed on to update the model, I chose to significantly increase the buffer size. The time horizon, which is the number of steps that were incorporated in calculating the reward function, was increased to 256, considering that this task requires learning a longer sequence of actions. Lastly, the number of hidden units was decreased to 256 as it was discovered in the trials, there was little difference in the

[8] PPO addresses the challenges of policy gradients – the performance being sensitive to the choice of step size and having poor sample efficiency, taking millions or billions of time steps to learn a simple task. However it has a simpler implementation than Trust Region Policy Optimization (TROP) and Actor Critic with Experience Replay (ACER) which seek to do the same but require extensive hyperparameter tuning. PPO does not require analytical results, constraint-based programs or second-order derivatives but instead performs the update using popular stochastic gradient descent algorithms like ADAM, and adjusts the penalty when updates are too large or too small. (Schulman et al. 2017)

[9] Environment where the agent needs to press a button to spawn a pyramid, then navigate to the pyramid, knock it over, and move to the gold brick at the top. Vector Observation space: 148 corresponding to local ray-casts detecting switch, bricks, golden brick, and walls, plus variable indicating switch state Actions: 1 discrete action branch, with 4 actions corresponding to agent rotation and forward/backward movement. (Juliani et al. 2020) Config file : https://github.com/Unity-Technologies/ml-agents/blob/develop/config/ppo/Pyramids.yaml

agents training with 512 units, but larger network size would require longer training. In Appendix B the complete set of hyperparameters are shown.

3.5 Interactive Training Framework

The second objective of this thesis is to explore an interactive training framework where the designer is involved in the feedback loop of the training of the generative agent to further articulate the design and design criteria. Instead of having the agent generating forms based solely on the initial 2D silhouettes that were provided, would it be possible to give it feedback within the training process, and have the agent adapt its behavior to generate forms that satisfy the intermittent feedback?

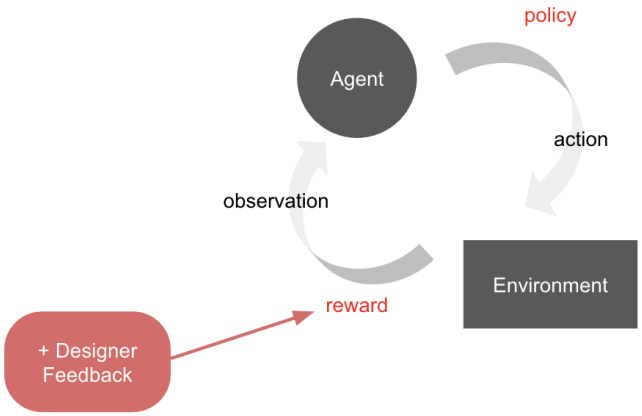


Figure 3-11. Diagram of interactive training framework where adding designer feedback affects reward function and eventually the policy during the training process.

This study is built upon the following distinctive characteristics of the design exploration process and the training of neural networks. The first is that the objective of this framework is for design exploration, where any partial forms that are interesting to the designer have value. Throughout its training process, although the quantitative performance may not be optimized, the Builder agent is generating some form. Even when the form does not satisfy the 2D silhouettes completely, they can be interesting enough to serve as a seed for exploration for the designer. Moreso, the partial forms that emerge as the Builder Agent is

sequentially building can inspire designers to discover new criteria or further articulate the design artifact.

I proposed the following framework for interactive training of the Builder Agent. First, the designer pauses the training process at a point where the Builder agent would likely produce partial forms that are interesting. The agent is initialized to perform a series of actions that would consecutively build a 3D form within the voxel grid. The designer pauses the action within its execution and provides feedback in the form of cubes the voxel grid encouraging the agent to build in those locations. This feedback is added as a signal in the reward function. The Builder agent continues to train from the point with the modified reward function which would likely affect the agent's policy and its behavior. The hypothesis is that the agent would adopt its policy to modify the reward function, and the newly generated forms would take into account the designer's feedback.

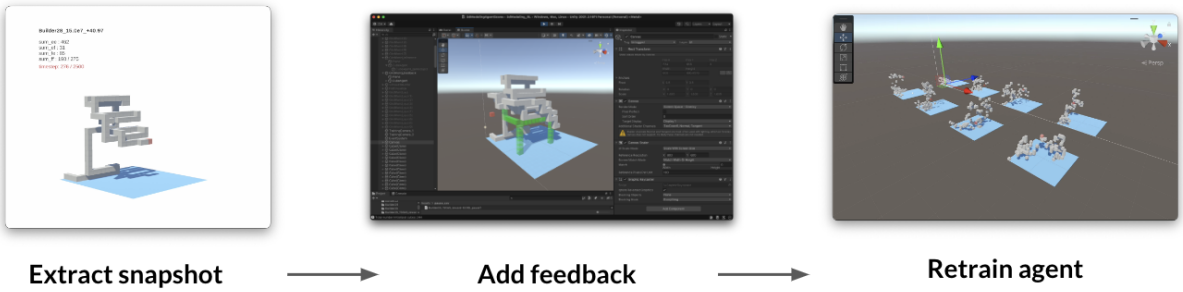


Figure 3-12. Interactive framework from designer/user perspective.

The implementation of this framework involves creating three modes - inference mode, feedback mode and training mode within the Unity game environment that each support the running of an agent, providing feedback and retraining with additional feedback. First the inference mode takes a neural network onnx file as an input and at initialization has that agent run its policy in the grid environment. The designer can press the key 'p' to pause the agent in the middle of its execution to take a snapshot of the current voxel grid composition, which is a csv file that contains the (x,y,z) voxel grid coordinates of the currently occupied voxel cells.

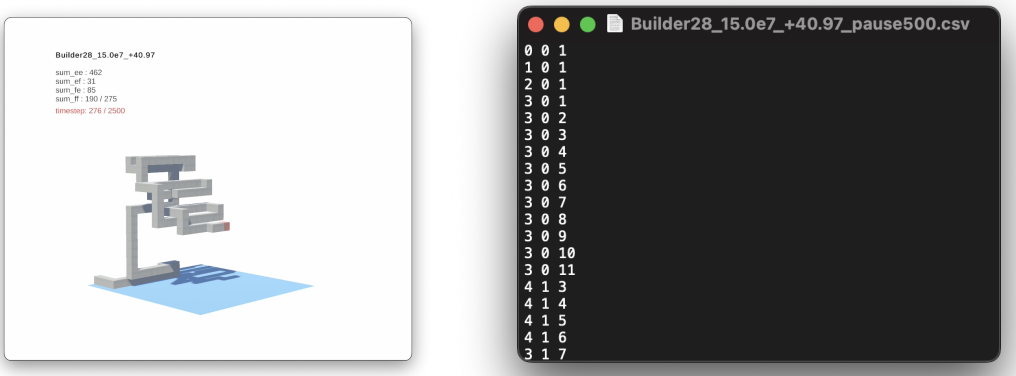


Figure 3-13. Interface for inference mode. The number of time steps and sensor map information is stated on the top left (left). The snapshot on the left is recorded as a csv file (right).

The feedback mode takes the snapshot csv file from the inference mode as input and at initialization creates the voxel grid composition of the snapshot. The designer can then move a cursor within the voxel grid to add feedback cubes. When they are finished with drawing their feedback, at the press of the enter key, the coordinates of the feedback cubes are extracted into a csv file.

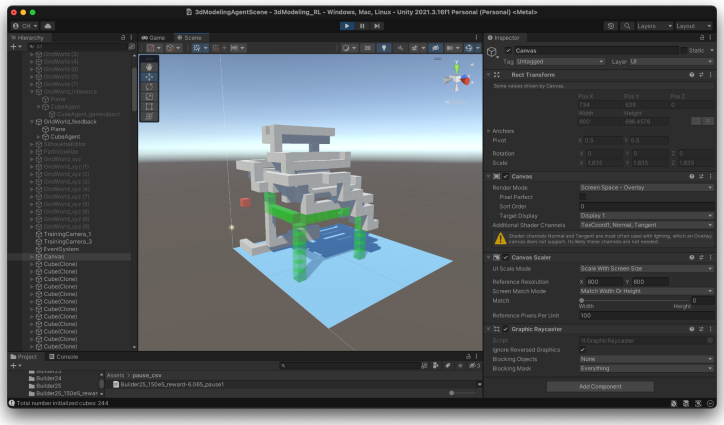


Figure 3-14. interface within unity game engine for feedback mode. The gray cubes are the composition from the snapshot and the green are feedback cubes.

The training mode takes in the feedback csv file as input and continues to train the agent with a modified reward function that incorporates the provided feedback. A large positive reward signal for building a cube in the coordinates of the feedback was added to the reward function. As with formulating the specifics of the Builder Agent, a trial and error process of

fine tuning the magnitude of this feedback was required. In the end I decided to give a reward signal of 1.5 for each time the agent built in one of the feedback coordinates.

$$Reward = r_{sensormap} + r_{no\ action} + r_{not\ allowed} + r_{move} + r_{feedback}$$

where

$$r_{sensormap} = +0.25\ (at\ build\ cube\ if\ filled\_empty\ ->\ filled\_filled\ in\ each\ silhouette)$$
$$-0.025\ (at\ build\ cube\ if\ empty\_empty\ ->\ empty\_filled\ in\ each\ silhouette)$$
$$-0.0125\ (at\ build\ cube\ if\ empty\_filled\ ->\ empty\_filled\ in\ each\ silhouette)$$

$$r_{no\ action} = -0.125\ (when\ default\ action\ 0\ is\ taken)$$

$$r_{not\ allowed} = -0.25\ (when\ builds\ where\ cube\ already\ exists)$$

$$r_{move} = -0.05\ (when\ selects\ a\ moving\ action)$$

$$r_{feedback} = +1.5\ (when\ builds\ in\ the\ coordinate\ of\ the\ feedback)$$

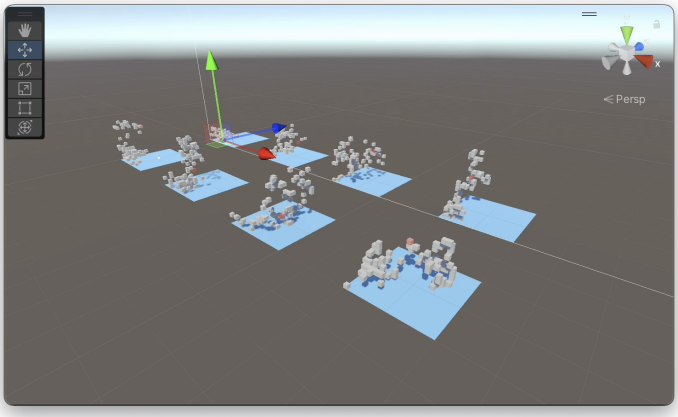


Figure 3-15. Interface at training mode. The environment is duplicated to collect trajectories simultaneously and expedite the training.

4 Results

4.1 Evaluation of the Builder Agent

In this section, I evaluate the Builder agent's performance with the objective of it being employed as part of a generative design framework. It is important to note that the objective of this framework is for design exploration, where any partial forms generated within the process that is interesting to the designer may have value. More so, the partial forms that emerge as the Builder Agent is training or executing a policy can inspire designers to discover new criteria or further articulate the design artifact. This is why it is important for the evaluation of the agent in this study to look at the agent's behavior and the forms it generated across the training process, as opposed to only looking at its optimal state. This is distinctive from traditional reinforcement learning tasks, that have an explicit terminal state, and the goal is known to be achievable. In these cases, the focus of the agent's training is to find the most efficient way of achieving the goal and the evaluation is surrounded by the optimal return value and the agent at that point. However, in this study, even when the return value has not reached an optimal value, there is a possibility that the agent is generating 3D forms that are useful for design exploration.

The Builder agent's performance can be evaluated in terms of quantitative performance - changes in the cumulative reward, if it increases and stabilizes and the qualitative performance - what behaviors the agent displays, more specifically if it is generating forms that correspond to the target silhouettes. After a trial-and-error process of 28 iterations, I formulated a Builder agent that shows both satisfactory performance quantitatively and qualitatively that is used for evaluation. The following evaluation presents the quantitative training graph in parallel with the qualitative agent behaviors at certain intervals of the training to observe how the Builder agent learns certain behaviors and the 3D forms it generates within its execution of its policy.

First, the quantitative training reveals that the agent before 50 million steps, experiences a phase of drastically low returns with high fluctuation, and then gradually plateaus at a return of +45 by 150 million training steps. I speculate that the agent gradually learns fundamental behaviors to build and create 3D forms before 50 million steps, and then wildly explores later to reach an optimal policy where it makes little changes. The first step is to identify points of interest within the training process to observe agent behaviors. The figure below shows the quantitative performance of the

Builder agent with the cumulative reward of the agent (y axis) for each step within the entire training process of 260 million time steps(x axis). I identify segments *a,b,c,d* that show different patterns within the performance. The assumption is that the agent's policy varies in these segments, and thus the agent behaviors would be distinctive as well. The changes in the agent's behavior across these segments can reveal how the Builder agent learns certain behaviors and the different 3D forms it generates. The reasoning behind the selection of each segment and the agent's behaviors within those segments are further elaborated later on.

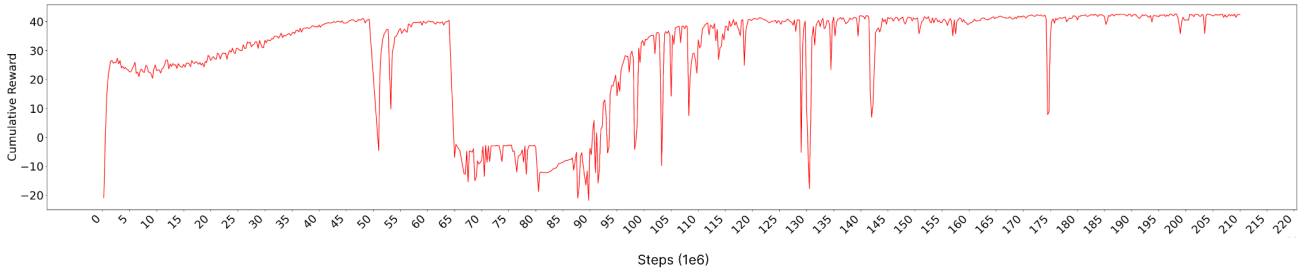


Figure 4-1. Quantitative performance of Builder that shows cumulative rewards agent over 220 million training steps.

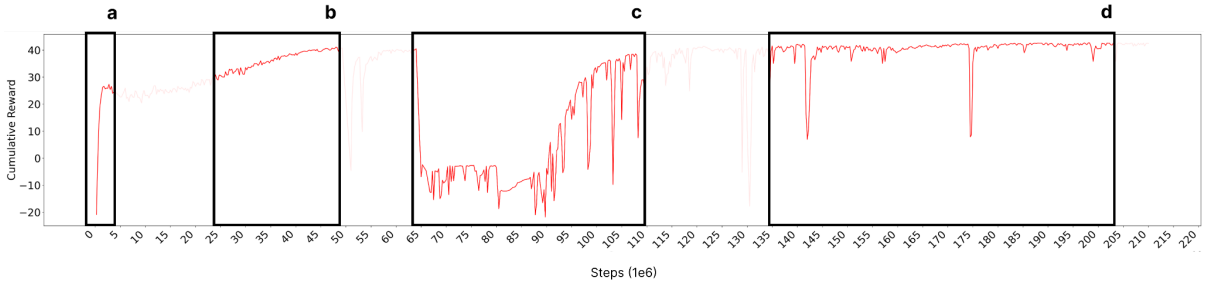


Figure 4-2. Quantitative performance of Builder agent labeled with intervals of interest.

Three or four points of interest are taken from the four segments that can show the even fine grained changes in agent behavior. During the training, the Builder agent's policy, a neural network is saved as checkpoints at every 250000 time steps. I retrieve the checkpoint at the points of interest, and run and record the agent's policy in the environment. The qualitative figure on the bottom half lays out snapshots of the agent performing it's policy within one episode of 2500 time steps. The main purpose of this figure is to examine the different forms the agent creates at various points in the training, as well as the trajectory it takes within one episode to create that form. The time intervals at which the screenshot was taken are closer together in the beginning of the episode which is reflected in the

relatively small spacing between screenshots. This choice was made because I noticed the form the agent creates changes rapidly in the beginning of the episode as opposed to towards the end.

Lastly, I employ a heatmap of red colored areas surrounding the screenshots that can indicate how good of a job the agent is doing at satisfying the initial constraints of the target silhouette. The target silhouette given for this task has a total of 275 cells that should be filled. During the training, the number of filled cells in target silhouette that are filled in the current silhouette are kept count with the index 3 of the sensor map which I here denote and *filled_filled* states. With areas of different opacity, I mark the time step within the episode at which the percentage of *filled_filled* cells passes the thresholds of 30% (73/275), 75% (188/275), and 90% (248/275), where the higher the threshold, the darker the red color is. The heat map across all segments of the training reveals that the Builder agent effectively learns to find a trajectory that satisfies the silhouettes sooner in its execution of one episode. It is only after 50 million steps in segment b, when the cumulative returns reach the value of +45, that the agent also fills over 90% of the *filled_filled* cells. Likewise in segment d where the performance is stabilized at +45, the agent is reaching the 90% threshold as soon as 250 steps into the episode.



Figure 4-3a. Quantitative and qualitative performance of segment a (0 ~ 6 million training steps) within the training of the Builder agent

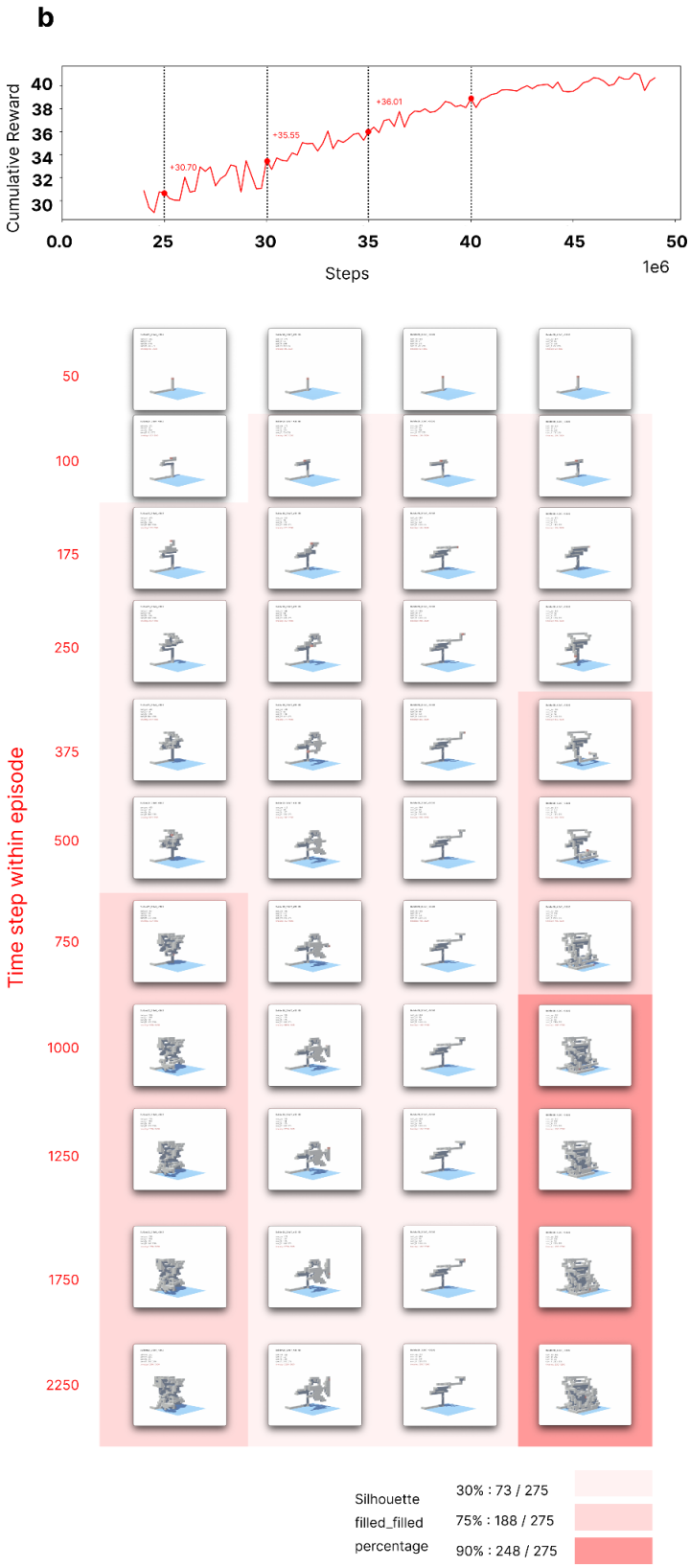


Figure 4-3b. Quantitative and qualitative performance of segment b (25 ~ 50 million training steps) within the training of the Builder agent

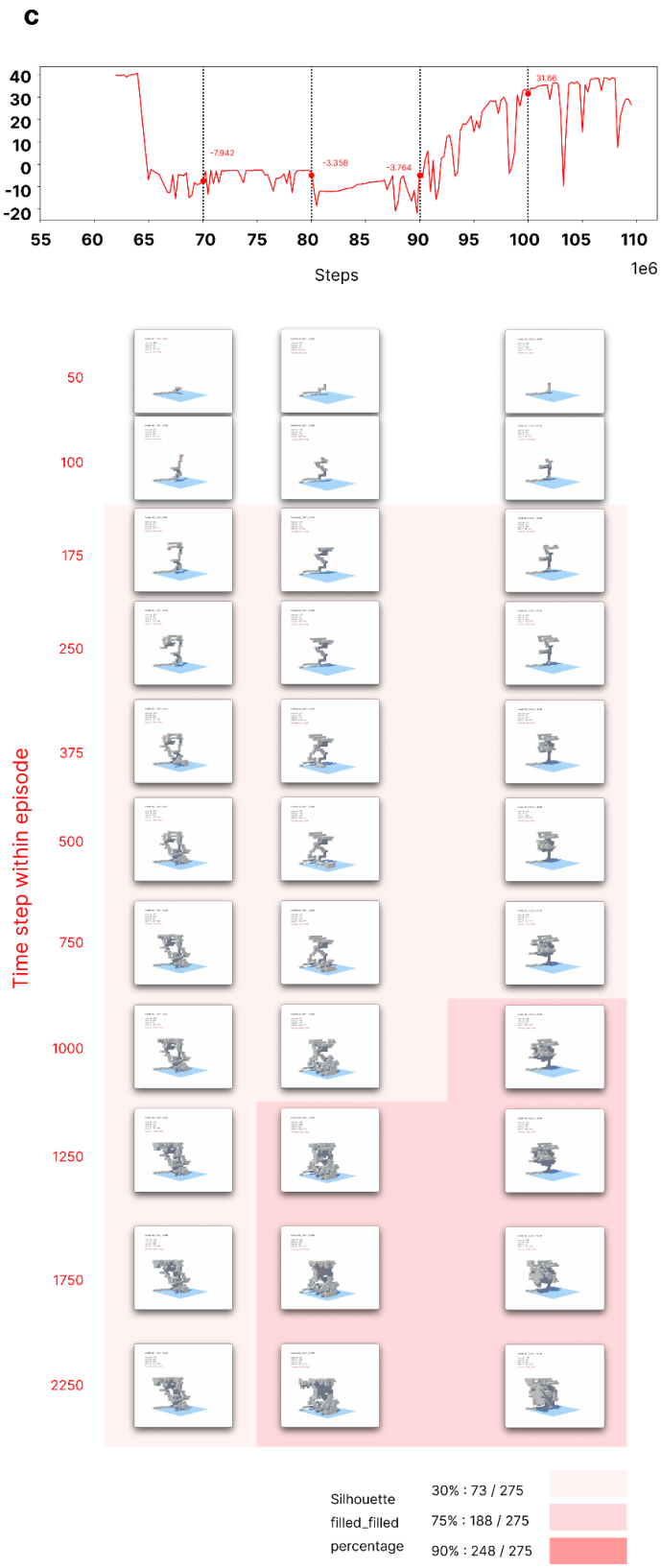


Figure 4-3c. Quantitative and qualitative performance of segment c (65 ~ 110 million training steps) within the training of the Builder agent

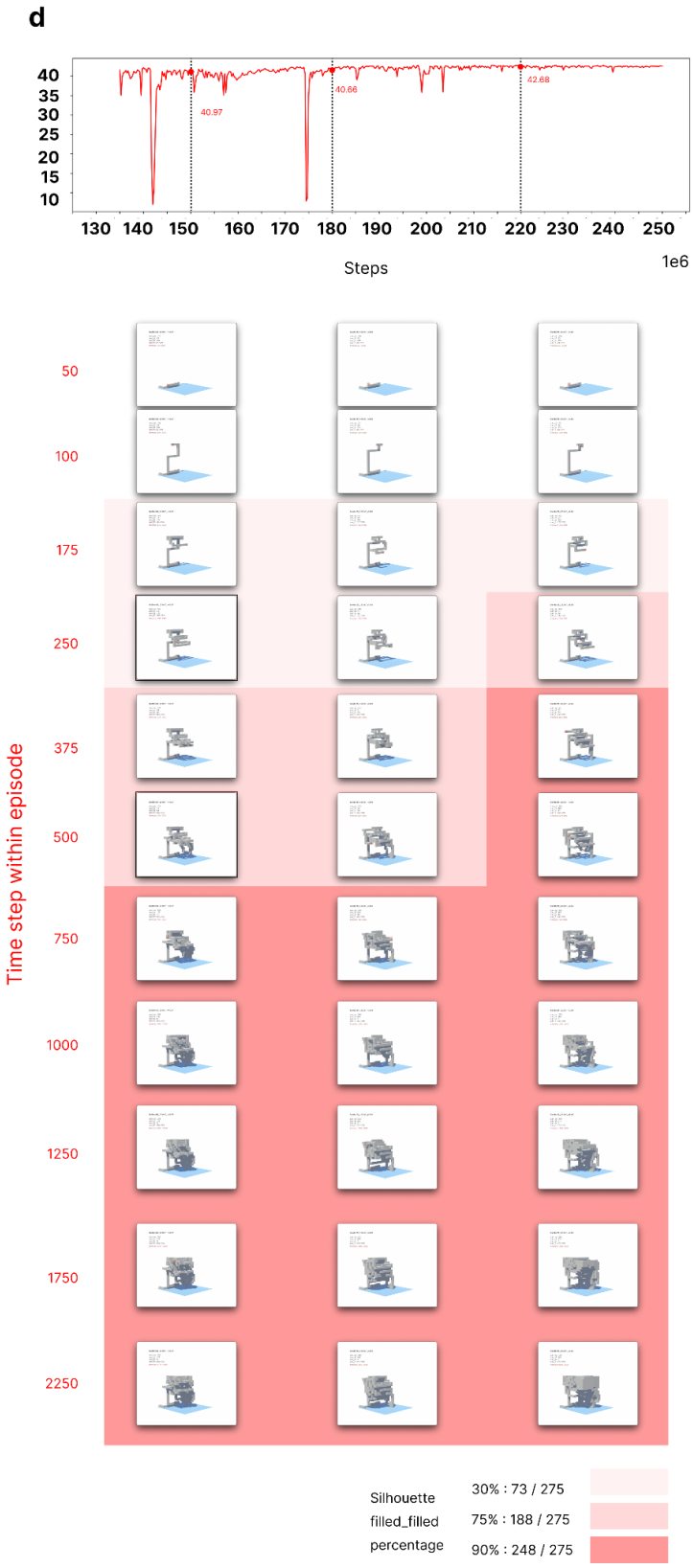


Figure 4-3d. Quantitative and qualitative performance of segment d (130 ~ 250 million training steps) within the training of the Builder agent

Looking into each segment and the change in agent behaviors shows how it is learning to perform the task and most importantly, where the designer can find forms for design exploration. The first segment a from the start of the training up to 5 million steps where the performance starts from a low negative value and rapidly increases to +20. One can see here that the agent starts off by trying different actions randomly, moving in all directions and building cubes. It soon learns to build consecutively by moving one over to an empty voxel cell and building a cube. The second segment b is between 25-50 million steps and is where the training performance steadily increases to reach around +45 which is revealed later to be the maximum reward value. Here one can see that the agent learns a certain trajectory of building the cubes, especially early on in the episode, but towards the end it explores different trajectories.

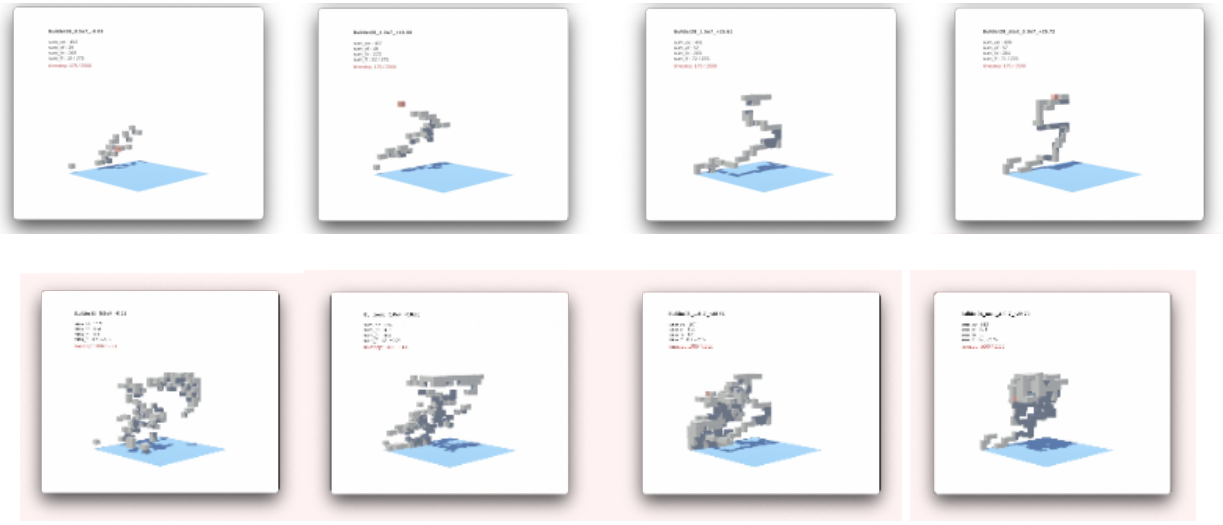


Figure 4-4. Snapshot of Builder agent from training segment a at time steps 175 (top) and 1000 (bottom).

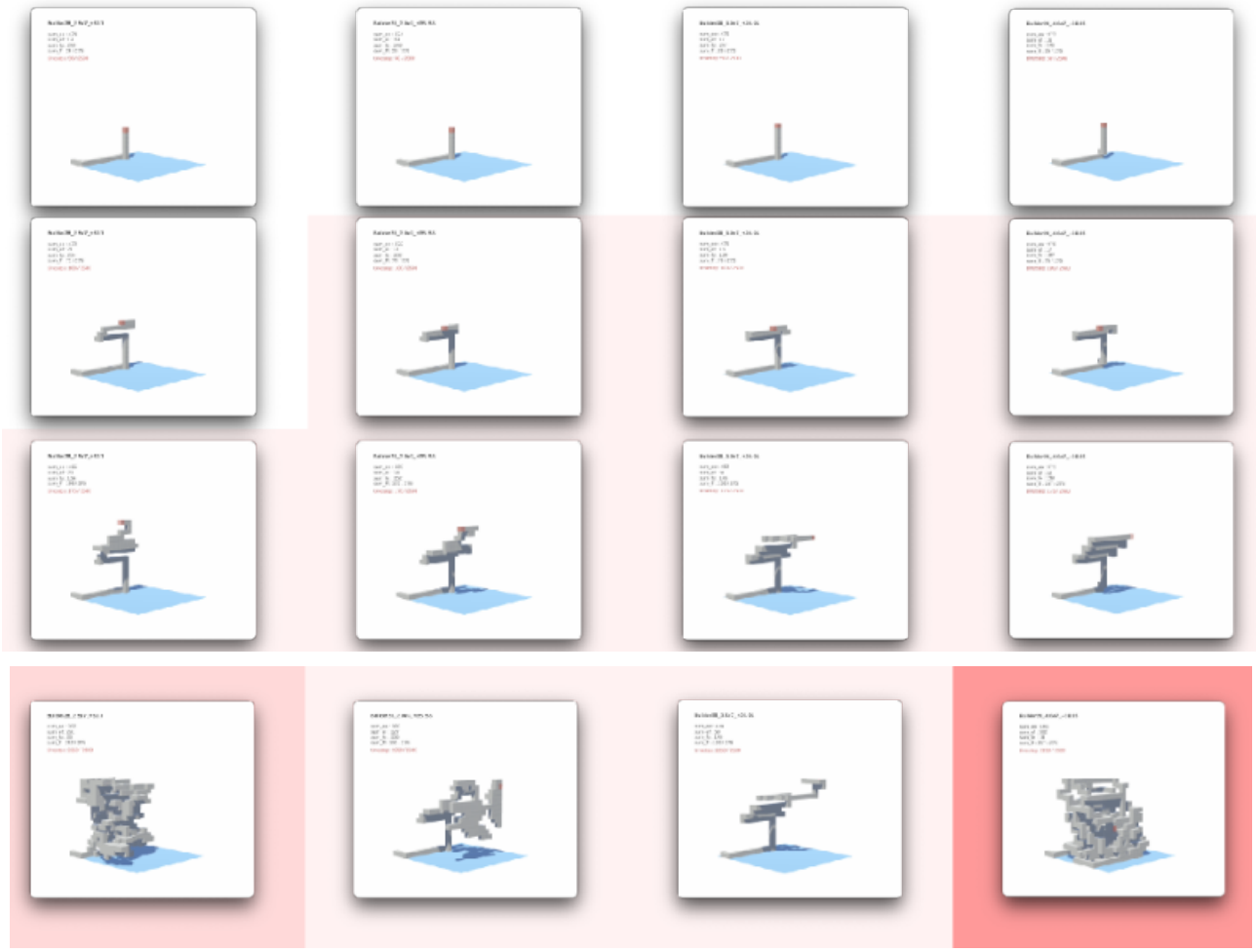


Figure 4-5. Snapshot of Builder agent from training segment b at time steps 50, 100, 175, 1000 (from top to bottom).

In segment c, the performance decreases to a negative value and fluctuates greatly. Here one can see that the agent is exploring vastly different trajectories that show little resemblance with the initial silhouettes. Finally in segment d, around 140 million steps, the training stabilizes at its maximum value around +45. In its execution, one can see that the agent has found a certain trajectory that it continues to initialize in the beginning of the episode. This is likely because it is certain that this trajectory is expected to yield the most rewards. Whereas towards the end of the episode, it is still in the process of learning the optimal path, or the trajectory that would yield the most rewards for the whole episodes. Also one can observe that the forms generated by the agent, while different at each point in training, closely resemble the initial silhouette of the chair that was provided as constraint for the task.

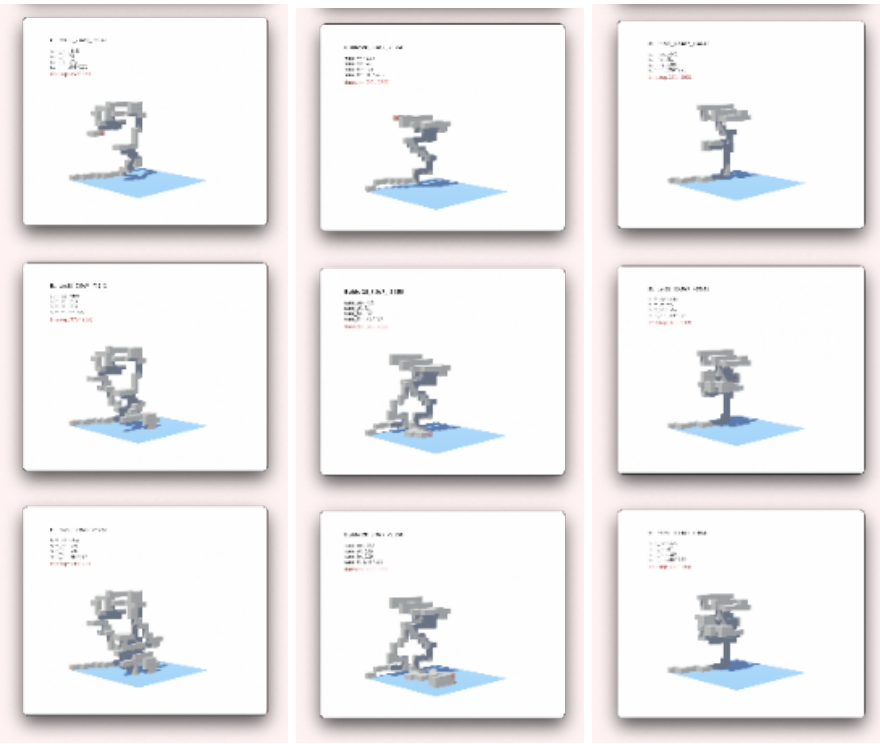


Figure 4-6. Snapshot of Builder agent from training segment c at time steps 375, 500 and 750 (from top to bottom).



Figure 4-7. Snapshot of Builder agent from training segment c at time steps 50, 100, 175 (from top to bottom). The agent is taking a similar trajectory at the beginning of

each episode.



Figure 4-8. Snapshot of Builder agent from training segment c at time steps 350, 500, 750 (from top to bottom). The forms that the agent creates closely resemble the initial silhouettes of the chair.

At this point, I was able to identify where in the training process, and where within the agent's execution forms for design exploration can be found. First, within the training the agent should reach a point where the quantitative performance plateaus at its optimal value because it is only after this point where it generates forms that satisfy the target silhouettes to show visible resemblance. Within the execution of the agent's policy, when it achieves 75% ~ 90% of the *filled_filled* cells the form the agent created most closely resembled a chair-like form that the target silhouette was intended to create. After the 90% threshold, the agent shows behavior that I denote as "overbuilding", where the is continuously building cubes to fill in the remaining cells that should be filled. When the agent is overbuilding, there is an increase in *empty_filled* cells, which are cells that should be empty according to the target silhouette but are filled by the agent. This results in a form that is less defined and loses its resemblance to the target silhouette. Identifying these points where forms for further design exploration could be found is critical to proceed to the next part of the study, as the input for the interactive training is a partial shape that the designer can further iterate on.

4.2 Evaluation of Interactive Framework

This section intends to evaluate the proposed interactive generative design framework through the qualitative results, more specifically the agent trajectory and the forms it creates after being retrained with the designer's feedback. The evaluation is based on the following qualitative criteria.

- ➔ Does the agent retrain while preserving the features that have initially been selected by the designer in the paused instance (snapshot)?
- ➔ Does the agent retrain to adapt to the designer's feedback?

Four instances of using the iterative framework are presented to validate that the interactive framework may work under specific circumstances. Instance 1 is set as a base line example that is considered to satisfy the evaluation criteria. In comparison with this baseline, instance 2,3,4 reveal variables that may affect the results of the interactive training process.

First, the interactive framework first requires a Builder agent that has been successfully trained to generate forms that can serve as a seed for further design exploration. From the evaluation of the Builder Agent, I was able to identify points within the training process, and within the Builder agent's execution of its policy where such forms are generated. For all four instances, I use the Builder agent at 150 million training steps. In *Inference mode*, the agent is initialized to run for one episode and at step 300, the execution is paused and a *snapshot* is taken. The assumption is that the *snapshot* was taken at the point where the form had features that met the design criteria, but the designer wanted to change the form created from that point. In these instances, the moment at which the snapshot is taken is based on my judgment that the form generated at this point has started to show resemblance to the chair silhouettes especially with its leg and backrest-like features. Switching to *Feedback mode*, I draw feedback that are leg-like features in the front of the chair. I select this shape for feedback because in the agent's original execution, the generated form seemingly only has two back legs. Given the additional feedback, would the agent generate a form that has all four legs of the chair? The feedback is set as input in the *Training mode*, where the training is initialized with the neural network from the 150 million training step of the original training and is trained for an additional 30 million steps.

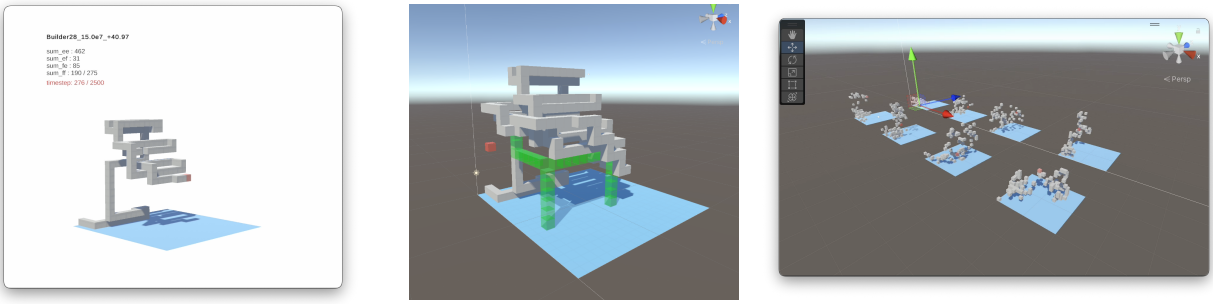


Figure 4-9. There are three modes in the interactive training process. In *Inference mode*, the user runs the agent and pauses to output a snapshot. (left) In *Feedback mode*, the user can add feedback blocks to the snapshot. (center) In *Training mode,* the agent's training is resumed with added feedback. (right)

Evaluation of the instances involves comparing the trajectory the Builder agent takes and the form it generates with its *original policy* to that with its *retrained policy* incorporating the feedback. The first evaluation criteria is that the form trained with feedback should preserve features from the original training. With instances 1,2 and 3 one can see that the trajectories up to 100 steps are similar to the original in their sequences of moving in the positive z direction, positive y direction, negative z direction, positive y direction, positive z direction, positive y direction and negative z direction to create a meandering shape along one yz-plane. The sequence of actions up to 375 are also similar to the original in that the agents meanders down the y axis while taking paths parallel to the yz-plane to create a form resembling the backrest of a chair which is a distinctive feature in snapshot taken with the original training (figure). Thus, one can say that instance 1,2,3 all preserve the features that have initially been selected by the designer in the paused instance.

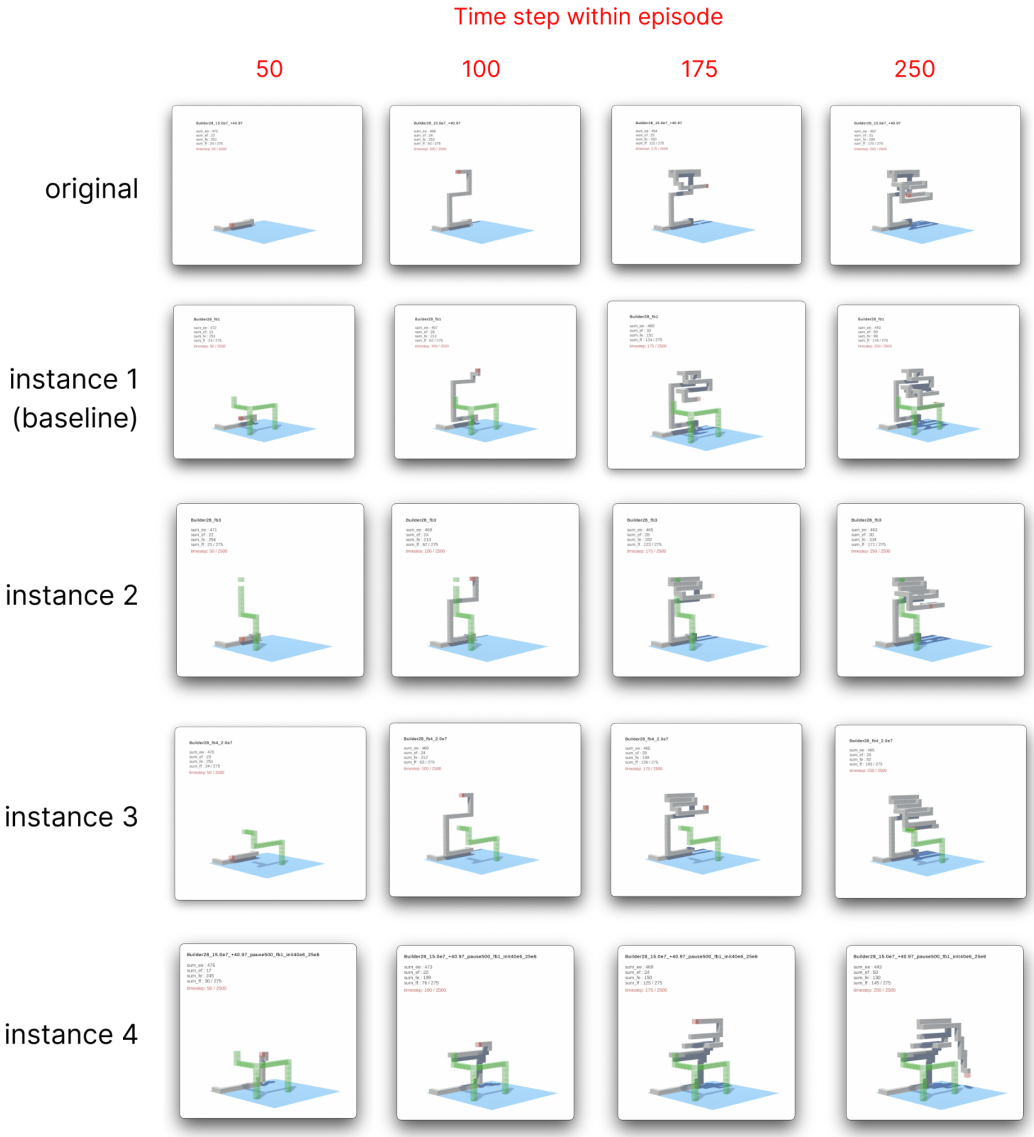


Figure 4-10a. Execution of one episode for agents from original training and 4 instances retrained with feedback (time step 50~250).

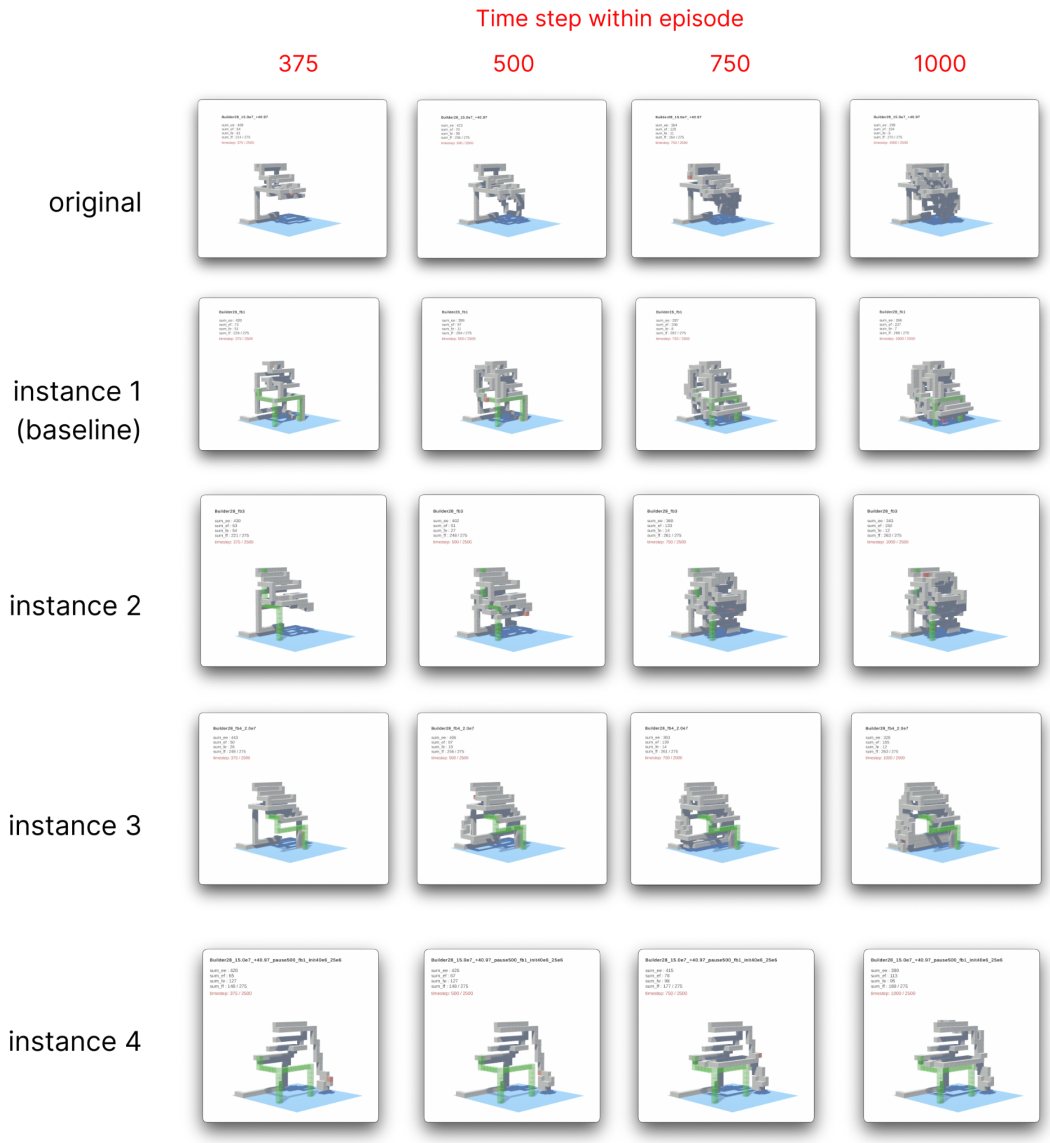


Figure 4-10b. Execution of one episode for agent from original training and 4 instances retrained with feedback (time step 375~1000

By the 500 step mark within the episode, one can observe the agent showing behaviors of changing its trajectory to meet the additional feedback. For example, the agent retrained in instance 1, has filled 17/31(55%) cells of the given feedback, which is the most out of all instances and the path the agent takes to fill the feedback aligns with direction used to draw the feedback by the author. On the other hand, in instance 2, while 11/23 (48%) of the feedback voxels have been filled, the agents paths which are loops parallel to the xz plane, do not align with the gestures used to draw the feedback which was vertical along the y axis. For instance 3,4 only 1 or none of the feedback voxels were filled, and even later on in the episode the agent's trajectory does not pass the feedback voxels. With these observations I make the conclusion that only in instance 1 has the agent retrained to adapt to the designer's feedback, satisfying the second criteria of the evaluation.

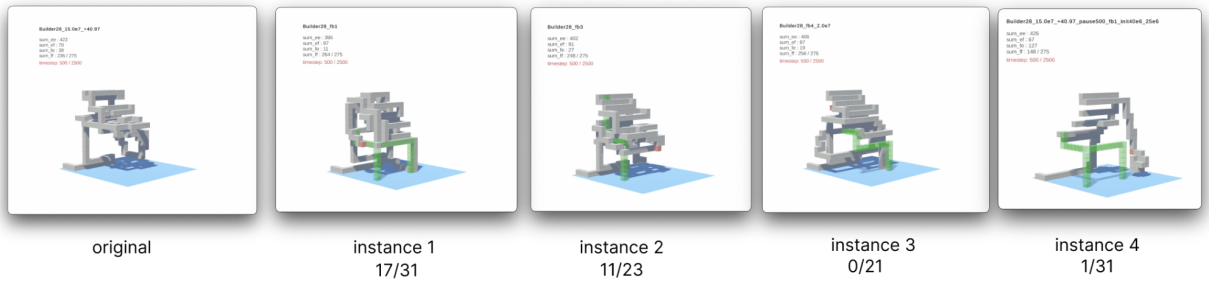


Figure 4-11. Comparison of each instance at step 500 / 2500 of the episode with the number of feedback voxel cells that the agent has built in.

As a result, instance 1 was the only instance to satisfy both criteria of the evaluation to validate the proposed interactive generative design framework. By comparing the settings of the four instances, I seek to identify what variable may affect the training process to make it successful or unsuccessful. From instance 2, one can make the speculation that the gestures in creating the feedback should be similar to the original agent's gestures in its trajectory. At instance 2, after 400 steps, the agent makes circular gestures along the xz axis that create horizontal lines whereas the feedback provided was in a vertical direction along the y axis. It might be that abrupt changes in directions may be difficult for the agent to learn within the short retraining time. For Instance 3 and 4, where the agent's both did not pass the feedback voxels, reveals that the retraining time of 30 million steps might have not been enough for the agent to pass the feedback in its trials. In other words, it only chose the trajectory that does not go through

the feedback because it never experienced the reward it could yield. This reveals that the time in retraining may have to be longer for the agent to adjust to any kind of feedback with high certainty. Lastly, instance 4 was the only instance that was initialized with a neural network at an earlier point in the original training (40 million steps in) as opposed to at the point when the snapshot was taken (150 million steps in). This may be the reason why instance 4 was the only instance that did not show resemblance to the original agent's trajectory in the beginning of the episode. This is to be expected as the policy, or the neural network that is used to initialize the retraining may have vastly different weights, and thus is likely to start to train from a different trajectory. This goes to confirm that the retraining should be initialized from the training point where the snapshot was taken.

In conclusion the experiments show that the proposed framework can produce results that can be used for interactive design exploration, but at the same time reveals the many variables that can affect the training, thus need to be fine tuned. For instance, at what point in the training should the agent be extracted and at what point within the episode of the agent's execution can the snapshot be taken? Another consideration is what forms of feedback require more or less time to train to reach a point where it can be seen that the agent has adjusted to the feedback?

5 Conclusion

5.1 Summary

This thesis presets a detailed account of formulating an interactive agent-based generative design system for the purpose of design exploration. First, I explore how to formulate and train a reinforcement learning agent to perform the task of building a 3D form given 2d silhouettes. I provide a feasible set of training environment, agent capabilities and reward function that can be used to train an RL agent to sequentially generate 3D form. Furthermore, with documentation of process in devising the environment and agent, I reveal major and minor conditions, which can serve as guideline for future works looking to develop reinforcement learning agents with similar tasks 3D form finding. Next, I present an interactive framework where the designer is able to react to the generated design artifact, and the generative agent adjusts its behavior accordingly. With experiments using varying inputs, I reveal the framework's capabilities and limitations. The thesis argues that the proposed interactive agent-based generative design framework is effective in allowing a iterative and evaluative design process while leveraging generative capabilities of learning-based techniques and as a result it has potential to facilitate the concurrent and gradual articulation of design criteria and design artifact that is critical in the early stages of design.

5.2 Limitations and Future works

Here I identify some limitations of this thesis, especially in the implementation of the proof-of-concept that provide insight to future works. I acknowledge that the proof-of-concept in this thesis is limited to one example of generating a chair-like form within a 16x16x16 resolution with gestural feedback. Further studies can be done to compare different forms, resolutions and modalities of feedback, which can reveal the framework's capabilities and additional considerations. There were challenges in the implementation of this work that came from the innate difficulty of the task. The agent is learning a consecutive trajectory that would satisfy all silhouettes, which is uncertain to even be feasible and accompanies a large action space. As a result, there were limitations to the quantitative performance of the agent within the training time in this study. While it was not critical for the agent to optimize rewards for the objective of this study, especially when the interactive framework uses "mid-training" results, improving the training may reveal useful information and more importantly, reduce time of training. It is worth

noting that the studies in this thesis did incorporate modifications to the basic PPO algorithm by transfer learning, adding recurrent neural networks and curiosity, yet it has been noted that for complex operations and huge actions spaces, off-the-shelf RL frameworks may not be capable to learn good policies (Lin et al., 2020, p. 2). There is potential for future work to point to novel RL algorithms that target to perform better on tasks of this nature.

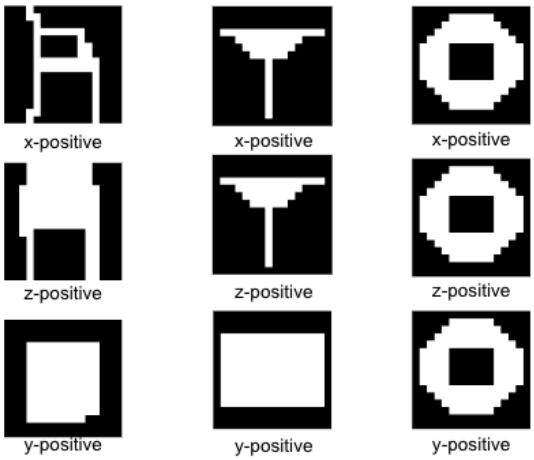


Figure 5-1. Potential silhouettes to explore the capabilities of the agent.

One shortcoming of the proof-of-concept is that the time to train the Builder agent is not ideal, considering it is incorporated in an interactive system. The Builder agent trained from scratch only showed satisfactory performance after 150 million training steps, which required approximately 100 hours of training on the authors device (Apple silicon chip that was not accelerated with GPU) and the retraining was done with 30 million additional steps which required close to 20 hours. Aside from accelerating the training time with different RL algorithms and hyperparameters, there are ways to reformulate the framework. One possibility is to make the task easier by dividing the expected 3D form into small parts. For instance the 16x16x16 grid can be divided into 4 8x8x8 parts or a chair-like form can be divided into semantic parts - the seat, back rest, arms and legs. These smaller parts would be trained separately, and are joined by the designer later. This would be reducing the training time in exchange for incorporating heuristics or user's judgment . Yet this can be effective not only in alleviating the task difficulty for the RL agent, but for designer's to have more control over design outcome. Another possibility is to train a generalizable Builder agent up front, which can generate 3D forms given any kind of silhouette. The training of this agent will require an immensely long time, as different silhouettes are given as different environments at each episode

for the agent to adjust to. However, assuming that this agent is successfully trained, it can then be incorporated into the interactive generative workflow, leaving the designer or user to only do the retraining which is relatively short.

Due to challenges in reducing the training time, the proof-of-concept was limited to one interaction. Future works can employ multiple interactions to observe how the generated form can change with evolving feedback. In the following figure I proposed a temporal decay function to integrate the gradual feedback in the reward function within the interactive training framework proposed in this thesis.

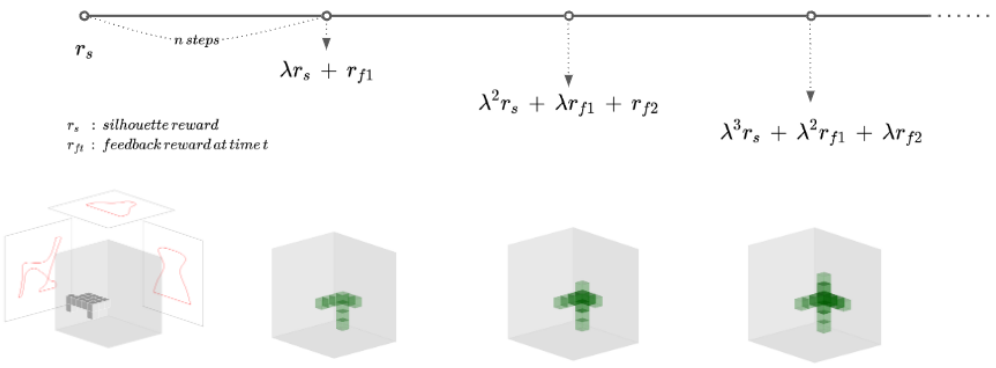


Figure 5-2. Scheme for incorporating evolving feedback in temporal decaying reward function.

Lastly, the selection and evaluation of results in the proposed framework rely in part on subjective criteria. This is inevitable to an extent because the framework is being used for design exploration for the user's custom purpose. Yet, there is the possibility to incorporate quantitative metrics. For instance, in the evaluation of the interactive framework, when comparing forms generated by the original training and retrained training in the interactive framework, a metric of similarity and dissimilarities of forms could be used instead of relying solely on the author's judgment.

5.3 Contributions and Discussion

This thesis ultimately aims to open the discussion to the possible future directions of generative design, design tools and professed automation in design. In its efforts to do so, it offers a number of technical contributions to both fields of computational design and learning-based methods which are addressed in this section.

How can designers have agency in formulating learning-based design tools?

This thesis introduces new ways designer or user can be involved in creating, determining the capabilities and functionalities of the computational medium, especially those using learning-based methods. In the current state of applied machine learning, the domain experts' or the end-users' engagement in the toolmaking process is limited to reflecting on the result, whereas the machine learning engineer, who might not have a strong grasp of the specific practice would curate data sets, develop algorithms, train and fine-tune models, evaluate the models' performance and deploy solutions (Amershi, Cakmak, Knox, & Kulesza, 2014, p. 105 cited by Bidgoli) The proof-of-concept in this thesis provides a frame for designers to indirectly fine-tune the generator of the system by providing gestural, intuitive feedback that fits their specific needs and intents. The designer, by providing partial examples alters the micro-scale rules imposed to the generative agent which govern its behaviors, leading to emergent results.

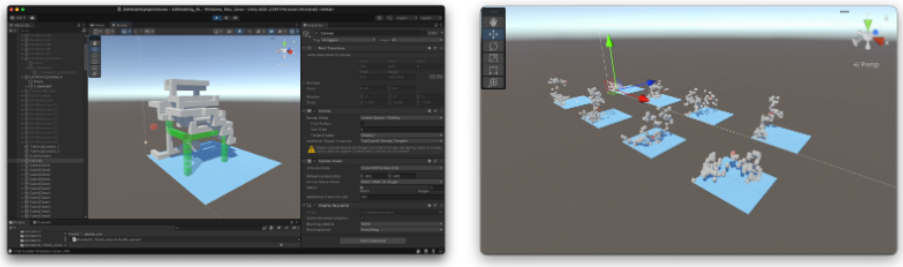


Figure 5-3. Retraining of the RL agent based on intermittent feedback provided by the designer.

Furthermore, this thesis intends to shed light on the "mutually constitutive" (Lave 1991) relationship between learning-based methods and designer as toolmakers. The common discourse surrounding learning-based methods is for computational algorithms or agents to have human-like intelligent capabilities that would serve the automation of tasks. However, I argue that learning-based tools should be considered with respect to the context and user rather than stand-alone and autonomous systems, especially as design processes that are nuance-rich and idiosyncratic. As a start, it is essential to identify synergetic roles for computer and human and how they can be compatible. In this study, I set the RL agent to delegate the creation of cohesive 3D forms accompanied by constraint satisfaction while the designer takes the role of holistic evaluation with implicit constraints.

How can computational design tools adapt to domain, context, problem specific design processes?

The limitations of current computational design tools, particularly CAD software, is that they are closed systems which once deployed their affordances cannot be altered. From their creation, the usage of these tools are anticipated and predetermined in their structure. However how these softwares are used - for what designs that are used for, what information is incorporated and how results are represented vary with domain, context, and problem, and are subject to change, ever more frequently today. As a result, there has been inefficiencies such as the need for softwares to be reengineered and individuals who are willing to go through the learning curve of using the API's of these softwares making proprietary plugins for their specific needs. One motivation for this thesis is to explore ways in which computational design tools can be customized with user and problem-specific feedback, as well as dynamically adapt to changes in design requirements. For instance, the proof-of-concept's use of voxel representation allows the embedding of gestural feedback that is unique to the user. This highlights future opportunities to incorporate multimodal information in the customized development of generative design systems that vary from explicit, analytical information to implicit, holistic information.

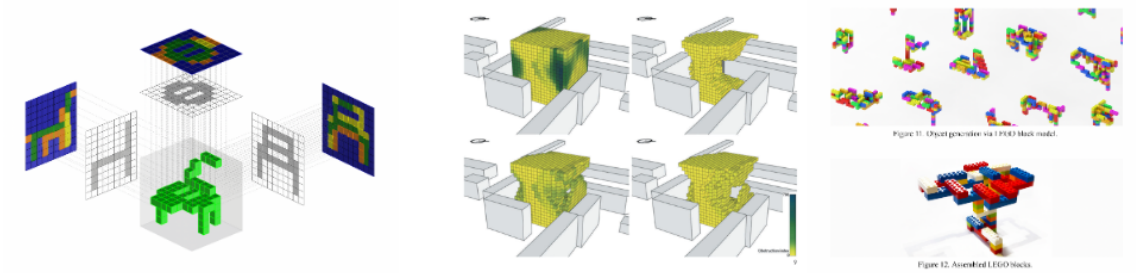


Figure 5-4. Information and constraints that can be expressed with voxel representation : form (left), environment analysis (center), structure (right).

How can learning-based methods be used for design tasks with ambiguous goals and changing criteria?

Lastly, the findings in this work reveal useful information for implementing learning-based methods for design tasks. It introduces 3D design synthesis as an RL task and expands RL's capabilities to be used for tasks with open-ended goals and changes in criteria which are common in design. One of the main contributions of this work is in its formulation of 3D design synthesis into Markov Decision Process where the agent makes decisions to

sequentially create forms within the voxel grid environment. From the few works that have employed reinforcement learning for 3D design synthesis, very few have provided detailed documentation of their implementation. The experiments within this thesis serves as precedent for future research using RL for 3D form finding, revealing relevant hyperparameters both in the formation and training of a generative RL agent. Also it demonstrates how intermediate results during the training can be valuable for the interactive generative design system for design exploration. Finally this work demonstrates the capabilities of agent-based generative design systems that have been less explored compared to other learning-based methods.

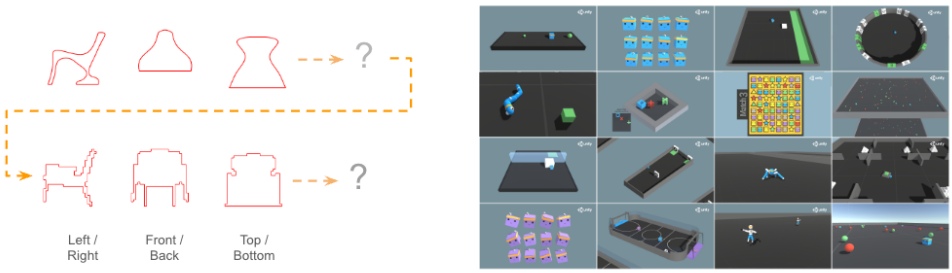


Figure 5-5. Example of design task with ambiguous goal and evolving constraints presented in this thesis (left). Suite of RL tasks provided by Unity-ML (right).

REFERENCES

Agrawal, Akash, and Christopher McComb. 2023. “Reinforcement Learning for Efficient Design Space Exploration With Variable Fidelity Analysis Models.” *Journal of Computing and Information Science in Engineering* 23 (4). https://doi.org/10.1115/1.4056297.

Akin, Bharat Dave, Shakunthala Pithavadian, and Carnegie Mellon University Engineering Design Research Center. 1987. “Problem Structuring in Architectural Design,” January. https://doi.org/10.1184/R1/6076109.v1.

“An Introduction to Reinforcement Learning with OpenAI Gym, RLlib, and Google Colab | Anyscale.” n.d. Accessed April 26, 2023. https://www.anyscale.com/blog/an-introduction-to-reinforcement-learning-with-openai-gym-rllib-and-google.

“Backpropagation.” 2023. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1150946223.

Barkstrom, B. R. 2004. “The Standard Waterfall Model for Systems Development.” *Atmospheric Sciences Data Center*. http://asd-www. larc. nasa. gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development. htm.

Bidgoli, Ardavan, and Pedro Veloso. 2019. “DeepCloud. The Application of a Data-Driven, Generative Model in Design.” arXiv. https://doi.org/10.48550/arXiv.1904.01083.

Cagan, Jonathan, William J. Mitchell, and Carnegie Mellon University Engineering Design Research Center. 1991. “Shape Annealing : A New Approach for Controlling Shape Generation,” January. https://doi.org/10.1184/R1/6490151.v1.

Chaillou, Stanislas. 2020. “Space Layouts & GANs.” *Spacemaker Research* (blog). September 15, 2020. https://medium.com/spacemaker-research-blog/space-layouts-gans-2329c8f85fe8.

Chang, Angel X., Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, et al. 2015. “ShapeNet: An Information-Rich 3D Model Repository.” arXiv. https://doi.org/10.48550/arXiv.1512.03012.

Cheliotis, Kostas. 2021. “ABMU: An Agent-Based Modelling Framework for Unity3D.” *SoftwareX* 15 (July): 100771. https://doi.org/10.1016/j.softx.2021.100771.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. “Decision Transformer: Reinforcement Learning via Sequence Modeling.” In *Advances in Neural Information Processing Systems*, 34:15084–97. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html.

Chen, Wei. 2019. “Data-Driven Geometric Design Space Exploration and Design Synthesis.” https://doi.org/10.13016/qwkq-qmtw.

Clancey, William J. 1997. *Situated Cognition: On Human Knowledge and Computer Representations*. Situated Cognition: On Human Knowledge and Computer Representations. New York, NY, US: Cambridge University Press.

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

Cross, Nigel. 2006. *Designerly Ways of Knowing*. London: Springer-Verlag. https://www.springer.com/gb/book/9781846283000.

Davis, D. 2013. "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." (Doctoral dissertation, RMIT)., RMIT, Melbourne. Retrieved from https://researchrepository. rmit.edu.au/.

De Wolf, Tom, and Tom Holvoet. 2005. "Emergence Versus Self-Organisation: Different Concepts but Promising When Combined." In *Engineering Self-Organising Systems*, edited by Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, 1–15. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/11494676_1.

"Deep Reinforcement Learning." 2023. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Deep_reinforcement_learning&oldid=1140528695#Deep_reinforcement_learning.

"DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning: ACM Transactions on Graphics: Vol 36, No 4." n.d. Accessed April 25, 2023. https://dl-acm-org.cmu.idm.oclc.org/doi/abs/10.1145/3072959.3073602?casa_token=SRZkc3Rpf0oAAAAA:nq2yDrjtkodKLaVL6K_GVtU50gej41SIvMZ-H21fC1JfdvHi42wC4Wjr3NcWn1s2rmSlex0ggEkd.

"Define Reward Signals - MATLAB & Simulink - MathWorks España." n.d. Accessed May 8, 2023. https://es.mathworks.com/help/reinforcement-learning/ug/define-reward-signals.html.

Dworschak, Fabian, Sebastian Dietze, Maximilian Wittmann, Benjamin Schleich, and Sandro Wartzack. 2022. "Reinforcement Learning for Engineering Design Automation." *Advanced Engineering Informatics* 52 (April): 101612. https://doi.org/10.1016/j.aei.2022.101612.

Gänshirt, Christian. 2012. *Tools for Ideas: Introduction to Architectural Design*. Walter de Gruyter.

Green, Josh. 2021. "Design Space: An Overview." Citrine Informatics. June 25, 2021. https://citrine.io/what-is-a-design-space/.

Groenewolt, Abel, Tobias Schwinn, Long Nguyen, and Achim Menges. 2018. "An Interactive Agent-Based Framework for Materialization-Informed Architectural Design." *Swarm Intelligence* 12 (2): 155–86. https://doi.org/10.1007/s11721-017-0151-8.

Habowski, Tyler. 2020. "AI for 3D Generative Design." Medium. March 20, 2020. https://blog.insightdatascience.com/ai-for-3d-generative-design-17503d0b3943.

Harding, John E., and Paul Shepherd. 2017. “Meta-Parametric Design.” *Design Studies* 52 (September): 73–95. https://doi.org/10.1016/j.destud.2016.09.005.

Holland, John H. 1999. *Emergence: From Chaos To Order*. Basic Books.

Holzer, Dominik, Richard Hough, and Mark Burry. 2007. “Parametric Design and Structural Optimisation for Early Design Exploration.” *International Journal of Architectural Computing* 5 (4): 625–43. https://doi.org/10.1260/147807707783600780.

Israilov, Sardor, Li Fu, Jesús Sánchez-Rodríguez, Franco Fusco, Guillaume Allibert, Christophe Raufaste, and Médéric Argentina. 2023. “Reinforcement Learning Approach to Control an Inverted Pendulum: A General Framework for Educational Purposes.” *PLOS ONE* 18 (2): e0280071. https://doi.org/10.1371/journal.pone.0280071.

Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, et al. 2020. “Unity: A General Platform for Intelligent Agents.” arXiv. https://doi.org/10.48550/arXiv.1809.02627.

Khan, Shahroz, and Muhammad Junaid Awan. 2018. “A Generative Design Technique for Exploring Shape Variations.” *Advanced Engineering Informatics* 38 (October): 712–24. https://doi.org/10.1016/j.aei.2018.10.005.

Koh, Immanuel. 2017. “Decoding Parametric Design Data - Towards a Heterogeneous Design Search Space Remix.” In *P. Janssen, P. Loh, A. Raonic, M. A. Schnabel (Eds.), Protocols, Flows, and Glitches - Proceedings of the 22nd CAADRIA Conference, Xi’an Jiaotong-Liverpool University, Suzhou, China, 5-8 April 2017, Pp. 117-126*. CUMINCAD. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?caadria2017_086.

Lave, Jean. 1991. *Situated Learning: Legitimate Peripheral Participation*. Learning in Doing. Cambridge [England] ; Cambridge University Press.

Lawson, Bryan. 2005. *How Designers Think: Demystifying the Design Process*. Jordan Hill, UNITED STATES: Taylor & Francis Group. http://ebookcentral.proquest.com/lib/cm/detail.action?docID=269907.

Lin, Cheng, Tingxiang Fan, Wenping Wang, and Matthias Nießner. 2020. “Modeling 3D Shapes by Reinforcement Learning.” arXiv. https://doi.org/10.48550/arXiv.2003.12397.

Liu, Hong, Liping Gao, and Xiyu Liu. 2005. “Generative Design in an Agent Based Collaborative Design System.” In *Computer Supported Cooperative Work in Design I*, edited by Weiming Shen, Zongkai Lin, Jean-Paul A. Barthès, and Tangqiu Li, 105–16. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/11568421_11.

Ma, Y.-S., G. Chen, and G. Thimm. 2008. “Paradigm Shift: Unified and Associative Feature-Based Concurrent and Collaborative Engineering.” *Journal of Intelligent Manufacturing* 19 (6): 625–41. https://doi.org/10.1007/s10845-008-0128-y.

Markus, Thomas A. 1969. “The Role of Building Performance Measurement and Appraisal in Design Method.” *Design Methods in Architecture*, 109–17.

Matejka, Justin, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. “Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets.” In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. CHI ’18. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3173574.3173943.

Maver, T. W. 1970. “A Theory of Architectural Design in Which the Role of the Computer Is Identified.” *Building Science* 4 (4): 199–207. https://doi.org/10.1016/0007-3628(70)90022-8.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. “Playing Atari with Deep Reinforcement Learning.” arXiv. https://doi.org/10.48550/arXiv.1312.5602.

Narahara, Taro. 2017. “Collective Construction Modeling and Machine Learning: Potential for Architectural Design.” In *Shock - Sharing of Computable Knowledge*, 593–600. Education and research in Computer Aided Architectural Design in Europe. https://researchwith.njit.edu/en/publications/collective-construction-modeling-and-machine-learning-potential-f.

“Onformative — AI Sculpting.” n.d. Accessed May 7, 2023. https://onformative.com/work/ai-sculpting.

Ororbia, Maximilian E., and Gordon P. Warn. 2021. “Design Synthesis Through a Markov Decision Process and Reinforcement Learning Framework.” *Journal of Computing and Information Science in Engineering* 22 (2). https://doi.org/10.1115/1.4051598.

Regenwetter, Lyle, Amin Heyrani Nobari, and Faez Ahmed. 2022. “Deep Generative Models in Engineering Design: A Review.” *Journal of Mechanical Design* 144 (7). https://doi.org/10.1115/1.4053859.

“Reinforcement Learning.” 2023. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1153070423.

Russell, Stuart, and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*. 3rd edition. Upper Saddle River: Pearson.

Schon, Donald A., and Glenn Wiggins. 1992. “Kinds of Seeing and Their Functions in Designing.” *Design Studies* 13 (2): 135–56. https://doi.org/10.1016/0142-694X(92)90268-F.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. “Proximal Policy Optimization Algorithms.” arXiv. https://doi.org/10.48550/arXiv.1707.06347.

Schulz, Adriana, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. 2018. “Interactive Exploration of Design Trade-Offs.” *ACM Transactions on Graphics* 37 (4): 131:1-131:14. https://doi.org/10.1145/3197517.3201385.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, et al. 2018. “A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play.” *Science (New York, N.Y.)* 362 (6419):

1140–44. https://doi.org/10.1126/science.aar6404.

Simon, Herbert A. 1973. “The Structure of Ill Structured Problems.” *Artificial Intelligence* 4 (3–4): 181–201. https://doi.org/10.1016/0004-3702(73)90011-8.

———. 2019. *The Sciences of the Artificial, Reissue of the Third Edition with a New Introduction by John Laird*. MIT Press.

Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. Second edition. Cambridge, Mass: A Bradford Book.

Toulkeridou, Varvara. 2019. “Steps towards AI Augmented Parametric Modeling Systems for Supporting Design Exploration.” In , 3. https://doi.org/10.5151/proceedings-ecaadesigradi2019_602.

Turrin, Michela, Peter von Buelow, and Rudi Stouffs. 2011. “Design Explorations of Performance Driven Geometry in Architectural Design Using Parametric Modeling and Genetic Algorithms.” *Advanced Engineering Informatics*, Special Section: Advances and Challenges in Computing in Civil and Building Engineering, 25 (4): 656–75. https://doi.org/10.1016/j.aei.2011.07.009.

Veloso, Pedro, and Ramesh Krishnamurti. 2021a. “Mapping Generative Models for Architectural Design.” In *The Routledge Companion to Artificial Intelligence in Architecture*. Routledge.

———. 2021b. “Self-Learning Agents for Spatial Synthesis.” In *Formal Methods in Architecture*, 265–76. Springer.

Vinyals, Oriol, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, et al. 2019. “Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning.” *Nature* 575 (7782): 350–54. https://doi.org/10.1038/s41586-019-1724-z.

Xu, Kai, Vladimir G. Kim, Qixing Huang, and Evangelos Kalogerakis. 2015. “Data-Driven Shape Analysis and Processing.” arXiv. https://doi.org/10.48550/arXiv.1502.06686.

Yumer, Mehmet, Siddhartha Chaudhuri, Jessica Hodgins, and Levent Kara. 2015. “Semantic Shape Editing Using Deformation Handles.” *ACM Transactions on Graphics* 34 (4): 1–12. https://doi.org/10.1145/2766908.

LIST OF FIGURES AND TABLES

Figure 2-1. Artist sculpting an animal body as an example of the iterative and evaluative process of design[10]

Figure 2-2. Generalized map of the feedback loop within the design process borrowing from (Markus 1969; Maver 1970) taken from (Lawson 2005, 40). *Analysis is the ordering and structuring of the problem. Synthesis is .. the generation of solutions. [Evaluation] (Appraisal) involves the critical evaluation of suggested solutions against the objectives identified in the analysis phase.*

Figure 2-3. Illustration of algorithmically searching a continuous design space from Green 2021.

Figure 2-4. The design space is used to visualize and find design solutions. Illustration from (Schulz et al. 2018) where the mapping from manifolds in performance space back to design space allows designers to explore performance trade-offs interactively while visualizing the corresponding geometry and gaining an understanding of a model's underlying properties.

Figure 2-5. A parametric representation of a wine glass model by (Khan and Awan 2018). Design alternatives of the glass model are generated by a space-filling-generative design technique which generates design alternatives that cover the entire design space to give a designer a better understanding of possible design options

Figure 2-6. (Habowski 2020) presents an example of a data-driven generative design system. The TSNE[11] map of the latent space vectors is colored according to the object category.

Figure 2-7. Example of a custom-made dataset for data-driven floor plan generation from (Chaillou 2020). The left side encodes the footprint of the apartment unit in black and the position of the facade openings in red. On the right, the pair displays the corresponding room layout with their programs encoded with the color scheme below.

Figure 2-8. Examples of interactive generative design frameworks. (Yumer et al. 2015) maps semantic attributes to geometry for smooth shape editing using deformation handles (left). (Bidgoli and Veloso 2019) employs Leap Motion for users to manipulate the latent space representation of a point cloud design space (right).

Figure 2-9. Veloso and Krishnamurti 2021b employ multiagent deep reinforcement for spatial synthesis. Simulation of six agents that are trained to satisfy shape, area individually, and adjacency collectively.

Figure 2-10. Diagram explaining builder's actions : step, flock, attach, and build (top) and 3D urban block results (bottom) from Narahara (2017)

[10]screen shots from *(Erik Durant 2021)* taken by author with annotations added by author

[11] t-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map.

Figure 2-11. Exploration behavior of RL agents is visualized using PCA embeddings from Agrawal and McComb (2023).

Figure 2-12. The step-by-step procedure of 3D shape modeling from Lin et al. 2020. The first row of each figure shows how the target shape is approximated with primitive shapes. The second row shows the process of mesh editing given the primitives.

Figure 2-12. "Onformative — AI Sculpting" n.d. Presents an RL agent for 3D sculpting. *In the voxel-based environment infinite data and a clear reward structure was provided for the agent to move through. The starting state of the environment was one big cube. Out of this, the agent needed to remove mass to get closer to a predefined target state. With each step, the agent could decide where to go and if to remove a mass of voxels around itself and how.*

Figure 3-1. An abstraction of the interactions of the agent with the environment for reinforcement learning.[12]

Figure 3-2. Screenshot of the voxel grid environment created within the Unity Game Engine. The environment is duplicated 8 times to accelerate the training process, as Unity ML-agent allows the training of multiple concurrent environment instances. Created by author.

Figure 3-3. Screenshot of the terminal connecting the Python Trainer module to the Unity environment(left) and logging training results(right). Training the agent is possible through the Low-Level Python API provided by Unity ML-agent that allows communication between the Python Trainer modules and the Unity environment. Created by author.

Figure 3-4. The 3D Form Finding Task is defined as finding a sequence of building actions to create a voxel 3D form given silhouettes from three sides. This example created by the author[13] illustrates the input and expected output.

Figure 3-5. Original chair model from ShapeNet. Screenshot taken from top, front, right direction by author.

Figure 3-5. 16×16 size chair-like target silhouettes used in the experiments of the proof-of-concept.

Figure 3-6. Comparison of expressibility of different resolutions in target silhouettes.

Table 3-1. The maximum environment length is estimated as an indicator of training complexity for different voxel grid resolutions.

Figure 3-7. Move or Create action set mapping organized in action branch and index format used in Unity ML-agent.

[12] Image from ("An Introduction to Reinforcement Learning with OpenAI Gym, RLlib, and Google Colab | Anyscale" n.d.)
[13] The example was created by taking an example chair 3D mesh model from ShapeNet, voxelizing it to a 32*32*32 resolution, and tracing the silhouettes from the three sides left, front and top.

Figure 3-8. XYZ Create action set mapping organized in action branch and index format used in Unity ML-agent.

Figure 3-9. Concurrent Create action set mapping organized in action branch and index format used in Unity ML-agent.

Table 3-2. Sensor map indices assigned by current state of silhouette cell in relation to target silhouette cell at the same location.

Figure 3-10. Illustration of sensor map where the current silhouette of the form is indexed in relation to the target silhouette.

Figure 3-11. Diagram of interactive training framework where adding designer feedback affects reward function and eventually the policy during the training process.

Figure 3-12. Interactive framework from designer/user perspective.

Figure 3-13. Interface for inference mode. The number of timesteps and sensor map information is stated on the top left (left). The snapshot on the left is recorded as a csv file (right).

Figure 3-14. interface within unity game engine for feedback mode. The gray cubes are the composition from the snapshot and the green are feedback cubes.

Figure 3-15. Interface at training mode. The environment is duplicated to collect trajectories simultaneously and expedite the training.

Figure 4-1. Quantitative performance of Builder that shows cumulative rewards agent over 260 million training steps.

Figure 4-2. Quantitative performance of Builder agent labeled with intervals of interest.

Figure 4-3a. Quantitative and qualitative performance of segment a (0 ~ 6 million training steps) within the training of the Builder agent

Figure 4-3b. Quantitative and qualitative performance of segment b (25 ~ 50 million training steps) within the training of the Builder agent

Figure 4-3c. Quantitative and qualitative performance of segment c (65 ~ 110 million training steps)within the training of the Builder agent

Figure 4-3d. Quantitative and qualitative performance of segment d (130 ~ 250 million training steps)within the training of the Builder agent

Figure 4-4. Snapshot of Builder agent from training segment at time step 175 (top) and 1000 (bottom)

Figure 4-5. Snapshot of Builder agent from training segment b at time steps 50, 100, 175, 1000 from top to bottom.

Figure 4-6. Snapshot of Builder agent from training segment c at time steps 375, 500, and 750 from top to bottom.

Figure 4-7. Snapshot of Builder agent from training segment c at time steps 50, 100, 175, from top to bottom. The agent is taking a similar trajectory at the beginning of each episode.

Figure 4-8. Snapshot of Builder agent from training segment c at time step 350, 500, 750 from top to bottom. The forms that the agent creates closely resemble the initial silhouettes of the chair.

Figure 4-9. There are three modes in the interactive training process. In *Inference mode*, the user runs the agent and pauses to output a snapshot. (left) In *Feedback mode*, the user can add feedback blocks to the snapshot. (center) In *Training mode,* the agent's training is resumed with added feedback. (right)

Figure 4-10a. Execution of one episode for agents from original training and 4 instances retrained with feedback (time step 50~250).

Figure 4-10b. Execution of one episode for agent from original training and 4 instances retrained with feedback (time step 375~1000).

Figure 4-11. Comparison of each instance at step 500 / 2500 of the episode with the number of feedback voxel cells that the agent has built in.

Figure 5-1. Potential silhouettes to explore the capabilities of the agent.

Figure 5-2. Scheme for incorporating evolving feedback in temporal decaying reward function.

Figure 5-3. Retraining of the RL agent based on intermittent feedback provided by the designer.

Figure 5-4. Information and constraints that can be expressed with voxel representation : form (left), environment analysis (center), structure (right).

Figure 5-5. Example of design task with ambiguous goal and evolving constraints presented in this thesis (left). Suite of RL tasks provided by Unity-ML (right).