# CARNEGIE MELLON UNIVERSITY

## School of Architecture

College of Fine Arts

**Thesis**

Submitted in Partial Fulfillment of the requirements for the degree of

## Master of Science in Computational Design

TITLE:

Designing Agents for Co-Creation in Virtual Reality with Deep Reinforcement Learning

AUTHOR:

**Mitchell Foo**

**ACCEPTED BY ADVISORY COMMITTEE:**

| | | |
|---|---|---|
| Daniel Cardoso-Llach | Principal Advisor | May 10, 2023<br>DATE |
| Thomas Corbett | Advisor | May 10, 2023<br>DATE |

# Designing Agents for Co-Creation in Virtual Reality with Deep Reinforcement Learning

Mitchell Foo, MSCD ‘23

## Abstract

The ways that we embody interactions in virtual spaces are rapidly evolving with developments in the fields of deep learning and virtual reality. This is significant because these spaces are increasingly becoming extensions of different human social and creative practices. That being said, there is a lack of perspectives offered towards shaping technological usage that considers creative practices alongside immersive platforms and cutting-edge AI technologies. How then might we as designers, developers, and users, interrogate the intersection of deep learning and VR to better understand how the future landscape of design and virtual interaction might change?

This thesis offers insight into how these technologies might overlap in future design environments. This thesis provides a framework for a design interaction between human and agent co-creator in VR supported by reinforcement learning, an unsupervised training method focused on crafting AI behavior. This thesis contributes in two ways, by firstly providing insight into how human-agent interactions can be facilitated given open-ended design problems, and secondly, by exploring RL as a design medium for crafting agent behaviors through creating human demonstrations in VR. This thesis helps create and guide future questioning on the relevancy of emerging AI technologies in open-ended design contexts.

# Table of Contents


**Abstract** **1**
**Table of Contents** **2**
**1. Introduction** **3**
1.1 Context 4
1.2 Problem 5
1.3 Outline 6
**2. Background** **7**
2.1 Simulated Environments for Deep Learning 7
2.2 Designing for RL 11
2.3 Embedded Data Environments 16
2.4 Virtual Reality and Deep Reinforcement Learning 23
2.5 From Prototypes to Co-Creation 26
**3. Hypothesis** **37**
3.1 The Design Game 37
3.2 Implementation Expectations 39
**4. Methods** **41**
4.1 Prototype Overview 41
4.2 Environment Design 52
4.3 User VR Controls 56
4.4 Agent Training and Design 59
4.5 Latent Agent Visualization 73
**5. Results** **80**
5.1 Quantitative Training Results 80
5.2 Latent Evaluation 88
5.3 Qualitative Agent Observations 93
5.4 Reflection on Co-Creation 99
**6. Conclusion** **103**
6.1 Evaluating Methods 103
6.2 Limitations 109
6.3 Contributions 110
6.4 Future Work 111
**References** **116**
**Appendix A: Glossary of Custom Scripts** **121**
**Appendix B: Training Parameters** **123**

# 1. Introduction

This thesis introduces a VR environment that supports the development pipeline for training an agent with reinforcement learning[1] (RL). Instead of traditional objectives for RL being explicit and task-based, to understand more nuanced interactions between humans and agents, I propose a framework for an open-ended design task where the agent co-creates with a human collaborator within a VR block-building environment. This novel approach helps formulate questions regarding how we can utilize RL for qualitative objectives in design and human-AI interaction. Through realizing a prototype that brings human user and agent together in a design environment, the thesis hopes to shed light on the impact these emerging technologies will have on the cultural landscape of technology usage and design.

## 1.1 Context

The evolution of 3D virtual environments has allowed for further speculation on the changing landscape of human-machine interaction. These virtual environments encompass a variety of different spaces facilitating different levels of human agency. They could be spaces for working in the sense of computer-aided design and 3D design, spaces for play in the context of video games, and spaces for simulation when modeling reality. That being said, our interactions with virtual spaces are physically distinct from the real, and often, a virtual environment lacks nuanced and rich responsiveness that is understood in real life.

Emerging technologies pose the potential to transform this lack of response with virtual spaces. As a result, it is important to address technologies that have the potential to affect substantially how virtual spaces are perceived and interacted with. The thesis places focus on human-AI interaction, VR environments, and deep-learning models. Alone these technological fields have plenty of vested interest but brought together, there is a lack of research that expands on their potential applications.

In response, this thesis strives to create an environment that contextualizes the user and AI agent together in a shared VR space, collaborating on an open-ended creative task. In contrast

[1] A machine learning method of training an agent through trial and error with corresponding rewards and punishments within a given environment

to deep learning models such as stable diffusion[2] that draw from large labeled datasets to generate possible visual outcomes, this thesis proposes an agent-based approach in RL. Instead of thinking of a deep learning model trained to understand a design space from a labeled dataset, training an autonomous agent instead considers the experience gained in an environment designed to teach the agent how to do the task. Though this may not be the optimal model for some if not many design tasks, the emergence of VR and immersive virtual environments is an opportunity for extending our interactive abilities with a creative AI.

## 1.2 Problem

With the emerging role of virtual environments in deep learning applications, more research is necessary for understanding the potential relationship between humans and AI agents within virtual spaces. In particular, research with deep reinforcement learning has revealed that this method is unique to other machine learning paradigms due to the nature of its training, which could heavily weigh human feedback and demonstration. To extend the speculative potential of engaging with virtual spaces with AI agents, this thesis develops a virtual reality environment by which human user and agent can collaborate on a design task from which it is trained to do.

By including VR, we begin to question further the role that immersive virtual platforms might play in integrating humans with these aforementioned technologies. VR offers a unique lens and experience in virtual environments, but what is not often considered is that the kind of data it captures from the user is also unique to other digital interfaces. In the context of deep learning, it is important to question what can be done with this data to impact RL training and virtual experiences. We might even consider creating data purposefully in VR that augments deep learning training. The potential for users to design different approaches and behaviors in the agent needs further exploration. Implementing an environment and overall development process that interrogates these technologies should begin providing clearer insight into the future of human and AI agent interaction in virtual spaces.

[2] A generative deep learning model that creates images based on natural language text prompts

## 1.3 Outline

- Chapter 2 provides a wide background of topics surrounding virtual environments as it relates to deep learning as well as a more focused look at reinforcement learning both technically and for its potential to in developing embodied agents. Furthermore, topics on data and visualization are addressed to make consideration of user experience in interacting with agents.
- Chapter 3 introduces the prototype of this thesis and how it connects to the topics introduced in the background. It concludes by stating why said prototype will help bridge these questions together for reflection and evaluation.
- Chapter 4 details the technical development framework of the prototype, from the designing of the environment, agent design, application interface, and reinforcement learning training.
- Chapter 5 reflects on the qualitative and quantitative evaluation of the prototype. This includes a personal reflection on the outcome of the prototype's development, the quantitative results of agent training, and user testing on their experience with agent collaboration.
- Chapter 6 concludes the thesis's findings, its contributions, limitations, and suggestions for future work.

# 2. Background

3D engines are a popular tool for a variety of applications, whether it's crafting experiences in video games, to creating models that can simulate realistic dynamics. Being removed from realistic constraints, many deep learning applications are utilizing simulated environments to execute their training, especially in the fields of robotics, autonomous driving, and computer vision. As a result, these engines provide opportunities for augmenting the deep learning process when considering how one can interact with the environments being developed.

This section provides an overview of the emerging practice of deep learning with simulated environments and how that might be extended given recent developments in virtual reality. By identifying the current state of deep learning and virtual reality, this background should highlight the opportunities for this thesis to expand and define an emerging multi-disciplinary field.

## 2.1 Simulated Environments for Deep Learning

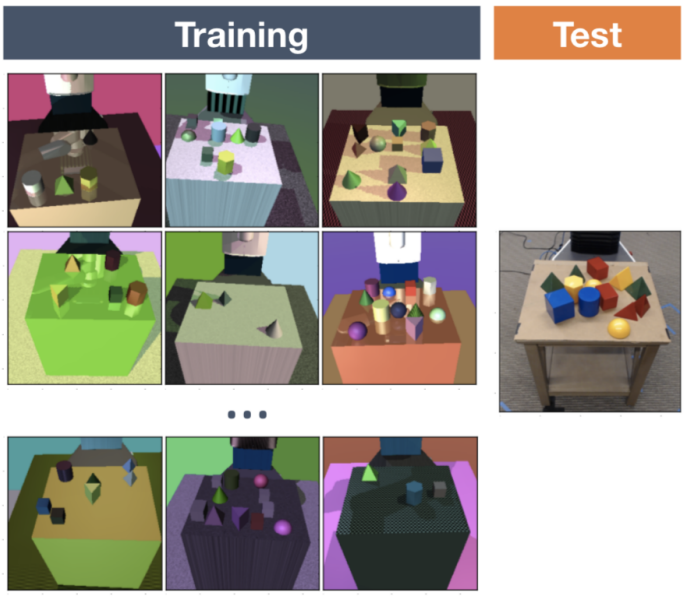
Simulating environments in 3D provides a platform for designing the experience from which deep learning training is derived and is an area of research only recently been documented as its own field[36]. Emerging applications for computer vision models and 3D environments enable the creation of synthetic data, removing the cost and time constraint of obtaining ground-truth datasets in the real world. This idea of ‘sim-to-real’ can help suggest methodologies for thinking more critically about what our crafted 3D environment suggests synthetically to the agent during training.

### 2.1.1 Synthetic Data Environments

Crafting synthetic datasets for deep learning is concerned with how simulation can transfer successfully to a real context. Bridging this gap between reality and simulation, 3D engines have emerged as a useful tool for producing datasets where realistic rendering, randomization, and physical dynamics are required [45]. That being said, there are still gaps between simulation and

reality, which often include the ability to accurately reproduce noise and dynamics that are difficult to model or predict.

One of the earliest applications of 3D engines for synthetic data creation that attempts to address this gap is in a paper that applies domain randomization within a 3D rendering engine [49]. Their task was in training a simple object detector where a training dataset was purely rendered images and the test dataset real images of objects arranged on a table. What they discovered was that instead of training on a single environment, having the simulator randomize camera position, lighting, object textures, and pose, they were able to train a precise and generalizable detector purely on rendered images. Not only can one train a computer vision detector via purely rendered images, but the level of control over the randomization in the dataset also suggests a way to engineer the final deep learning model through the 3D simulation. A more recent example of this application is in Microsoft's 3D facial reconstruction model [53], which is trained on randomized human faces, realistically rendered, and automatically annotated. Not only is synthetic data generation more cost-efficient, but it also ensures that all types of faces and possible headwear are accounted for in the dataset.




Example synthetic dataset made using domain randomization for sim-to-real [49]

Though the examples above are successful in one-shot vision detection, more complex deep learning applications require simulated environments that provide dynamic interaction to

address the sim-to-real gap. The most common application for these environments is in robotics and autonomous vehicular navigation. Typically built atop popular games engines like Unity[3] and Unreal Engine[4], reality simulations have been used for creating synthetic image-based datasets [46] as well as full dynamics simulation that also extends beyond just object detection to include support for RL and imitation learning (IL) [12][14]. Particular to CARLA [14], a platform to support autonomous driving, recent developments to extend the platform has been concerned with providing additional realism to the simulated environments, from pedestrian animations to vehicle behaviors, and weather. These qualitative developments to the simulation environment emphasize the importance of authoring environments for deep learning training.



In-engine capture of CARLA, comparing segmented (left) and realistic (right) views [14]

In addition to simulation environments being used for a directly functional application, researchers with AI2-THOR explore the potential to be a platform for exploring embodied AI [12], agents that can adapt and generalize in the way humans can. Their exploration was through training a robot to perform navigation tasks in simulation using a vision-based sensor. The simulated environment used for training was a replica of a real space to explore the gap between sim-to-real. They hypothesized that an agent learning to detect objects in simulation can learn a visual semantic logic for navigating a space. Though their experiments did not produce successful results in transferring from sim-to-real, asking questions about the potential for simulations to explore embodied AI is a question worth continual exploration.

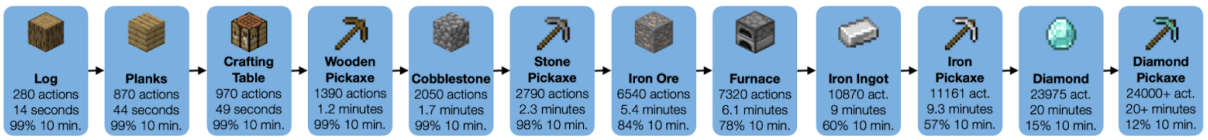
[3] Unity is a cross-platform game engine designed to be highly-customizable for a variety of 3D applications
[4] Unreal Engine is a game engine specialized in cutting-edge computer graphics

### 2.1.2 Environments as a Training Testbed

Due to the dynamic nature of 3D engines, in particular, those developed on top of game engines, 3D environments are also often used to benchmark RL algorithms. In this instance, the dataset comes as a result of experience from exploring the designed environment. An agent is tasked with exploring the environment's dynamics given an objective and iteratively improves itself to better accomplish this task. As a result, in order to benchmark and improve RL algorithms, baseline simulations are used universally by researchers, the most popular being MuJoCo [51].

Another popular benchmark for RL is video games, as not only does it allow for observing how well the agent plays the game, but also emergent strategies and similarities to the way humans would play the same game [36]. Even in complex strategy games [40][52], the agents are able to outperform the best human level of play and are able to discover known strategies of a game strictly through iterations of playing the game and evaluating how well it is performing [33].

In addition, the popular video game *Minecraft* has inspired a competition to train an RL agent to play the game using prior human demonstration [6][21][25]. A game with more open-ended exploration and ways to accomplish tasks serves as a helpful testbed for training agents with human demonstration. In this case, agents are rewarded for their ability to achieve a specified sequence of tasks through interaction with the open-world environment. What could furthermore be interesting about this example is the potential to explore new tasks given that the environment is open-ended where human demonstration can prescribe multiple different ways to achieve the same task. More often the case, video games are a way to test training methodologies and performance rather than explore the agent in its particular setting.




Description of the time it takes an agent trained with Video PreTraining to build a Diamond Pickaxe in Minecraft [6]

## 2.2 Designing for RL

In order to facilitate recent advancements in RL, much research has been put toward developing training environments. Frameworks and APIs have been developed to help support the creation of these environments, most popularly extending from OpenAI's Gym [8]. This section goes into defining the relevant RL concepts as they relate to developing environments for training agents. Furthermore, it also presents Unity's ML-Agents package as a tool for designing environments and agents [24].

### 2.2.1 RL Overview

RL is concerned with agent-based decision-making to maximize rewards in a given environment. Unlike other deep learning applications where a dataset is given for training a model, RL algorithms instead must learn unsupervised through sparse reward signals [33]. As a result, RL can be difficult to evaluate in comparison to supervised learning, which is more simply evaluated based on a training and validation dataset. Instead, in more applied settings, one must consider the scalar rewards given to an agent that can guide the agent to complete its ultimate task. For this section, a general overview of RL methods is provided based on the slides from the course *Statistical Techniques in Robotics* at Carnegie Mellon University.

An RL problem is also called a Markov Decision Process (MDP), which is defined by having: a set of states, a set of actions, a transition function, a reward function, and a start state. Agents are evaluated based on a trajectory of state-action pairs otherwise known as an episode. A given episode is evaluated for a given number of states-action pairs from which the sum of cumulative rewards in the environment determines the agent's return. The goal is for the agent to maximize the expected return. For any MDP, we can define a policy $\pi$, which maps an action to a given state. An optimal policy $\pi^*$ is the goal of RL, knowing what actions to take given any state to maximize the expected return.

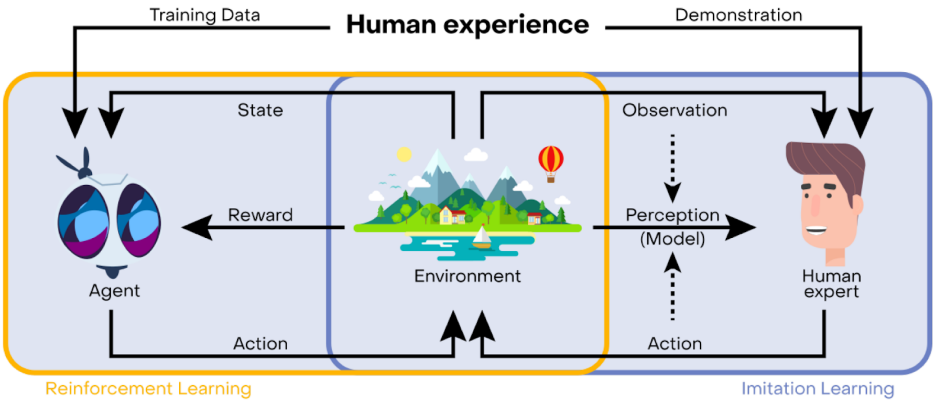



Diagram describing how RL and IL are a combination of state-action pairs learned by interacting with an environment[5]

There are three main categories for RL methods: gradient-based, Q-function-based, and model-based. The most popular methods in a simulation are gradient-based methods like Proximal Policy Optimization (PPO) [47], Q-function-based methods like Deep Q-Learning (DQN) [33], and methods that combine the two like Soft Actor-Critic (SAC). The primary difference between these two approaches is how actions are selected whilst learning a policy. Vanilla Q-learning's goal is to learn a single deterministic action from a set of discrete actions that would maximize the reward. Discrete actions, for example, could be moving a character in a video game via arrow keys. In contrast, gradient-based methods are learning a probability distribution of actions given a state. As a result, gradient-based methods are better suited for more complex environments and for continuous action spaces. Continuous actions, for example, might include the angle of a steering wheel and the pressure applied to a pedal.

For this thesis, we will be primarily focused on using gradient-based methods in PPO. While gradient-based methods are the least efficient, in the long run, they usually provide the best-performing policies and require less hyperparameter tuning. Because we are concerned with training in simulation instead of real-world training, sample efficiency is less of an issue.

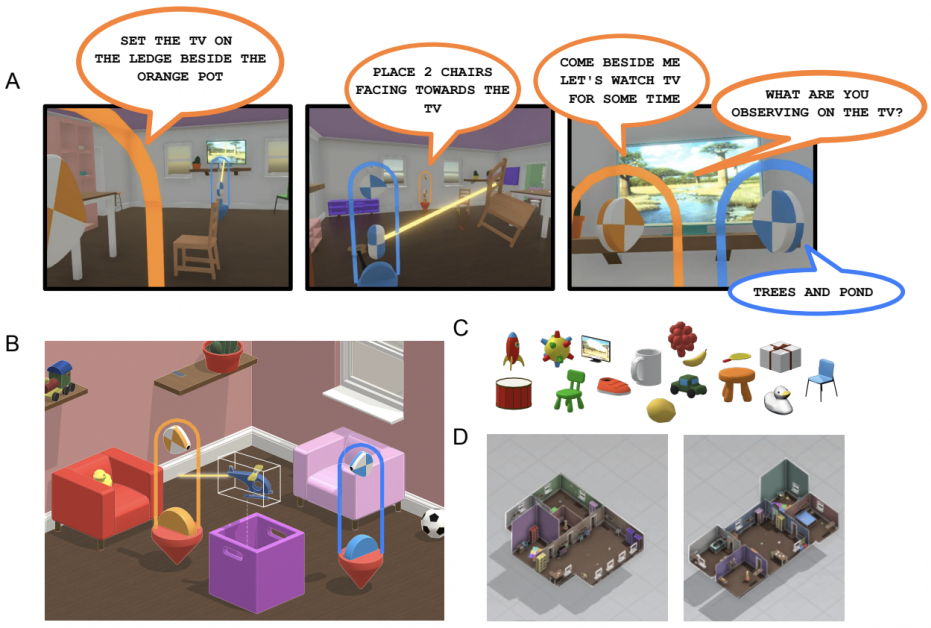
### 2.2.3 Imitation Learning

IL is a method adjacent to RL that trains an agent based on a given set of demonstrations produced by either a human demonstrator or an expert agent. Common methods in IL are

[5] Navidi, Neda, and Rene Landry. "New Approach in Human-AI Interaction by Reinforcement-Imitation Learning." *Applied Sciences* 11 (March 30, 2021): 3068. https://doi.org/10.3390/app11073068.

Behavior Cloning (BC) [25][26], and Generative Adversarial Imitation Learning (GAIL) [23]. These methods all require demonstration data prior to training. This data is a combination of state-action pairs that can either pre-train the agent or be a reward signal during training. IL techniques have been shown to greatly accelerate policy convergence [23]. IL also facilitates a unique dynamic between the human demonstrator and the agent, allowing for a sense of control over the agent's policy via the weight of their demonstration [4]. This thesis will be most concerned with using GAIL for IL.

IL has also recently been studied by researchers at DeepMind for its potential in creating multi-modal human-like agents in virtual environments [2][11]. What this work presents are novel training methods for creating agents with naturalistic behaviors through pairing tasks in a room with natural-language-based directions. What they identify as a great challenge in their research is in capturing the human-embodied intent in the demonstration data. A recognized limitation in their research is that demonstrations were crafted from humans interfacing with the environment via computer, limiting the physical embodiment of the agent to the human demonstration. In their future works, they also comment that an exciting direction would be to also consider reward signals like that used in training RL agents.

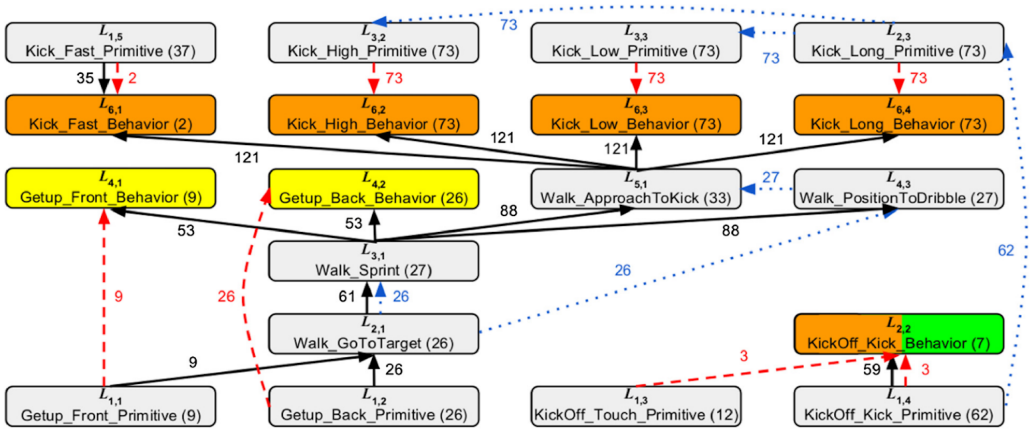



Overview of MIA's AI avatars performing actions in a virtual environment based on human natural language [11]

## 2.2.4 Curriculum Learning

Curriculum learning (CL) is a framework within RL that breaks down a larger task into smaller sequences of tasks to help guide and accelerate an agent's training [35][41]. CL builds on a more general concept in machine learning called transfer learning, where instead of models being trained for new tasks from scratch, you would initialize training for a new task with a model that has been already trained for something related. Hence, CL uses transfer learning to train models on simpler tasks and graduate to more complex tasks using the knowledge learned from the previous training.

Task generation and sequencing are critical to the design workflow in CL. Though there are studies to find ways to automate these steps [29], typically these tasks are handcrafted by domain experts. An example of this application being expanded successfully is a team of researchers training robotic agents to play soccer via simulation [32]. They proposed a layered approach to learning, where in order to train their agent on nineteen different behaviors, each behavior was separated into its own training task and sequenced such that the simpler behaviors were trained first and more complex behaviors that may depend on the simple behaviors were trained later. Relying on handcrafting the sequence seemingly worked to improve agent training. A study even showed that tasking non-domain experts to create agent curriculums helped optimize training [35], showing how CL can be a useful means for transferring human intuition to agent training outcomes.



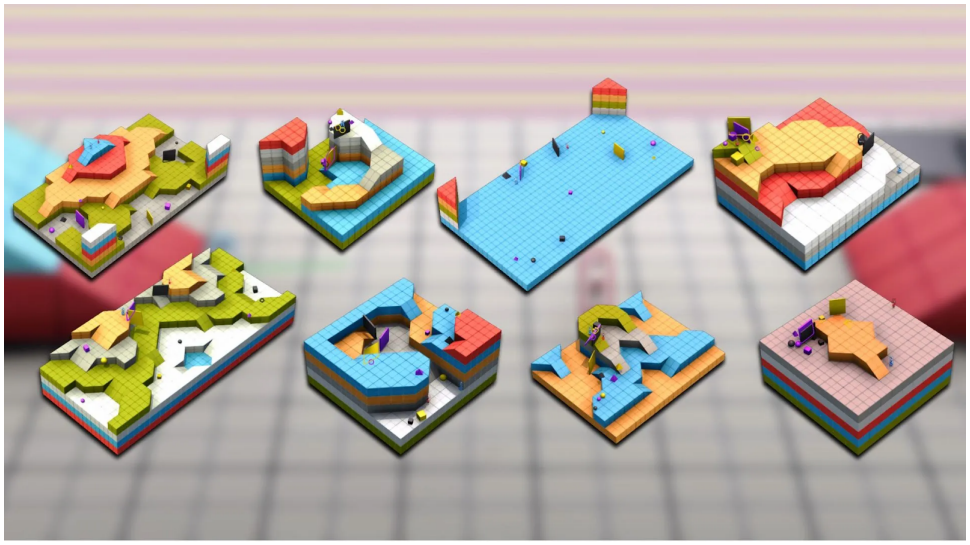

A curriculum describing the sequence of behaviors to be learned for an agent playing soccer [32]

### 2.2.5 Designing Training Environments with Unity

Testing RL algorithms in simulation for the sake of the algorithms often does not require one to build their own platform for conducting the simulation. Often, popular environments pre-packaged with OpenAI Gym [8], like MuJoCo [51], Atari, and Lunar Lander, are sufficient for benchmarking algorithms. But what if one wants to extend RL to a specific application? As a result, especially when the application is 3D, game engines like Unity and Unreal Engine are being extended to support RL algorithms and Gym [54].

Unlike Unreal Engine, Unity Technologies has developed a native package for training RL agents within Unity's game engine called ML-Agents[23]. Taking advantage of Unity's engine itself is a great opportunity for RL research because of how powerful its interface is for developing customizable environments and interactions. ML-Agents is a Unity-developer-friendly package, which wraps RL algorithms within a simple Python API and is integrated within Unity's development framework. This means that Unity developers can conceptualize and implement RL training through a more familiar lens.

As researchers have begun using Unity's ML-Agents to explore novel RL ideas and design their own environments for training [17][20][28][48], there still is a relatively small amount of publications given the wide range of potential applications for Unity. Much of the current research uses Unity as a tool for creating custom environments, especially in robotics and sim-to-real work, but what is lacking is reflecting on the platform itself, as a 3D engine for design. Some of the most promising work that begins to look at deep learning and game engines are coming from DeepMind [39] and game studios themselves like Ubisoft [19]. What makes these works promising is that it is less so using the game engine to model a simulation but rather using the context of video games as a means to inspire novel applications and training methods for deep learning.



A series of 3D environments developed by DeepMind to diversify training of a generalizable agent [39]

## 2.3 Embedded Data Environments

In the context of simulation, virtual environments are developed with large dimensions of data in order for their parts to appear and operate as one unified whole. One can say the same about real environments too, the only difference being that virtual environments are man-made and hence have all of their underlying parts and interactions documented and understood. As for real environments, as humans, we develop frameworks for observation and formulating data. Much like the RL agent training in virtual environments, we as humans are agents collecting observations in our environment. This section takes a broader view of what it means to have data embedded into one's observable environment, reflecting on the potential for human means of data representation to influence the design of 'data environments' for RL agents.

### 2.3.1 Contextual Data

In RL, data is developed through experiences in a given environment that are designed to provide information as feedback. Similarly, as humans learn through observation and interactions, we learn through both tangible and digital feedback. Regarding human interfaces with the digital world, Paul Dourish writes in The Stuff of Bits [15], "Shifting from a language of virtualization to a language of rematerialization turns our attention from questions of absence to questions of

presence…software and digital information have their own materialities that manifest themselves in our encounters with them." (85) Regarding Dourish, because the digital is becoming increasingly intertwined with our physical world, how one sees this virtual world might actually be closer to our physical understanding of the environment than we think.

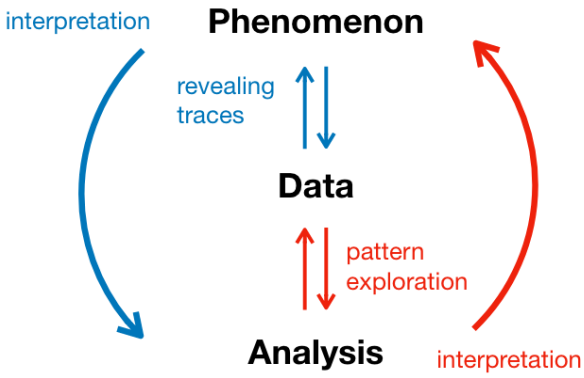
It is critical to consider the data that supplements our perceived world as it relates to producing environments for learning. From producing data to its access, there is a series of conscious abstractions being made, as if creating data was its own language. For this thesis, these two relationships are important to consider when designing environments for learning: how data is being abstracted to represent the world is learned and how that data is being represented as feedback for interaction. Considering designing environments for learning as a practice situated around data, it is important to distinguish that this is not any kind of data but instead, highly contextualized data.

Contextualized data refers to data that is anything but generic, instead specific to describing place. Especially in the context of machine learning, where often time datasets are generic or standardized for the sake of benchmarking algorithms. For the purpose of this thesis, data is given the opposite treatment. Data theorist Yanni Alexander Loukissas explores this idea by proposing a practice around data that he refers to as 'local readings'[31]. Not only does he argue that the place from which the data comes is as important as the data itself, but data is also a representation of culture and hence bears an interpretive and relative dimension to it. What he proposes is to explore data visualization as a means to connect place and dataset, and as he puts it, "...focus on showing the structure and texture of data, rather than offering clear visual patterns, telling stories, or answering narrowly defined questions, as more conventional instances of data visualization might do." (8) Though thinking about visualization techniques alone will not train an agent for a task, it can inform how one might consider the design of simulation environments. This consideration might inform the designer to provide an environment that goes beyond just a task to be completed but to also consider more subjectively how to complete said task. Furthermore, we should also think of these data environments beyond just being occupied by the agent, but too, by the designer.

### 2.3.2 Material Data

In data environments particular to training RL agents, there are a number of ways to frame data from the perspective of either the agent or the human designer. For the agent, it senses by its design and from interacting with the given environment. The designer observes the agent whilst training and collects data on the basis of its quantitative performance and qualitative behaviors. These observations inform the decision of the designer to either alter the agent or environment, hence creating a relationship between the agent and human designer around the embeddings of data in the simulation environment. Though practices in RL have aptly presented techniques for quantitative observation such as calculating the agent's average reward over training steps, methods for a behavioral evaluation are instead lesser explored.

To explore a system for qualitative observations in RL environments, one approach could be to look at ways we might document and observe the real world in the production of qualitative data. In particular, a useful framework can be found using autographic visualization, a concept proposed by Dietmar Offenhuber as a counter-model to traditional forms of data visualization reliant on numbers and abstracted symbols [37]. Offenhuber argues for a way to represent data reliant on the phenomenon itself rather than through the introduction of other systems. As Offenhuber puts it, "Autographic visualizations can be accidental, such as the desired paths across grass areas in the city or the uneven traces of wear on a staircase or a computer keyboard. But in general, they are the outcome of design operations that aim to reveal, isolate, amplify, conserve, and present material traces as records of past processes and events." (2) Framing data collection as a material trace not only contextualizes the data to its environments, it furthermore explains the phenomena of its events.




Offenhuber's diagram for autographic visualization [37]

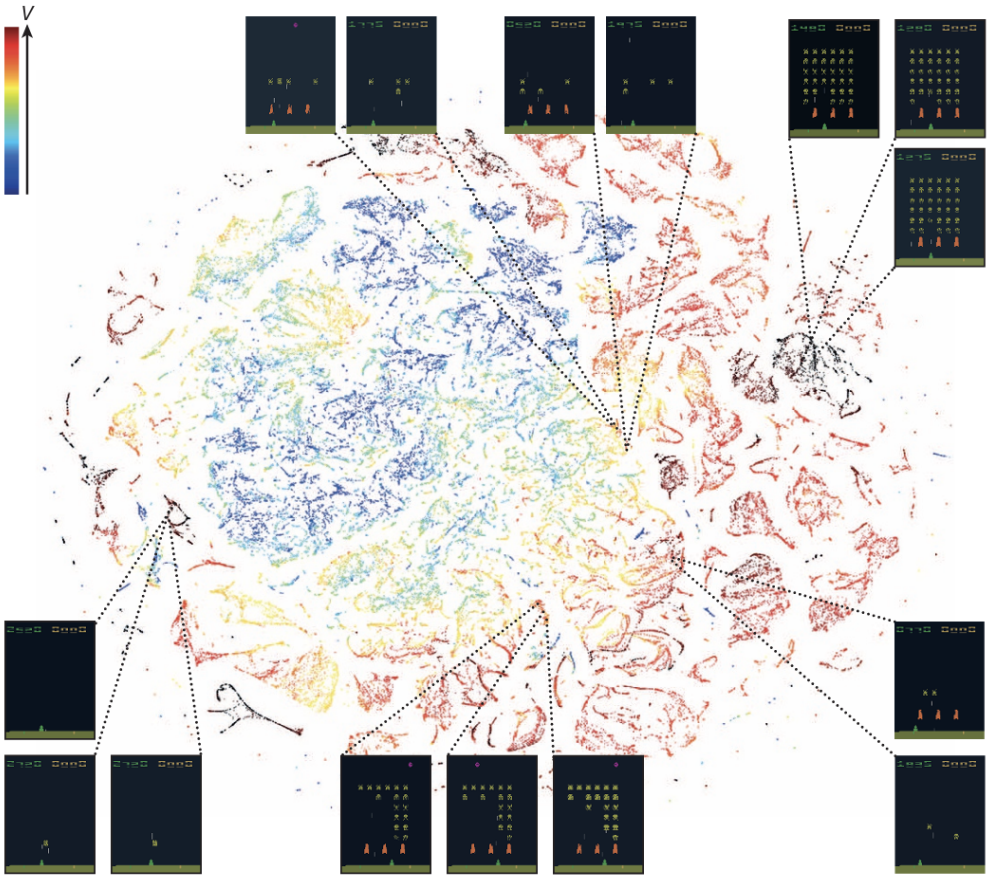
By focusing on the environment as a container for natural phenomena, we might also consider agent learning through exploration as its own contained phenomena. Through the lens of autographic visualization, one might consider the following: treating the agent and its interactions as inherently material and hence traceable as well as designing into the environment the means to observe these traces for observation and data collection. What makes this theory of autographic so translatable to a simulated environment for RL, is the fact that even though the environment is explicitly constructed, the impact of an agent interacting within is a phenomenon that requires ways of tracing.

### 2.3.3 Environments for Real and Virtual Data

To expand upon contextualized data and data environments, a simulation needs to serve as a space for both human designers and agents to operate in. The reason why it is important for the designer to understand the data environment is that through the environment, the designer can understand the agent and its training. In current RL practices, perhaps the best representation of a contextual analysis of an agent is through treating the agent's action space as a latent space that can be explored in two dimensions with dimensionality reduction techniques like t-SNE[6] [34]. Through this approach, an agent's observation, visual or vectorized, can be mapped to a relative space of actions. Considering 3D simulated environments, can such a representation translate to

[6] T-distributed stochastic neighbor embedding is a method often used in data science to help visualize clustering and relationships in large high-dimensional datasets at lower dimensions

a greater dimension and furthermore, incorporate more immersive and familiar representations that can help situate the designer?

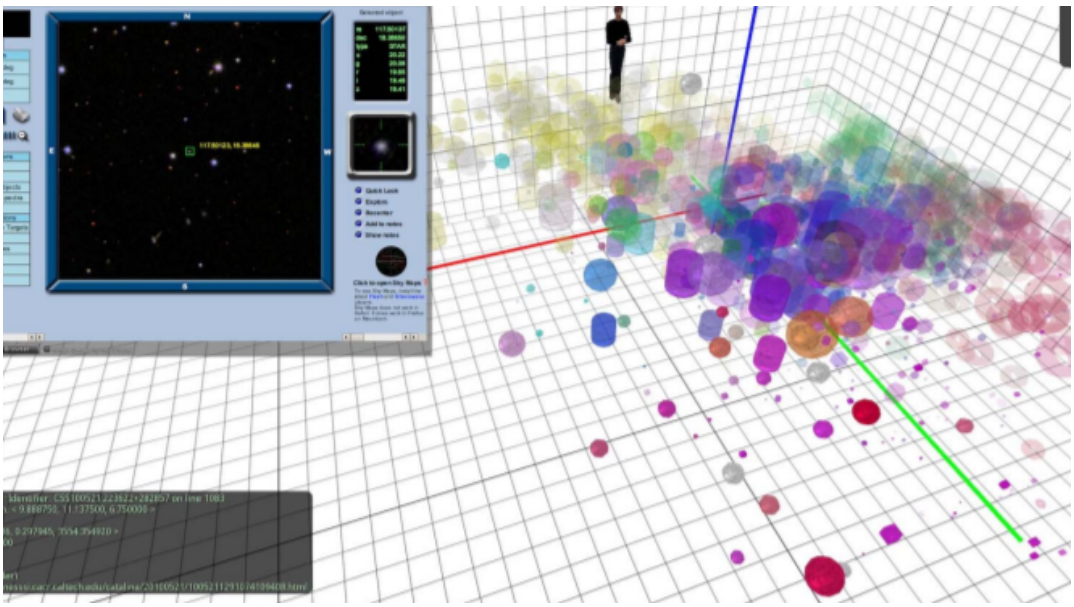



2D mapping of latent action embeddings taken by a trained RL agent [34]

An important quality of an immersive data environment is the ability to extend beyond the limitations of 2D representation. As discussed in DeepMind's MIA project [11], a 2D interface for interaction with a 3D environment is limiting the extent to which human users can interact intuitively with the data environment. Though they conclude that expensive telecommunication devices are required to extend the embodiment of the user in simulated space, I would instead argue that we already have accessibility to said devices in the form of virtual reality (VR) headsets and controllers.

In exploring VR as a method for engaging with data environments, Donalek et al. explore this possibility by creating a VR exploration and curation environment for high-dimensional data [13]. Donalek argues that VR allows for more intuitive interaction and experience with data

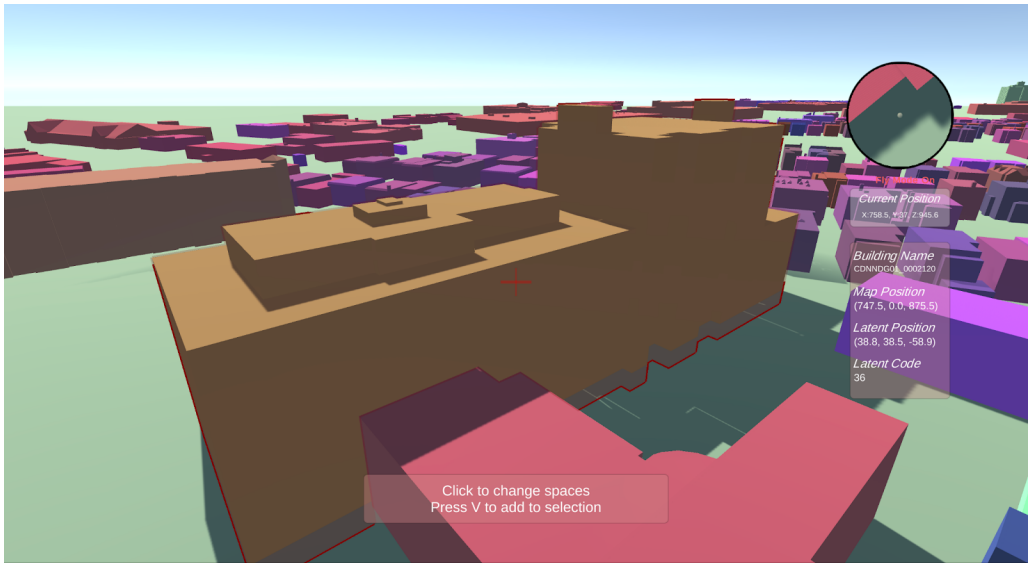
points. By being in 3D, data visualization can be extended beyond three dimensions through the use of symbols, scale, and color. Furthermore, Donalek extends their VR application to be a collaborative platform for multiple users to operate within the same data space. Being able to operate intuitively within a high-dimensional environment with another user is what makes VR a promising tool for data exploration and interaction. That being said, to extend from Donalek's research we might consider autographic visualization for a more contextualized data environment. Instead of focusing on the representation of the data points, rather consider the entire virtual environment as a representation of the data's locality.



View within *iViz* [13]

To consider the locality of data and the representation of high dimensional data as it relates to machine learning, I explore my own visualization platform to explore concepts together. What I present in my application is a linkage between a variational autoencoder to its 3D dataset, which is a collection of 3D models of buildings in Montreal. Through using the Unity game engine, I present a first-person interactive approach for exploring data by linking two data spaces: the buildings of Montreal in their geographic location at a 1:1 scale and a 3D latent space that locates the buildings by their latent embeddings. In this environment, the user can move and select a building in either space and by doing so, will teleport the user to the corresponding location in the other space. Two key outcomes from this research are the following: firstly, datasets can be presented in an interactive manner through a representation of

their original locality, and secondly, 3D environments are a powerful tool for connecting the latent space of data objects to their more familiar representation.




Building's geospatial context in *A Walking in Latent City*




3D model embedding located in latent space in *A Walking in Latent City*

Connecting my research to that of Donalek, I consider how implementing VR controls would affect the immersion between two data spaces. VR requires different considerations for the user, where locomotion and scale are more connected to the user's physical body rather than just a 2D window into a virtual space. On one hand, immersion into a contextualized

environment might be even greater but on the other hand, flying in a large urban space might feel too discomforting and as a result ineffective as a data interface. To address this, one can think of two helpful constraints: smaller-scaled environments, and a presentation of the physical and data space as one layered space rather than two distinct ones. We see a similar consideration emerge with datasets of real homes modeled in 3D, to serve as a connection between sim-to-real for different deep learning applications [12][42][43][49]. There lies an opportunity to extend the interface of these datasets during training, by allowing human engagement with the environment through VR controls and embedding the environment with visual access to its underlying latent data.

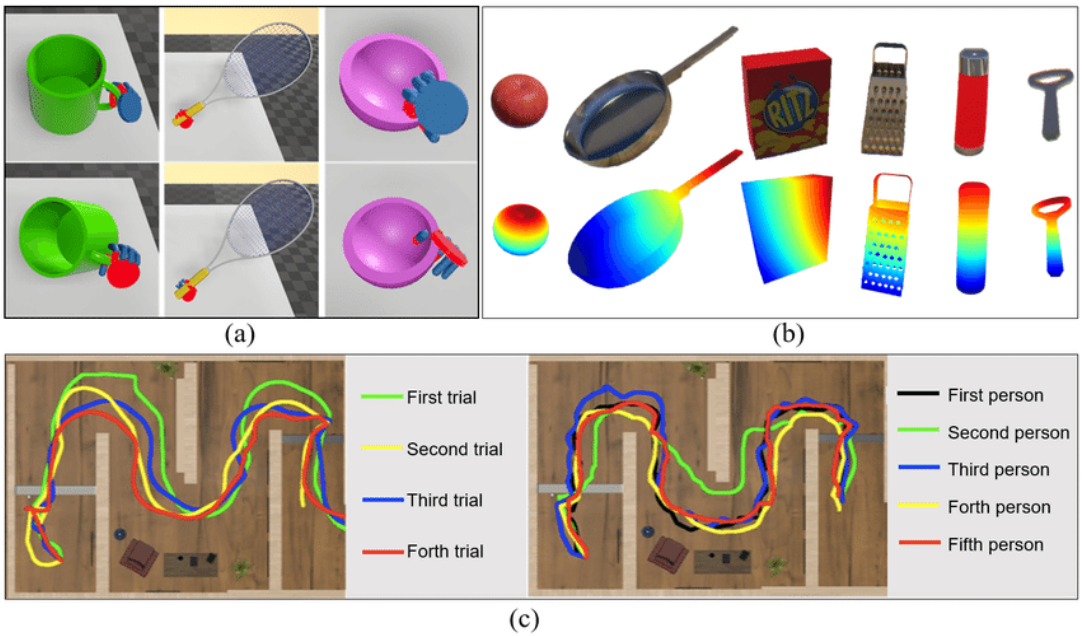
## 2.4 Virtual Reality and Deep Reinforcement Learning

VR poses the opportunity to engage more immersively with virtual environments than other interactive platforms. Not only is VR becoming more accessible to the lay consumer from its increasing affordability and ease of use, but there is also the potential large cultural impact it could have regarding the human occupation of virtual spaces. As a result, human interaction with the digital, especially with data, is entering an unexplored domain of both application and impact, posing both areas for opportunity and potential risks. In order to support this thesis' exploration into this domain, there are a number of more technically driven research aimed at using VR to support deep learning and RL training [16][44]. In this section, I explore research that can help support the methodologies of my thesis, from technical implementation to agent evaluation. This also will help define a unique perspective this thesis can take at this particular intersection of technologies.

### 2.4.1 Integrated VR Training Environments

Regarding research into VR and deep learning and the topics of this thesis, *VRGym* [56] is the most realized example of a platform for human-in-the-loop robotic training in simulated environments. This research contributes a comprehensive integration of VR into a robotic simulation, including extended reality hardware integration, deep learning training, techniques for visualizing agent interaction data in the scene, and human-embodied avatars for agent

interaction. Underpinning this research is a sim-to-real adaptation for robot agents, where in a safer digital environment, exploring human-robot interaction is made possible.




*VRGym* showcasing evaluative techniques for describing VR and agent interaction [56]

*VRGym* is an exemplary general platform for human-agent interaction in VR and can be extended by considering its simulation context further. Doing so might consider an application beyond sim-to-real and robotics. *VRKitchen* [18] is another similar VR training environment but instead positions the agent and user in control of the same digital avatar with the objective of completing virtual kitchen-based tasks. *VRKitchen* has capabilities for RL training as well as recording user demonstrations for IL. Positioning the user and agent within the same avatar, allowed by virtual reality and virtual environments suggests a wholly virtual context for human-agent interaction. Though *VRKitchen* does not have capabilities for human-agent interaction, we can speculate what those impacts might be of interacting with an agent that embodies a similar appearance and task trajectory to the human user.

### 2.4.2 VR for Embodied Demonstration

The ability to demonstrate in VR has also become a practical tool for training agents and robots. Some early examples include controlling robotic arms [20] and creating demonstration datasets

using VR hand controls [17]. Koganti et al. suggest Unity as an ideal platform for training agents with VR demonstration via BC and GAIL [27]. In their research, they used the VR headset to track and demonstrate gaze direction for firing at targets in a simple game environment. They report that using BC, 20 minutes of the demonstration was adequate for an agent to outperform the human user in their game environment. But aside from using VR controls functionally for demonstration, there is also more to consider regarding the embodied nature of the data being recorded for training.

A useful perspective on VR demonstration is to consider it as another method for generating human-based synthetic data. Given the simulation environment, how a user responds via VR can be recorded and used to train various ML models, examples including gaze tracking [30] and fall detection [57]. Particular to the research for fall detection, Zherdez et al. combine embodied avatars using VR with vision-based synthetic data generation techniques, including automatic labeling, rendered depth, and instance segmentation. In the way that simulation is useful for designers training agents and ML models, those affordances can also be reflected back to users entering the virtual environment in VR. We are able to now provide a richness of data back to the observing agent to impact their training.

### 2.4.3 Risks and Implications of VR and Deep Learning

VR being an emergent technology that can support novel approaches in deep learning is exciting but bears a number of risks and assumptions designers should be wary of when using it for development. These risks are concerned with the manifestation of a new form of data or as Carter and Egliston coin it, ‘virtual reality data’ [10]. Much in line with the advantages of embodied data through VR, Carter, and Egliston write on the very risk of said VR data. In summary, they highlight the risks in three key points:

1. A fantasy of perfect data - we cannot assume that VR data is a comprehensive representation of what it is attempting to capture
2. Quantifying the qualitative - there is an impulse to quantify intangibles like human emotion, which ultimately cannot be directly measured

3. Data empires - VR data can be captured and commoditized for machine learning applications

The above considerations might not be directly addressed within the scope of this thesis but are important to acknowledge. While the thesis presents a practice towards more embodied data, there are risks to interacting with increasingly more realistic AI in virtual environments. One particular consideration is the potential necessity for AI to be represented and designed distinct from human users. We see such concerns already emerge in research regarding AI avatars in social platforms in the late 2000s [9], but such questions remain relevant when considering the emerging nature of deep learning and VR.

That being said, the virtual does provide positive implications for VR data too. VR and synthetic data pose the opportunity to allow for greater control of the datasets and remove biases that might be found from collecting real data [36]. Furthermore, training by demonstration in VR can allow users to create agents that are customized through their own training data and environment. Though this practice needs more research, emergent from this thesis is the potential to promote an interface that can help democratize AI.

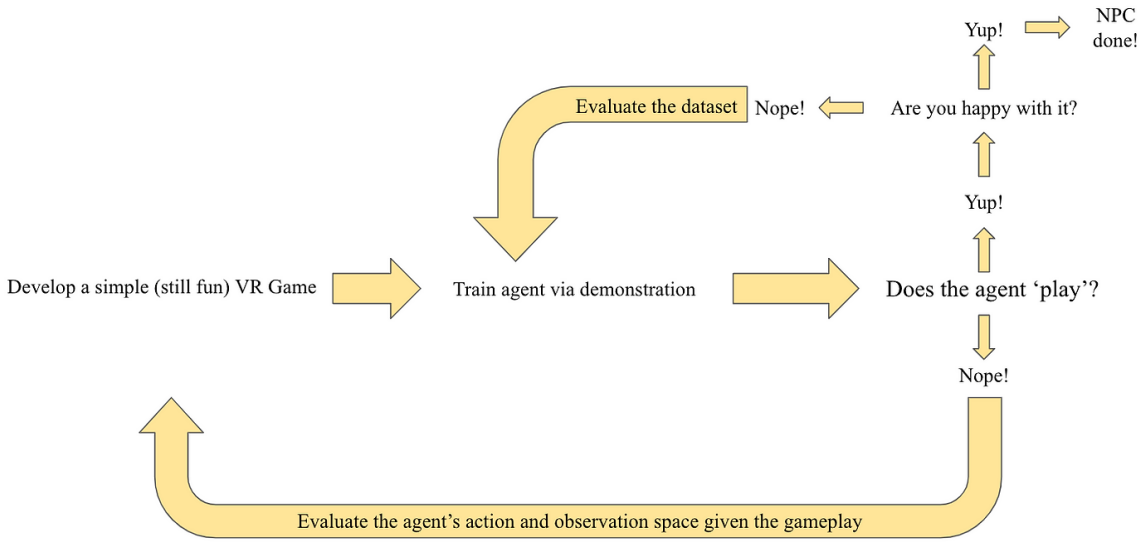
## 2.5 From Prototypes to Co-Creation

In supporting this thesis, I have undertaken prototypes involving VR and RL in order to begin to explore the questions being investigated in the presented background material. These questions involve addressing human-agent interaction and design. My explorations look towards understanding how we can design agents behaviorally through the environment and by setting up competitive game-like scenarios. Ultimately, building from these explorations, we can uncover motivations for a virtual environment capable of supporting explorations of human-agent co-creation.

### 2.5.1 Imitation Ball

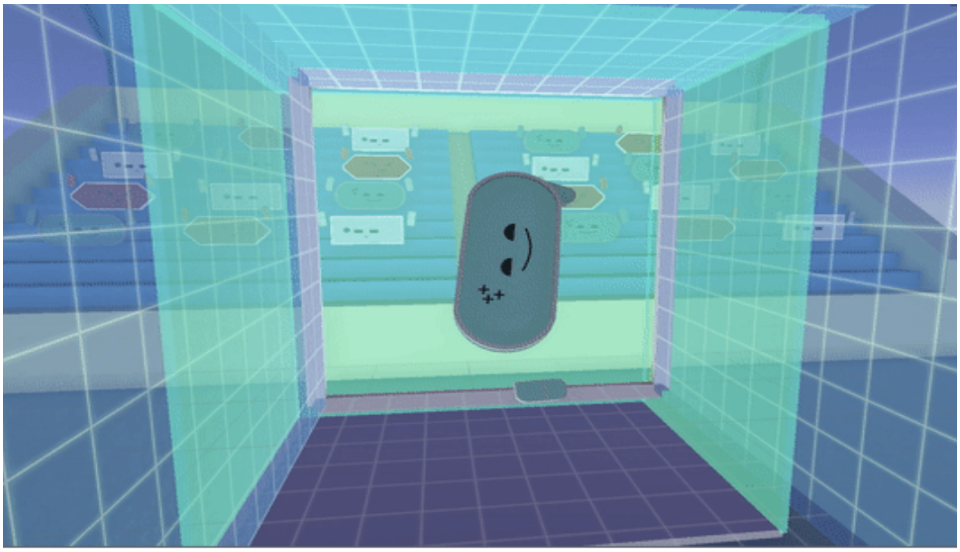
*Imitation Ball* is a VR project inspired by the likes of *VRGym* [56] to facilitate a range of human-agent interactions and training. *Imitation Ball* is a VR game best described as a 3D Pong, where the player stands at one end of the hallway and rallies a ball back and forth with an

opposing AI paddle with the objective to score on the opponent. The game was designed to be simple enough for an agent to learn how to play, while also being complex enough that gameplay styles could be explored for the agent. Because in modern RL, it is evident that enough training would likely result in an undefeatable opponent AI, the objective of this project was to explore how to craft different types of AI opponents' behaviors incorporating IL, instead of the traditional notion of optimizing agent performance, a concept with initial exploration in the work of Borovikov et al. [7]



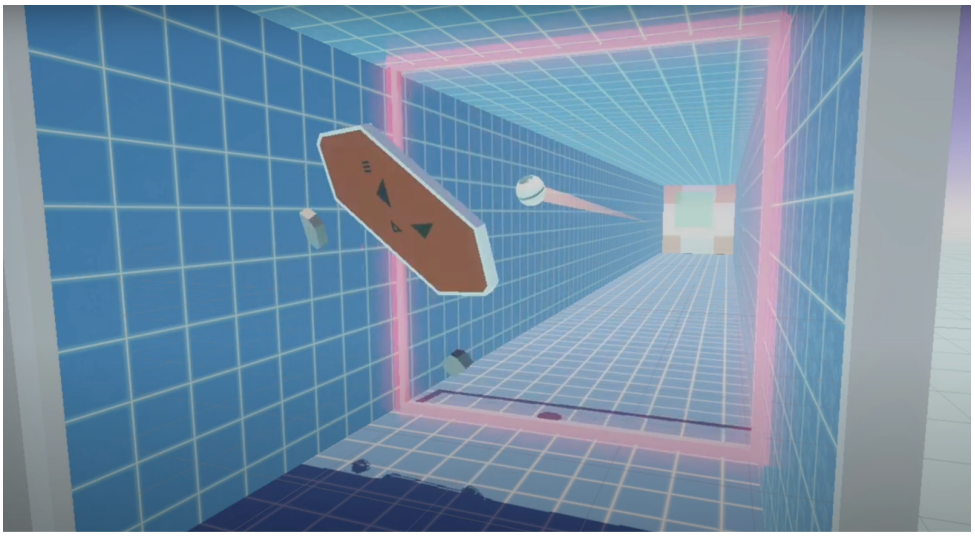

An early proposed development cycle for *Imitation Ball*

Important for this project was that it served as both an investigation into training processes as well as entertaining gameplay. This is because, in an attempt to frame this process as a design problem, it is essential to establish a rich context to which the training is applied. As a result, designing unique visual characteristics for the agents help enrichen the appearance of this game by personifying the AI opponents based on their emergent AI behaviors.



Agent opponent in the VR game environment

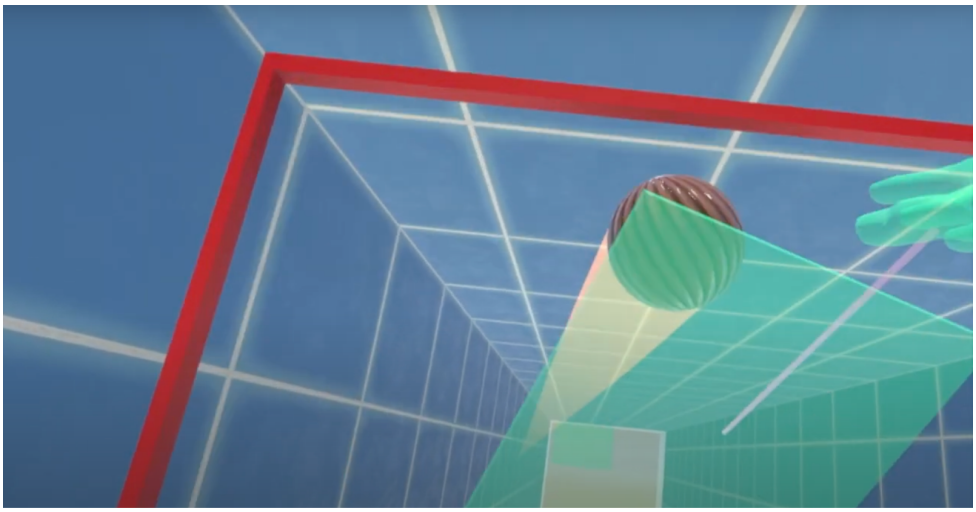
As an exploration into training processes, the project was also a platform for a variety of different types of agent training. Within the Unity project, there are different scenes for different training environments that include the basic empty hallway for initial agent training, hallways with moving obstacles, hallways with targets for the agent, and a hallway for multi-agent training.



Agent in the training environment

In the final build of the game, four distinct agents were included, each with their own distinct behaviors that were crafted by tuning their movement parameters and demonstration

weights prior to training. The key finding was that weighting GAIL during training has a visible effect on the way the agent performed their objective task. Their task was to simply move and deflect the ball back, but an agent trained without demonstration would simply move without rotating itself. In contrast, in the human demonstration, there were instances where the player would swing and twist their wrist, almost as if playing tennis or squash. As a result, agents trained with GAIL on this demonstration projected this behavior, flipping and rotating whenever the ball would near their position.



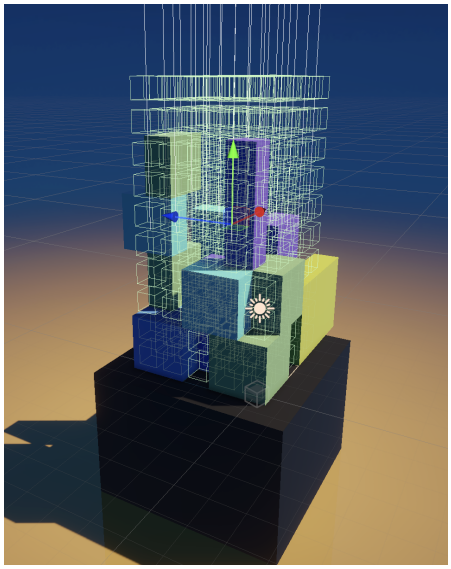
Creating demonstrations for the agent using VR controls

Regarding this thesis, the gameplay in *Imitation Ball* does not require one to do any designing, instead, the design of the gameplay and AI is a design problem in which RL played an essential role. What developing this game offers for this thesis is an insight into how creating agents for a virtual experience is a design process that can be supported by RL.

### 2.5.2 Stacky-Block Environment

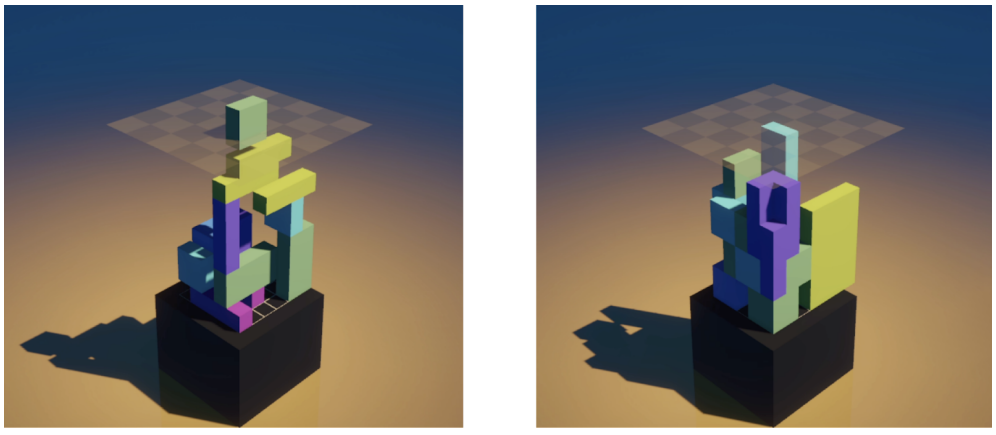
The *Stacky-Block Environment* is an environment built using Unity for exploring a design-orientated agent whose task involves observing the spatial qualities of a 3D space. *Stacky-Block Environment* poses the agent with a packing problem in the form of a puzzle. The agent has five different block types and is periodically required to drop the blocks on top of a grid. The objective of the agent is to fit as many blocks within the 3D space before reaching the height limit, similar to the game Tetris. The inspiration for this environment was to see whether

the agent can solve a design-like problem in 3D and remove the agent from being viewed as an entity but that which observes and acts upon the entire 3D space.



3D observed grid given the block positions

This exploration revealed that the agent in fact does improve and figures out a strategy to pack approximately twice as many blocks as a random agent within a 5 x 5 x 10 cubic grid. The observation space was the entire cubic grid plus the block to be dropped and is thus represented by a 251-length vector and the agents were rewarded for each block placed within the 3D grid space. This means in order to maximize rewards, the agent must learn how to fit as many blocks into the space before the stacked blocks reached the top, resetting the episode, the episode being the evaluation period of the agent in the environment.



Comparison of a random agent (left) versus a trained agent (right)

Though the initial test of this system was successful, this project has limitations needing further addressing. Firstly, the agent was trained with the same sequence of blocks, meaning that it fits that sequence and does not learn to generalize. Secondly, grid sizes were kept smaller to allow for a smaller observed space. Similar RL problems using agent-voxel interaction address this problem [38], and could potentially help inform future prototypes.

Though this thesis will not be looking to improve upon these technical drawbacks, this prototype offers insight into how we can train an RL agent capable of forming spatial design solutions. A lot of these implementations, particularly with the 3D observed grid, will inform how to approach the prototype developed in this thesis.

### 2.5.3 Building from the Prototypes

To contextualize the formulation of a novel co-creation prototype environment, the two preceding prototypes described provide a methodological and conceptional framework as a basis for its development. The previous prototypes abstract larger conceptual ideas into objectified tasks, which can be used as building blocks to address how to develop an environment that supports VR co-creation.

For this thesis, co-creation is being framed as a way to expand human-agent interaction within a design context. Fundamental is that the agent is not being presented as a tool or an assistive aid in a design environment, but rather as an agent with its own capabilities and sensibilities that might mirror a human user. We see the emergence of a design-capable agent made possible in *Stacky-Block*, where the agent is able to learn a sequence of block placements that begin to maximize the constraints of the given observed space.

Learning a sequence alone is not necessarily noteworthy, but what is worth noting is the expression of a design constraint for the agent to learn and become adaptable to. Setting a loose reward function, which simply rewards an agent for each block placed within the grid, but punishes the agent for exceeding the tower's height limit, implies a spatial objective that the agent presumably discovers a solution. Although we can view this particular RL agent as solving an optimization problem, where altering certain parameters or the reward function itself can train an agent to provide an objectively optimized solution, this is but one inferred outcome of

*Stacky-Block*'s exploration. For this thesis, we are most concerned with the basic idea that we can develop an agent that makes spatial design decisions reactively such that it can respond to changes in the design environment.

Though the *Stacky-Block* exploration does not go so far as to explore the dynamic potential of this agent, the co-creation prototype can by taking advantage of the affordances provided by RL. The agent is able to fulfill the design constraint by learning the actions to take given its current spatial observation. The trained agent thus can operate in the environment even if it is changing, and in a collaborative sense, changing due to the actions of another contributing actor, which could very well be a human user.

This is similar to the nature of the agents trained in *Imitation Ball* despite the environment being more competitive than collaborative. In *Imitation Ball*, the agent's observation of the environment is based on observing the ball and its own position so whether the ball is bouncing off a wall or being deflected back to the agent by a human player, the agent is not trained to behave any differently, it will simply make sure to bounce the ball back. When translated to a design problem similar to *Stacky Block*, a hypothetical RL agent will keep placing blocks given the trained design constraint by observing the current state of the environment. A human actor involved in that design problem adding or removing blocks would hence not prevent the agent from continuing to try fulfilling its design objective.

Functionally, these preceding prototypes exemplify how RL can develop an agent a user can co-create with, but furthermore, these prototypes have also begun posing questions on how to evaluate and design the desired agents for human-agent interaction. When developing the agents for *Imitation Ball*, once the basic agent was trained such that you could rally with it, the next question was how one goes about creating agents with distinct playstyle characteristics. Do we make adjustments to the reward function to reward behaviors atop general gameplay? Can we demonstrate to the agent playstyles for it to imitate? *Imitation Ball* demonstrated ways to address both questions to create agents we are then able to discern qualitatively.

This thesis seeks to build upon the developmental process of designing agents from *Imitation Ball*, where the user, developer, and designer are engaging in a variety of different observations in order to train and evaluate an agent suited for co-creation. Firstly, there is an

observation being made on the environment and the design task that informs the reward function to train the agent with. Secondly, there is then the observation of the agent itself as a result of different factors: quantitative in accordance with training convergence, and qualitative regarding the given task. And henceforth, what the thesis prototype aims to explore is the qualitative methodologies for judging the agent to locate the significance of facilitating human-agent co-creation.

### 2.5.4 The Design Game

As discussed in previous Background sections and prototypes, current research regarding RL and autonomous agents actively use video games and competitive rule-based scenarios to train and test modern algorithms. These games are well-suited for these types of tasks because they are defined by a clear objective by which an agent can quantitatively optimize, whether it is attaining the highest score or reaching an objective in the fastest time. But as previously discussed, open-ended environments like *Minecraft* do have potentially creative outcomes but in the case of *MineRL* [21], restrict the agent's evaluation to an explicit objective. Desiring agents trained with open-ended outcomes, we must reevaluate the functionality of games for exploring agent design.

We can start framing the way we develop more open-ended problems through the ideas explored in *Concept Design Games* [22] by Habraken and Gross to help answer the question on qualitative methodologies for judging agents. Within the context of designing the built environment, the authors propose abstracting the design process within the context of a board game environment in order to highlight the interaction between actors who negotiate and evolve their own objectives throughout the course of the design game. For this thesis, we can think of these two actors as the agent and the human user, and instead of creating a physical artifact, we can imagine the creation of a digital one.

The key principle of *Concept Design Games* is that design includes many participating actors with different responsibilities and expertise. As a result, essential to designing is mediating these differences through rule-making. Rules allow for limiting moves within a design context, for instance, the rules for moving checker pieces on a checkered board, but within these rules lies enough flexibility for arrangements to emerge that express a negotiation between the

actors at play. In the context of RL, these actors are the agent and the human user developing the agent and potentially participating in the game with the agent.

It is clear that rules related to RL can suggest explicitly a reward function and the action space programmed into the agent. These are necessarily explicit given the training process in RL, but considering how to apply these agents, a game suggested in *Concept Design Games* can allow for more implicit means of evaluating the agents in a participatory design setting. In the author's second book on the topic, they go into detail about more specific games and their conceptual outcome. One such game is the *Silent Game*, a game of two actors, one which makes a pattern within the space of the design game, the other trying to guess and complement it with their own moves.

Habraken and Gross's reflections are a valuable insight into the outcomes of this game, revealing the relationship the design artifact have between participants. The outcome of the artifact revealed intent to achieve patterns that are either material or spatial, where material indicates the relationship between piece types and spatial configuration of pieces. Their conclusion on the participant interaction in *The Silent Game* is as follows:

> The game has no collective program, it has only individual goals. Since all parties must jointly make only one configuration on a same site, and nobody can fully control the final result, players must adjust their original individual programs in the course of play by taking other players' intentions into consideration. Very likely, different players have conflicting intentions of form making. Players must cooperate and try to understand each other's intentions. Cooperation is more plausible than competition, and players learn that unless they can understand and respect other players' intentions, and unless all players share certain rules as common ground, their efforts in pursuing individual goals will not be successful.

A helpful conclusion from their reflection is to consider what the *Silent Game* can inform us if playing with an AI agent. Because these games are designed to be played with two humans, observing 'intentions', and 'individual goals' must be described more explicitly in the context of AI. But how explicitly is less clear, and developing a similar situation found in the *Silent Game* for an RL agent can help answer questions about how to develop an agent for implicit tasks like design and interaction.

### 2.5.5 Objectives for Co-Creation

Co-creation in design is open-ended in the possible modes of designing and collaborating. Critical for this thesis and its prototype is to frame the work as being openly suggestive of different design dynamics despite the specificity of the task being presented. This is largely different than a typical objective game-like scenario as seen in *Imitation Ball* or even in many popular benchmarks for RL algorithms like the Atari games. That being said, in order to define the design task, it is necessary to frame it in an objectified game-like way in order for the agent to learn.

There are a number of papers within the field of human-computer interaction that have begun manifesting the field of creative co-creation between humans and AI [27][55]. Currently, the definition of co-creation is seemingly malleable in the context of AI and has been generally applied to any creative task that an AI can help assist a human with and is largely concerned with facilitating human-computer interaction. This thesis is less concerned with defining a conceptual model of co-creation and rather simplifies co-creation as a link between human and agent collaborators on an open-ended task. Work at MIT Media Lab most closely defines co-creation this way by designing games for child-robot co-play [3].

The prototype must address a way to balance explicit game objectives with allowance for a novel design to take shape through collaborative interaction. This is functionally why the block environment seen in *Stacky-Block* is useful in that it facilitates design decisions while narrowing the large potentiality of a collaborative design task. That being said, it is important to acknowledge that this type of voxelized design is far more restricted in the potential outcome and resolution of models created within CAD software. Though this thesis can only speculate about the potential human-agent interaction that can occur in a continuous 3D design space, a discretized action and design space does two important things: take advantage of current accessible RL approaches and facilitate an experience more focused on agent interaction in VR.

The premise of agent interaction in VR is distinct from other modes of co-creation, whether it is human-human or human-AI. Firstly, the immersive nature of VR changes how one perceives space and hence operates. Secondly, the human user occupies an avatar controller that is more likened to how a potential AI agent might operate. This potential with VR to bring user

and agent into a similar control schema creates a novel form of co-creation where the user can demonstrate and craft a particular behaving agent from both a first and third-person evaluation. As a result, it is important to implement in this prototype a design interaction in VR that bridges the gap between human and AI co-creator.

# 3. Hypothesis

Given the Background, how and to what extent will building a prototype begins to address these questions? The Background set out to establish distinct fields of interest to this thesis, and where overlaps between said fields are beginning to emerge. It is this overlap of fields and techniques this thesis is most interested in exploring in the form of building a prototype that contextualizes these fields into one conceptual design exercise. From this motivation, this section makes clear how the given broader technologies regarding VR, RL, and data interfaces might become woven into developing virtual design environments.

## 3.1 The Design Game

Following the format of Habraken and Gross' Concept Design Game [22], the Silent Game, I speculate a more open-ended game to be played with a hypothetical RL agent. The objective behind this game can be broken down into two more general objectives. The first is to facilitate a design environment where a human and an agent coordinate implicitly by each placing pieces on a site to fulfill a common design target. This allows us, as users, to reflect on the impact of our decisions in the design space when confronted with working together with an AI collaborator in VR. The second is to create a game where a dimension of behaviors can be observed to support an investigation into developing qualitative agent behaviors with RL. This can predominantly be understood as behaviors regarding the choice of design actions and behaviors regarding executing the design actions.

Habraken and Gross' design game had a clear objective in its attempt to abstract the communicative aspect of design interaction. To address broader topics and needs, consequently, I propose a version of a Concept Design Game that focuses less on the game itself but rather on the context it provides. The reason is a conscious decision to scope this thesis as laying a foundation for exploring human-agent interaction within design. The thesis for this reason tries not to engage directly with a specific design question and rather propose methods for potential novel design paradigms. Consider the following *Design Co-Creation Game* as a template for further design exploration with AI agents, but a template that heavily influences the technological developments of this thesis.

# DESIGN CO-CREATION GAME

**Summary**

The Design Co-Creation Game is about exploring human-AI interaction through the medium of a 3D interactive block matrix. Both human and AI are given prior experience with a target 3D model that they are trying to reproduce in block form. On this site, both human and agent players have the ability to place blocks according to their knowledge and desired outcomes for the final 3D model. There are no turns and no victory conditions. The player can toggle the presence of the AI and toggle it off. The model develops through the control of the player, deciding when they choose to co-create with the agent. Communication remains implicit throughout the game.

## Game Rules

**1. Technical Universe**

Selection: The pieces in this game are not predetermined for the human player, but the agent is trained with a stylistic selection. Human player selection rules are made implicitly during play.

Distribution: The agent places pieces in relation to the objective of the model while the human player's technical rules for placing blocks are made during play.

**2. Site**

A VR environment portraying a room with an empty 3D matrix for where block pieces are to be located and populated.

**3. Program**

There is no explicit program.

**4. Roles**

This game has two roles: both have the role of a creator and are hence co-creators, but the human player also has the role of agent controller.

**5. Moves**

There are five kinds of moves that can be made:

- Remove a block from the site

- Place a white block in the site
- Place a red block in the site
- Place a green block in the site
- Place a blue block in the site

**6. Goals**

The goal is to reconstruct the objective 3D model, but this goal is explicit for the agent and suggested to the human player.

**7. Protocol**

- The agent observes the spatial configuration of the site and not its block colors.
- The human player can erase away all the blocks on the site at any time.
- The game ends when the human player decides to end.

There is a purposeful lack of constraint, as expressed by the lack of an explicit program for this game, where in contrast to the *Silent Game*, the program was to speculate and continue your collaborator's secret pattern. With such a format, what we might expect is less insight into evaluating the output design, but instead an insight into the behaviors of how human and agent negotiate their bodies in VR and affect each other's decisions. As a result, it is less important to condition the agent with a reward function defined by a design objective similar to that in *Stacky-Block*. Instead, we just need an RL agent to learn to the best of its ability the target 3D model so that we can observe how it affects the way it designs in developing the agent.

## 3.2 Implementation Expectations

The expectation for this prototype is that it can begin a conversation on theoretical frameworks and applications that might suggest the implication of human-agent interaction and development. It is not expected that the implemented methodology is optimal but at the very least highlights the capabilities of these technologies such that future work can expand on these ideas.

The most important expectation for this prototype is that it provides insight into ways to evaluate agents through different perspectives, whether through traditional ML inference or through qualitative frameworks in VR. In order to achieve meaningful evaluation, also means a

level of development to be expected with RL and IL implementation used to implicitly develop these agents. As a result, we also get insight into this development process and the usefulness and viability of VR and/or IL as tools for augmenting agent development, which we already can begin to see in precious prototypes.

From developing *Imitation Ball*, my expectation is that we will see a clear variation in agents as a result of different explicit styles of demonstration in VR. Furthermore, observing these agents in VR and interacting with them opens up a more subjective evaluation of these agents rather than an evaluation based purely on how well the agents learned a task. By framing the task more openly under the context of design co-creation, we are also less interested in questions of an explicit task but rather how the agent goes about the task and how the agent's operation within the design space affects our own design decisions as well.

Lastly, by integrating data visualization techniques similarly seen in *iViz*[13] and *A Walk in Latent City*, I expect these data spaces to stimulate a conversation on how we can begin to create static representations for evaluating dynamic agents for qualitative applications like design and interaction. What should emerge is that these spaces can allow us to compare different agents against each other, similar to the comparison made against human and agent actions while playing *Space Invaders* [34]. These visualizations should suggest future metrics for similar design processes involving RL, where ideally, a realized evaluation metric can provide feedback to designers and developers on ways they can change the training of their agents.

# 4. Methods

The thesis's prototype builds atop previous ideas explored in the two prototypes *Imitation Ball* and *Stacky-Block*, borrowing from their methods to support the development of a prototype exploring human-agent co-creation. The resulting prototype is a self-contained VR application that presents a collaborative design task between the user and agent. Furthermore, the process of developing the VR application, including agent design, user interaction, and data interpretation, is presented to speculate on future methods for exploring human-agent co-creation.

## 4.1 Prototype Overview

From motivation to its final design, the prototype for this thesis is laid out to help describe how these different technological interests become contextualized by a human-agent design game. The objective of this overview is to outline all of the parts of the VR experience and how they are connected to support both human-agent co-creation and agent training.

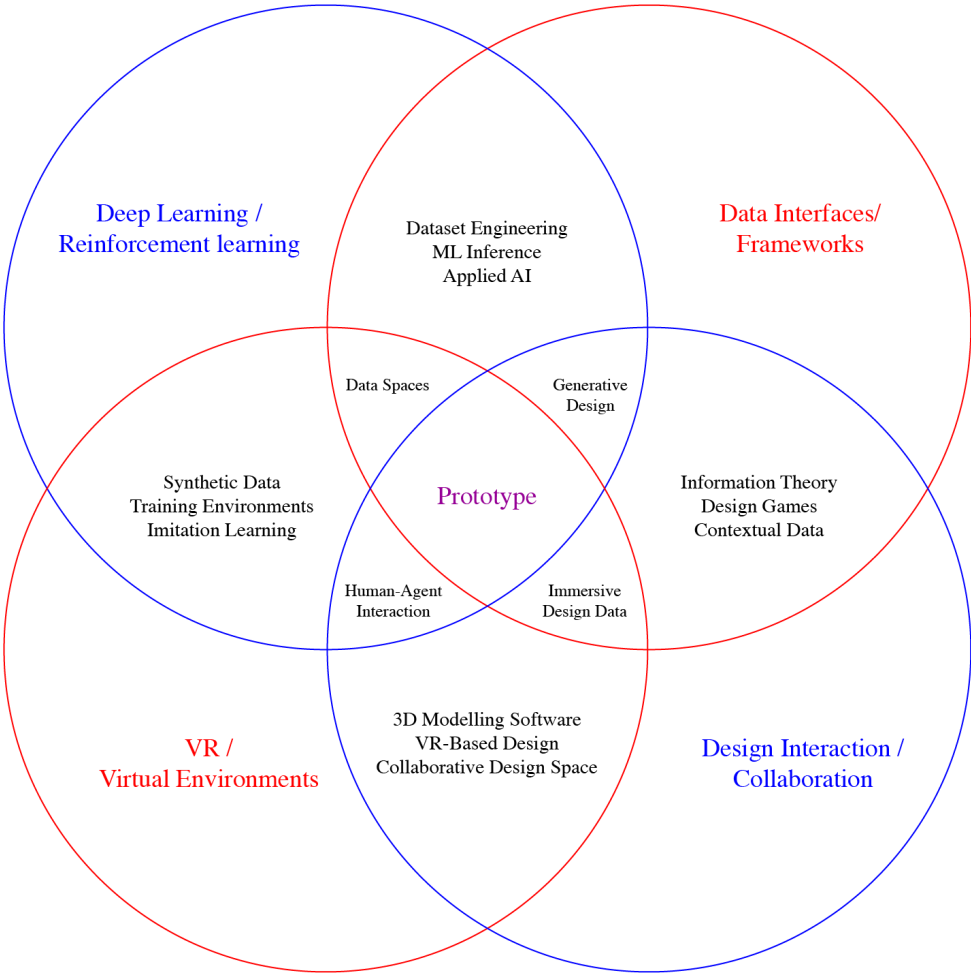
### 4.1.1 The Speculative Prototype

To frame the goals of the prototype given the material presented in the Background section, the following diagram aids in describing the key topics being considered and how their relationship with one supports a novel context for this thesis. To better understand the diagram, the largest ideas are categorized and individualized at a higher level based on the following distinction of being an *Action* or a *Platform*:

| **Action** | **Platform** |
|---|---|
| Deep Learning / Reinforcement Learning | Data Interfaces & Frameworks |
| Design Interaction & Collaboration | Virtual Reality & Environments |

This distinction claims that an *Action* needs a *Platform* for which it is used and applied. Hence, specific methods and works always lie at the intersection of an *Action* and a *Platform*, and two *Action*s are related based on a common *Platform*. This sets up a logical framework for defining

the relationships of methods relevant to this thesis. It also helps highlight how the thesis is taking a novel stance at the confluence of different design and technical methodologies

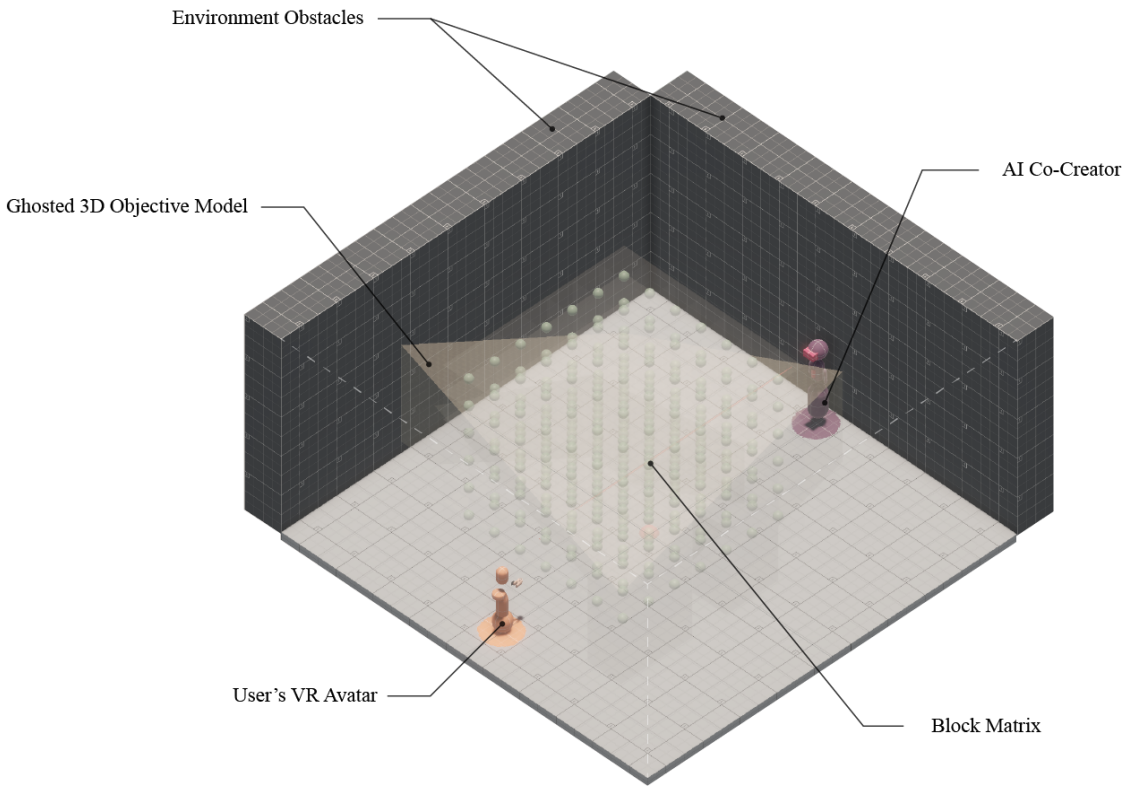



A diagram framing the relationship between relevant technologies addressed by this thesis

By placing the prototype at this intersection, the goal is to project outcomes for future human-AI interactions through a design interaction lens. That being said, design interaction is a loose term due to the large span of fields ‘design’ qualifies as. As a result, the thesis is more concerned with how design can help investigate novel dynamics of human-AI interaction through the nuanced nature of design interaction and collaboration. Design is a conversation of evaluating objectively or subjectively, or rather in the context of AI, explicitly or implicitly.

### 4.1.2 Environment

The core VR environment is a room where the user and AI agent can move within and interact with a centered 3D grid, titled the Block Matrix, that supports the collaborative design exercise. The design exercise is a block reconstruction of a predetermined 3D model for the user and trained agent to collaborate on, supported by placing blocks within the grid system. This environment is at its core what supports RL training, human-agent interaction, and the final VR application.




Overview of the human-AI co-creation environment

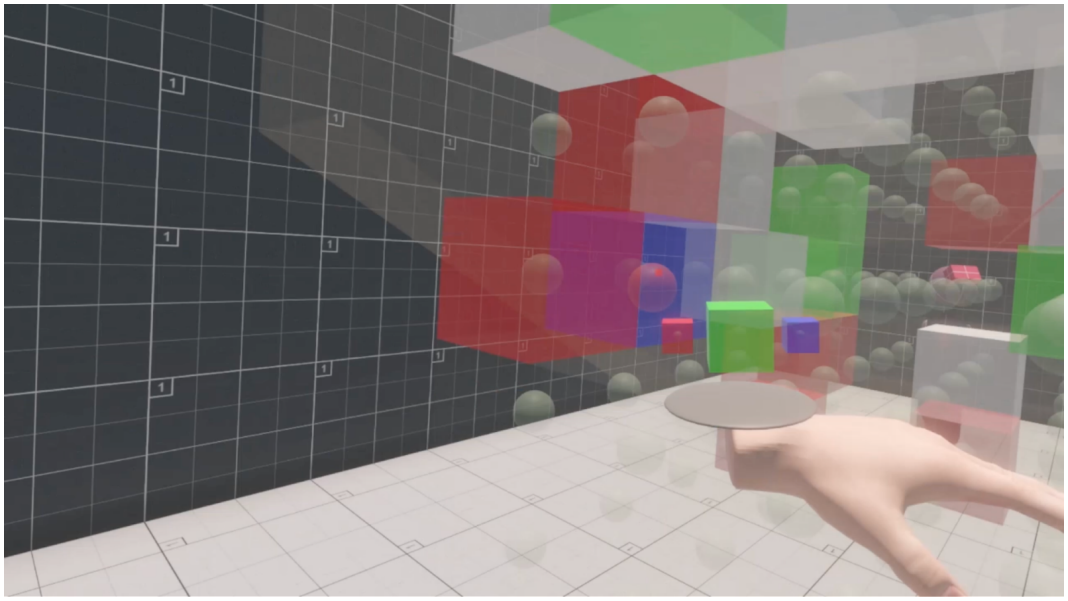
The Block Matrix is intended to support the design of a pavilion-like structure within the room environment. One block unit in the grid system is intended to be at a scale that balances being large enough to be perceived spatially at room scale while also being fine enough to provide resolution in capturing the source 3D model. This approach to design is notably distinct from more third-person approaches as seen in most 3D modeling software. Adopting a more perceptual first-person approach to design takes advantage of VR's immersive spatial integration into the virtual environment.

On initialization, the grid system is comprised of a 3D array of empty slots that can be filled in by the user and agent in order to perform the 3D reconstruction. At the center of the empty cubes is a spherical toggle that when cursored over, the agent and user can select a block input to place a block type at that spot. The user and the agent can move freely within and around this grid system, altering the design of the current 3D model.

### 4.1.3 User Control

In the VR environment, the user takes over a virtual avatar with the capabilities of moving in the space and placing or removing blocks within the 3D grid system. The experience of performing these actions takes advantage of the ability to perceive spatially with the VR headset.

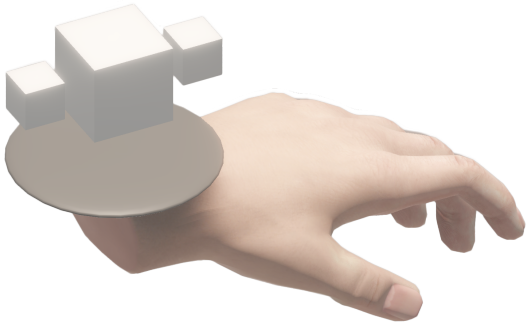
Unlike traditional VR applications that make use of hand controls to make selections, similar to using a laser pointer, the prototype instead uses the headset as a way to cursor through the toggles within the 3D grid. The center of the user's gaze in the VR headset is able to cursor over the current block selection and with the hand controls, the user decides when and which block type to place.



User view cursoring over a with their gaze and deciding which color to assign it

With the VR hand controllers users have on their left hand a Palette Watch, where they can cycle through their current block type. If cursoring over a target block, the user can set that

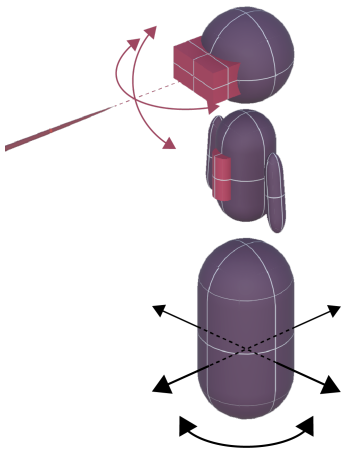
block's type to that currently selected on the Palette Watch. Another control of the user is buttons that reset the Block Matrix and agent, as well as pause the agent.



The Palette Watch design, indicating previous, current, and subsequent Block Types

### 4.1.4 Agent Design

The agent is designed based on the user's VR controls and avatar. The user controls in VR are based on controller input to move and rotate the body, and head rotation is determined by the orientation of the headset. The agent has five degrees of freedom modeled after the VR user's avatar: left/right, forward/back, body rotation left/right, head rotation left/right, and head rotation up/down. By making both user control and agent control similar, it allows for both actors to operate within a similar action space. This means that it is possible to both compare agent and user actions given the same observation and allow the agent to learn from a VR user demonstration.

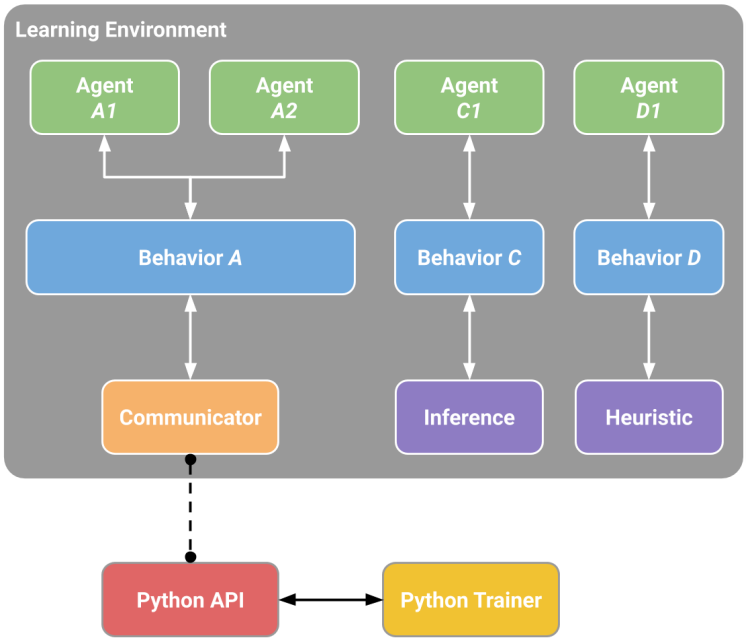


The design of the agent with its five degrees of freedom

The agent itself is trained to do, in a functional sense, what a VR user is capable of. The agent chooses what blocks to place, and is able to move and rotate its gaze to place a block and then continues to place blocks given an observation of the Block Matrix. Because the agent is trained with RL and potentially IL, the behavior that emerges from the agent to do these actions is directly based on how it is trained. Similar to *Imitation Ball*, different agent behaviors can emerge and have a large impact on the experience of co-creation. A key part of designing this agent is designing it to be adaptable and flexible to different training parameters and environments.

### 4.1.5 RL Framework

Training is done outside of the Unity environment via Unity's ML-Agents toolkit, which allows for the training environment designed in Unity to communicate externally with ML-Agent's PyTorch RL implementations. Through communicating with the Unity environment, ML-Agents is able to train a neural network to maximize the rewards of an agent and save locally the neural network. The neural network is then brought into the Unity project and associated with the corresponding agent to be integrated into the application's final build.




Example ML-Agents framework from Unity's documentation [24]

Creating a Unity environment ready for RL training with ML-Agents is centered around developing the agent whose actions and observations can be translated into each their own vector. Three classes are coupled together when defining an agent: Behavior Parameters, Agent, and Decision Requester. Behavior Parameters are parameterized within the Unity editor requiring explicit information about the size of the observation vector, the action vector, and reference to an ONNX file[7], which represents a trained neural network. The Decision Requester sets the frequency in which an action is requested from an agent per step. There also is a toggle for the agent to keep taking that same action within the same period of steps. The Agent class contains the bulk of the Agent design and is meant to serve as a base class to be inherited by your specific agent's class.

There are three core function overrides that need to be considered when writing a new agent class, they are OnEpisodeBegin, OnActionRecieved, and CollectObservations. OnEpisodeBegin is called at the beginning of a new episode, which occurs after a predetermined number of steps or a specified condition is achieved. Generally, actions like resetting the environment and agent to an initial default state occur here. OnActionReceived determines what happens within the Unity environment when given an action vector. This means determining how an index in the action vector affects the scene and/or agent. CollectObservation determines how the observation vector is constructed. This means drawing from the objects within the Unity environment to get information to feed as input into the neural network during training.

In RL we often describe a reward function that scores the agent per training step given its actions in the environment. ML-Agents does not require an explicit mathematical function in this sense, rather, a function call to add a reward given an event is all that is required. For example, if we want the agent to stay away from the outer boundaries, detecting a collision with the boundaries will call a function to give the agent a negative reward. The reward from this event is then factored into the reward calculation for that current training step.

The last key part of training with ML-Agents is configuring each training run. Training is configured from a high-level YAML[8] file that provides all the necessary information for training

[7] Open Neural Network Exchange is an open format for describing neural networks used between different ML frameworks
[8] Yet another markup language is a text file format that prioritizes making the program humanly legible

the neural network. The main sections include the type of RL algorithm, hyperparameters, network settings, and reward signals.

The RL algorithm in this thesis is parameterized with PPO [47], which is in the family of policy gradient RL methods. The reason why PPO works well in this context is due to its higher performance trade-off for less sample efficiency when compared to Q-learning-based RL methods. Because training is done in a simulated environment, sample efficiency is less of an issue.

Determining hyperparameters and network settings is relatively straightforward in the sense that settings can be parameterized from similar RL problems. On the other hand, there are a number of different reward signals that can augment training, potentially affecting agent convergence. By default, RL is concerned with the agent learning from gaining extrinsic rewards from its environment. Alternatively, we can also include intrinsic rewards for our agent, which include using demonstrations with GAIL and BC, as well as incentives for exploration with Curiosity[9].

The ML-Agents toolkit abstracts much of the algorithmic complexity behind RL, allowing the focus of RL to be on creating the environment and formulating the agent's problem. Once networks are trained, they act as individual assets that can be assigned to the agent, allowing for quick testing of agent variants.

### 4.1.6 Application Architecture

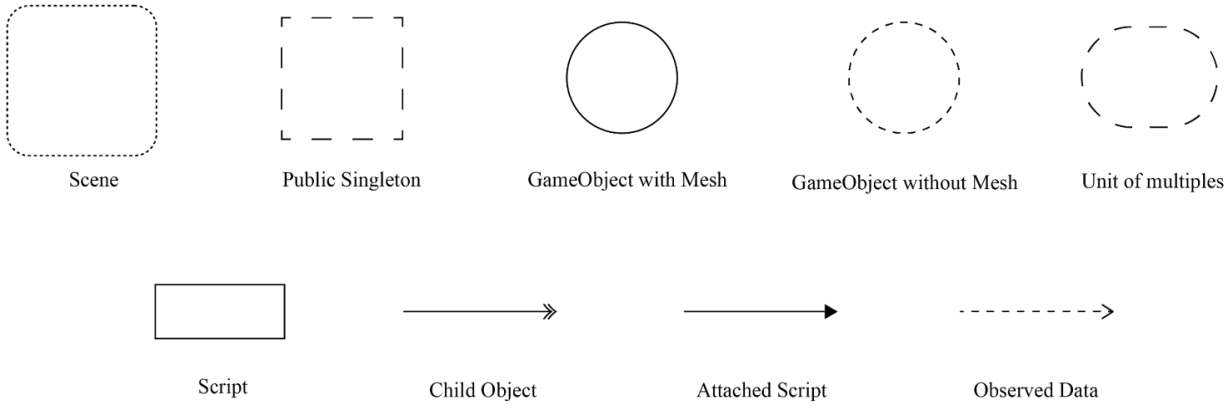
To outline the prototype of this thesis, the following are application architecture diagrams that explain the relationship between scene objects and custom scripts written to provide the functionality for human-agent co-creation. Though not all of these terminologies will make sense right now, the relationship between scripts will be better explained in the following sections. For now, this is an abstract mapping of how the scenes are constructed within Unity and will be a helpful reference for later sections.

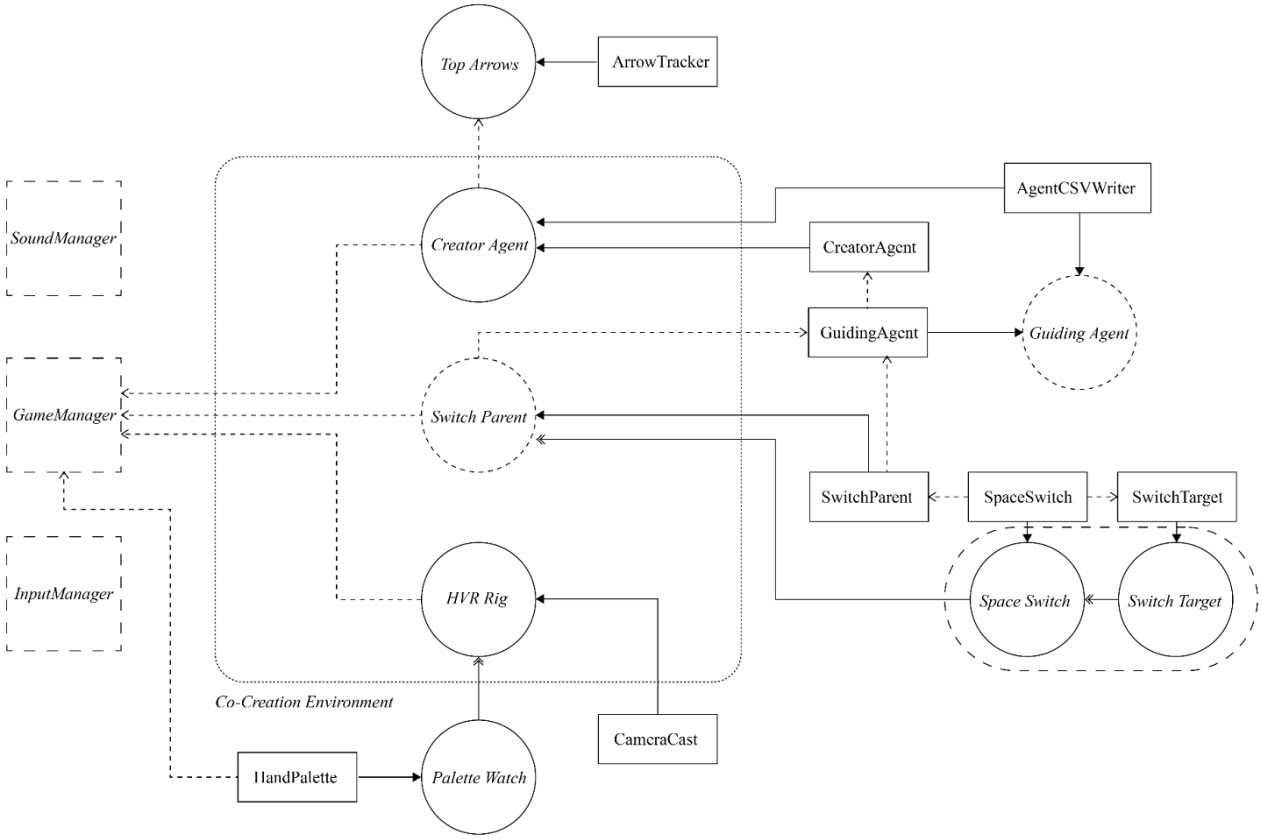
[9] A method in RL implementing a second neural network that helps encourage the agent to perform actions with unexpected outcomes

There are some Unity concepts that can be clarified to help understand these diagrams. Firstly, a *scene* is an environment that contains all of the assets and entities that are contextualized and interact with one another. For this thesis, there is the main scene for co-creation, two versions of that scene for agent training, and then lastly a separate scene for evaluating agents through their latent action embeddings. A *GameObject* is an empty object in Unity, which by default has a Transform, the object's pose, and can have attachable scripts to it, such as the custom ones written for this prototype, 3D meshes, and Unity generic components like colliders. There are also public singletons called Managers that contain publicly accessible functions and references for the rest of the scripts in the scene. Each script has its own sole class inherited from Unity's default MonoBehavior class, and there are no other instances of inheritance used in this prototype. Also within the diagrams, GameObjects inside the scene means that they are spatially located within it in the editor, otherwise, they are instantiated dynamically during runtime.

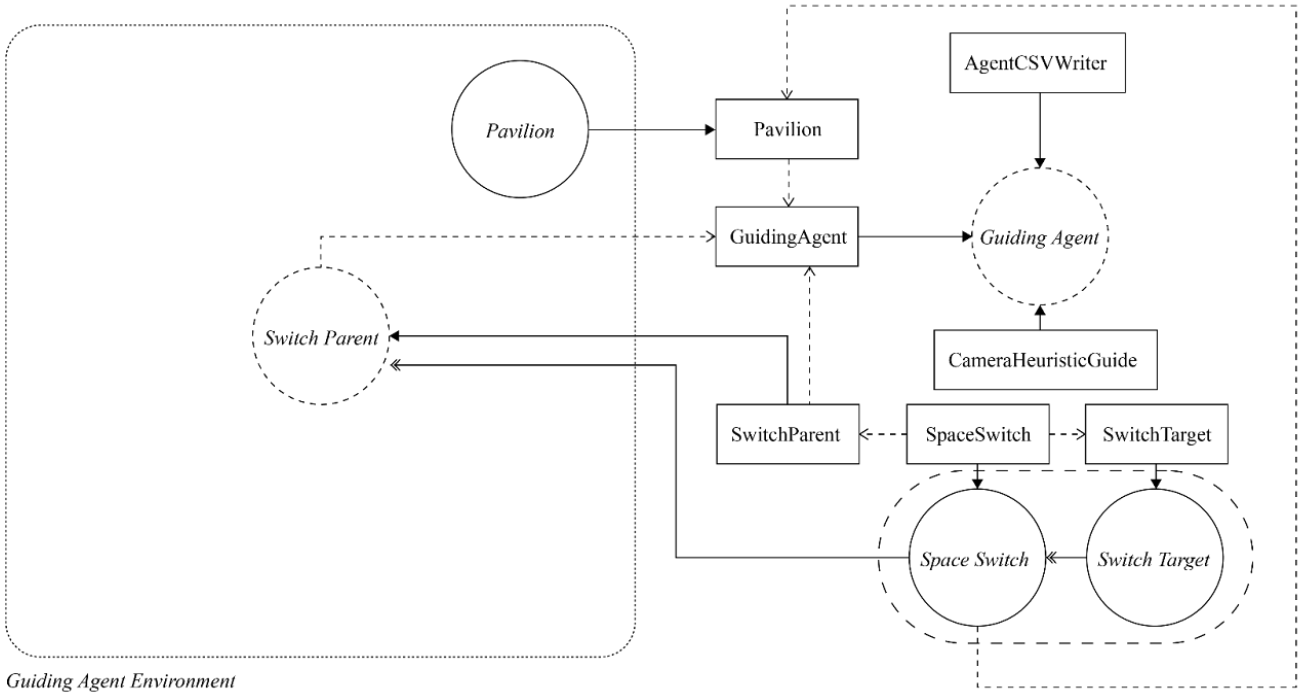
For a glossary of scripts and a brief description, it is recommended to visit *Appendix A*. It will aid in understanding the architectural diagrams and references made throughout the Methods section.

*Diagram Legend*

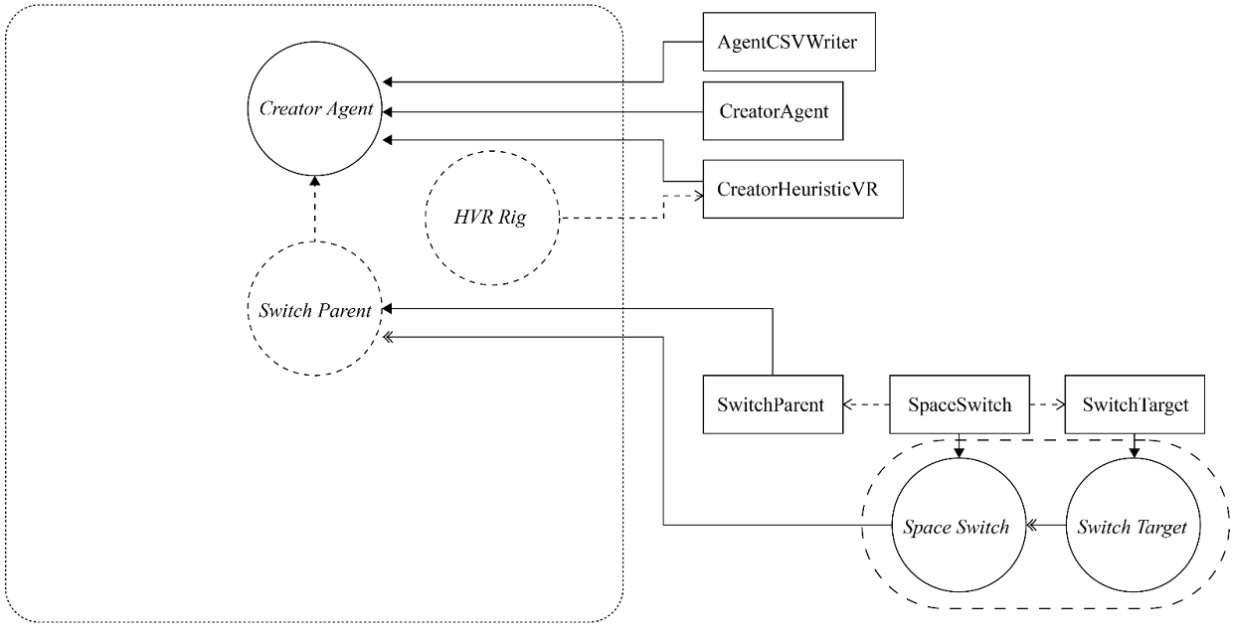




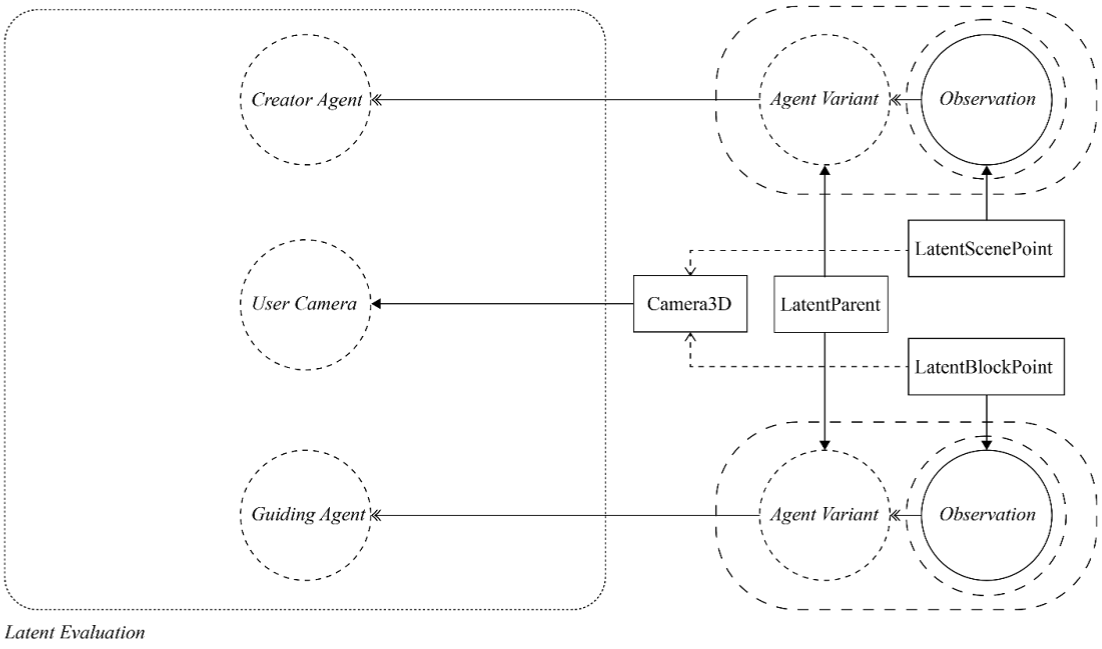

Architecture of the Co-Creation scene



Architecture of the Guiding Agent training scene




Architecture of the Creator Agent training scene



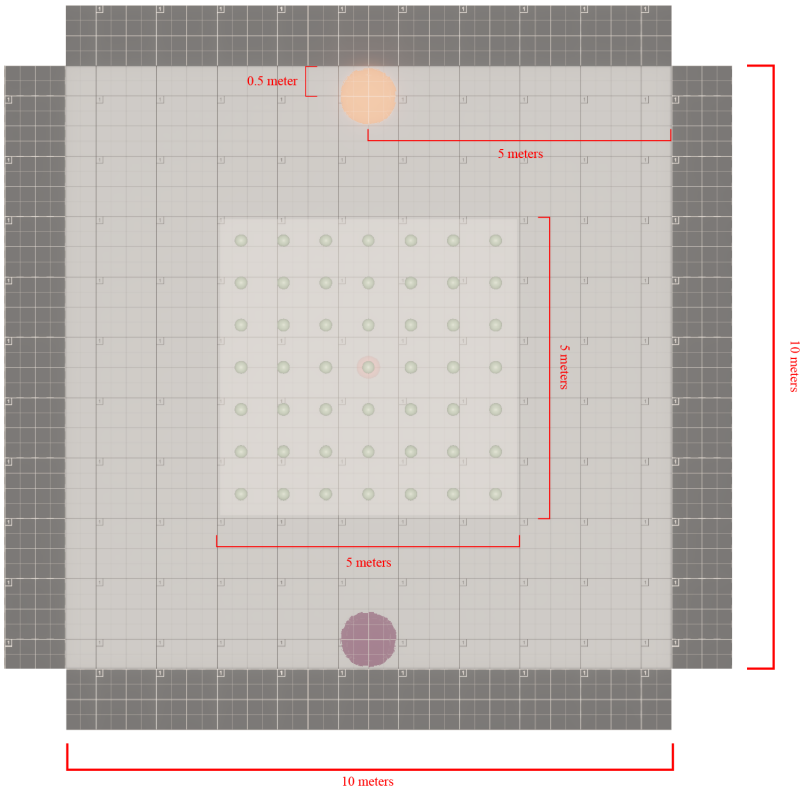

Architecture of the scene for latent evaluation

## 4.2 Environment Design

An overview is provided of the 3D elements that spatially define the co-creation and training environment. These elements are all implemented with consideration of how the agents are able to interact and observe their features as they are essential to providing the spatial framework for the design interaction aspect of this thesis.

### 4.2.1 Spatial Features and Variants

The core design environment is defined by simple 3D features: a floor, four surrounding walls, two spawn locations for the user and the agent, and the middle empty to allow space for the Block Matrix to be instantiated at runtime. The dimensions of the floor area are 10 x 10 meters and the wall heights are 5 meters tall, though the walls there are meant to act as boundaries for the user and agent to prevent them from leaving the space. The walls and floors are set as simple box colliders.

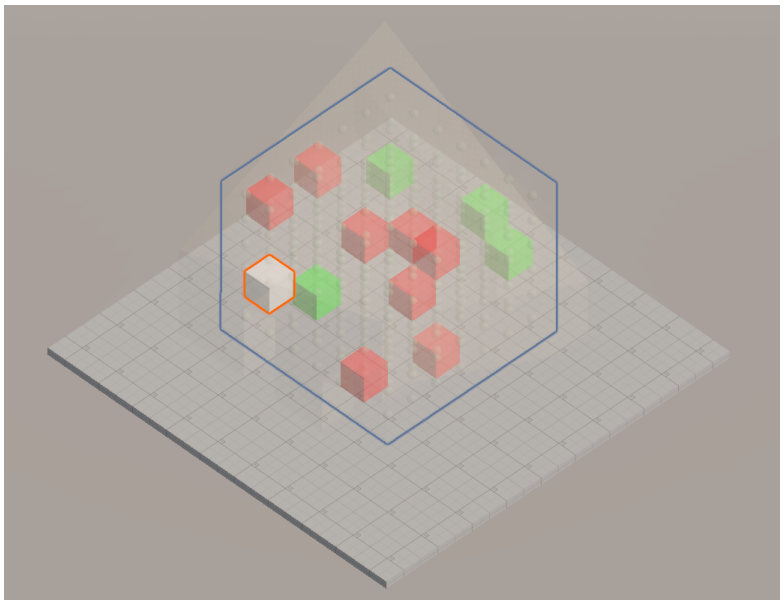



Measured floor plan of the default design space

For agent training, the environment is meant to be exactly the same as the core design environment, the only difference being that it does not account for there being a player spawn. Too, the walls are assigned an '*Obstacle*' tag within the Unity editor. Doing so allows for unique identification for the agent to know that it is looking or colliding with the boundary. This also can be used for calculating agent rewards, for instance, providing a negative reward to disincentivize the agent from colliding with the walls.

### 4.2.2 Block Matrix

The implementation for the Block Matrix is comprised of a parent script that constructs the 3D matrix upon runtime and the blocks within the Block Matrix itself. These empty blocks within the Block Matrix are known as a Space Switch because they can be thought of as something static but switched on to different states or in this case block types. Within each Space Switch is a Switch Target, which is used for the user and the agent to cursor over and toggle the state of the corresponding Space Switch.



An initialized Block Matrix made up of units of Space Switches

**Switch Parent**

The Switch Parent class is used for instantiating the Block Matrix, setting the scale of the Block Matrix, and keeping track of the state of the Block Matrix for agent observation. The Switch

Parent's dimensions are parameterized within the Unity editor and can be scaled via its local transform. In this current prototype, the dimension of the Block Matrix is set to 7 x 6 x 7 and the scale to 0.7, meaning that each unit within the matrix is a block 70 centimeters in each dimension.

Upon runtime of the environment, the Switch Parent sequentially instantiates the individual Space Switches as its child, while simultaneously building a list of integers that keeps track of the switch states. Each switch upon instantiation is assigned a unique index that corresponds to a location in the switch states list. Switch Parent also builds a second more simplified list that tracks a boolean value per switch, indicating whether that switch has been activated at least once before. Having these two lists provides options for different observation types when training agent variants. Lastly, the Switch Parent is also responsible for resetting the switches back to their default state as well as the two lists during an episode reset during training or during the co-creation process if the user desires to start the design over again.

**Space Switch**

A Space Switch is a prefab that represents an individual unit within the Block Matrix and is the object that the Switch Parent references when constructing the matrix. The main role of the Space Switch class is to control the different block types that the space switch can be toggled to as well as update the Switch Parent's list of switch states. Each Space Switch is stored with its own unique index value that matches a corresponding index in the Switch Parent's state lists. Within the Unity editor, the Space Switch class can be configured in three different ways: its initial state, which by default is None, an empty block, the switch's possible colors, and its transparencies.

Defining the block colors is an enum called SwitchState, which in this implementation is comprised of five different elements. The five elements are, in order, None, White, Red, Green, and Blue, the names corresponding to the possible colors of the blocks. The enum type is useful in this scenario because each element can also be represented by an integer type. This means that when updating a Space Switch by indicating a new SwitchState, that enum element can be passed directly to the Switch Parent's list to be updated as an integer.

The model of the Space Switch prefab uses Unity's default cube mesh with no changes to its default transform. This means that by default a Space Switch is 1 meter in each dimension. Centered within the Space Switch is a child Switch Target prefab, which handles toggling of the Space Switch's current state. The need to distinguish between the switch and a target is to allow for the targeting of a specific Space Switch by the gaze of the user or the agent. The Space Switch also has a box collider attached to it, set as a Trigger. This is to accompany the implementation for defining the objective 3D model.

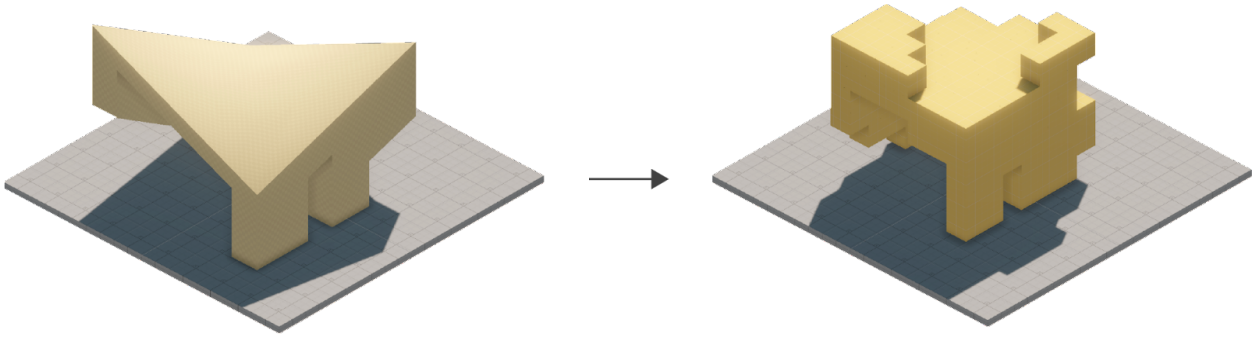
**Switch Target**

The Switch Target is a spherical mesh and collider, 20 centimeters in diameter, residing within the Space Switch. The Switch Target class's role is to indicate and track whether its parent is the current targetted Space Switch. If the gaze of an actor collides with that Switch Target, the Switch Target has functions for changing its color and updating the Game Manager with its parent as the current Space Switch. When not targeted, the Switch Target will revert the current Space Switch to a null type.

### 4.2.3 Objective 3D Model

The objective 3D model is the target 3D model that the agent is trained to reproduce. Training the agent to learn a spatial configuration requires importing a 3D model into the Unity editor and attaching the Pavilion class to the imported object. The main role of the Pavilion class is to define the given objective 3D model within the context of the Block Matrix. The given 3D model is placed into the training environment prior to the Block Matrix initialization.

On runtime, when the Space Switches are procedurally generated, if their box colliders intersect with the mesh collider of the objective 3D model, the Pavilion class stores in a list of indices the index of the collided Space Switch. After the Block Matrix is initialized, the Pavilion class has the indices of every Space Switch that it collides with, which can be thought of as a list that describes the 3D model's voxelized form. This list is referenced by the agent during training indicating whether its block selection is considered a part of the objective 3D model or not.



7 x 6 x 7  Block Matrix voxelization of the objective 3D model

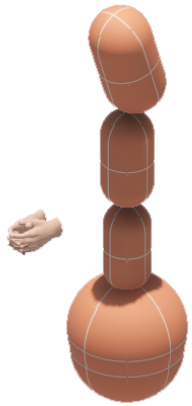
For developing the agents and their variants in this thesis, a custom pavilion was modeled in Rhino 3D and imported as a mesh into the Unity environment to serve as an example objective 3D model. The motivation behind this particular design was to formulate an object that provided spatial variety given the constraints of the Block Matrix, as well as a complex geometric feature in the roof form to communicate that the input 3D model does not need to be orthogonal despite becoming voxelized. The model was used for developmental purposes and is a baseline for all tests in this thesis, but only suggests one potential objective 3D model, which otherwise could take on an alternate form. It should be acknowledged that different geometries might benefit from alternate optimization of rewards and parameters.

## 4.3 User VR Controls

All VR work is being built for testing on the Meta Quest 2. With Unity, developing a target VR platform is made very simple given native packages that can be easily integrated including the Oculus XR Plugin. The majority of VR controls and frameworks are similar and hence development for one type of headset is easily translatable to another.

### 4.3.1 VR Framework

To manage VR input and controls, the prototype uses externally developed packages Hurricane VR and Hexabody VR as a comprehensive solution for creating VR interactions and handling controller inputs. Both packages are developed together and include a prefabricated VR rig that can be simply dropped into the Unity scene.

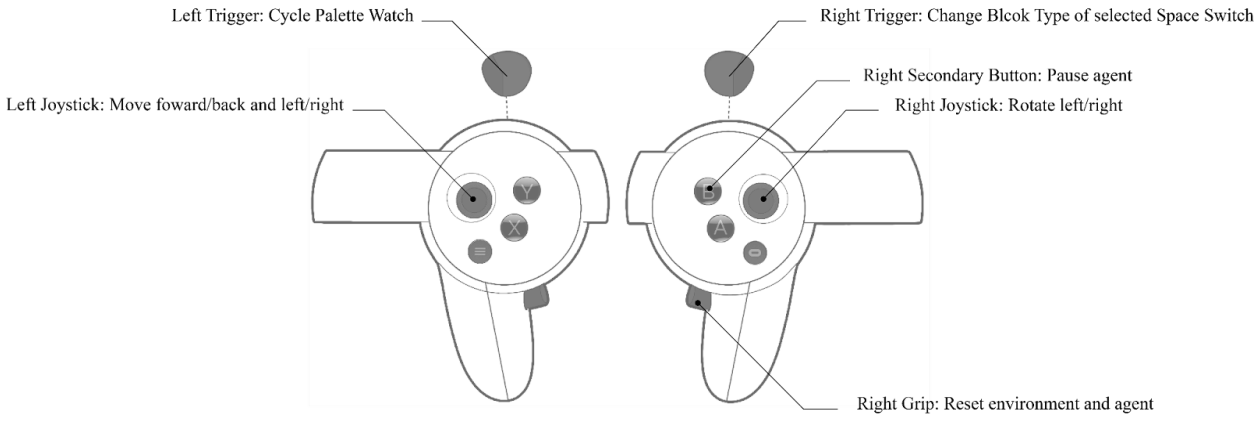


Default Hurricane VR player controller

Customizing VR controls within the Hurricane VR framework is very straightforward due to the majority of their scripts having their parameters serialized within the Unity editor. This also means that the framework can allow for a high level of customization without requiring any additional code as well as be integrated with your own code base.

An example of integrating with the Hurricane VR framework is through programming the VR control inputs. By having a separate class called Input Manager reside in the scene, this script can have functions that access and interact with other scene objects. The HVR Controller Events scripts, which reside in the scene in the HVRGlobal prefab, have a UI for designating VR controls with functions available within the Unity scene.

### 4.3.2 Controller Layout




Controller layout for the Quest 2 VR hand controllers

### 4.3.3 Gaze Interaction

The gaze interaction is implemented within the Hurricane VR player rig setup by attaching a custom script to the rig's primary camera called Camera Cast. This script is responsible for detecting whether the user is looking at a Switch Target or not. This detection is implemented using Unity's Raycast system, which is a vector ray fired from a point along a direction that returns the first object it collides with. In this case, the ray is fired forward from the center of the user's camera view and returns the first Switch Target that it collides with. If colliding with a Switch Target, that Switch Target is on and sets its parent as the active Space Switch for further interaction.

### 4.3.4 Recording Demonstrations in VR for GAIL

Recording demonstrations is a built-in feature of the ML-Agents toolkit that allows users within the Unity editor to record themselves controlling the agent and fulfilling the given task. These recordings are saved as time-series data pairing user action to observation, then used to supplement agent training via GAIL. These rewards are considered intrinsic, in contrast to the extrinsic rewards provided by the training environment.

The intrinsic rewards from GAIL influence the policy the agent learns as they are rewarded based on how closely their actions correspond to the demonstration. As a result, one can weigh the impact of the rewards from GAIL against the environment's extrinsic rewards. Previous findings in *Imitation Ball* found that weighing the reward from GAIL between 0.1 to 0.3 has an effect on training without impacting too negatively the extrinsic rewards. This is in contrast to typical practices using GAIL that provide a weight of 0.01 often to just aid the agent in the initial phases of training [23]. This thesis will sample a range of weights to create different agent variants and observe their qualitative outcome.

A novelty this thesis proposes is the use of VR controls to record demonstrations for the agent. That being said, users controlling their avatar in VR is different input-wise than how an agent would control their avatar. As a result, a step is required to map VR controls to the same action vector an agent would provide. For example, a user in VR turning their head would not think of their head turning as an explicit discrete directional action in contrast to moving a

character with a keyboard or mouse. As a result, such continuous actions need a level of discretized interpretation. The following implementation of mapping VR controls to agent demonstration will be further detailed in the next section.

## 4.4 Agent Training and Design

The AI agent is tasked with a number of complex tasks which include locomotion within the design space, targeting blocks to activate, and making decisions on which blocks to contribute to the 3D design. As a result, to break down a more complex problem, the final agent the user co-creates with is actually a product of two separate trained neural networks. Training these separate networks is like training two different agents, which in this thesis will be called the Guiding Agent and the Creator Agent. The Guiding Agent is responsible for looking at the Block Matrix and choosing which block to place next, while the Creator Agent calls the Guiding Agent for its next target block and is responsible for moving in the space and placing its given block. Training these two agents requires two different training environments and parameters built using the ML-Agents toolkit.

### 4.4.1 Guiding Agent

The Guiding Agent is trained to learn block placements given an observation of the Block Matrix and the objective 3D model. This means that for every different 3D model that is included in the prototype, a new network must be trained for the Guiding Agent to learn its configuration. The Guiding Agent is responsible for making the design decisions when co-creating with the user. Hence, crafting the behaviors of the Guiding Agent would be interpreted to conditions such as the order in which blocks are placed, the accuracy of blocks given the objective 3D model, and the decision the agent would make when the user removes or adds blocks themselves.

**Action Space**

The Guiding Agent uses a discrete action vector to interact with the Block Matrix. The action vector has four elements or branches that determine when to place a block and at what location the block should be placed. The following chart describes the action branches of the agent, where a branch is an index of the action vector, the size being the range of possible values of that

branch starting from 0, and the process action is how that branch's value is interpreted within the Unity environment.

| Branch | Size | Processed Action |
|---|---|---|
| 0 | 6 | This branch determines whether to place a block or not or which block type to place, where 0 is to do nothing and 1-5 is a different color or Switch State |
| 1 | 7 | This branch indicates the *x* value of a block selection in the Block Matrix |
| 2 | 6 | This branch indicates the *y* value of a block selection in the Block Matrix |
| 3 | 7 | This branch indicates the *z* value of a block selection in the Block Matrix |

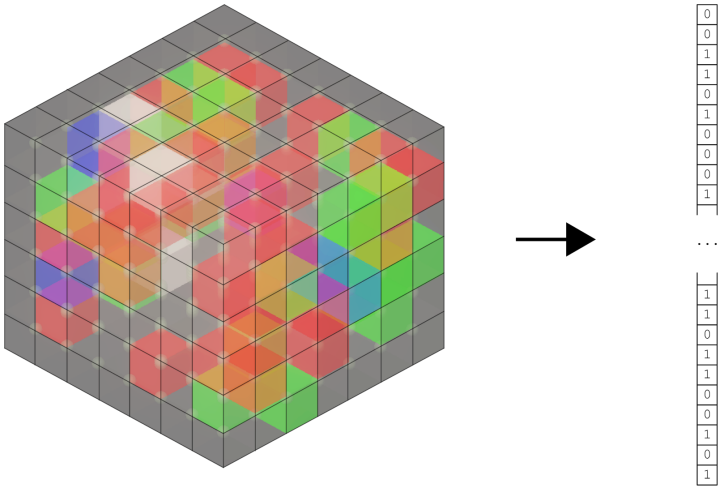
The size of the last three branches of the action vector must reflect the size of the dimensions given to parameterize the Block Matrix. That being said, the agent querying the Block Matrix by giving a 3D vector does not correlate directly with how the Block Matrix references are stored in the Switch Parent as a 1D list.

In order to translate the given 3D coordinates into the 1D list, a simple equation is used where *x* is the given value from Branch 1, *y* is the value given from Branch 2, *z* is the value given from Branch 3, and *X*, *Y*, and *Z* being the parameterized dimensions of the Block Matrix:

$$Index = z + y * Z + x * Y * X$$

**Observation Space**

The observation vector observes the state of each Space Switch in Block Matrix, and hence the vector is the length of the total number of switches. Given that in this prototype the Block Matrix dimension is 7 x 6 x 7, the observation vector is length 294. This observation vector comes directly from the Switch Parent's list of switch states. By default, the Guiding Agent translates the Switch Parent's list of switch states into a binary vector, converting any state not 0 to 1. This option can be toggled as a training parameter for the Guiding Agent, depending on whether block color is a desired factor in the agent training.




Conversion from 3D cube to flat binary vector

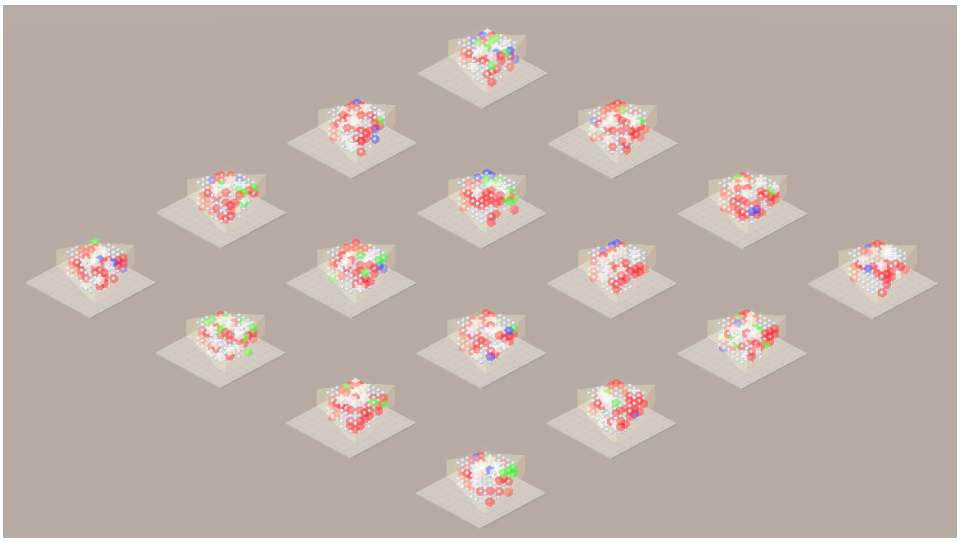
**Reward Function**

The Guiding Agent has its reward calculated per action taken, evaluating more particularly the action a block is being placed. The following provides the list of implemented reward variables and how they are parameterized:

| **Reward Name** | **Value** | **Description** |
|---|---|---|
| Place Block | 0 | Given for when a block is placed (unused but implemented) |
| Good Block | 5 | Given for when a block is placed located within the objective 3D model |
| Bad Block | -5 | Given for a block placed not inside the objective 3D model |
| Misfire | -1 | When a block placement or removal is valid but the action is redundant |
| Good Removal | 3 | Given for removing a block that is not in the objective 3D model |
| Bad Removal | –10 | Given for removing a block that is in the objective 3D model |
| Do Nothing | -0.01 | Given if the agent does nothing for their action |

### Training Environment

An environment for training the Guiding Agent requires three parts: the Guiding Agent, Switch Parent, and the objective 3D model.



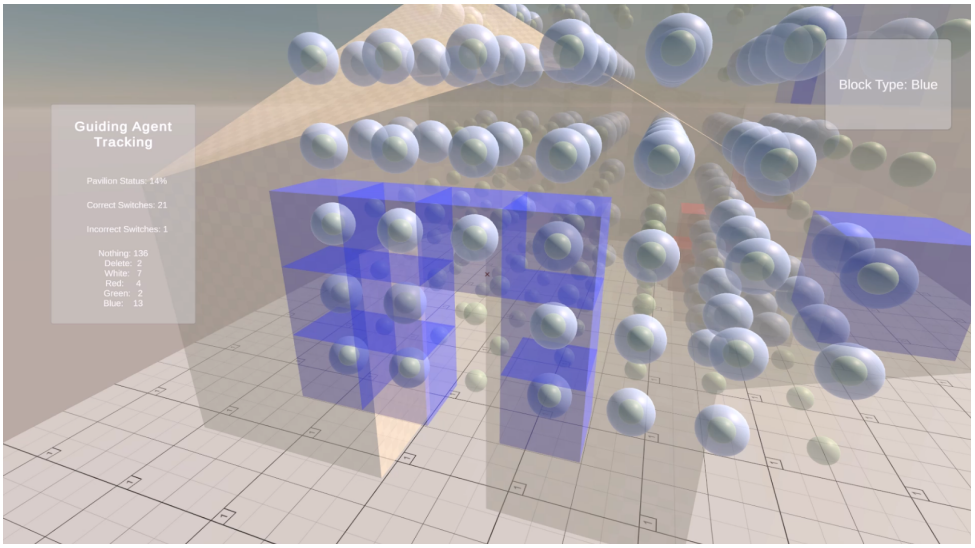
Guiding Agent training environment

### Training Parameters

The Guiding Agent is trained with just extrinsic reward signals from its environment, with GAIL the only addition when training with weighted demonstration. Visit *Appendix B* for full details on network and training parameterization.

### Heuristic Behavior for Imitation Learning

Because the Guiding Agent is learning a more top-down design-oriented task, the implementation for the human user to create demonstrations and control the Guiding Agent follows a similar approach. A separate camera object and script are written specifically for user demonstration for the Guiding Agent. The camera controls match that of the Unity editor controls, likening the experience to that similar to other 3D design programs. This interaction approach was deemed better suited for a design-related task, rather than doing the demonstration in VR because it allows for a more comprehensive viewing of the Block Matrix and faster iteration of design moves.




Creating demonstrations for the Guiding Agent via a point-and-click mouse and camera interaction

The attached script called Camera Heuristic Guide implements a simple point-and-click interaction with the Block Matrix, where a user can click on a Switch Trigger with a given Block Type to set its color. Toggling these Block Types are done with the number keys 1 through 5 on the keyboard and a basic UI updates and indicates to the user which Block Type they have actively selected. The Camera Heuristic Guide has two public variables exposed to the Guiding Agent, one for the most recent Space Switch selected by the user, and the second for the current Block Type of the user. Because the Guiding Agent is parameterized to perform an action given a set Academy interval, when the Guiding Agent does take an action, it looks at the most current Space Switch and Block Type saved within the Camera Heuristic Guide.

When a heuristic action is requested from the Guiding Agent, it must translate the variables given by the Camera Heuristic Guide into a discrete action vector informing the agent's actions. Recall the Guiding Agent's four-length discrete action vector comprised of indicating a Block Type, and the rest of the vector determining the 3D coordinates of the chosen Space Switch in the Block Matrix. If the Camera Heuristic Guide's current Space Switch is null, then the Guiding Agent passes 0 as its first index, which indicates no action is taken. Otherwise, if the current Space Switch is given by a user selection, the Guiding Agent infers the Space Switch's 3D coordinates and passes them into the action vector. The action is taken and the Camera Heuristic Guide's current Space Switch is set back to null until the user makes another selection.

**Agent Variants**

Developing agent variants is a result of using different training parameters, rewards, and demonstrations. The approach for creating a demonstration is to try and show the agent a particular way of doing the design problem. This includes specifying a deliberate approach kept in mind for doing the demonstration. For this particular environment, the Guiding Agent is stylistically determined by two factors: block type placements and block placement order. For the Guiding Agent in particular, it is important to note that these two factors are not at all accounted for in the reward function of the agent, and hence purely an experiment based on whether the current implementation of the Guiding Agent will pick up on these demonstrated stylistic choices. It should also be noted that the human demonstration may help increase the general performance and/or speed of convergence of the agent against the reward function.

For this experiment, there were two different demonstrations recorded to compare the outputs of the Guiding Agent. The following demonstrations are recorded by myself using the Guiding Agent's heuristic controls and each recording approximately captures twenty minutes of design interaction:

| **Demo Name** | **Block Type Logic** | **Placement Logic** | **Avg. Reward** | **Steps** |
|---|---|---|---|---|
| Red-Bottom | Only red block types | Start with blocks at the bottom of the matrix | 519 | 1837 |
| Multi-Sides | Place a unique block type per side | Start with each side at a time, then lastly the roof | 666 | 1988 |

Given these demonstrations, a total of five trained agent variants. The following chart details the agents and their training parameters, and of these five agents, three are considered final to be used for later comparison:

| **Agent Name** | **Version** | **Initialization** | **Configuration** | **Demo** | **Demo Weight** |
|---|---|---|---|---|---|
| Base | 1 | None | Vanilla | None | N/A |
| Base* | 2 | Base 1 | Vanilla | None | N/A |
| Red | 1 | None | BC + GAIL | Red-Bottom | 0.5 |

| | | | | | |
|---|---|---|---|---|---|
| Red* | 2 | None | BC + GAIL | Red-Bottom | 0.8 |
| Multi* | 1 | None | BC +GAIL | Multi-Sides | 0.8 |

* final agents for comparison

### 4.4.2 Creator Agent

The Creator Agent is trained to move in the design space and locate a designated block to trigger and construct a block at its location. The Creator Agent is designed similarly to the Hurricane VR rig that the user embodies with their VR controls. As a result, the agent is trained to accomplish a more straightforward task of finding blocks within the Block Matrix. Where RL plays a larger role in training a behavior in the agent, whether through modifying the training environment, reward function, or augmenting the training with IL. The Creator Agent is a representation and extension of the Guiding Agent's design decision-making and is meant to explore the interface between the user and AI in a co-creation environment.

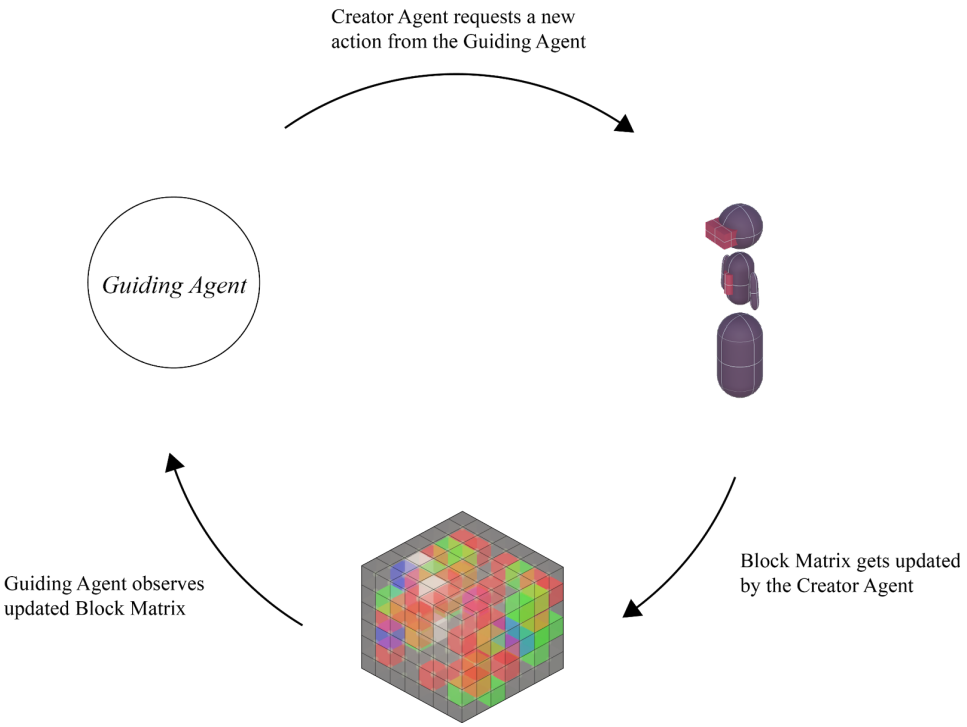
**Components**

In the Unity environment, the Creator Agent is constructed by the top-most parent object, which the Creator Agent class is attached to, and child objects which are the Body, Head, and ray perception sensors used for agent observation. The Body object contains child objects that include the agent's 3D mesh or visual appearance, and the agent's physical colliders. The Head object sits at the top of the Creator Agent model containing the model of the head. The object is linked to the Creator Agent class as a reference.

There is a total of two ray perception sensors within the Creator Agent, one on the body and one on the head. The ray perception sensor is a script that deploys a Raycast that can be converted into an observation for the agent during the training. The use of these sensors is to indicate to the agent what is directly in front of it.

The last core part of the Creator Agent's functionality is the integration of the Guiding Agent. The Guiding Agent's role is to provide actions from observing the Block Matrix to be carried out by the Creator Agent. The Creator Agent has a reference to the Guiding Agent so that when it needs a new target Space Switch to find and trigger, the Creator Agent can request an

action from the Guiding Agent. This sampling is done once at the beginning of the Creator Agent initialization to find the first target, and whenever a current target is triggered and a new one is needed. That being said, the Guiding Agent has the choice to not place a block. In this situation, the Creator Agent's location is frozen and it stays listening until the Guiding Agent decides otherwise, in which the Creator Agent is free to move again.




Communication loop between agents via Block Matrix

**Action Space**

The Creator Agent uses a discrete action vector to interact with the Block Matrix. The action vector has six elements or branches that determine the movement, rotation, and block placement of the agent as described in the following chart. For all but the last branch, the values between [0, 2] are converted to [-1, 1] indicating a positive value, negative value, or to do nothing.

| Branch | Size | Processed Action |
|---|---|---|
| 0 | 3 | This branch determines agent movement along the *x*-axis |
| 1 | 3 | This branch determines agent movement along the *z*-axis |
| 2 | 3 | This branch determines the agent's Euler rotation in the *y*-axis |
| 3 | 3 | This branch indicates the agent head's Euler rotation in the *x*-axis |
| 4 | 3 | This branch indicates the agent head's Euler rotation in the *y*-axis |

All movement and rotation are calculated as a normalized vector by a factor of the time between frames and a speed constant. This speed constant is a parameter that affects the agent's speed and consequently the behavioral outcome of its training. Too, agent head rotation is clamped between a positive and negative angular value, matching the way a human user in VR is only able to rotate their head vertically and horizontally to a limited degree. The following chart describes the parameters and their values used for the Creator Agent.

| Parameter Name | Value | Description |
|---|---|---|
| Move Force | 2 | Constant speed value used for moving the agent's body |
| Move Limit | 5 | Clamps the agent's location to the area of the design environment, preventing any unpredictable boundary bypass |
| Rotation Force | 100 | Constant speed value used for rotating the agent's body |
| Head Rotation Force | 70 | Constant speed value used for rotating the agent's head |
| Head Rotation Limit | 45 | Positive and negative range for which the agent can rotate its head in the vertical and horizontal axis |

For triggering changes in the Switch States, the agent uses the forward direction from its head to detect whether the agent is looking at a Switch Trigger or not. The implementation for toggling a Switch Trigger is exactly the same as implemented with a user's VR headset as described in 4.3.3. The only difference is that it does not fire a Raycast from the center of a

camera view as the agent does not use a camera instead firing from the center of the head's world position. Another difference is that the Creator Agent is not responsible for determining the Switch State to toggle the Space Switch to as this is instead the role of the Guiding Agent.

**Observation Space**

The observation space is a twelve-length vector indicating to the agent its own local position, its local rotation, its local head rotation, the direction to a target switch, the local position of the target switch, and whether the Creator Agent is currently looking at a Space Trigger or not. The following chart breaks down the indices of the observation vector:

| **Indices** | **Observation Description** |
|---|---|
| 0 -1 | The agent's local *x* position and *z* position |
| 2 -3 | The agent's normalized local rotation (quaternion *y*, *w*) |
| 4 - 7 | The agent head's normalized local rotation (quaternion) |
| 8 | Dot product from agent's head to target Space Switch trigger |
| 9-11 | The location vector of the target Space Switch |

**Reward Function**

The Creator Agent has its reward calculated per action. The following provides the list of implemented reward variables and how they are parameterized:

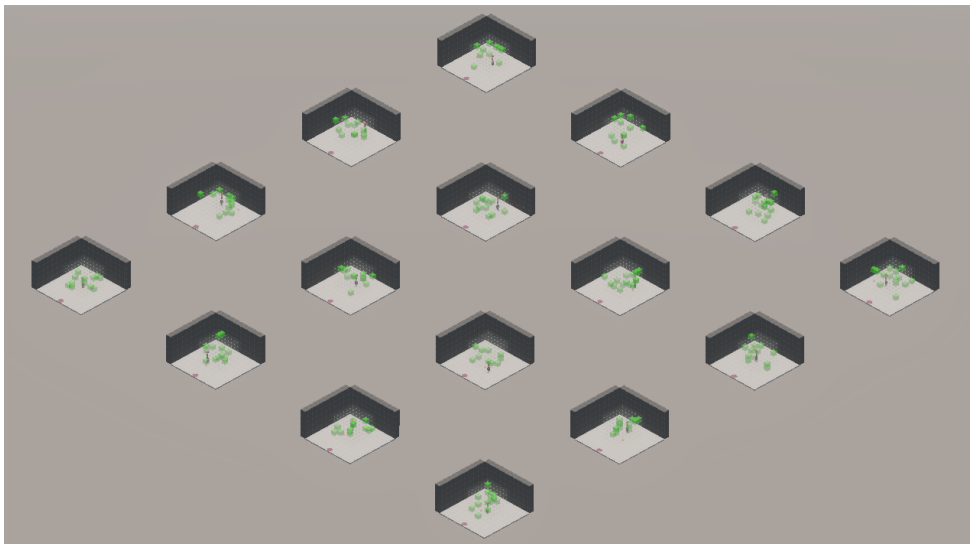
| **Reward Name** | **Value** | **Description** |
|---|---|---|
| Target Block | 100 | Given for when the target trigger is gazed at via Raycast |
| Look Towards Factor | 0.1 | Factor for weighing the negative dot product reward |
| Touch Obstacle | -10 | Given per frame the agent is touching a collider with an 'Obstacle' tag |

The Look Towards Factor differs from other agent rewards as it weighs a constantly updated function rather than describing the value of a more sparse reward like looking at the Target Block. This particular reward function uses the same dot product value calculated for the agent observation, which is determined using the forward vector from the point of the agent's gaze and the vector from the agent's gaze to the position of the current target Space Switch. What this tells the agent is how far off its current gaze vector is from the gaze it needs to have in order to locate the target switch.

Using this observed value, we can create a reward function that suggests to the agent the direction it needs to look towards to find the target Space Switch. This function uses the dot product value as a negative reward for the agent, forcing the agent to minimize the difference in direction between its gaze and the target vector. To do so, the dot product is subtracted by 1 to convert it into a negative range between [-2,0]. Furthermore, this value is then squared to exponentially increase the negative reward for looking away from the target. The function is given below, where ***a*** is the forward vector of the agent's current gaze and ***b*** is the vector from the agent's gaze to the target Space Switch:

$$Look\ Towards\ Reward = -(a \bullet b^T - 1)^2 \times Look\ Towards\ Factor$$

**Training Environment**



Creator Agent training environment

An environment for training the Creator Agent requires three parts: the room environment, Creator Agent, and the Switch Parent. In practice, the agent is given a target by the Guiding Agent to find, and once triggered, is given a new target Space Switch to seek out. Instead, during training, there are three options implemented within the training environment that employs this interaction. These options are, finding the closest trigger to the agent, selecting a random Space Switch, and using a trained Guiding Agent. For most purposes, we want a high-performing base Creator Agent that can find Switch Triggers from a random selection. This ensures that the agent is able to properly generalize locating targets within the Block Matrix.

**Training Parameters**

The Creator Agent is trained with just extrinsic reward signals from its environment and Curiosity to encourage exploration. For demonstration, BC and GAIL are used, and Curiosity is not. Visit *Appendix B* for full details on network and training parameterization.

**Heuristic Behaviors for Imitation Learning**

Developing heuristic controls for the Creator Agent is most concerned with implementing a way for a demonstrating user to use VR controls to control the Creator Agent. Because the Creator Agent is designed around default VR controls, this translation can be made rather directly. Recall that the Creator Agent's action vector controls the agent's movement, rotation, and head rotation. Like with the user VR controls, the same inputs can control the agent in the same way. The Left Joystick indicates moving forward or backward and turning left or right, and the Right Joystick determines the left or right rotation of the agent. The control of the agent's head is not explicit, where instead the pitch and yaw of the agent's head are updated to turn in the direction of the user's current head position in the VR headset.

Tracking the VR inputs is done in a separate script called Creator Heuristic VR, which is attached to the Creator Agent. This script exposes five public integers between [0,2] that correspond to the agent's discrete actions. The Creator Agent accesses these variables to determine its actions. Per frame, the script updates these values by listening to Hurricane VR's Input Manager, exposing the state of all of the buttons on the VR hand controllers. The script also compares the current Euler rotations of the agent's head compared to the VR user's head.

Given a threshold parameter, if the difference between rotations is larger than the threshold, the script will indicate the discrete direction the agent's head must turn in.

Since the agent's head direction tries to match that of the user's headset orientation while doing the demonstration, it was discovered the importance of creating a separate camera on the agent's head for the user. What this means is that the user's camera is separate from the agent's head direction while still being located at the same location. This is for two reasons, that being the agent is constantly trying to catch up to the user's head rotation, and the agent's action space is discretized, meaning that the agent's head rotation is not a smooth accelerating and decelerating motion. A user sharing this view with the agent would be a nauseating experience, as it does not respond naturally to the user's head movement. Instead, the demonstrating user's camera directly tracker their head movement while a red dot in their camera view represents the gaze of the agent. Thus the outcome is the agent's gaze visibly catching up to the center of the demonstrating user's gaze.

**Agent Variants**

Similar to the Guiding Agent, the Creator Agent's variants come as a result of augmenting training with different parameters and human demonstration. Unlike the Guiding Agent though, the Creator Agent and the user in VR have more similar modes of operation within the design space and the task is not open-ended. Instead, the task is a clear matter of aligning one's gaze with the target Space Switch. As a result, there does not need to be a clear logic defining the demonstration, and instead, the demonstration can be retroactively described and labeled as being human-like. I recorded one demonstration performing the same task as the agent, continually aligning my gaze to a random target for approximately 45 minutes:

| **Demo** | **Retroactive Demonstration Description** | **Avg. Reward** | **Steps** |
|---|---|---|---|
| Human | Stayed close to the side of the Block Matrix near the agent spawn and relied primarily on head rotation rather than body rotation to align my gaze with the target Space Switch. | 7560 | 18286 |

Given these demonstrations, a total of four trained agent variants were developed. The following chart details the agents and their training parameters, and of these five agents, three are considered final to be used for later comparison:

| Agent Name | Version | Initialization | Configuration | Demo | Demo Weight |
|---|---|---|---|---|---|
| Base* | 1 | None | Vanilla | None | N/A |
| Base-Demo | 1 | Base 1 | BC + GAIL | Human | 0.3 |
| Base-Demo* | 2 | Base 1 | BC + GAIL | Human | 0.8 |
| Demo* | 1 | None | BC +GAIL | Human | 0.8 |

* final agents for comparison

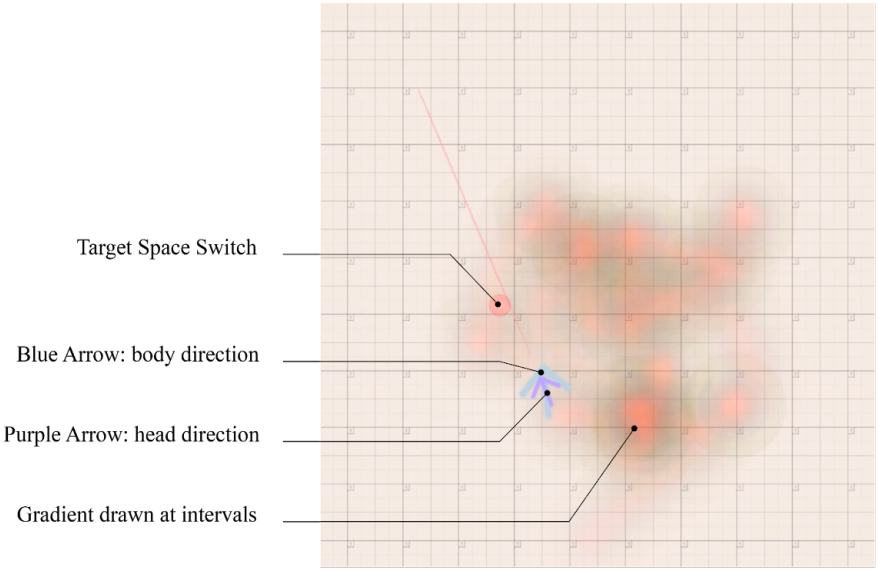
### 4.4.3 Inference

For evaluating agents and tracking progression, Tensorboard is used to see how the agent is training over time based on an ideal accumulation of rewards within its environment. Typically, once an agent is unable to gain any more rewards in its environment, the network has converged at an agent that can perform the task to the best of its ability. Any further training would likely lead to higher rewards but the payoff for that increase compared to the time taken for more training is not worth it in an iterative development process.

Particularly to the Guiding Agent, outside of tracking its cumulative rewards, we can also quantitatively track its actions and its actions in relation to the Block Matrix. Used for later comparing different trained Guiding Agents against each other, simple tracking metrics are made visible during inference within Unity. These metrics are the percentage of the objective 3D model completed based on its voxel representation, the number of correctly placed blocks, the number of incorrectly placed blocks, and the count of each Block Type given by an action from the Guiding Agent. These simple metrics will be helpful in analyzing the capacity of the Guiding Agent as well as the effect IL has on augmenting the training process.

Though we could apply similar tracking techniques for the Creator Agent as well, it was found that qualitative observations are more useful for the Creator Agent. This is because the Creator Agent's ability to do the task is far more behavioral in comparison to sequential design moves. That being said, there are ways to create a quantified record of the behaviors of the Creator Agent. One way, and the only way explored for this prototype, is a heatmap of the

Creator Agent. While the Guiding Agent is still giving the Creator Agent targets, the position of the Creator Agent is saved in the form of a circular gradient to a cumulative collection that makes up a heatmap. A heatmap is saved per episode, visualizing where the agent spends their time in the design space. This method is one way of visualizing behavior between agents for direct comparisons, especially when considering the impact of demonstration data.




Heatmap view or evaluating the Creator Agent

Otherwise, qualitative observations of the agents are done through an established reflection, whereas both developer and observer, I answer a set of questions posed to myself about the nature of the final agents. Furthermore, I also reflect as a user within the space with these final agents, documenting the overall design interaction experience to be used for later discussion. These questions and their corresponding answers are presented in greater detail in the following Results section.

## 4.5 Latent Agent Visualization

When evaluating agent training, we can determine the quantitative success through whether it has converged in training via rewards attained in the environment, and qualitatively through pure observation of its behaviors. That being said, this comparison is best suited for benchmarking RL

algorithms and hyperparameters, in contrast to more subjective objectives that this thesis concerns itself with like designing and crafting agent behaviors. As a result, this thesis explores the potential of a visualization method as previously discussed in 2.3.3 in the paper Human-Level Control Through Deep Reinforcement Learning [34] using t-SNE and principal component analysis[10] (PCA) to describe a 3D embedded action space. Furthermore, borrowing from ideas in *A Walk in Latent City*, the latent action space can be visually customized with agent training data to depict observations from training in its 3D context.

### 4.5.1 Latent Space Generation

In brief, t-SNE takes dataset entries described by *n*-dimensional vectors and projects them onto a lower dimensional manifold where ideally datapoint clusters are more easily distinguishable. In the context of RL, creating a 3D latent space of action embeddings means taking samples of an agent's actions as a dataset and projecting the action vector dimension to three dimensions.

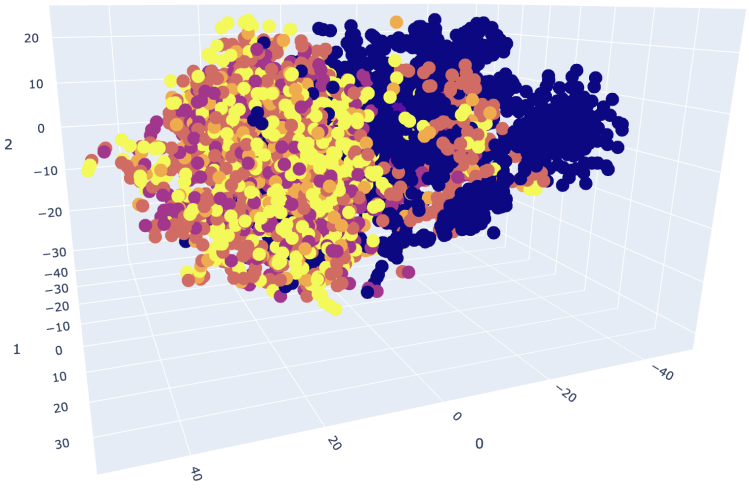
A 3D point in this space represents an agent's action, and the proximity between points describes the relative similarity of those actions. This is useful because we can then pair each action embedding with a given observation vector, indicating that the given observation influenced the agent to take that particular action. Furthermore, we can also use this latent space to include multiple different agents. This means that we can locate similar or the same action taken by different agents and compare their given observations.

For this prototype, two latent action spaces are created for the Creator Agent and the Guiding Agent. Producing the latent embeddings includes sampling data from a trained agent. This is done by letting a trained agent operate within its environment and by a set interval, sampling the agent's action and observation pair, and writing them to two different actions and observation CSV files. Because we are only concerned with generating a latent action space, we only need to run the t-SNE algorithm with the action CSV files.

The t-SNE latent space is generated via scikit-learn[11]'s implementation of the algorithm in Python. The script also uses pandas[12] DataFrame data structure to load and store the action

[10] Principal component analysis is a dimensionality reduction technique for large dataset
[11] scikit-learn is a machine learning library for Python
[12] pandas is a library for Python that provides tools for data analysis

CSVs. Because an action space comprises different agents' actions, we want to run t-SNE on all of the different agent actions collectively. Hence, when processing the CSVs, they are concatenated into one DataFrame to be fitted into the t-SNE model. The t-SNE model is parameterized with having 3 components, a perplexity of 40, a random state of 0, and is initialized with PCA. After the projection is complete, the t-SNE DataFrame is split up into individual tables corresponding to the original agents and then written to a new CSV file of three columns, each column corresponding to a projected dimension.




A more standard visualization of the Guiding Agent's action embeddings via the Plotly Python toolkit

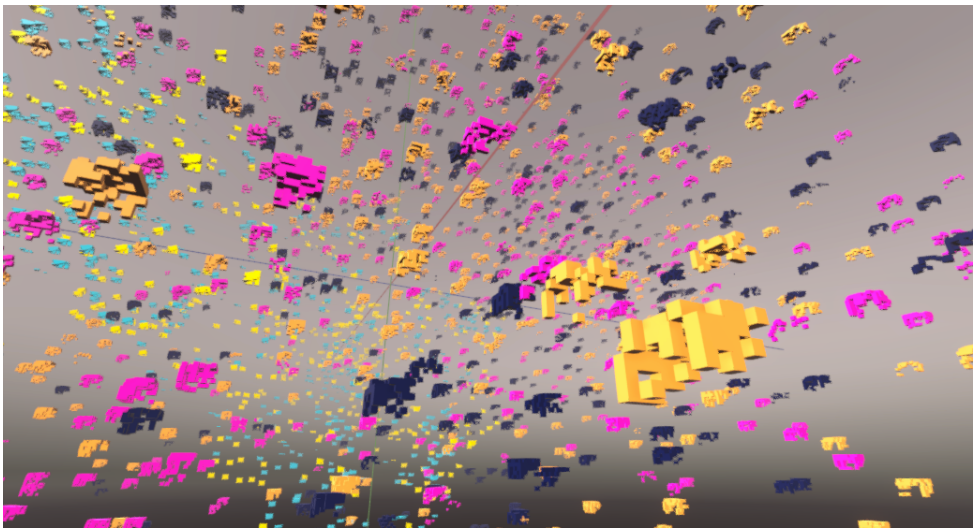
To add richness to the action embeddings, a parameter was also introduced to combine the last $t$ embeddings to the current action vector. What this means is that instead of the action vector being projected from its original $n$ vector length, by concatenating the last $t$ actions, each embedding is actually of size $n \times (t + 1)$. Early actions without prior entries are just concatenated with zeros. The reason for doing this is because, in RL, actions taken do account for previous actions and hence are not taken in isolation. What this also adds is a greater dimension to the latent action space creating a distinction between otherwise overlapping action embeddings. For the latent space in this prototype, $t$ is set to 3 prior actions.

### 4.5.2 Visualizing Latent Embeddings

Given the aforementioned process of obtaining the sampled agent action vectors as a projected latent embedding in CSV format, we can now use this data to locate points within 3D that

correspond to agent actions taken. Building a representation of agent actions in a latent space has been implemented by creating a general class and object that defines the collection of points for one agent.

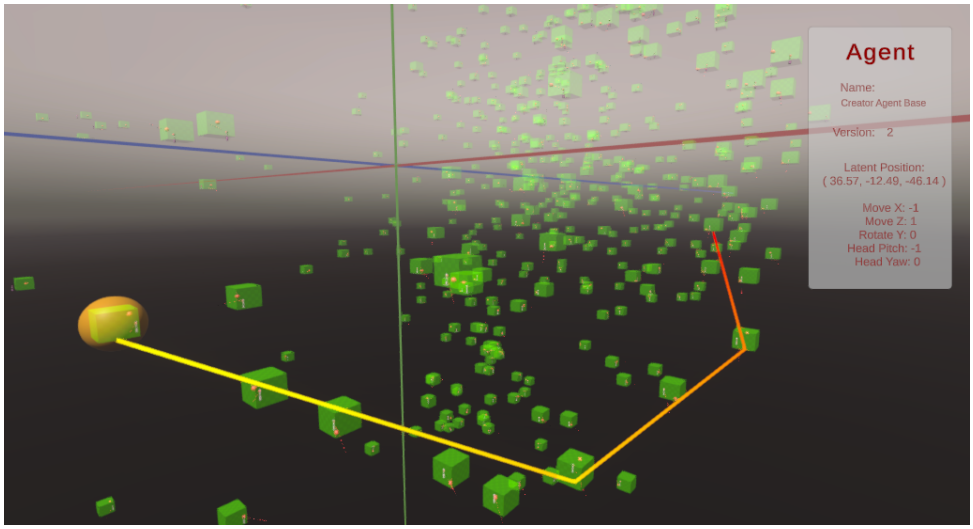
This class called the Latent Parent requires a reference for all three of the CSVs associated with its agent, which includes the CSV of observations, actions, and projected actions. These CSVs do as follows: the projected actions indicate the position of a data point, the observations inform the visualization at that point, and the actions inform the metadata read when toggling over a particular data point selection. The Latent Parent also needs a reference and indication of the observation type it is meant to process, which in this prototype is between the Guiding Agent and Creator Agent. For each action, the data point is tagged with a custom representation of its associate observation given the agent from which it is derived. Lastly, each Latent Parent has their own unique color to help differentiate between agents.



View inside the Guiding Agent's space of action embeddings

The camera script implemented for navigating this 3D latent space resembles that of the 3D navigational controls familiar in the Unity editor and other 3D applications like Blender. The user is also able to point and click in this space to read action metadata in a popup panel and zoom into data points for closer inspection. In addition, because in this latent space, its embeddings are parameterized by previous actions, a gradated line is drawn starting from the

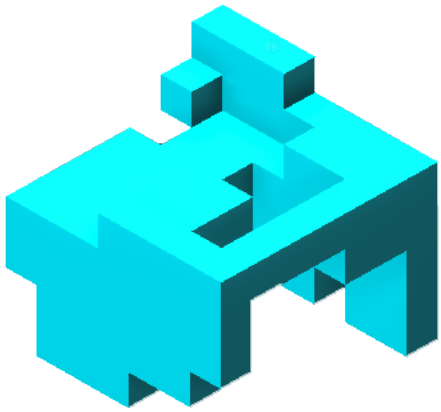
selected embedding to sequentially the previous *t* action embeddings. Lastly, users can toggle on and off between different Latent Parents depending on which agents they want to view.



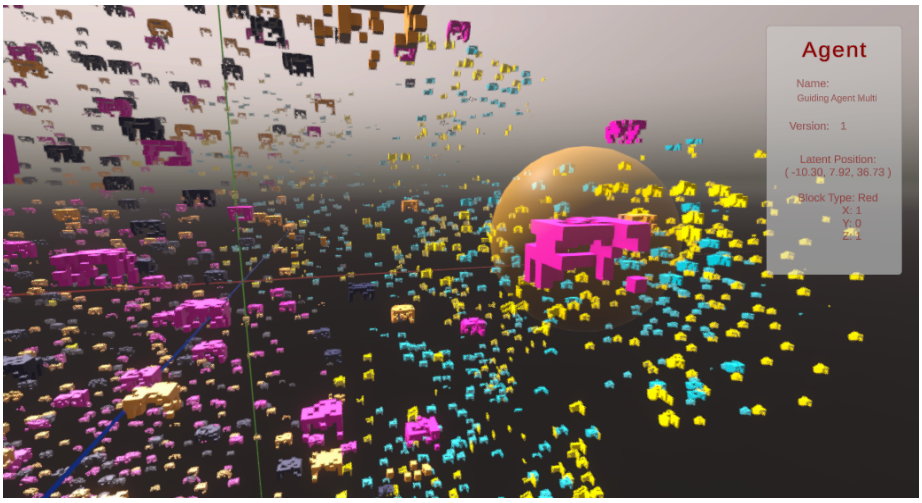

A selected embedding is highlighted by a yellow sphere, has a UI description, and a line drawn to each preceding action

**Guiding Agent Observation Point**

The observation point for the Guiding Agent represents the current state of the Block Matrix when the agent takes an action. At each action point, the corresponding observation is a binary vector that details whether a block is at a position or not. This vector is used for reconstructing the Space Switches in the Block Matrix that were active on that action being taken.



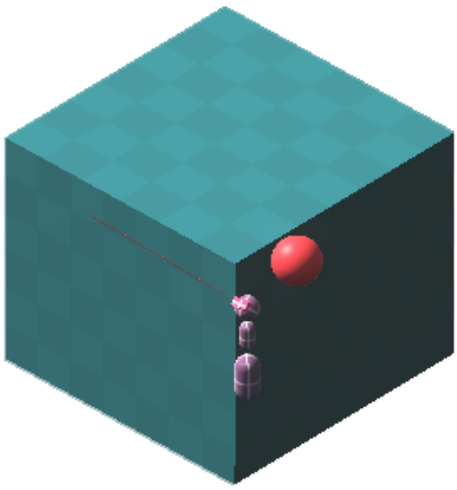
Sample reconstruction of a Guiding Agent observation



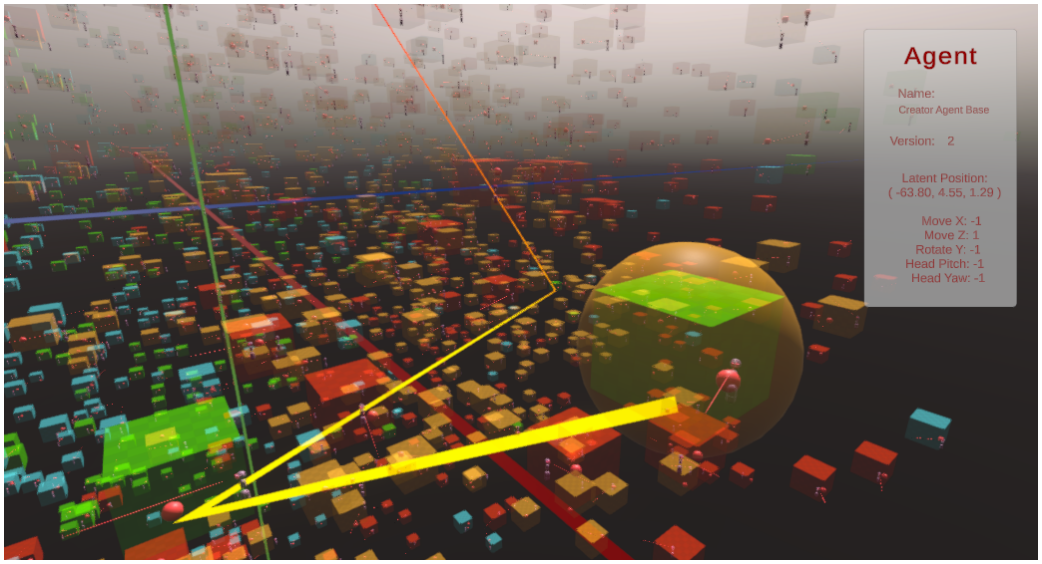

A selected Guiding Agent observation within the latent evaluation interface

### Creator Agent Observation Point

For the Creator Agent, its observations depict an abstracted scene of the agent's body and head pose, as well as the relative position of the Block Matrix and the target Space Switch. The agent observation vector stores all of this information making its exact reconstruction feasible. What this helps the user with is seeing relative to the current agent's body positioning and its target Space Switch, what actually it chooses to do next.



Sample reconstruction of a Creator Agent observation




A selected Creator Agent observation within the latent evaluation interface

# 5. Results

## 5.1 Quantitative Training Results

The quantitative metrics used and developed for this thesis help shed light on the technical performances of the methods. The importance of this is in validating the proficiency in the use of deep learning technologies to support the conceptual claims of this thesis. Too, the quantitative results help reflect promising results towards designing agents through human demonstration in RL training. It also reflects shortcomings too, providing insight into how the methods in the thesis can be further improved.

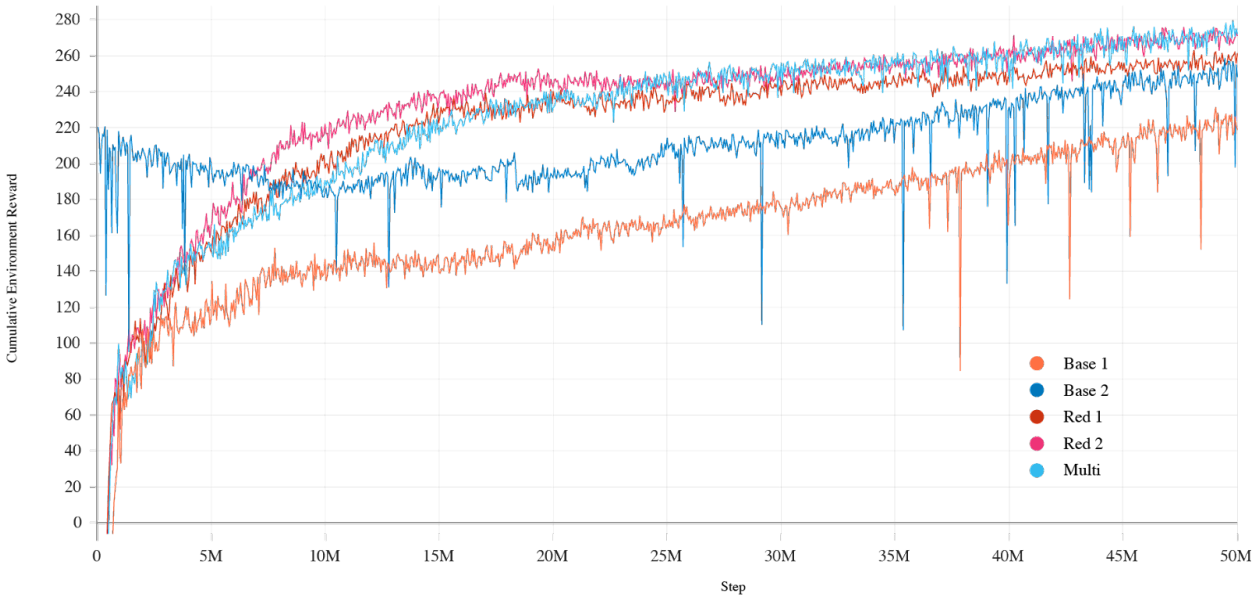
### 5.1.1 Guiding Agent

**Training Progression**

The training progression of the Guiding Agents can be seen in the following graph. A note on the Base 2 agent is that its weights are initialized from the Base 1 agent, which is why its initial environment reward begins approximately when the Base 1 agent ends after training for 50 million steps. Another clear observation is that we notice the agents trained with GAIL learn the design task faster than the base agent and learn to achieve a higher reward. If combining the training for Base 1 and Base 2 agents, it takes about 100 million steps to achieve a reward similar to the Red 1 agent, which is only trained after 50 million steps.

If we recall the rewards achieved by my human demonstration, which are 519 for Red-Bottom and 666 for Multi-Sides, the agents are far from achieving human-level performance for this task. That being said, we can observe the positive effect demonstration has on training. This is further tested when comparing the Red 1 and Red 2 agent. The difference between these agents is that the intrinsic reward from GAIL is weighted at 0.5 for Red 1 and 0.8 for Red 2. This stronger weighting of the intrinsic reward results in improvement during training.

*Guiding Agent Rewards Over Training Steps*



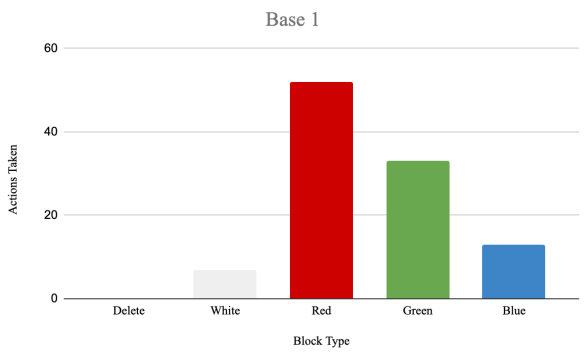

| **Agent** | **Final Reward** |
|---|---|
| Base 1 | 218 |
| Base 2 | 248 |
| Red 1 | 261 |
| Red 2 | 271 |
| Multi | 271 |

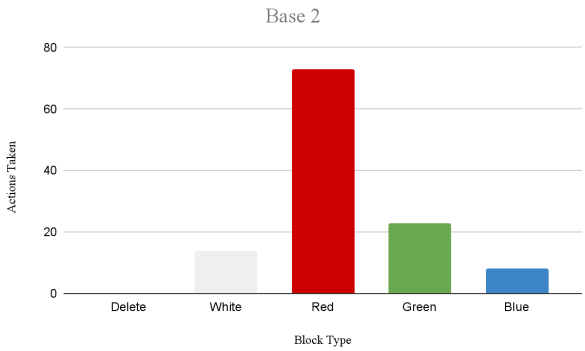
**Action Distribution**

The following charts detail the action distribution of the Guiding Agent given a single episode or run of designing the given 3D objective model in the Block Matrix. We can observe that from a performance point of view in regards to learning the block placements for the objective 3D model, the agents all perform very similarly in terms of total correct block placements. The main difference is that we can observe the agents trained with GAIL performing fewer incorrect block placements, which is why the Red and Multi agents achieve a higher average reward after training than the Base agent.

| Agent | Model Completion | Correct Placement | Incorrect Placements |
|---|---|---|---|
| Base 1 | 41 % | 60 | 7 |
| Base 2 | 50 % | 72 | 7 |
| Red 1 | 45 % | 65 | 5 |
| Red 2 | 49 % | 71 | 3 |
| Multi | 48 % | 70 | 5 |

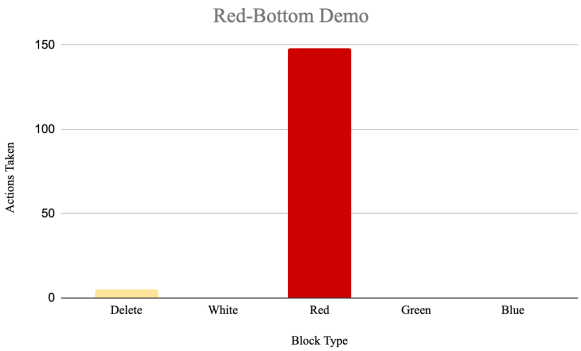
Though we see performative improvement from augmenting the training with demonstration, do the stylistic conditions also translate? Quantitatively we can compare the action distribution between my own demonstration to the training of the Guiding Agents. As a reminder, Red-Bottom only used red block types, while Multi-Sides tried to use a balance of the four different block types. We are interested in whether the style embedded in the demonstration data would translate into the agent's trained behavior despite not being made explicit in the reward function.

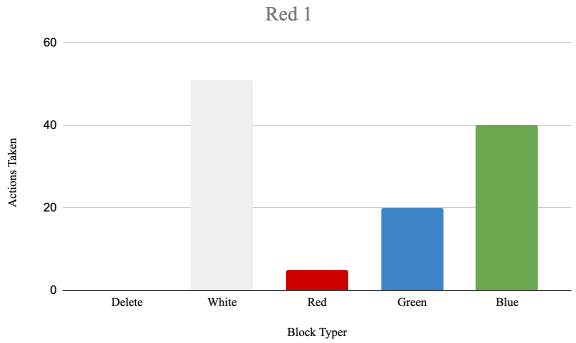


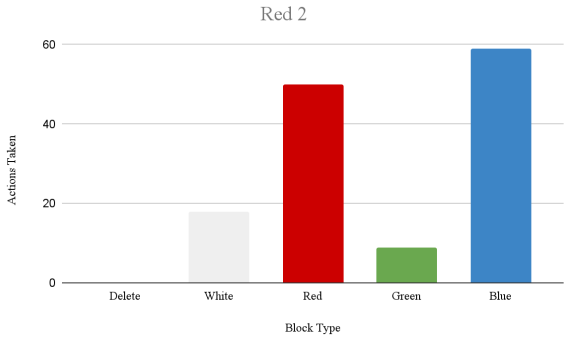



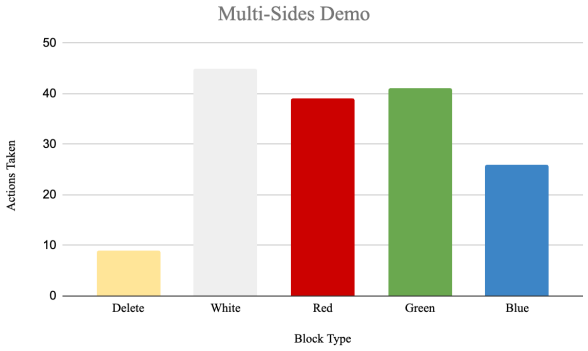

Before evaluating the agents augmented by demonstration, we can observe the differences in Base agents and compare the impact on Block Types used given initializing weights from a prior agent. As you can see between Base 1 and Base 2 agents, the action distribution is fairly similar, meaning that the Block Type selection of the agent is likely established more early on in the training cycle than later. We also see that the Base 2 agent becomes further reinforced to pick the red Block Type. This might be the case that by random chance, the red Block Type became associated with positively reinforced actions for the agent,

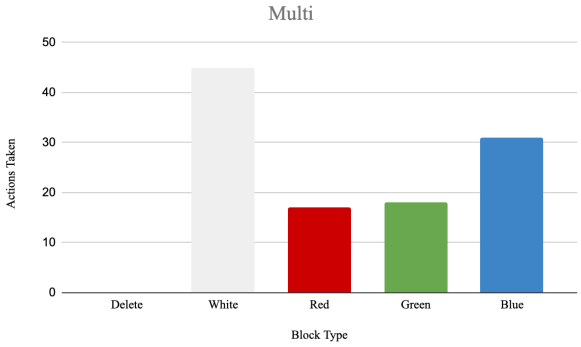
and further training reveals that the red Block Type becomes associated with those positive moves.




Observing the action histograms comparing the human demonstration to the demonstration augmented agents, we see that it is difficult to conclude that there is a correlation. We see that even with a demonstration of picking just one Block Type, even though the agent fixates on two Block Types rather than an even distribution, it is evident, especially with Red 1, that the given Block Type being singled out is not necessarily going to carry over to the trained

agent. Though we might see a more even distribution of actions taken in the Multi-agent, as intended by the Multi-Sides demo, it is still not conclusively correlated. Some evidence as to why this is the case will be more discussed in the latent evaluation of the Guiding Agent.

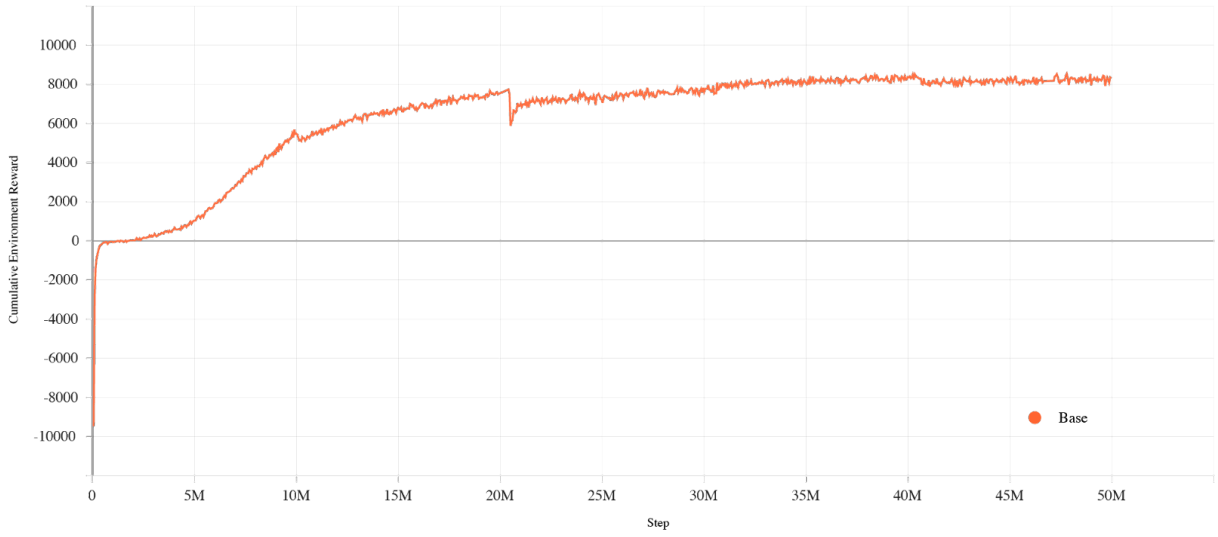
### 5.1.2 Creator Agent

**Training Progression**

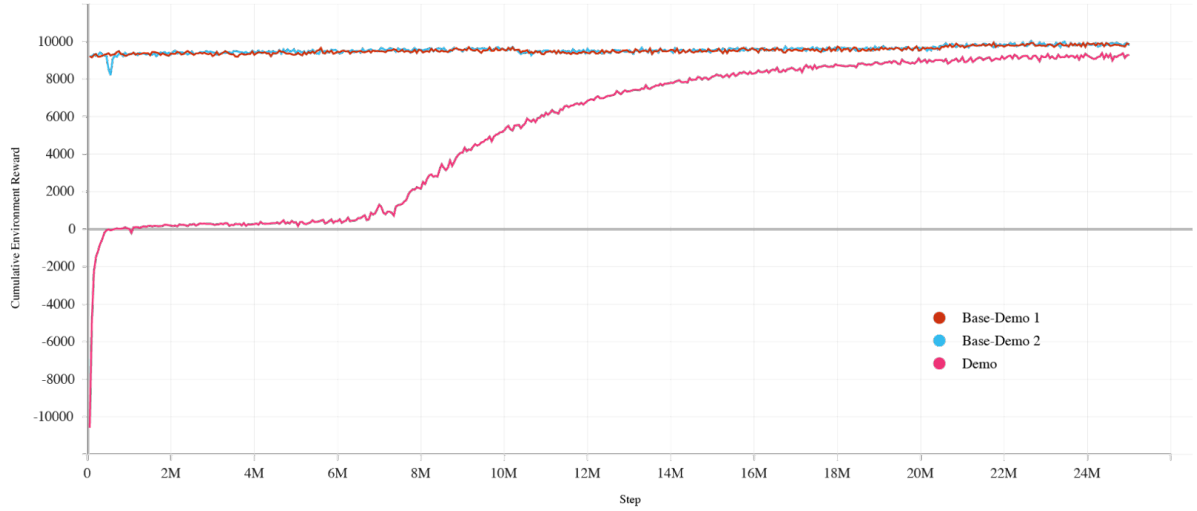
The following training progression of the Creator Agent is separated into two graphs, one with just the Base agent, and the other with the agents trained with augmentations from the demonstration. As a reminder, the difference between Base-Demo 1 and Base-Demo 2 is that Base-Demo 1 is trained with a weight of 0.3 to its intrinsic GAIL reward, and Base-Demo 2 with a weight of 0.8. Also, note that for both Base-Demo agents, their weights are initialized from the Base agent, and hence have accumulated 75 million training steps in comparison to the Base agent's 50 million, and Demo agent's 25 million.

Recalling that my demonstration averaged a reward of 7560 per episode, all of the agents outperformed my performance on this specific task. This is in contrast to the Guiding Agent, which likely reveals something regarding the setup and complexity of the task itself. But like the Guiding Agent, using the demonstration and GAIL made a performance and training difference when comparing the trained agents. This is evident when comparing the Base agent to the Demo agent, where the Demo agent reflects about a 12% increase in rewards gained with half the training steps. Furthermore, the Base agent after 25 million steps only achieves an average reward of 7217, which is still below my demonstrated performance.

*Creator Agent Rewards Over Training Steps: Base Agent*



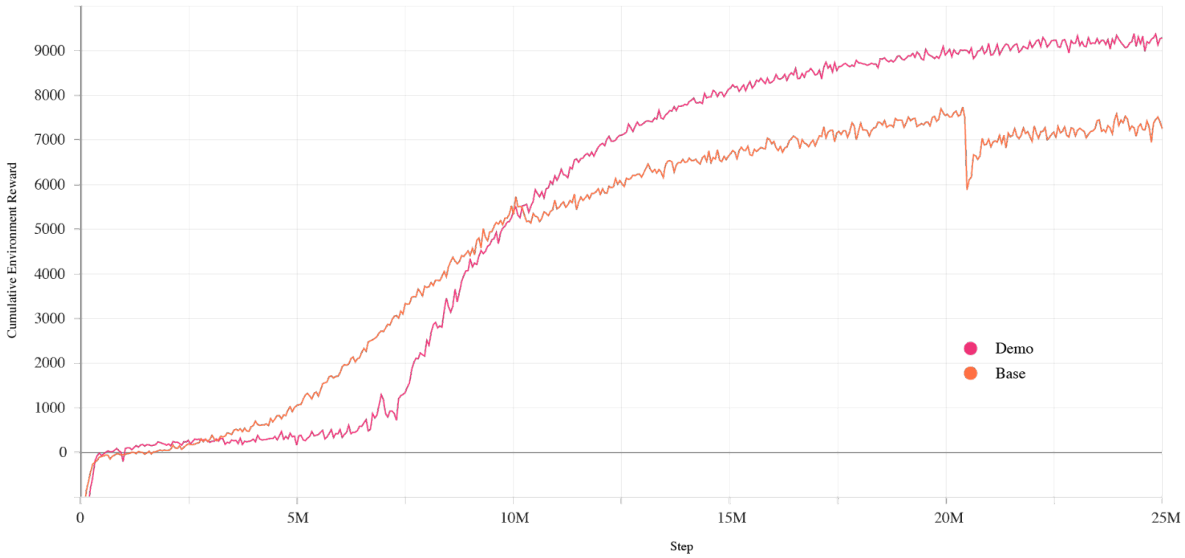

*Creator Agent Rewards Over Training Steps: Demo Augmented*




Another intriguing observation comparing the Base and Demo agent is their training trajectories. The Base agent after 3 million steps start to exponentially increase in reward, compared to the Demo agent starts to see significant training progress after 7 million steps. Due to GAIL's heavy weighting, this can be expected as the Demo agent is required to also maximize its intrinsic rewards alongside the extrinsic environment rewards. With that said, once the Demo agent does find a method that maximizes environment and intrinsic rewards, its method outperforms the strategy the Base agent converges to.

Regarding the Base-Demo agents, more comparisons are needed to validate their positive impact on the Base agent prior. Though we do see that the demonstration might potentially increase performance by maintaining a steady upward trend to an otherwise converging Base agent.

*Creator Agent Rewards Over Training Steps: Demo vs Base Agent*

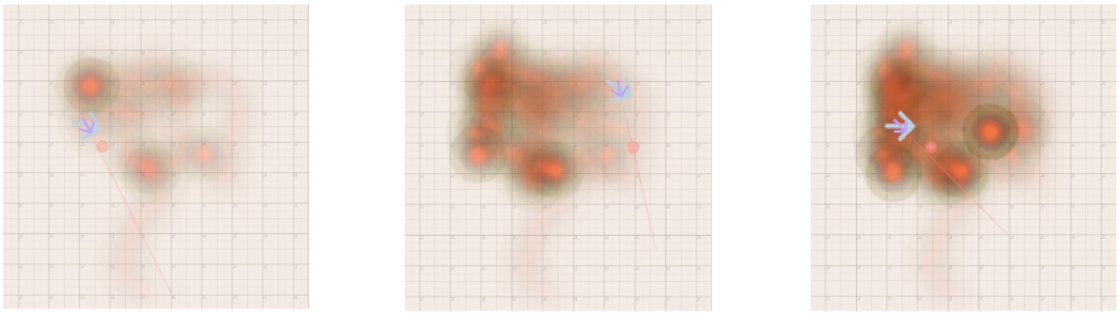



| Agent | Final Reward |
|---|---|
| Base | 8300 |
| Base-Demo 1 | 9840 |
| Base-Demo 2 | 9792 |
| Demo | 9288 |

**Action Distribution**

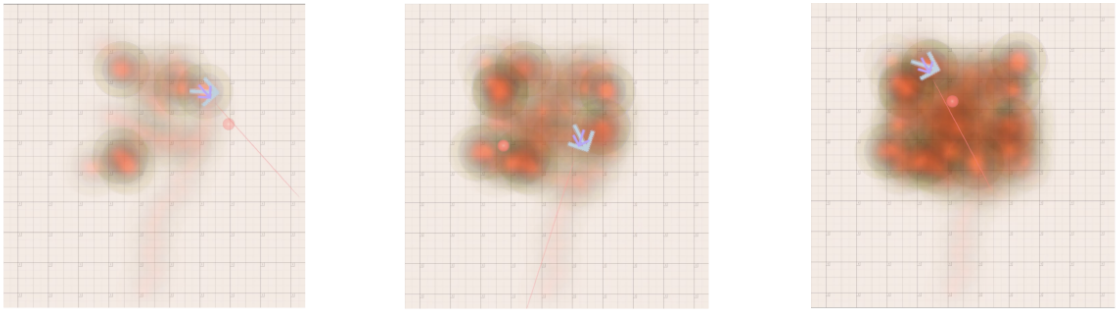
As discussed in Methods section 4.4.2, quantifying the Creator Agent's actions is not as discrete as the Guiding Agent. As a result, the heatmap is employed to compare the movement patterns of the agent, keeping track of its location in the design space, as well as the agent's head and body rotation direction. It takes each Creator Agent about ~2 minutes to complete the Guiding Agent's design, so the following captures the live heatmap at different time intervals of the agent.

*Base Agent Heatmap*



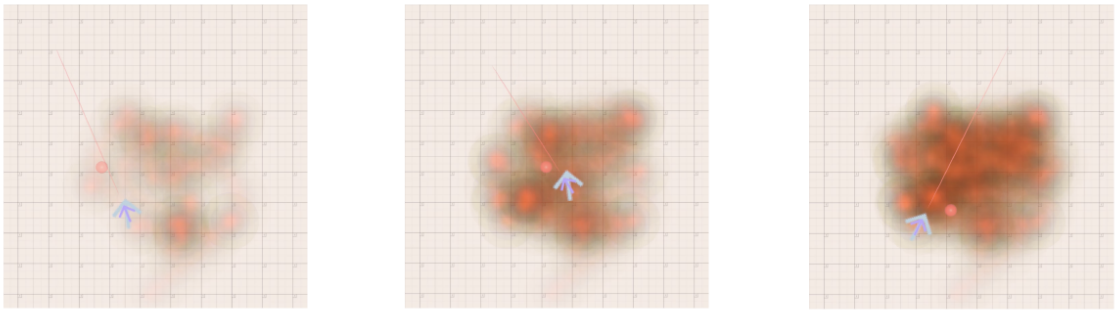
15 seconds | 75 seconds | 120 seconds

*Base-Demo 2 Agent Heatmap*



15 seconds | 75 seconds | 120 seconds

*Demo Agent Heatmap*



15 seconds | 75 seconds | 120 seconds

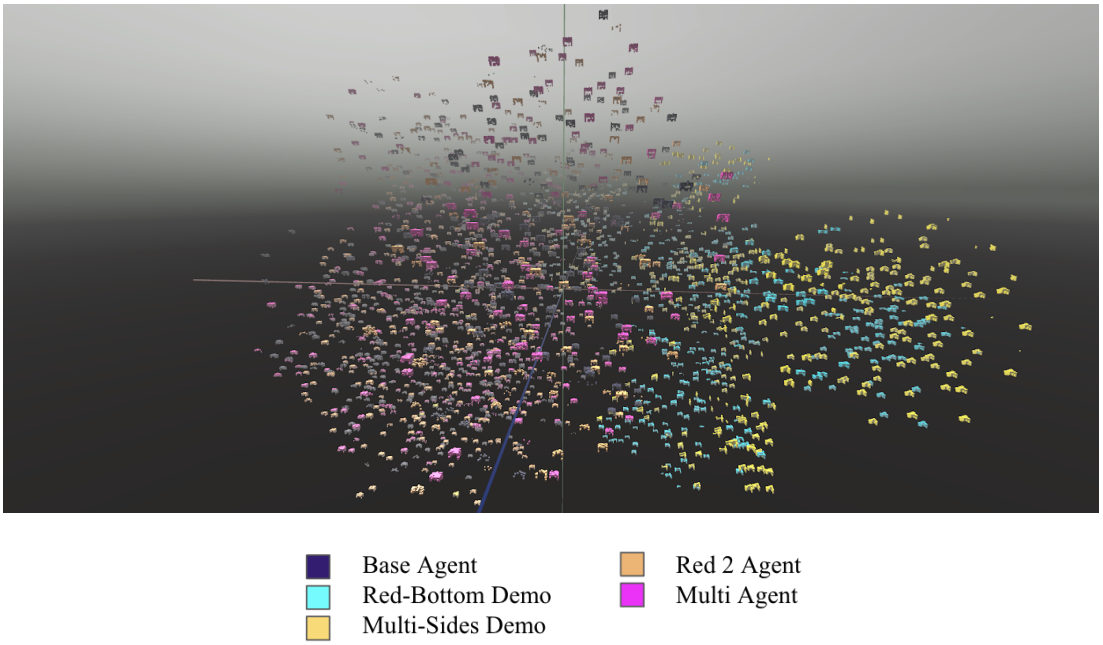
It is evident that each of these three compared agents has different movement patterns within the design space. The Base agent is more concentrated in specific areas in the upper-left of the design space, the Base-Demo 2 agent is more spread out in the upper middle of the design space, and the Demo agent is spread in the lower middle design space. Drawing observations from these static images, what is qualitatively noticeable is the difference between the Base agent

and the Demo agent. This is likely due to the fact that the demonstration used in GAIL used less movement and more head rotation, maintaining a positioning in the bottom half of the design space. Furthermore, when looking at the Base-Demo 2 agent, there is evidence for a clear blend of behaviors between the Base and Demo agent. The Base-Demo 2 agent remains in the upper half of the design space like the Base agent but has more centralized and spread-out movements like the Demo agent. With these heatmaps alone though, it might be too early to make too definite of a claim because we know that the Base-Demo and Demo agents converge to better solutions than the Base agent, so these movement patterns could be a result of that. That being said, we will look at our latent evaluations to help support claims of such correlation.

## 5.2 Latent Evaluation

Latent evaluation is a novel approach introduced to evaluate agent and human behaviors within the same embedded spatial context, building on representational methodologies from Mnih et al.'s [34]. Using PCA and t-SNE, we can create a projected latent space of agent and human action embeddings from an accumulation of recorded action-observation pairs. This space hence represents a relative spatial structure of action sequences taken by each agent given an observation. As a result, we are able to compare the structures and collective coordinates of each individual agent and compare them to each other. This platform for latent evaluation allows us to statically compare agents, allowing us to see where agents take similar sets of actions from a given observation. Furthermore, this section helps reveal insight given the quantitative results of the Guiding Agent and Creator Agent.

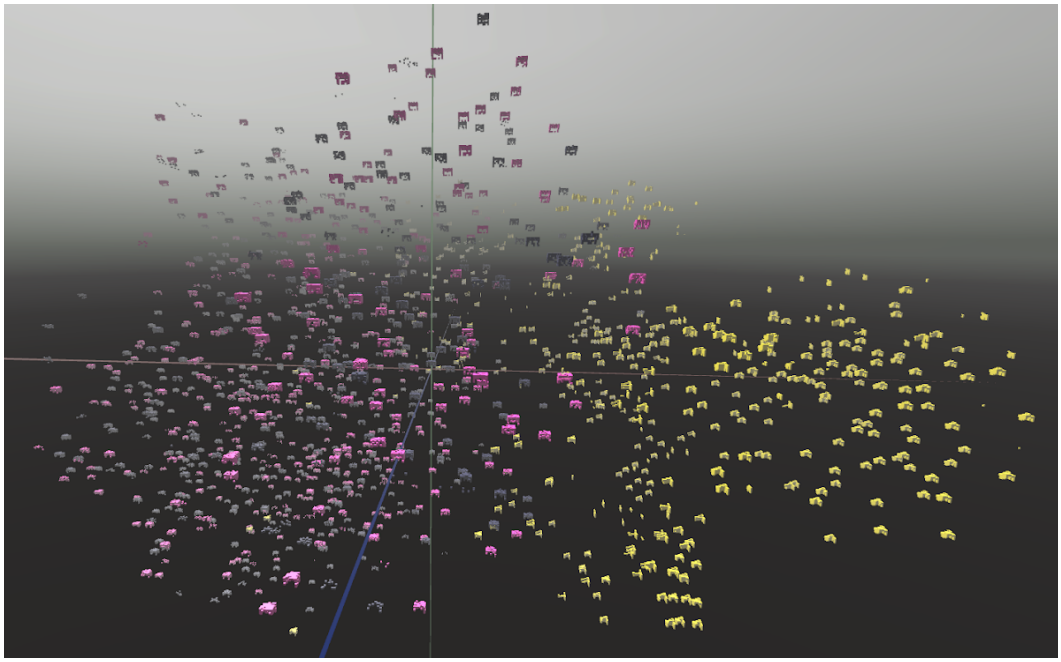
### 5.2.1 Guiding Agent



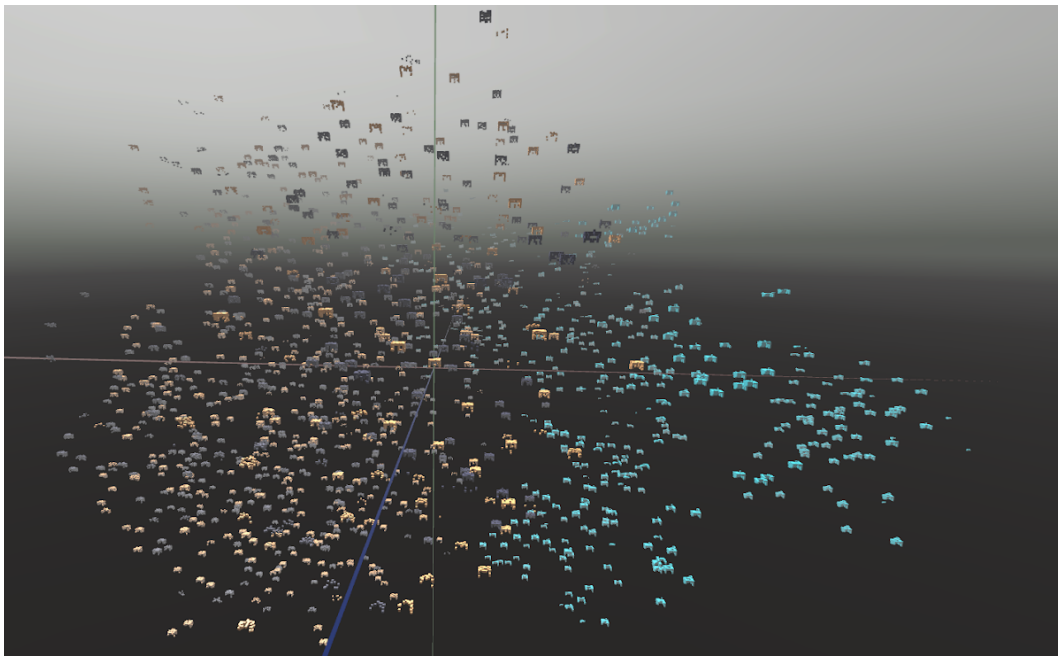

Guiding Agent 3D action embedding space

The overall spatial quality of the Agent's latent action embeddings reveals a noticeable distinction between trained agents and human demonstrations. We can note this based on the distinguishable clustering of both demonstrations as compared to the actions taken by agents trained with RL. All of the agents are defined by a similar spatial structure, meaning that they operate within a similar general logic of action sequences as compared to my demonstrations, which are clearly clustered separately away.

Recalling the action distributions reported in section 5.1.1, it was noted that it was difficult to identify a strong correlation between the actions taken by the agents compared to the human demonstration. A case can be made for explaining this within this latent space structure. Simply, the demonstrations and their corresponding agents take very different action sequences. This is later confirmed by qualitative observations where the stylistic sequence of Block Types embedded in the human demonstration does not translate to the trained agent. This also suggests that there might be too large of a discrepancy in translating the Guiding Agent's heuristic interface for recording demonstrations.

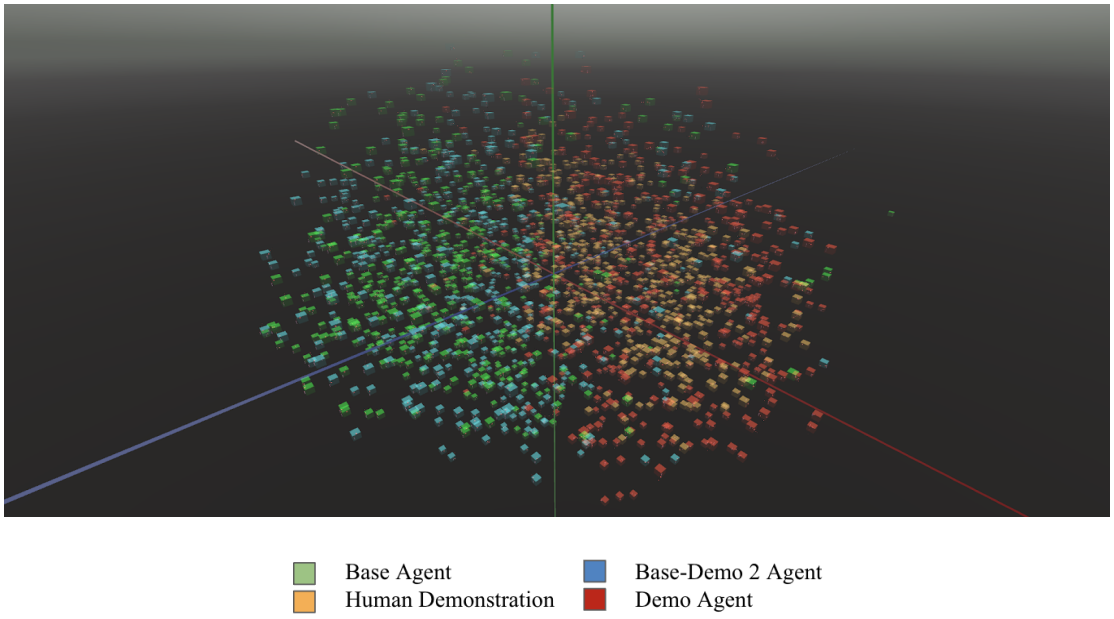


Base and Multi agent to the left, and Multi-Side human demonstration separated on the right



Base and Red 2 agent to the left, and Red-Bottom human demonstration separated on the right

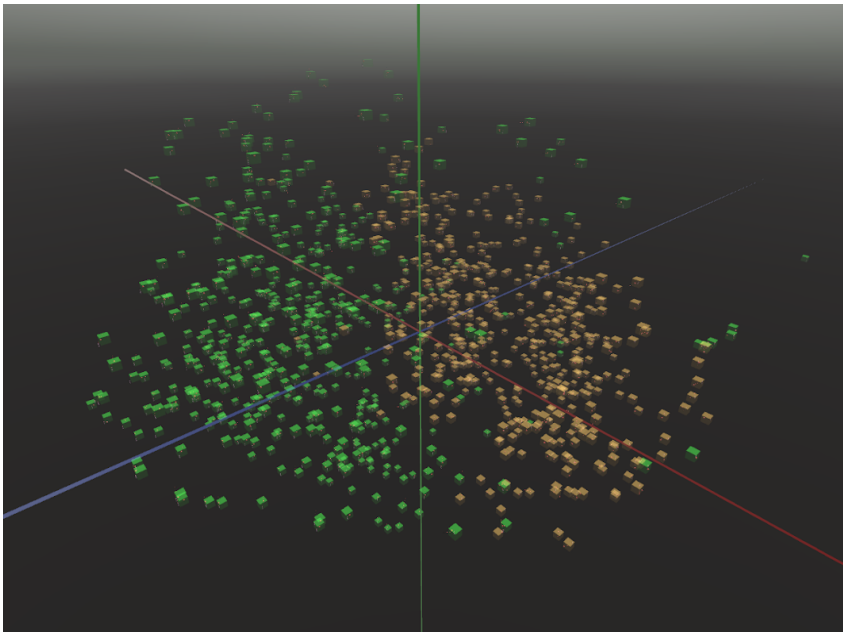
### 5.2.2 Creator Agent



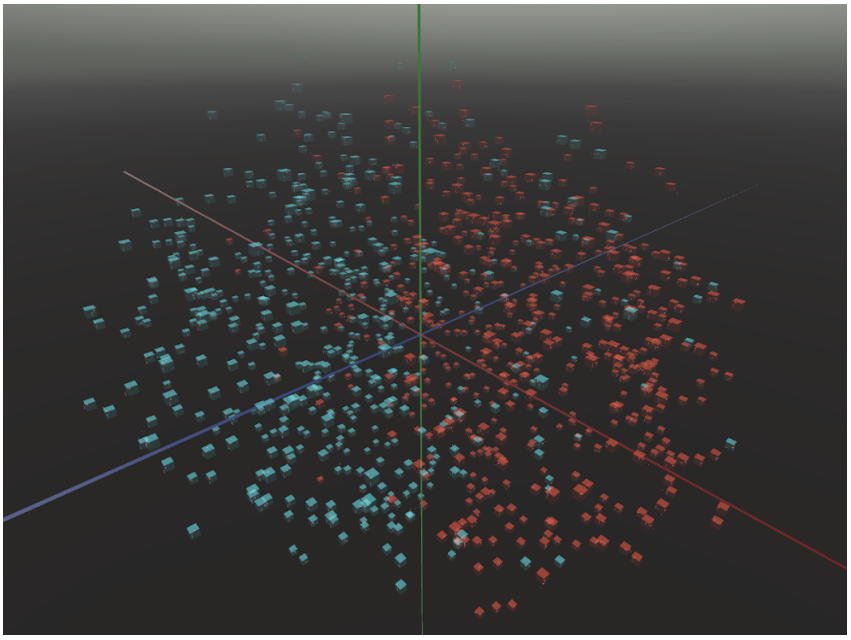

Creator Agent 3D action embedding space

Again referencing the quantitative results, as pointed out in 5.1.2, an argument can be made for there being evidence of the Creator Agent being augmented to behave similarly to my VR demonstration of the same task. Furthermore, the Base-Demo agents displayed evidence of being influenced by additional training with GAIL. These claims can be further evaluated in the context of the Creator Agents' action embeddings, which structurally support these direct claims.

The latent space and its projected embeddings show a very evident divide between the four recorded agent actions used to construct this space. The above image shows along a 2D axis, the four agents split into two distinct clusterings. The first clustering is composed of the Base agent and the Base-Demo 2 agent, and the other clustering is the human demonstration and the Demo agent. The clusterings suggest a distinguishable difference in action sequences between the two clusters. This means that within the relative context of these four recorded actions, the Demo agent learns to take an action sequence that resembles the human demonstration, whereas the Base-Demo agent's action sequences are still far more similar to the Base agent's. What is evident though, is that an agent initialized with BC and trained with GAIL learns a behaviorally similar approach represented in the given human demonstration.



Base agent on the left, and human demonstration on the right



Base-Demo 2 agent on the left, and Demo agent on the right

We also make claims in 5.1.2 that the Base-Demo 2 agent starts to take actions more sequentially similar to the Demo agent. This can be qualitatively observed when comparing the structure of the Base-Demo 2 agent's action embeddings to that of the Base agent. Notice the centrality of the clustering of Base-Demo 2 agent's embeddings, whereas the Base agent's

actions are further and more spread from the center. Distance-wise, the other action embeddings closest to the center are not the Demo agent, but in fact, are the human demonstration. This potentially qualifies the claim that training the pre-trained Base agent with demonstration does affect the behavioral nature of sequential actions taken by the agent and brings it closer to what is shown by the demonstration. Ideally, we want to quantify these claims, where having constructed the latent action space, it is possible to use Euclidean distance to help formulate a metric measuring the total similarity between two different agents. But even without creating quantifiable metrics, there is enough overall visual clarity provided by the latent evaluation to help draw conclusions about agent behavioral relationships at a macro level.

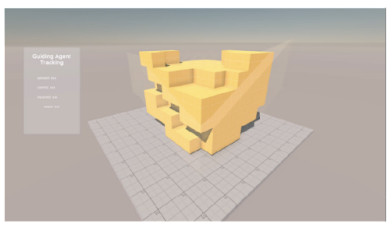
## 5.3 Qualitative Agent Observations

Interactions and designs are inherently visual and qualitative. We can also visually identify behavioral patterns that help distinguish between the agents and their variants. This is potentially the best way to analyze the results of an implicit design process through the demonstration, observing the dynamic likeness of the agent in comparison to the demonstration. Furthermore, what we ultimately are designing for is an experience where the Guiding Agent and Creator Agent are perceived as a singular entity a user co-creates with. We must revisit the contribution these methods have to an immersive user design experience. As a result, questions are prompted to help guide this analysis process, which I answer as a reflection on the qualitative aspects of co-creating with the agent.

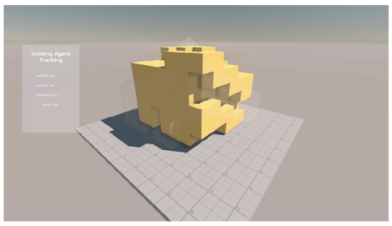
### 5.3.1 Guiding Agent

To understand the qualitative nature of the Guiding Agent variants, these sequences of views are taken at selected arbitrary timesteps of a turntable video recording the building progression to help depict the unique solution between the different agents. The videos from which the images are derived are used for qualitative analysis. The first set of images is to serve as a reference to the views, depicting the complete voxelized version of the objective 3D model.

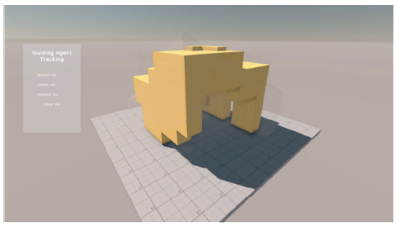
*Ground Truth Views*



6 seconds

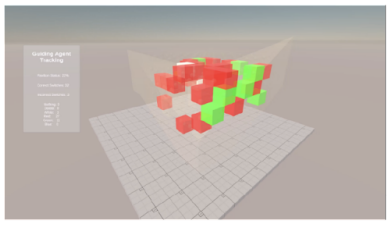


12 seconds

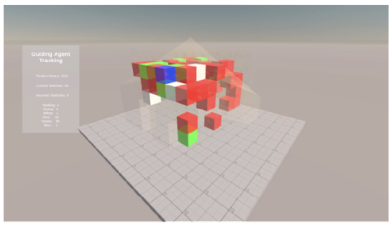


20 seconds

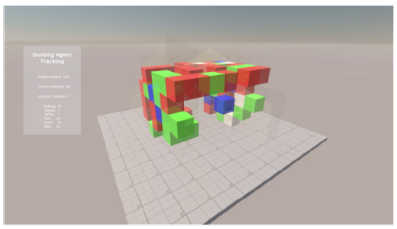
*Base 1 Agent Building Progression*



6 seconds

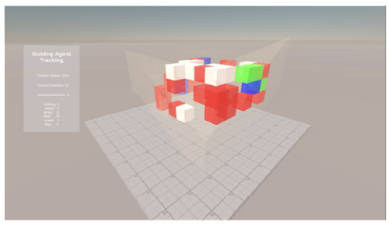


12 seconds

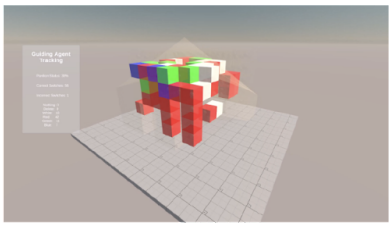


20 seconds

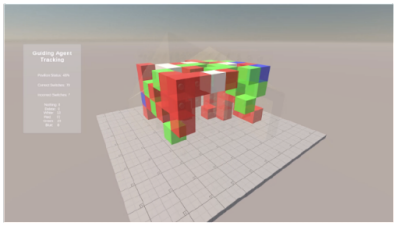
*Base 2 Agent Building Progression*



6 seconds

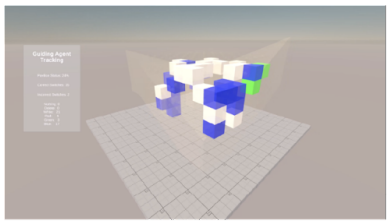


12 seconds

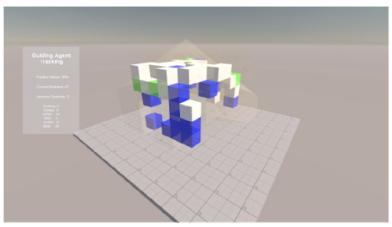


20 seconds

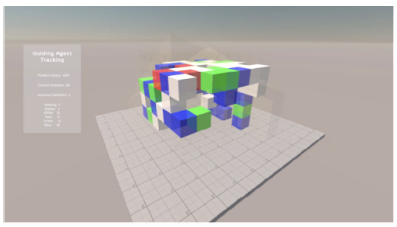
*Red 1 Agent Building Progression*



6 seconds

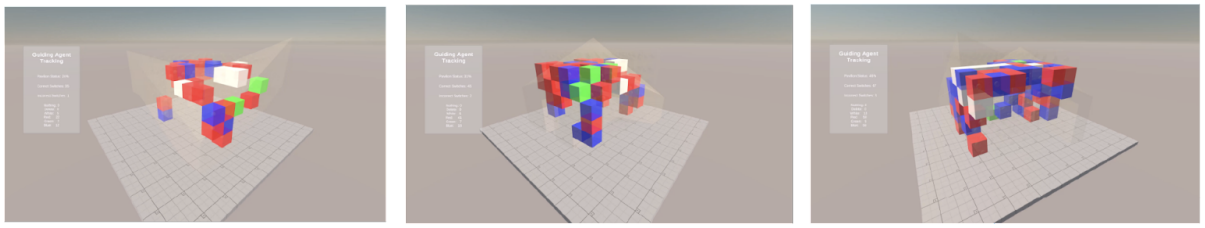


12 seconds



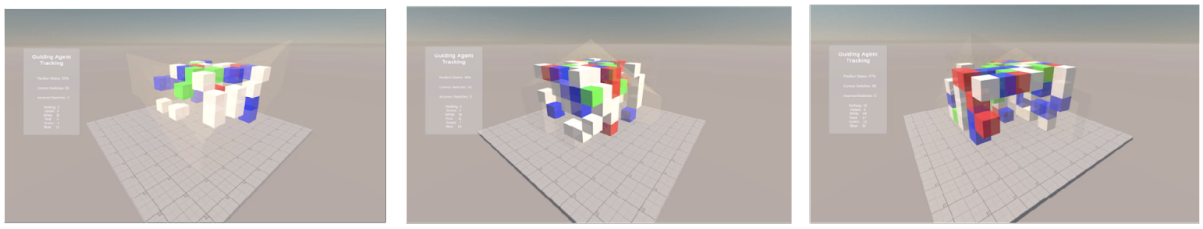
20 seconds

*Red 2 Agent Building Progression*



6 seconds | 12 seconds | 20 seconds

*Multi Agent Building Progression*



6 seconds | 12 seconds | 20 seconds

**Sequential Logic**

Despite efforts to embed a sequential logic within the demonstrations, across all of the agents, they are all prone to starting their construction in the upper-central area of the Block Matrix. This differs from the bottom-up, and sided logic of the two demonstrations. This means that alongside the action distribution proposed by the demonstrations, neither logics of the demonstrations are apparently present in the trained agent. This does not mean that this is not possible in the context of RL, but just within the methods presented in this thesis. A likely reason is that within the observations alone, actions not correlated with the environment's extrinsic reward function might not have enough weighting to translate into the agent's final behavior. This is also in addition to the observations found in the latent evaluation, which reveals a large contrast between human demonstrations and agent action sequences. So did the demonstrations have any effect on the training outcome?

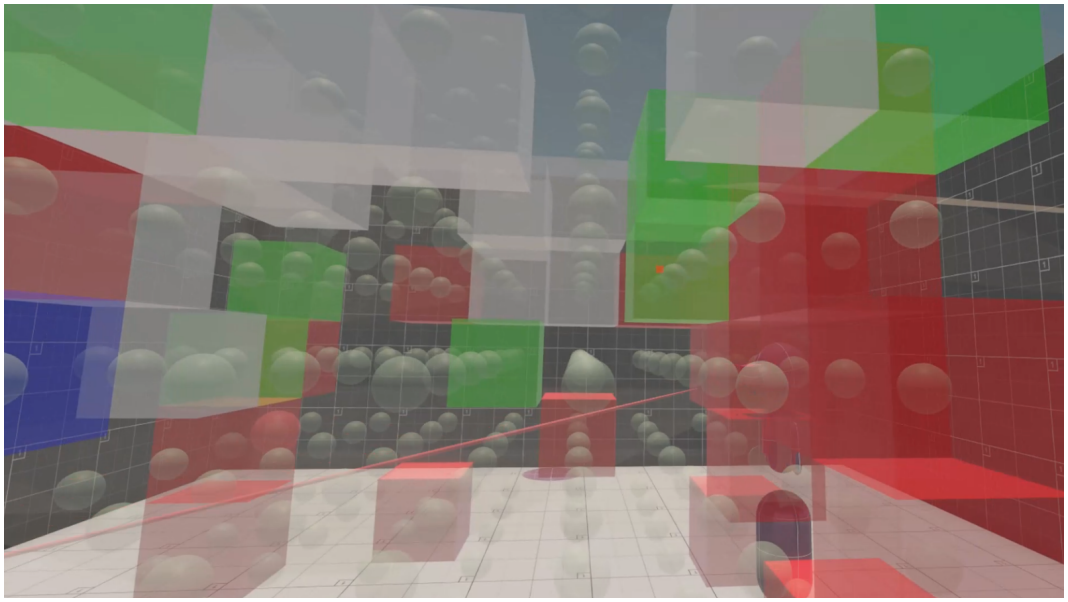
Though more variants are needed to say conclusively that my demonstrations affect the agents, we see a possible sign of this occurring between the two Red agents. Recalling these two variants, both are trained without prior initialization and with different weightings of the same

demonstration, 0.5 for Red 1 and 0.8 for Red 2. Also, recall that Red 2 gets six more block placements correct and two fewer incorrect than Red 1, so naturally will have some differences in the final voxel models. Across the five variants, Red 1 and Red 2 qualitatively have the most similar sequential and final spatial configurations, even more than the Base agents, where Base 2 was initialized from Base 1. This could potentially suggest that the demonstrations do have a consistent effect on the outcome sequential and spatial results of the Guiding Agent despite no sequential logic being made specific in the extrinsic reward function. This though only correlates with the sequential logic rather than the Block Type logic, which likely remains too implicit.

We can consider why this is the case given the training setup of the Guiding Agent. Why a sequential pattern might be more likely to be implicitly learned from a demonstration because placing blocks directly affect the outcome of the extrinsic reward function. Hence the agent can be understood to be learning how the demonstration is solving the problem. In the case of Block Type, aside from the removal block action, which we can note is never used by the agent looking at our actions distributions, the Block Type has no effect on the extrinsic rewards. Furthermore, in an original attempt to make the observation space smaller, the agents do not observe the Block Type of a given Space Switch, but instead just its binary spatial observation. To investigate further the effect of implicit demonstrated action distributions, it would be worth studying the effect of giving the agent observation of the Block Types in the current Block Matrix.

**Spatial Quality**

Within the 7 x 6 x 7 sized Block Matrix, the already-course voxel resolution of the objective 3D model certainly makes for a less detailed comparison in terms of quality between the Guiding Agent's solution to that of the ground truth voxel representation. As a result, the question at this scale matters more from a qualitative design perspective whether the Guiding Agent was able to reconstruct the original model with subjective spatial similarity to a VR user. Also, recall that all the agent variants were only able to construct around 50% of the original model, potentially impeding the ability to capture the original model's spatial features.



View in VR inside an agent-reconstructed 3D model

Despite this lack of completeness, I find that across the Guiding Agent variants, the agents do capture the general features of the given objective 3D model, which happens to be its three walls, two of which are cut angularly, its one open side, and the roof. This is not necessarily a guarantee that at 50% completeness, we see the gist of the objective model. An example of the agents not capturing the objective model is in the case that the agent only completed the roof and not the walls. Instead, blocks are generally scattered across the model giving a sparse but spatially complete reading. This might be helped with further training, but what this indicates is that using RL searches throughout the entire 3D space of possibilities with general success.

### 5.3.2 Creator Agent

**Performance**

We can observe qualitatively the solutions of the agent relative to average rewards, which helps provide insight as to what agents struggle with in particular situations and why some agents perform better than others. The visible difficulties observed in VR of the agents are quite clear and correspond well to the recorded heatmap videos.

| Agent | Final Reward | Observed Issues |
|---|---|---|
| Base | 8300 | Struggles with high targets, getting stuck for a varying number of seconds looking close to the target but not at it to trigger it |
| Base-Demo 1 | 9840 | Occasional issue described in the Base agent, and when stuck does not stay stuck for a long time (<1 second) |
| Base-Demo 2 | 9792 | Similar to Base-Demo 1 |
| Demo | 9288 | Does not struggle with any particular targets, but comparative to the other agents is slower at finding targets and hence is rewarded less |

Notably in the Base agent, it gets close to aligning its gaze with the target Switch Trigger. But because the movement inputs are discrete and do not allow for finer continuous control of movement, the agent will find that a movement input will displace its gaze further from the Switch Trigger. As a result, the agent will remain in its position, head rotated as much as possible to the maximum of 45°, moving its body back and forth until activating the trigger. This is a likely explanation for some of the movement patterns observed in the heatmaps for the Base agent, consisting of strong concentrated areas where likely the agent was stationary looking for an upper Switch Trigger. With further training though, evident in the Base-Demo agents, we see this behavior less present with a more performant learned solution.

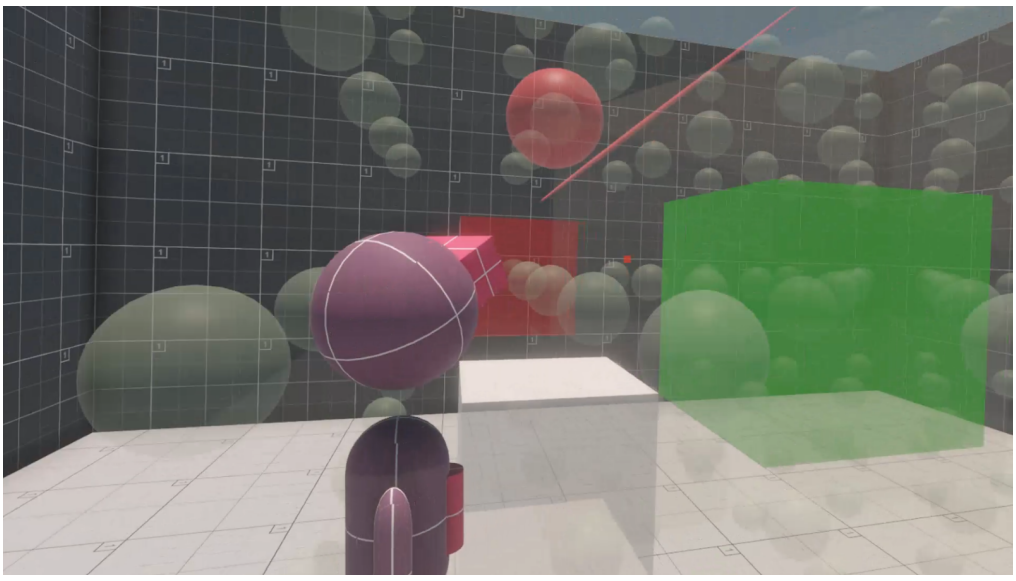
**Behavior Comparisons**

Noticeable in both VR and in the abstracted heatmap view, the agents trained from the Base agent vs the Demo agent have very different movement behaviors when aligning their gaze with the Switch Target. The Base and Base-Demo agents employ a strategy where moving and rotating the body of the agent was preferred, typically reducing the distance between the agent to the target Space Switch. In contrast, in my demonstration, I use a different strategy, where I do not move my VR avatar's body unless I felt like I needed a better view of the target, and instead just turned my head to align my gaze to the target. I would also extend that my demonstration represents a more natural human VR user's actions, where head movement is often preferred over using the joystick to control the human user's avatar. The qualities of my demonstration are visually evident in the solution of the Demo agent.

In contrast to the Base agents, the Demo agent prefers its head rotation more to finding targets, moving its body less. What stands out characteristically is noting what the agents do next after receiving a new target Space Switch. Simply, the Base agents move their bodies and the Demo agent moves its head. This is especially noticeable due to the parametrized movement factors of the agents, which are constant during training and inference. Input for body movement causes greater displacement than the input for head rotation. As a result, we notice jagged movements from the Base agent, whereas the Demo agent is comparatively smoother and more direct, including its body movement. Though the Demo agent is not performing the solution in the most optimal way in comparison to the Base-Demo agent, it is apparent that is performing a solution that is more human-like to the demonstration, which is also quantified by the observed heatmap and latent evaluation. This has an effect on co-creation and this observation will be more thoroughly explained.

## 5.4 Reflection on Co-Creation

To better observe the qualitative observations of co-creating with the variants is done more so as a personal reflection from a set of given questions generated to help reflect critically on the overall experience created by these methodologies. This is due to a lack of facilitating a proper user test in the co-creation environment.



View in VR next to the Creator Agent

*How does the spatial relationship between you and the agent variants affect the way you operate in the design space?*

Operating in the design space without the agent as compared to with the agent is a very different experience. The VR experience strongly conveys the spatial presence of the agent, especially due to the scale of the agent matching your own. As a result, you cannot help but notice the agent in your own space, and naturally, you make space for the agent, conscious to keep separation as you would with other human beings in a physical space. The agent has no knowledge and hence no reaction towards your presence, but you do it and this is strongly apparent. Interestingly though, the degree to which the agent's presence intrudes on your own varies between the agent variants.

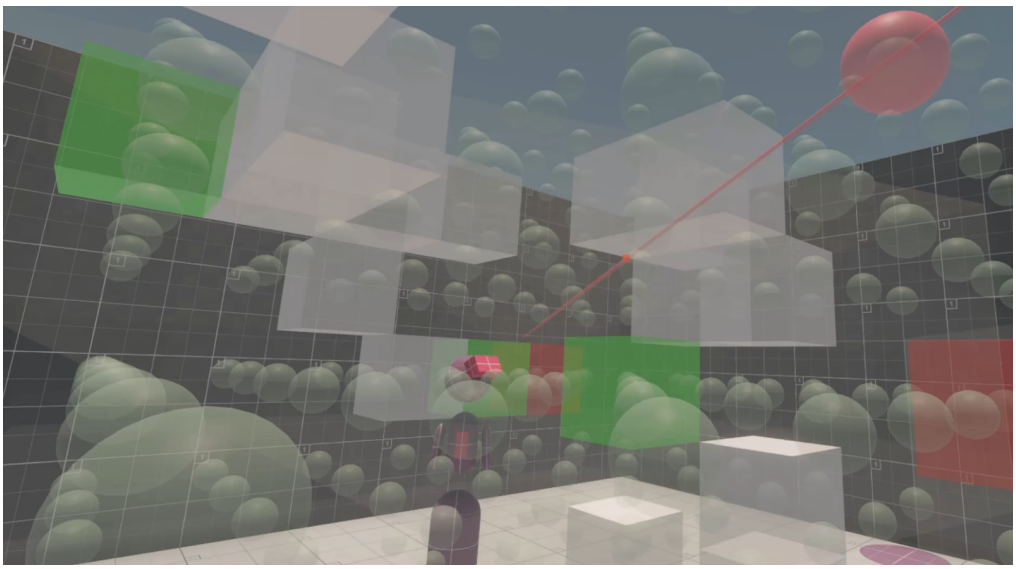
When comparing variants, my own subjective feeling is far more comfortable working in the design space with the Demo agent in comparison to the Base agents. This is due to the movement behaviors of the Demo agent being less erratic, and the Demo agent's movement patterns being more predictable. You can more easily see that the Demo agent will stick to moving in one direction, and often when it moves into the design space, it moves back out of it. Though there is still keep a spatial degree of separation between the Demo agent and myself, I definitely felt like I could contribute to the Block Matrix without having to always remain conscious of the agent I am collaborating with. This contrasts the Base agent that notably moves within the center of the Block Matrix, leaving yourself often at the periphery of the design space.

*What do interactions through the Block Matrix look like?*

Part of the intent of this thesis was to use the Block Matrix as a medium for silently interacting with the agent, similar to the Silent Game. When reflecting on the interactions between the agent through the Block Matrix, it becomes evident the lack of nuance these agents possess. A sequence of the design is learned by the agent and once learned, from a qualitative perspective in VR, is repeated between iterations, despite my contribution to the design in the Block Matrix. Too, because none of the agents learn to remove blocks, which is understandable since that was never introduced into the agent's training curriculum, the agents will not remove your block placements despite the capability to do so.

The most notable interaction that occurred in the Block Matrix is when I removed blocks that the agent placed. The agent after placing a few blocks usually would come back to replace

the block that you had removed. Also, when the agent is finished placing blocks, after removing five or more blocks, the agent would begin placing blocks again. From a design perspective, it is hard to negotiate with the agent in this regard, where you feel like as a collaborator, you can only add to the Block Matrix rather than edit what currently existed. From a human-AI interaction standpoint, it did not make me feel too uneasy, in fact, it made me feel like I was instead working with an adolescent. I would describe the situation as a child playing with legos with a design very well envisioned in their head that they are not willing to compromise on. I felt like the agent did have a sense of what it wanted to build, it just has not adapted yet to be capable of understanding the concerns of other actors in the same design space.



Observing the agent in VR

*Do you spend more time observing the agent rather than building with the agent?*

I spent more time observing the agent in VR, that being said, my concerns and perspective are skewed toward evaluating and being critical of the agent's development. I wanted to pose this question because it would have been essential in asking during a hypothetical user test. From my experience, I would imagine that because the agent's presence is so strongly felt, and the nature of the design interaction itself relatively simple, the agent would occupy much of the user's attention. As a result, I would rather conduct an experiment that played with this interaction between user building and observing more deliberately.

In response to considering this question, a small implementation I added was to give the user button controls of the agent's presence directly on their VR controller. From my own experience, being able to toggle the agent on and off and actively choosing when to clear the Block Matrix felt more conducive for an iterative design process with the agent. It does make the agent feel like a tool, or there to ‘fill in the blanks’, but until the agent becomes more aware and reactive of your presence, this is the best design interaction to facilitate between the user and the agents trained in this thesis.

*What are the differences between observing the agent in VR versus in the editor?*

The difference in observation is that agents are felt spatially, and their movements affect your psyche much further. As a result, the details to make the agents more behaviorally comfortable are important, and hence I personally felt that the approach to use human demonstrations in this context is well validated and one method for creating a better user experience in VR.

Another point worth noting is that development occurs in 2D on your computer’s development platform, which in contrast to designing traditional screen-based video games, and perhaps more similar to architectural design, the design work applied is experienced within another perceptual lens. Because of this change in platforms, I think there is a valued gain in implicit design methods, such as using the demonstration to train RL agents with GAIL. Such methods can help translate design intent between the target platform to the development platform where data from the target platform is richly embedded in nuances difficult to explicitly recreate.

# 6. Conclusion

In the conclusion of this thesis, the primary goal is to reflect on the results of the methodologies of this thesis, assessing the insights they might provide in understanding human-agent co-creation and strategies for developing agents for novel virtual platforms. In much of the previous sections, the distinction between Guiding Agent and Creator Agent is made clear from a developmental standpoint, but ultimately, the perceived agent the user interacts with is a synthesized version of the two. More importantly, the reflection's goal is to evaluate the qualitative effect their synthesis has on the co-creation process in VR. By doing so, it allows us to further speculate on what this thesis suggests for future research for the many overlapping disciplines concerned with the prototype.

## 6.1 Evaluating Methods

I recall the entire process of the thesis prototype, reflecting on how certain implementations came to be and my personal thoughts on their outcome. The thesis was intentionally explorative from the start, with no clear method for implementing or designing the prototype, just clear motivations. As a result, the evolution of the methodologies is key to this work as a body of research, including my own criticality on the iterations of development that shaped the final prototype.

### 6.1.1 Developmental Process

Not made explicit in the Methods section are the steps taken to get to the conclusive implementation we see for the co-creation agent. As a result, an important part of this thesis is reflecting on the overall design process of its many parts, comparing the initial ideas behind implementations to their final outcome.

**Mediating Between User and Agent**

At the original outset of this thesis, the template given by Habraken and Gross' Silent Game [22] motivated the idea that one can create a similar situation at the scale of moving game pieces in VR. Instead of the 'design space', it was instead more of a 'playroom', focused on simulating

tangible interactions to interface with an abstract game of placing blocks within a tabletop-scaled grid space. As a result, the agent would learn hand movement and controls, which are typically the means of interacting in VR, and what I previously explored in *Imitation Ball*. The issue faced with this setup was the prioritization of developing an agent that learned human VR gestures, rather than design concepts. It also, in my opinion, does not offer enough insight regarding a design exploration that took advantage of VR's unique spatial immersion.

The conclusion to increase the scale and make the design interaction more bodily was for creating a unique way of designing that required active spatial awareness from both the user and agent. For both a user and an agent to be spatially aware and to use that awareness to perform design moves, was thought of as a means to better express the novelties and possibilities made available by virtual platforms like VR.

**Training a Co-Creating Agent**

Initially, the idea was to train one network using RL that could control an agent capable of both locomotion and design moves given the objective 3D model. The decision that eventually ended up in developing two separate training environments and networks came as a result of an evolving process due to the lack of success found in training a single capable network. The original setup was to train the agent with a single vector observation that included its own pose, whether it was looking at a Switch Target or not, and then the 256 additional binary spatial values representing the Block Matrix. This was also coupled with the agent's actions being a combination of Creator Agent and Guiding Agent, being required to navigate its own gaze to a Switch Trigger and decide to make a Block Type selection or not.

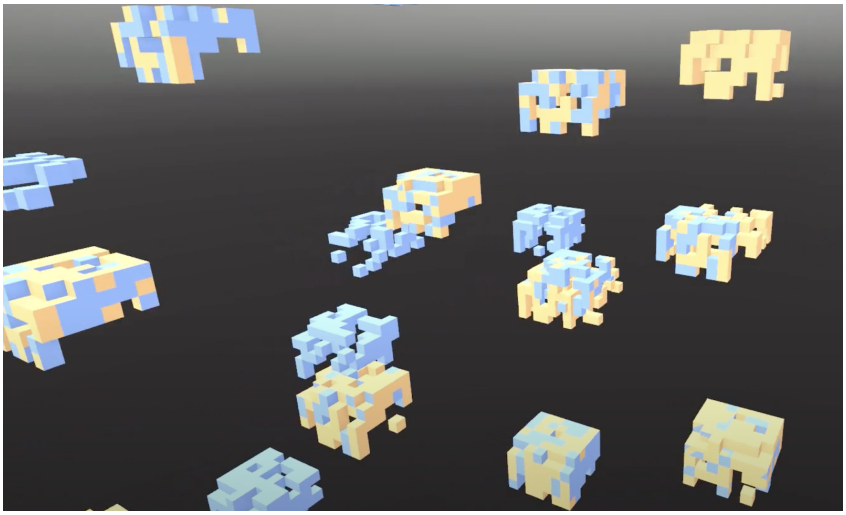
From experience, given an RL task, there are two ways to address the problem if the agent is not able to learn it. Firstly, question whether the action and observation space can be made smaller, and secondly, whether there is more environmental information you can offer the agent without resorting to explicit heuristics. Thinking through this approach and a number of iterations is how the Creator Agent and Guiding Agent emerged to help the research of this thesis.

On this topic, another question to reflect upon is the possibility that there was a way to train a single neural network capable of what the Creator Agent and Guiding Agent do

separately. Given this hypothetical scenario, from the results of this thesis, I believe that in many ways it would have posed some difficulties in being critical about the design and evaluation of these agents. What was learned was the great benefit of creating different treatments for the distinct locomotion and design task, from the training setups, demonstrations, and evaluative methods. Furthermore, by having decoupled networks, one can also iterate specific qualities of the overall agent into separate environments. In the context of designing agents for the purpose of design problems, breaking down the agents in such a manner is arguably essential.

**Action as Sequences for Latent Evaluation**

The original implementation of creating the latent embeddings did not include a factor of *t* previous actions taken. As a result, the action embeddings often were overlapping between agents, which was likely a given since the action space is relatively small across both agent types. To make the embeddings richer in contextualizing the actions relative to each other, the introduction of making the embeddings a result of *t* previous actions helped distinguish these actions. Furthermore, this method also speaks well to the time horizon parameter in PPO where *n* previous actions are accounted for in the reward given by the current action. This means that agent actions are learned as sequences rather than in isolation. Furthermore, the sequential factor of actions taken largely indicated the unique behaviors of both agent types. This allows for more informed readings and clusterings in the latent evaluation, especially aiding evidence supporting the Creator Agent's causal relationship to the human demonstration.



Early iteration of the Guiding Agent action embeddings where overlapping embeddings are visible

### 6.1.2 Latent Evaluation

The latent evaluation in this thesis takes a novel approach to use t-SNE for creating clusterings that are visually expressive of its source embeddings and for helping compare action sequence behaviors between agents. It was successful because even though it was a relatively simplistic implementation, the latent space generated was very helpful in creating a static method for comparing otherwise active agents.

The latent space representation has the potential to provide far richer information for the sake of evaluating agents between each other. The thesis already had much to gain from the obvious qualitative structures of the latent action embeddings, but that is not the say that there are greater subtleties to be more closely examined and filtered within this space. To realize these subtleties, comparative metrics would be necessary to develop as well as further exploration into the user interface and interaction with the embeddings.

Another reflection on the latent evaluation is the representations of the observations themselves and how they lead to a micro-scale evaluation of the agents. I found that between the Guiding Agent and Creator Agent, the Guiding Agent had much more to benefit from this feature. Being able to compare the spatial configuration associated with the embedding to its nearby neighbors revealed a clearly legible contrast or similarity between agents at the level of action sequences taken. For example, one can compare at what level of completeness of the model one agent to the other decides to place a block at a similar 3D coordinate. This latent evaluation is certainly interesting and could potentially be important for future research in RL and 3D design.

There is one major drawback of the latent evaluation and that is that the latent space is only relative to the given data entries used to construct the space. That means whenever you would want to compare a new agent to a current space, you would need to reconstruct the space including the new agent's action data. Also, given the way t-SNE works, generating a space out of all possible entries might not be a desirable approach if, for instance, you want a finer distinction between two agents trained with similar parameters. It is likely that in that instance you would want to generate a space of just those two agents' action embeddings. In conclusion, the latent evaluation method holds a lot of promise but could be further explored as a dedicated

application with greater functionalities to support future design research at different levels of comparative granularity.

### 6.1.3 Developing and Performing Demonstrations

Instead of talking further about the impact of the demonstrations, this reflection instead comments on the act of doing the demonstrations. I wanted to approach this reflection from a highly personal perspective because my realizations were unexpected until recording myself doing the demonstrations. What occurred was that I began to understand the task from the perspective of the agent, and also, actively reflect on how differently the task was perceived and performed between myself and the agent.

To recall, recording yourself doing a demonstration is done by stepping into the agent's shoes, similar to the agent being a puppet. As a result, there were steps to implementing this translation between my natural actions to that of the agent. This is especially seen in the implementation of the Creator Agent heuristic controls. In my initial implementation, after translating VR controls to control the agent, I had the in-scene camera setup positioned such that it was centered on the agent's gaze so that what I saw in VR mimicked what the agent was seeing based on its gaze direction. What I realized trying to create the demonstration from the agent's perspective was how imprecise the discrete controls had in controlling the gaze of the agent, not allowing for the fine control of gaze that we as human VR users are accustomed to. Whether this is considered empathizing or not, I realized the difficulty of the Creator Agent to be precise and hence tuned the agent's head rotation speed to slow down the rate of rotation to allow for finer control.

This was a significant discovery for me because not only was it unexpected but it shed a different perspective on the dynamic of human-AI interaction. Due to the affordance of VR, instead of performing the demonstrations being just a means to an end, it allowed me to understand the agent's perspective better. It gave me practical insight into improving training but also an indescribable feeling of association with the agent I did not expect to feel.

### 6.1.4 Data Feedback

A theme addressed in the Background was drawing from information theorists on creating novel ways of visualizing and understanding the physicality of data. Because theorists like Loukissas try to find novel methods for contextualizing the physical world with its digital counterpart, this thesis being contextualized by virtual environments and hence creating digital data, had the opportunity to create novel visualizations unconstrained the way one would be collecting data in the physical world. Contemplating such methods for visualization data from virtual environments may before not have been so necessary due to the explicit nature of design virtual environment interactions and AI finite state machines. But in the context of deep learning and implicit methods of designing agents, these methods do become necessary.

We see this, particularly in creating the Creator Agent heatmaps. Not only were they helpful in abstracting the agent's movement patterns, but reflecting on the method for developing these visualization poses an opportunity for approaching visualizations unique to observations in virtual environments. For example, the agent and its head can be abstracted to just arrows, and potentially other aspects of its body and actions can graphically be described. But these graphics can always exist on the agents, it is just a matter of what camera we are displaying of the agent and what that particular camera sees in the scene.

Another way we see data explored with a novel lens is in developing the spatial features of the latent evaluation. An inspiration for framing the latent space as a means for comparing agents was introduced by Offenhuber, who suggests recontextualizing material artifacts to better compare their traces to one another. There is a semblance to this idea, and it is translated into virtual space, where materiality can be perceived differently at more malleable scales and representations. For instance, historical instances of a room-scaled pavilion be scaled to the relative size of a point in a clustered dataset. And developing a data format for identifying these traces allowed for the phenomenon of agent actions to be interpreted and fed back into a cycle of analysis and design iteration.

## 6.2 Limitations

The span of this thesis's topics is expansive and as it attempts to draw connections between a number of different themes, it too left a lot more to be desired in exploring more thoroughly the capabilities of its implementations.

### 6.2.1 Design-Capable Agent

In order to create an agent capable of designing for the practical purposes of this thesis, the thesis does not look to explore a critical definition of what designing might look like in the context of RL and deep learning in general. The implementation of the Guiding Agent is in fact very simplistic, to the extent that both technically and conceptually further work can be done to expand upon the idea that it suggests.

Technically we can quantifiably note that none of the agents learn more than 50% of the objective 3D model, meaning that there could be improvements in the current training setup. Especially if a finer resolution Block Matrix is desired, revisiting the work of Onformative [38] might be a better technical template to model the Guiding Agent's implementation. Furthermore, the actions taken by the Guiding Agent have little sequential relevance to the extrinsic environment reward, which is a core aspect of using PPO.

Conceptually, we can ask whether the Guiding Agent is actually an effective design aid. Is a supervised approach to learning a configuration a productive contributor to informing the design process, let alone the co-creation, of a user? That being said, I think that it is possible for an agent to be developed that understands certain abstract nuances, similar to the agents developed in MIA [11]. And though it is important to highlight the potential for such interactions, admittedly, the implementation in this thesis is limited in effectively capturing an agent with dynamic design sensibility.

### 6.2.2 Lack of Agent Variants

The claims made in this thesis are made on account of a small number of agents trained, and demonstrations made to explore more extensively agent variants. This is largely due to the

emphasis in this thesis to produce novel speculative human-AI interactions and evaluative methods. Consequently, the results in this thesis show potential for further exploration with these developed methodologies but could be better reinforced by a further investigation into the level of nuanced observations these methods can formulate about agent behaviors.

This thesis also does not go further to explore what the iterative process might look like for designing the agents. Given the results presented by the evaluative methods developed, how would they have informed the next iteration of agents? If having explored this question, this thesis would have perhaps been able to draw a better conclusion regarding the relationship between the role of design as a conduit for understanding the development of human-AI interaction.

### 6.2.3 User-Testing

As the sole researcher and developer of this thesis, my highly biased perspective offers little to understanding how users of different backgrounds would interpret the experience of interfacing with an agent in this co-creation VR environment. Furthermore, users of different design and engineering backgrounds might too be able to offer insight into the evaluative methods of this thesis, whether it is well understood or not at providing insight into the agent's behaviors.

Also, the lack of creating a structured user test meant that the thesis paid little attention to stitching together its parts into a single experience and package. As the developer of this overarching system, I have a perception of how its parts are connected, but this may not be as obvious depending on the user. Considering who the users are and how they interact with these methodologies would help ground them in more concrete design processes and contexts.

## 6.3 Contributions

To summarize the contributions of this thesis:

- Examines a range of techniques regarding RL and virtual environments for deep learning training, offering a novel lens for exploring their overlap with more open-ended design objectives

- Demonstrated that immersive technologies like VR are posed to be an accessible method for capturing a greater dimension of human data for deep learning applications
- Implements an application and development cycle that promotes evaluating RL agents qualitatively through their behaviors on the basis of design and human-AI interaction
- Proposes a novel 3D interface for evaluating RL agent action sequences between agent variants and human demonstration
- Highlights the domain gap that needs further exploring when training deep learning models to perform actions for open-ended stylistic design objectives
- Contributes personal reflections that aids exploring reactions to this novel dynamic of human-agent interaction in an immersive design space

To generalize these contributions, the thesis hopes to not necessarily find the optimal methods for developing applications using these technologies nor be too detailed in its approach, but instead work to produce new types of questions that speculate on the relationship of specific emerging technologies. To reiterate the motivations of the thesis, little work has been done to situate VR, 3D virtual environments, and deep learning together under a critical designerly lens. The hope is to create work that can inform both technical and design perspectives pertaining to human-AI collaboration because both have an equal stake in what is an emerging trend and interest. By developing a thesis that crosses and sits at the intersection of domains, I hope that my unique position can help establish common methodologies and ways of thinking that help better connect the technical and the design practitioner.

## 6.4 Future Work

Here is a wide range of proposed future directions that can be explored given the questions raised by this thesis. They range from addressing the technical needs of this thesis to exploring design methodologies with AI and forming a criticality on human relationships with technology. These brief ideas are more so suggestions for future areas that can be explored in a variety of different research directions.

### 6.2.1 Translating the Human-Agent Domain Gap for Design

As seen with the Guiding Agent, the implementation used in this thesis did not sufficiently capture human demonstration enough to augment agent behavior. This is likely due to the implementation of the agent itself and the lack of implementing a way to better translate the user's design interface into data that could be better transferable to RL training. As a result, I suggest three ways to address this domain gap given the prototype of this thesis.

The first method for approaching this problem is to implement more intentional ways of filtering the demonstration data from the user. In this instance, we treat the user and Guiding Agent the same, when in fact more work needed to be done to accommodate their differences. A simple strategy for doing so would be to omit instances where a user does not make a design decision, making the user's actions appear more similar to that of a trained Guiding Agent.

Secondly, we can look at the agent training environment itself to simulate what a user within a design interface would more realistically be doing. The Guiding Agent had the ability to remove blocks, but would consciously not because it is far better reward-wise to just build as correctly as possible. In real-life circumstances for 3D modeling, is often an iterative process. A user tries things and deletes what they do not like. Extending the design environment to facilitate more dynamic training where the agent learns to delete actions would be a step towards this. This potentially might be implemented through multi-agent training, where two agents occupy the same design space with different and hence competing design objectives, by which they must add and remove blocks accordingly.

Lastly, they are also training methods in RL we can consider, like that of curriculum learning introduced in the Background section, and that of inverse RL [1][5]. Observed in the implementation of the Guiding Agent was that its extrinsic rewards were not explicit enough to convey ideas like Block Type distribution and construction sequences. This can be introduced into the reward function but should be carefully balanced to remain mostly implicit. A way to do so is in inverse RL, which considers that instead of letting a reward function suggest a behavior, the desired behavior can come first, influencing the desired reward function. This gives additional functionality to human demonstrations, helping inform the reward function and contribute to training via GAIL

### 6.2.2 Latent Evaluation as Holistic Evaluative Method

The 3D visual method of the latent evaluation served as an early-stage prototype that could definitely be expanded into more functionality for the user's exploring visualizing the differences between high-dimensional datasets. Here are two different ways to further extend the latent evaluative method introduced by this thesis much of which draws from my earlier project *A Walk in Latent City*.

The first would be to explore quantifiable metrics for comparing embeddings in the latent space. For example, in *A Walk in Latent City*, users can select two embeddings at once and see the difference in measured Euclidean distance between the two. In the constructed latent space, a smaller relative distance means more similarity between embeddings. Questions to consider would be how to expand these measurements to better inform agent comparisons, for instance, using the averaged sum of distances between all entries between two agents. Of course, this can also be used at smaller scales, but using measured metrics allows for a greater depth of details if one wanted to use this method to explore more nuanced differences between agents.

Secondly, we can look to investigate latent evaluations in design workflows and integrate this method within the digital design platform itself. This would mean a level of interactivity that might include dynamic parameterization and generation of latent spaces with t-SNE. For instance, a user could be designing a 3D model, and during their design process, wonder how their current design fits into a relative context with the design of other users and AI. They could load their model's design data into a database of other designs and generate a visual context of their design with others. This is just one example of many ways an integrated latent evaluation model can inform design iteration, especially in the content of human-AI collaboration.

### 6.2.3 VR Data for Deep Learning and Design

As explored prior to this thesis in *Imitation Ball*, from an interaction standpoint, especially as it relates to VR and immersion, developing agents using data from VR can allow for agents that are behaviorally more similar to humans. There are obvious concerns with this dynamic from a data privacy perspective, but instead, I would like to suggest an alternate perspective that integrates VR, design, and deep learning.

The thesis highlights the difficulty of capturing the nuances of co-creation with AI. That being said, developing agents with projected behaviors is clearly possible. One idea to take advantage of this capability is to heighten the level of realism in simulations that help inform design decisions. For example, let us say we want to test designed spatial configurations that will include many human actors. Instead of having many users in VR explore the space, one could design agents that resemble VR avatars, like in this thesis, and train them with different VR demonstrations in order to both increase their performance and give them a variety of human-like behaviors. Unlike this thesis, this keeps the VR user and VR separate. Exploring the interactivity between human VR users and AI is a whole other topic that can also be further explored but will be harder to do within the context of design.

### 6.2.4 Revisit the Concept Design Game and Co-Creation

The goal of this thesis was to provide a novel lens regarding the emergence of co-creation and humans. Though this thesis tried to eliminate the notion that AI is just simply a tool, an embodied AI agent still can be as such, similar to how I implemented the control to turn off and on or pause the agent. The concept of an AI agent within design acting like automotive sentence completion is an intriguing concept to look further into, particularly in the realm of 3D design. And too, this would also not have to be done so to explicitly create the image that the human and agent are equal actors, it can be more similar to working with just the Guiding Agent.

I believe that the Concept Design Games still have much to offer in investigating this dynamic, and furthermore, could produce novel ideas for AI-integrated design interfaces. In retrospect, the Silent Game might be too implicit regarding a human-AI relationship, given that it was designed with the intent for human-human gameplay. Rules more conducive to human-AI relationships might be more promising as a research framework. For instance, the human player makes five moves, the agent makes five moves, and so on. Even an additional detail to the game could make the investigation of the agent both explicit and easier to evaluate.

Another way to make reinforce the idea of design sequences is to also introduce different types of pieces, where a piece is not just a color but instead a different shape such that each piece affects the site in a different way, similar to *Stacky-Block*. If pieces were to affect the site differently, this would extend the unique quality of a given move and its subsequent ones, rather

than just basing it on block color. This exploration would also allow for designs that require more complex execution, which should aid in pushing the limits of human-AI design collaboration.

# References


1. Abdulhai, Marwa, Natasha Jaques, and Sergey Levine. “Basis for Intentions: Efficient Inverse Reinforcement Learning Using Past Experience.” arXiv, August 9, 2022. https://doi.org/10.48550/arXiv.2208.04919.
2. Abramson, Josh, Arun Ahuja, Federico Carnevale, Petko Georgiev, Alex Goldin, Alden Hung, Jessica Landon, et al. “Improving Multimodal Interactive Agents with Reinforcement Learning from Human Feedback.” arXiv, November 21, 2022. https://doi.org/10.48550/arXiv.2211.11602.
3. Ali, Safinah, Nisha Devasia, and Cynthia Breazeal. “Designing Games for Enabling Co-Creation with Social Agents.” arXiv, May 1, 2021. https://doi.org/10.48550/arXiv.2105.00201.
4. Argall, Brenna, Brett Browning, and Manuela Veloso. “Learning by Demonstration with Critique from a Human Teacher.” In *2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 57–64, 2007. https://doi.org/10.1145/1228716.1228725.
5. Arora, Saurabh, and Prashant Doshi. “A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress.” arXiv, November 18, 2020. https://doi.org/10.48550/arXiv.1806.06877.
6. Baker, Bowen, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. “Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos.” arXiv, June 23, 2022. https://doi.org/10.48550/arXiv.2206.11795.
7. Borovikov, Igor, Jesse Harder, Michael Sadovsky, and Ahmad Beirami. “Towards Interactive Training of Non-Player Characters in Video Games,” June 3, 2019. https://doi.org/10.48550/arXiv.1906.00535.
8. Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. “OpenAI Gym.” arXiv, June 5, 2016. https://doi.org/10.48550/arXiv.1606.01540.
9. Burden, David J. H. “Deploying Embodied AI into Virtual Worlds.” In *Applications and Innovations in Intelligent Systems XVI*, edited by Tony Allen, Richard Ellis, and Miltos Petridis, 103–15. London: Springer, 2009. https://doi.org/10.1007/978-1-84882-215-3_8.
10. Carter, Marcus, and Ben Egliston. “What Are the Risks of Virtual Reality Data? Learning Analytics, Algorithmic Bias and a Fantasy of Perfect Data.” *New Media & Society*, May 9, 2021, 14614448211012794. https://doi.org/10.1177/14614448211012794.
11. DeepMind Interactive Agents Team, Josh Abramson, Arun Ahuja, Arthur Brussee, Federico Carnevale, Mary Cassin, Felix Fischer, et al. “Creating Multimodal Interactive Agents with Imitation and Self-Supervised Learning.” arXiv, February 2, 2022. https://doi.org/10.48550/arXiv.2112.03763.
12. Deitke, Matt, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, et al. “RoboTHOR: An Open Simulation-to-Real Embodied AI Platform.” arXiv, April 14, 2020. https://doi.org/10.48550/arXiv.2004.06799.
13. Donalek, Ciro, S. G. Djorgovski, Alex Cioc, Anwell Wang, Jerry Zhang, Elizabeth Lawler, Stacy Yeh, et al. “Immersive and Collaborative Data Visualization Using Virtual Reality Platforms.” In *2014 IEEE*

*International Conference on Big Data (Big Data)*, 609–14. Washington, DC, USA: IEEE, 2014. https://doi.org/10.1109/BigData.2014.7004282.

14. Dosovitskiy, Alexey, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator." arXiv, November 10, 2017. https://doi.org/10.48550/arXiv.1711.03938.
15. Dourish, Paul. *The Stuff of Bits*. Accessed October 5, 2022. https://mitpress.mit.edu/9780262546522/the-stuff-of-bits/.
16. Duan, Jiafei, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. "A Survey of Embodied AI: From Simulators to Research Tasks," March 8, 2021. https://doi.org/10.48550/arXiv.2103.04918.
17. Dyrstad, Jonatan S., and John Reidar Mathiassen. "Grasping Virtual Fish: A Step towards Robotic Deep Learning from Demonstration in Virtual Reality." In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 1181–87, 2017. https://doi.org/10.1109/ROBIO.2017.8324578.
18. Gao, Xiaofeng, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. "VRKitchen: An Interactive 3D Virtual Environment for Task-Oriented Learning." arXiv, March 13, 2019. http://arxiv.org/abs/1903.05757.
19. Ghorbani, Saeed, Ylva Ferstl, Daniel Holden, Nikolaus F. Troje, and Marc-André Carbonneau. "ZeroEGGS: Zero-Shot Example-Based Gesture Generation from Speech." arXiv, September 23, 2022. https://doi.org/10.48550/arXiv.2209.07556.
20. Giorgio, Andrea de, Mario Romero, Mauro Onori, and Lihui Wang. "Human-Machine Collaboration in Virtual Reality for Adaptive Production Engineering." *Procedia Manufacturing*, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy, 11 (January 1, 2017): 1279–87. https://doi.org/10.1016/j.promfg.2017.07.255.
21. Guss, William H., Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, et al. "The MineRL 2020 Competition on Sample Efficient Reinforcement Learning Using Human Priors." arXiv, January 26, 2021. http://arxiv.org/abs/2101.11071.
22. Habraken, N. John, and Mark D. Gross. "Concept Design Games." *Design Studies* 9, no. 3 (July 1, 1988): 150–58. https://doi.org/10.1016/0142-694X(88)90044-0.
23. Ho, Jonathan, and Stefano Ermon. "Generative Adversarial Imitation Learning." arXiv, June 10, 2016. https://doi.org/10.48550/arXiv.1606.03476.
24. Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, et al. "Unity: A General Platform for Intelligent Agents." arXiv, May 6, 2020. https://doi.org/10.48550/arXiv.1809.02627.
25. Kanervisto, Anssi, Janne Karttunen, and Ville Hautamäki. "Playing Minecraft with Behavioural Cloning." In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, 56–66. PMLR, 2020. https://proceedings.mlr.press/v123/kanervisto20a.html.
26. Kanervisto, Anssi, Joonas Pussinen, and Ville Hautamäki. "Benchmarking End-to-End Behavioural Cloning on Video Games." In *2020 IEEE Conference on Games (CoG)*, 558–65, 2020. https://doi.org/10.1109/CoG47356.2020.9231600.

27. Kantosalo, Anna, and Hannu (TT) Toivonen. “Modes for Creative Human-Computer Collaboration: Alternating and Task-Divided Co-Creativity,” 2016. https://www.semanticscholar.org/paper/Modes-for-Creative-Human-Computer-Collaboration%3A-Kantosalo-Toivonen/f504a2d2c751a7208bc311a19d0c0d6b96f08d45.
28. Koganti, Nishanth, Abdul Rahman H. A. G., Yusuke Iwasawa, Kotaro Nakayama, and Yutaka Matsuo. “Virtual Reality as a User-Friendly Interface for Learning from Demonstrations.” In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–4. CHI EA ’18. New York, NY, USA: Association for Computing Machinery, 2018. https://doi.org/10.1145/3170427.3186500.
29. Liu, Siqi, Guy Lever, Zhe Wang, Josh Merel, S. M. Ali Eslami, Daniel Hennes, Wojciech M. Czarnecki, et al. “From Motor Control to Team Play in Simulated Humanoid Football,” May 25, 2021. https://doi.org/10.48550/arXiv.2105.12196.
30. Lohr, Dillon J, Samantha Aziz, and Oleg Komogortsev. “Eye Movement Biometrics Using a New Dataset Collected in Virtual Reality.” In *ACM Symposium on Eye Tracking Research and Applications*, 1–3. ETRA ’20 Adjunct. New York, NY, USA: Association for Computing Machinery, 2020. https://doi.org/10.1145/3379157.3391420.
31. Loukissas, Yanni Alexander. *All Data Are Local: Thinking Critically in a Data-Driven Society*, 2019. https://doi.org/10.7551/mitpress/11543.001.0001.
32. MacAlpine, Patrick, and Peter Stone. “Overlapping Layered Learning.” *Artificial Intelligence* 254 (January 1, 2018): 21–43. https://doi.org/10.1016/j.artint.2017.09.001.
33. Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing Atari with Deep Reinforcement Learning.” arXiv, December 19, 2013. https://doi.org/10.48550/arXiv.1312.5602.
34. Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. “Human-Level Control through Deep Reinforcement Learning.” *Nature* 518, no. 7540 (February 2015): 529–33. https://doi.org/10.1038/nature14236.
35. Narvekar, Sanmit, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. “Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey.” arXiv, September 17, 2020. https://doi.org/10.48550/arXiv.2003.04960.
36. Nikolenko, Sergey I. *Synthetic Data for Deep Learning*. Vol. 174. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-75178-4.
37. Offenhuber, Dietmar. “Data by Proxy -- Material Traces as Autographic Visualizations.” arXiv, July 11, 2019. http://arxiv.org/abs/1907.05454.
38. “Onformative — AI Sculpting.” Accessed December 15, 2022. https://onformative.com/work/ai-sculpting.
39. Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, et al. “Open-Ended Learning Leads to Generally Capable Agents.” arXiv, July 31, 2021. http://arxiv.org/abs/2107.12808.

40. OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, et al. “Dota 2 with Large Scale Deep Reinforcement Learning.” arXiv, December 13, 2019. https://doi.org/10.48550/arXiv.1912.06680.
41. Peng, Bei, James MacGlashan, Robert Loftin, Michael L. Littman, David L. Roberts, and Matthew E. Taylor. “Curriculum Design for Machine Learners in Sequential Decision Tasks.” *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, no. 4 (August 2018): 268–77. https://doi.org/10.1109/TETCI.2018.2829980.
42. Puig, Xavier, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. “VirtualHome: Simulating Household Activities via Programs,” June 19, 2018. https://doi.org/10.48550/arXiv.1806.07011.
43. Ramakrishnan, Santhosh K., Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, et al. “Habitat-Matterport 3D Dataset (HM3D): 1000 Large-Scale 3D Environments for Embodied AI.” arXiv, September 16, 2021. https://doi.org/10.48550/arXiv.2109.08238.
44. Reiners, Dirk, Mohammad Reza Davahli, Waldemar Karwowski, and Carolina Cruz-Neira. “The Combination of Artificial Intelligence and Extended Reality: A Systematic Review.” *Frontiers in Virtual Reality* 2 (2021). https://www.frontiersin.org/articles/10.3389/frvir.2021.721933.
45. Risi, Sebastian, and Julian Togelius. “Increasing Generality in Machine Learning through Procedural Content Generation.” *Nature Machine Intelligence* 2, no. 8 (August 2020): 428–36. https://doi.org/10.1038/s42256-020-0208-z.
46. Ros, German, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. “The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes.” In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3234–43, 2016. https://doi.org/10.1109/CVPR.2016.352.
47. Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms.” arXiv, August 28, 2017. https://doi.org/10.48550/arXiv.1707.06347.
48. Stelldinger, Peer. “A Competitive 3D-Volleyball-Game Using Reinforcement Learning with Unity ML-Agents.” In *Autosys*, 2022. https://autosys.informatik.haw-hamburg.de/publication/2022sabrinag%C3%B6llner/.
49. Szot, Andrew, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, et al. “Habitat 2.0: Training Home Assistants to Rearrange Their Habitat.” arXiv, July 1, 2022. https://doi.org/10.48550/arXiv.2106.14405.
50. Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.” In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23–30, 2017. https://doi.org/10.1109/IROS.2017.8202133.
51. Todorov, Emanuel, Tom Erez, and Yuval Tassa. “MuJoCo: A Physics Engine for Model-Based Control.” In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–33, 2012. https://doi.org/10.1109/IROS.2012.6386109.

52. Vinyals, Oriol, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, et al. “StarCraft II: A New Challenge for Reinforcement Learning,” August 16, 2017. https://doi.org/10.48550/arXiv.1708.04782.
53. Wood, Erroll, Tadas Baltrusaitis, Charlie Hewitt, Matthew Johnson, Jingjing Shen, Nikola Milosavljevic, Daniel Wilde, et al. “3D Face Reconstruction with Dense Landmarks,” April 6, 2022. https://doi.org/10.48550/arXiv.2204.02776.
54. Wu, Yi, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. “Building Generalizable Agents with a Realistic and Rich 3D Environment.” arXiv, April 8, 2018. https://doi.org/10.48550/arXiv.1801.02209.
55. Wu, Zhuohao, Danwen Ji, Kaiwen Yu, Xianxu Zeng, Dingming Wu, and Mohammad Shidujaman. “AI Creativity and the Human-AI Co-Creation Model.” In *Human-Computer Interaction. Theory, Methods and Tools*, edited by Masaaki Kurosu, 171–90. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-78462-1_13.
56. Xie, Xu, Hangxin Liu, Zhenliang Zhang, Yuxing Qiu, Feng Gao, Siyuan Qi, Yixin Zhu, and Song-Chun Zhu. “VRGym: A Virtual Testbed for Physical and Interactive AI.” arXiv, April 2, 2019. https://doi.org/10.48550/arXiv.1904.01698.
57. Zherdev, Denis, Larisa Zherdeva, Sergey Agapov, Anton Sapozhnikov, Artem Nikonorov, and Sergej Chaplygin. “Producing Synthetic Dataset for Human Fall Detection in AR/VR Environments.” *Applied Sciences* 11, no. 24 (January 2021): 11938. https://doi.org/10.3390/app112411938.

# Appendix A: Glossary of Custom Scripts

## Agent

*AgentCSVWriter*: A script attached to either the Creator or Guiding Agent that can record agent or demonstration actions and its given observation to individual CSV files at set frame intervals.

*CameraHeuristicGuide*: For demonstration, this script designates a camera for the Guiding Agent that users control with their mouse, navigating the design environment and facilitating point-and-click selection in the Block Matrix.

*CreatorHeuristicVR*: A script attached to the Creator Agent that connects the user's VR rig to the agent allowing for VR inputs to be used for creating demonstrations.

*CreatorAgent*: Inheriting from the *Agent* class in the ML-Agents toolkit, this script provides all of the functionality for the Creator Agent, from reward, actions, and observations.

*GuidingAgent*: Inheriting from the *Agent* class in the ML-Agents toolkit, this script provides all of the functionality for the Guiding Agent, from reward, actions, and observations.

## Environment

*ArrowTracker*: A simple script attached to arrow sprites that track the Transforms of the Creator Agent's head and body.

*CameraCas*t: A script attached to the user's VR camera that determines whether or not the user is looking at a Switch Target or not, if so, which Space Switch it belongs to.

*HandPalette*: A script for controlling the functionality of the Palette Watch, indicating to the environment which is the user's current Block Type.

*Pavillion*: This script is attached to a 'wild' 3D model imported into Unity and brought into the design space such that it will register which Space Switches it intersects within the Block Matrix.

## Latent Space

*Camera3D*: A camera script that allows users to navigate in latent space, as well as click embeddings and view their metadata.

*LatentBlockPoint*: A script for the Guiding Agent observations, reading in a given binary vector from the observation CSV and reconstructing a corresponding representation of the Block Matrix.

*LatentParent*: Parameterizes the parent GameObject that contains an agent variant's embeddings, and initializes all of that agent's embeddings in the latent space at runtime.

*LatentScenePoint*: A script for the Creator Agent observations, reading in a given vector from the observation CSV and reconstructing a corresponding representation of the agent's pose in relation to its target Space Switch.

## Managers

*GameManager*: Public singleton meant for storing global references in the co-creation environment.

*InputManage*r: Public singleton that handles all VR input functionality in the co-creation environment.

*SoundManager*: Public singleton that provides functions for playing sounds.

## Switch

*SpaceSwitch*: Attached to the cube mesh that represents a Space Switch, this script keeps track of the current Block Type, changing colors and type upon agent or user selection.

*SwitchParen*t: Script for initializing the Block Matrix by a given dimension, parenting the grid of Space Switches, and keeping track within a public list, the current state of all of the Space Switches.

*SwitchTarget*: Script that tracks whether the parent Space Switch is the current switch or not based on whether the agent or user is gazing at it.

# Appendix B: Training Parameters

The following are text from the YAML files used to set up the training parameters of the various RL agent variants.

## Guiding Agent

| Base Agent Variants | Demo Agent Variants |
|---|---|
| behaviors:<br>  CreatorAgent:<br>  trainer_type: ppo<br>  hyperparameters:<br>  batch_size: 2048<br>  buffer_size: 20480<br>  learning_rate: 0.0003<br>  beta: 0.01<br>  epsilon: 0.2<br>  lambd: 0.95<br>  num_epoch: 3<br>  learning_rate_schedule: linear<br>  network_settings:<br>  normalize: false<br>  hidden_units: 256<br>  num_layers: 2<br>  vis_encode_type: simple<br>  reward_signals:<br>  extrinsic:<br>  gamma: 0.9<br>  strength: 1.0<br>  keep_checkpoints: 5<br>  max_steps: 50000000<br>  time_horizon: 128<br>  summary_freq: 50000 | behaviors:<br>  CreatorAgent:<br>  trainer_type: ppo<br>  hyperparameters:<br>  batch_size: 2048<br>  buffer_size: 20480<br>  learning_rate: 0.0003<br>  beta: 0.01<br>  epsilon: 0.2<br>  lambd: 0.95<br>  num_epoch: 3<br>  learning_rate_schedule: linear<br>  network_settings:<br>  normalize: false<br>  hidden_units: 256<br>  num_layers: 2<br>  vis_encode_type: simple<br>  behavioral_cloning:<br>  Demo_path: <local path><br>  strength: 0.5<br>  steps: 5000<br>  reward_signals:<br>  extrinsic:<br>  gamma: 0.9<br>  strength: 1.0<br>  gail:<br>  gamma: 0.99<br>  strength: 0.8<br>  learning_rate: 0.0003<br>  use_actions: false<br>  use_vail: false<br>  Demo_path: <local path><br>  keep_checkpoints: 5<br>  max_steps: 25000000<br>  time_horizon: 128<br>  summary_freq: 50000 |

# Creator Agent

| Base Agent Variants | Demo Agent Variants |
|---|---|
| behaviors:<br>  CreatorAgent:<br>  trainer_type: ppo<br>  hyperparameters:<br>  batch_size: 2048<br>  buffer_size: 20480<br>  learning_rate: 0.0003<br>  beta: 0.01<br>  epsilon: 0.2<br>  lambd: 0.95<br>  num_epoch: 3<br>  learning_rate_schedule: linear<br>  network_settings:<br>  normalize: false<br>  hidden_units: 256<br>  num_layers: 2<br>  vis_encode_type: simple<br>  reward_signals:<br>  extrinsic:<br>  gamma: 0.9<br>  strength: 1.0<br>  curiosity:<br>  strength: 0.02<br>  gamma: 0.99<br>  keep_checkpoints: 5<br>  max_steps: 50000000<br>  time_horizon: 128<br>  summary_freq: 50000 | behaviors:<br>  CreatorAgent:<br>  trainer_type: ppo<br>  hyperparameters:<br>  batch_size: 2048<br>  buffer_size: 20480<br>  learning_rate: 0.0003<br>  beta: 0.01<br>  epsilon: 0.2<br>  lambd: 0.95<br>  num_epoch: 3<br>  learning_rate_schedule: linear<br>  network_settings:<br>  normalize: false<br>  hidden_units: 256<br>  num_layers: 2<br>  vis_encode_type: simple<br>  behavioral_cloning:<br>  Demo_path: <local path><br>  strength: 0.5<br>  steps: 5000<br>  reward_signals:<br>  extrinsic:<br>  gamma: 0.9<br>  strength: 1.0<br>  gail:<br>  gamma: 0.99<br>  strength: 0.8<br>  learning_rate: 0.0003<br>  use_actions: false<br>  use_vail: false<br>  Demo_path: <local path><br>  keep_checkpoints: 5<br>  max_steps: 25000000<br>  time_horizon: 128<br>  summary_freq: 50000 |