POLITECNICO
MILANO 1863

SCHOOL OF ARCHITECTURE URBAN PLANNING CONSTRUCTION ENGINEERING

Master's Program Building and Architecture Engineering - Architecture

MASTER'S THESIS

TOWARDS REVIVING THE MASTER BUILDER

Autonomous design and construction using deep reinforcement learning.

Thesis Submitted by:
Ahmed Elmaraghy
Matricola: 914566

Supervisor:
Prof. Pierpaolo Ruttico

Co-supervisors:
Prof. Marcello Restelli
Dr. Jacopo Montali
Prof. Francesco Causone

Academic Year 2021 - 2022

Acknowledgment

To the love of my life, the only thing that made sense in this life I am living. To my mentor, my best friend, and my role model, To you Grandpa. Wish you were here now, yesterday and every single day of my life.

I would like to thank all who helped me with finally finishing this thesis.

I would like to start with thanking my supervisor, Prof. Ruttico for encouraging and supporting me during that period and for his continuous presence for advice. The same goes to my co-supervisors Prof. Restelli, Dr. Montali (who helped me keep going when I was about to bail out), and Prof. Causone. Special thanks to Prof. Masera for his advice and support during the whole duration of my program. Prof. Tadi for his instant help when I needed. Thank you all for your time and consideration.

Thanks to Arcoquattro Family, special thanks to Professors Franck and Massimo for their support during this period.

I would also like to thank all my friends, brothers, sisters and colleagues who supported me in that period:

Moustafa, Nadia, Costanza, Omar, Batran, Shoukry, Rashad, Samy, Tugrul, Mohadaseh, Bahaa, Helia,

Thanks to ETO family, Dearest friends and Principals who supported me for achieving this while working: Roberta, Roberto and Mr.Franchi (Feltri Marone), Andrea (IBS), and My Dear Mary from the office.

To my family, I can't thank you enough for everything. Thanks to my always supportive Uncle, my parents, my Aunt, my sister (who helped me in proof reading the Thesis, so please blame her for any mistakes :)), my lovely dog.

Finally, I would like to thank myself for bearing me all this period, for the tons of working and studying weekends that I had… You deserve some vacation, whenever possible.

Table of Contents

List of Figures

Sintesi

Il Deep Reinforcement Learning ha recentemente iniziato ad essere adottato nella ricerca architettonica. L'applicazione è stata principalmente focalizzata sulla risoluzione di compiti specifici sia nella costruzione che nell'ottimizzazione della progettazione. Data l'unicità di ogni progetto architettonico e la necessità di adattare diversi parametri di progettazione alle mutevoli caratteristiche ambientali e ai vincoli del sito, la natura di tali progetti potrebbe trarre vantaggio su scala più ampia dal successo dell'apprendimento per Reinforcement Learning nell' ambiente di complessi giochi strategici. Questa ricerca sperimenta l'uso di algoritmi di apprendimento per Reinforcement Learning in un ambiente simulato, abilitato alla fisica, per manipolare i vincoli di progettazione e sito verso la realizzazione di una struttura completa.

Questo processo è guidato da obiettivi gerarchici progettati dall'uomo (architetto) per realizzare un progetto appropriato. L'agente quindi cerca di capire un metodo appropriato per raggiungere questi obiettivi di progettazione sulla base dei parametri ambientali, rivedendo contemporaneamente l'effetto del suo comportamento e delle sue azioni sulla costruibilità del progetto.

Di conseguenza, questa ricerca introduce un nuovo quadro per la progettazione e la costruzione autonoma di un progetto. In tale contesto, l'architetto è disaccoppiato dai compiti di progettazione dettagliata ed è più sfidato a trasformare la conoscenza architettonica tacita in una sequenza di obiettivi che sarebbero sfruttati da un agente e implementati da robot di costruzione per realizzare questi obiettivi in una struttura abitabile reale. Pertanto, il ruolo dell'architetto si propone di essere spostato verso una forma rinnovata di " Mastro Costruttore " che si occupa piuttosto della "supervisione" del comportamento degli agenti e della definizione degli obiettivi strategici generali per il raggiungimento di un progetto di successo.

Come prova di concetto, questa ricerca dimostra un progetto di padiglione in mattoni sviluppato in Unity con un ambiente di fisica simulata e utilizza il framework di apprendimento per Reinforcement Learning abilitato da ML-Agents. Gli obiettivi sono divisi in obiettivi di progettazione e costruzione. Gli obiettivi progettuali comprendono, ad esempio, il raggiungimento di una corretta ventilazione naturale attraverso la scelta della posizione, dell'altezza e del numero delle aperture e del loro corretto allineamento con la direzione prevalente del vento o selezionando la giusta porosità o interasse tra i mattoni in base alle finalità architettoniche richieste in ogni fase. D'altra parte, gli

obiettivi di costruzione sono più legati al rispetto delle regole basate sulla fisica quando si posizionano i mattoni per evitare il collasso e ottimizzare la sequenza di posizionamento verso un minor consumo di energia da parte dell'agente.

Abstract

Deep Reinforcement learning has recently started to be adopted in Architectural research. The application has been mainly focused on solving specific tasks whether in construction or in design optimization. Given the uniqueness of each architectural project and the need to adapt different design parameters to the changing environmental characteristics and site constraints, the nature of such projects could benefit on a larger scale from the success of deep reinforcement learning in playing complex strategic games. This research experiments the use of state-of the art reinforcement learning algorithms in a simulated, physics enabled, environment to manipulate design and site constraints towards realizing a complete structure.

This process is guided by hierarchical goals designed by the human (Architect) to realize an appropriate design. The agent then tries to figure out an appropriate method to achieve these design goals based on the environment parameters while reviewing simultaneously the effect of its behavior and actions on the constructability of the project.

Accordingly, this research introduces a new framework for autonomously designing and constructing a project. In such a framework, the architect is decoupled from detailed design tasks and is more challenged to transform the tacit architectural knowledge into a sequence of goals that would be exploited by an agent and implemented by construction robots to realize these goals into a real habitable structure. Therefore, the architect's role is proposed to be shifted towards a newly revived form of a "Master Builder" that is rather concerned with the "supervision" of the agents' behavior and laying down the general strategic goals for achieving a successful design.

As a proof of concept, this research demonstrates a brick pavilion project developed in Unity with a simulated physics environment and uses the deep reinforcement learning framework enabled by ML-Agents. The goals are divided into design and construction driven goals. The design goals include, for instance, the achievement of proper natural ventilation through the selection of the position, height and number of openings and their proper alignment with prevailing wind direction or selecting the right porosity or spacing between bricks based on the required architectonic purposes in each phase. On the other hand, construction goals are more related to obeying the physics-based rules when placing bricks to avoid collapse and optimize the placement sequence towards less energy consumption by the agent.

Chapter 1

State of the Art

1.1. Artificial Intelligence and Machine Learning

1.1.1. Introduction to Artificial Intelligence

Despite the common use of the term “artificial intelligence” (AI) on many occasions, it turns out that it is actually difficult to have a consensus on a precise nature of its subject matter. The problem, in contextual point of view, could be addressed by stating how we could conceive the 2 words comprising this term; namely: Artificial and Intelligence [1].

What is considered to be “artificial” about artificial intelligence, has to be linked with its mode of creation in which a human contrivance has influenced the existence of such an “intangible form” through the creation and invention of “tangible” devices. This is opposed to something that is entirely incubated in nature or as a result of biological and evolutionary influence. These creatures would then possess a “natural” intelligence rather than an “artificial” one.

This latter hypothesis would lead to the exclusion of human beings from being “artificial” machines in case we could consider ourselves as machines in the first place. This term could fit on ourselves if we simply consider machines as things that are capable of performing work. Given that humans are able to do work and perform tasks, then they could be considered as machines as well.

At that point, there should be a distinction between animate and inanimate machines given that human beings remain biological in their origin. This would propose a more difficult question whose answer is less evident and is concerned with the second word of the term AI, namely: “Intelligence”.

The question is: Can “inanimate” or “artificial” machines acquire intelligence similar to the one exhibited by an “animate” machine (like human beings)? A great aspect of the difficulty of this question lies in defining the boundaries of what to counted by human beings as a sign of an “intelligent” behavior.

Therefore, if we start with the definition from the dictionary, according to Merriam Webster the term “Intelligence” is defined as [2]:

1. “The ability to learn or understand or to deal with new or trying situations”
2. “The ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria”

From the first definition, we could consider Turing Test as a test bed for the ability of a "machine" to try learning and understanding "human communication". It can then use what it learnt to be able to develop sort of "thinking" and "interaction" similar to that of humans. This is demonstrated by the ability of a machine to mimic human responses under specific conditions without getting "caught" or being recognized by a human candidate as a computer respondent.

During the test, three physically separated terminals are operated by 2 humans and 1 computer respectively. The goal of one of the humans is to decide, after interrogating both terminals for a sufficient period of time, which one is the machine, and which one is the human. After repeating the test a number of times, if the questioner failed to correctly determine on more than half of the tests who is the human and who is the machine, the machine, or the used algorithm, is then regarded to have artificial intelligence [3].

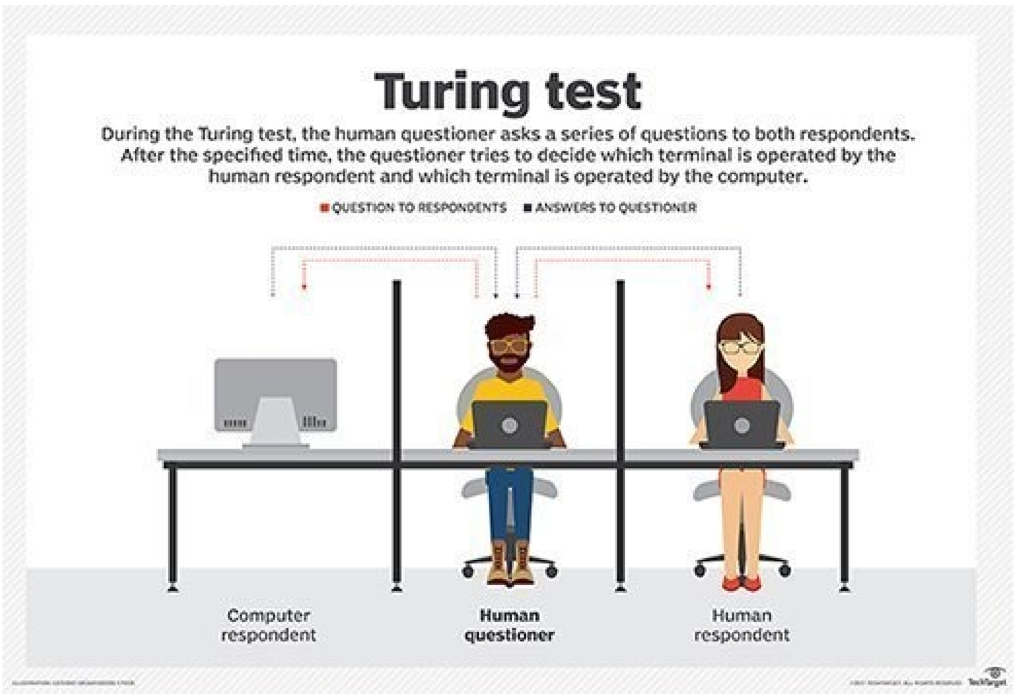

Turing test
During the Turing test, the human questioner asks a series of questions to both respondents. After the specified time, the questioner tries to decide which terminal is operated by the human respondent and which terminal is operated by the computer.
QUESTION TO RESPONDENTS
ANSWERS TO QUESTIONER
Computer respondent
Human questioner
Human respondent

Figure 1: Turing Test Demonstration

However, even if we assume that human beings are among the things that is considered to be intelligent and that they can decide on the intelligence of an object if it exhibits similar behavior to what they possess, the problem will still persist in isolating those specific human traits that are supposed to be "intelligent" from those that are not. For instance, humans exhibit anger, joy, jealousy, and rage. It might be asked which of these traits are considered as" signs of intelligence" if found in unanimated objects [1].

If such traits could be included in the AI boundary definition, it would lead to arising the matter of consciousness and the awareness of these objects with what's happening around them. For instance, Neil Leach in his paper "Do

Robots Dream of Digital Sleep?" [4] had discussed the issue of "Robot" or "Machine" creativity and its ability to "dream", "hallucinate" and produce something that may have never existed before.

In his research, he took as a starting point the science fiction movie "Blade Runner" (1982) [5]. This movie is based on the novel by Philip K Dick, Do Androids Dream of Electric Sheep [6]? which also inspired him in writing the title for his paper. The movie depicts a dystopian future world in which robots no longer possess an evident "artificial" nature but have rather acquired "bio-engineered" features. These features have rendered them behaving as "human replicants".

Therefore, they ended up acquiring sort of consciousness that it became nearly impossible to distinguish them or their behavior from real human beings. Although this speculation, coming from sci-fi perspective, is still regarded farfetched, it does highlight some questions like: Can AI be Creative? Or Can AI Dream? Or is it only humans that are capable of creating images that does not exist while machines cant as suggested by [7]?

This farfetched dream of machine consciousness would not be discussed unless we have seen evidence of "light" at the end of the tunnel. The impact AI has introduced even in domains that relies on pure "natural" creativity like design, art and architecture have created this sort of impulse.

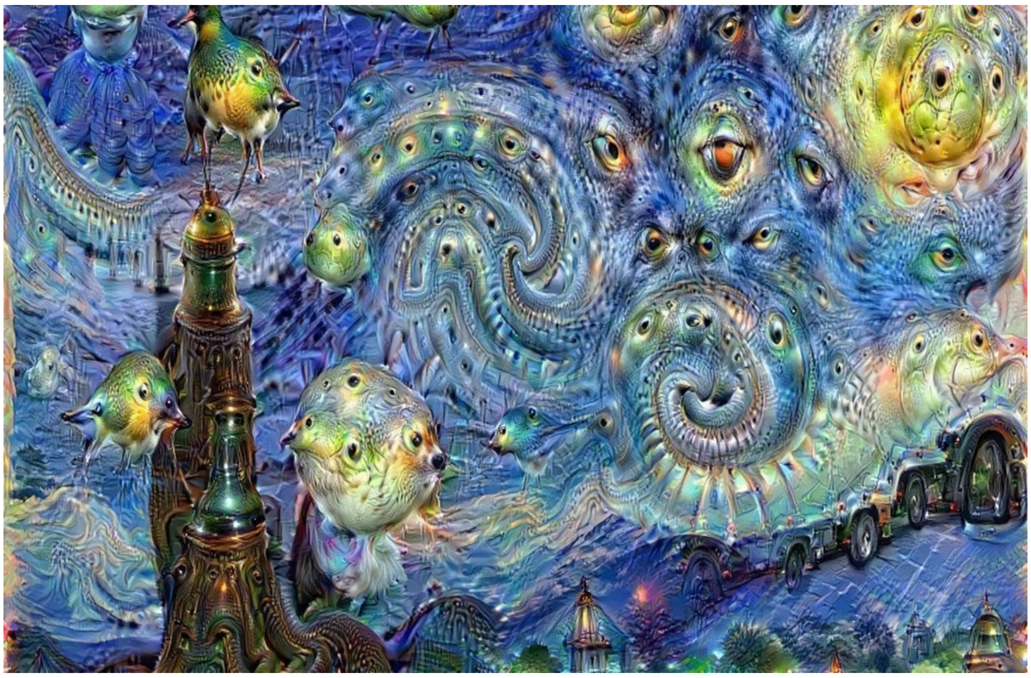


Figure 2: "Deep Dream Generator", A new form of psychedelic and abstract art being enabled through the use of deep neural networks [8]

An example of current adoption of "deep learning" techniques and methods in creating works of designs or artworks could be witnessed in the new era of architects and designers that use algorithms enabled by deep learning techniques like generative adversarial networks (GANs) and its different architectures (see Figure 2 and Figure 3).

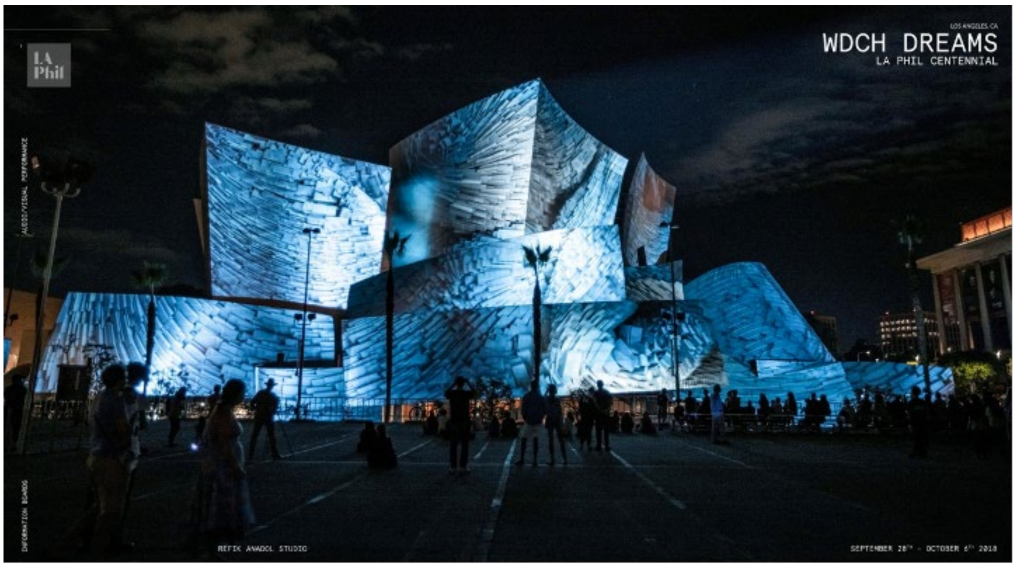


Figure 3: Refik Anadol, WDCH Dreams, Los Angeles, CA (2018) [9]

On the other hand, in terms of dreams, and despite that many AI applications introduce the word "dream" in their campaigns, there is still no clear evidence of the ability of machines to exactly reproduce the actual dreaming process of a human being. That said, there is no way to have a dream unless the machine could develop signs of consciousness; a challenge that is not yet entirely tackled by available algorithms. Despite that, it can be assumed that machines are currently on the path of developing its own consciousness, with demonstrations like a robot being able to identify its own reflection providing an evidence for such an assumption [10].

1.1.2. Introduction to Deep Learning

Over the last few years AI has witnessed a big boom especially since 2015. Much of that explosion is owed to the wide availability of GPUs that enabled parallel processing in a faster cheaper and a more powerful way. It also has to do with the simultaneous one-two punch of practically infinite storage and a flood of data of every stripe (that whole Big Data movement) – images, text, transactions, mapping data, etc. [11].

The easiest way to think of the relationship between AI, machine learning and deep learning is to visualize them in concentric circles (see Figure 4) with AI as the first circle inside of algorithms. AI is considered that initiated this whole branch of algorithms. Inside of the circle of AI comes machine learning, which blossomed later, and finally deep learning, which is driving today's AI explosion, fitting inside both.

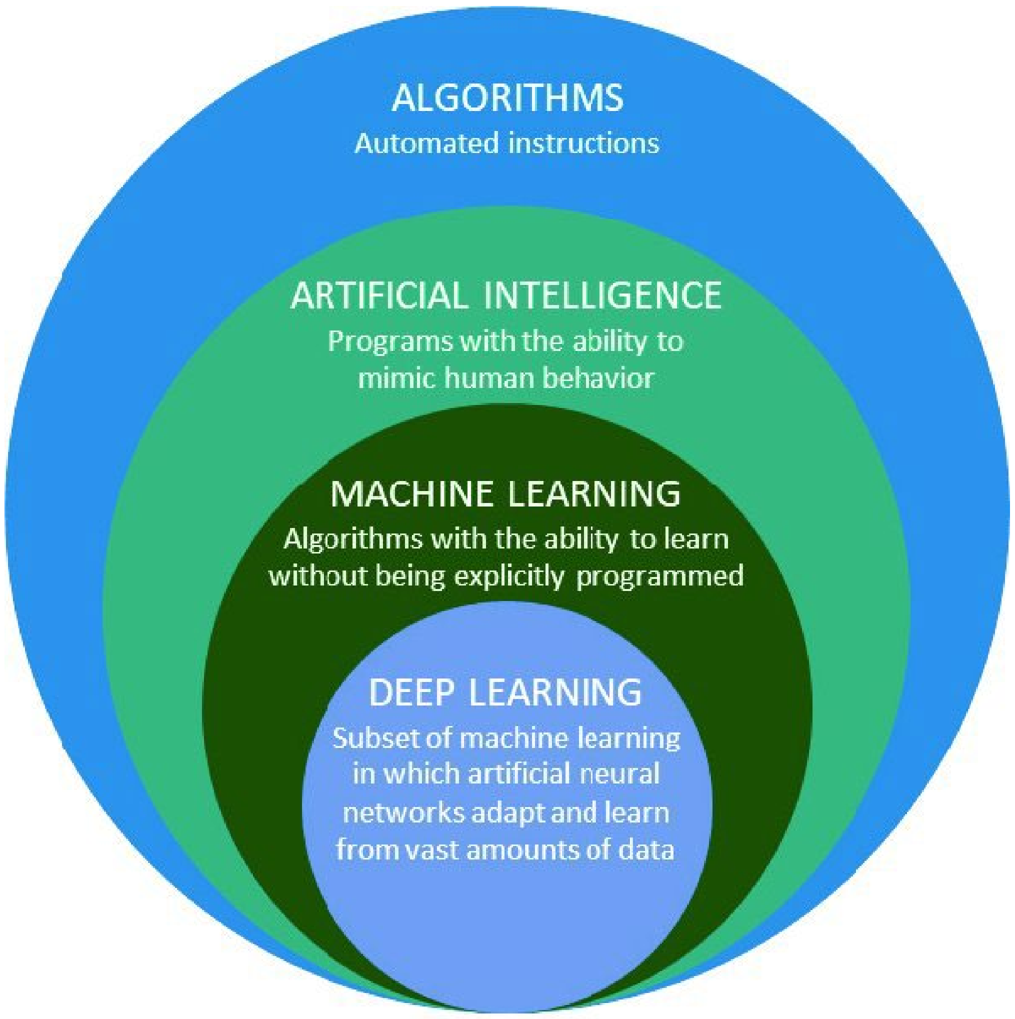

ALGORITHMS
Automated instructions
ARTIFICIAL INTELLIGENCE
Programs with the ability to mimic human behavior
MACHINE LEARNING
Algorithms with the ability to learn without being explicitly programmed
DEEP LEARNING
Subset of machine learning in which artificial neural networks adapt and learn from vast amounts of data

Figure 4: Visualization of algorithms vs. artificial intelligence vs. machine learning vs. deep learning as proposed by [12]

In brief, machine learning overarches branch of algorithms that parse data, learn from it, and then decides on or predicts something based on what was learnt. Therefore, the term "training" replaced the hard coded routines of algorithms in which the programmer needs to define each solution for a scenario of a problem that could happen and literally program a specific set of instructions to accomplish solving it. This training process is achieved using large amounts of data and algorithms that give it the ability to learn how to perform a task [11].

Deep learning was built on the same process of "machine learning" where we need a special kind of algorithms to be trained on large amount of data that would eventually lead to deciding on something. However, Deep Learning specialized in a certain kind of algorithms that depends at its core on neural networks.

An artificial neural network is a biologically inspired computational model that is patterned after the network of neurons present in the human brain.

Artificial neural networks can also be thought of as learning algorithms that model the input-output relationship.

An artificial neural network transforms input data by applying a nonlinear function to a weighted sum of the inputs. The neural network is comprised of a consecutive set of neurons knows as “layers”. The intermediate outputs of one layer, called features, are used as the input into the next layer. The neural network through repeated transformations learns multiple layers of nonlinear features (like edges and shapes), which it then combines in a final layer to create a prediction (of more complex objects). The neural net learns by varying the weights or parameters of a network so as to minimize the difference between the predictions of the neural network and the desired values. This phase where the artificial neural network learns from the data is called training [13].

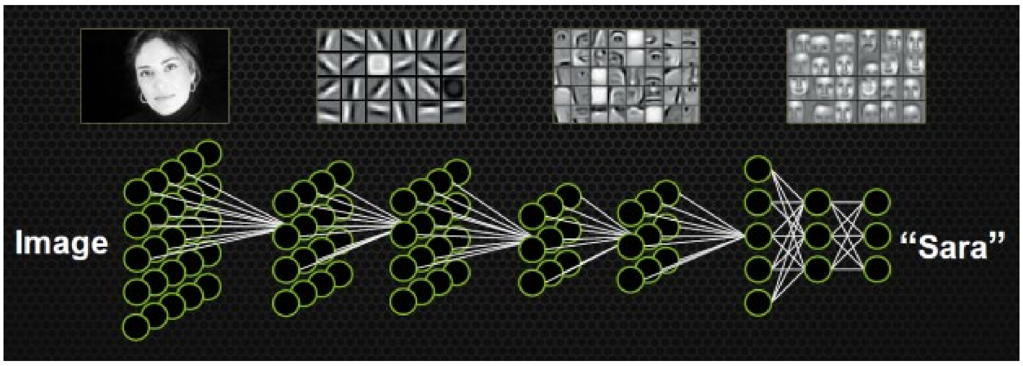


Figure 5: Schematic representation of a neural network according to [13]

Neural networks have been around since the earliest days of AI; however it was not widely spread like nowadays. The main issue was the massive amount of computation power needed. It wasn’t until GPUs were deployed that “deep” neural networks finally became popular. The word “deep” describes the use of multiple layers of neural networks with each layer comprising of several neurons. This complex architecture is considered a massive leap in the world of AI.

1.1.3. Deep Learning Algorithms Classification

“Deep” neural networks are currently supporting different kinds of algorithms. These algorithms fall under diverse branches of learning paradigms. Generally, learning paradigms can be classified into 3 groups: Supervised, Unsupervised, and Reinforcement learning. Recently, “Semi Supervised” leaning can be distinguished as a branch of its own.

Although this classification could be also adopted in general over machine learning algorithms (see Figure 6), it is widely perceived that the amount of advancements that deep learning brought to these branches is quite impactful. This is evident in most of the currently used algorithms that are trained using one or more deep neural networks. The following lines represent a brief explanation of the 3 branches added to them “Semi supervised learning” as a 4th branch combining some features from both supervised and unsupervised learning [14].

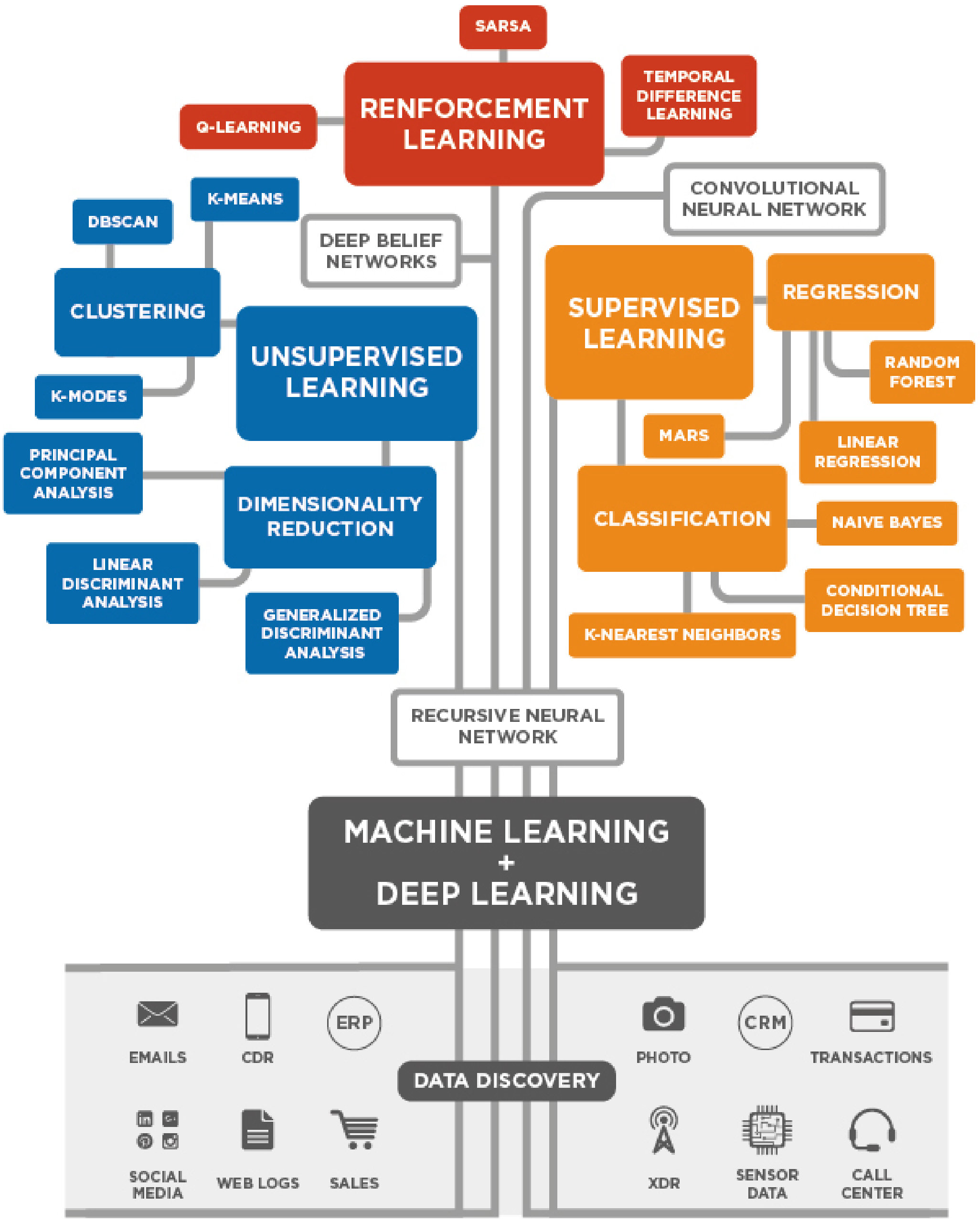


Figure 6: Tree of Machine Learning Algorithms [15]

- *Supervised Learning*

In supervised learning, the training data is fully labelled. Fully labeled means that each example in the training dataset is tagged with the answer the algorithm should come up with on its own. This label helps the neural network during training in adjusting its weights to be able to be able to predict the right label correctly.

There are two main areas where supervised learning is useful: classification problems and regression problems.

Classification problems looks at discrete categorical data, where the goal is to depict the category a certain input data, like an image, belong to. while regression focuses on continuous data. The simplest example could be predicting a certain value of variable y given a particular x.

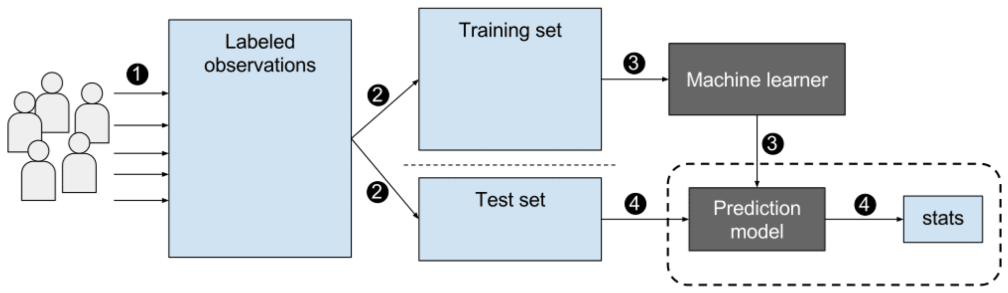


Figure 7: Supervised Learning Workflow, where the algorithm learns from the data [14]

- *Unsupervised Learning*

Unsupervised learning comes handy to problems where we have the data, but we don't know how to benefit from. This could be in the form of finding implicit patterns among the data, or perhaps clustering them in groups with similar features, etc.

In unsupervised learning, deep learning model is handed a dataset without explicitly mentioning what to do. There is no correct outcome or a specific desired output. The role of the neural network is to automatically find patterns by extracting useful features and analyzing its structure.

Therefore, the type of the problem could infer what kind of grouping of data is required. This data organization can take several forms like:

Clustering: The deep neural network tries to find training data having features similar to each other and groups them together.

Anomaly Detection: Using the deep neural network to detect outliers or very limited amount of data whose features do not conform with the majority of the elements in the dataset.

Association: By looking at a couple key attributes of a data point, an unsupervised learning model can predict the other attributes with which they're commonly associated.

Autoencoders: Autoencoders takes input data, compresses it, then try to recreate the input data from that summarized code. While a neat deep learning trick, there are fewer real-world cases where a simple auto coder is useful. But add a layer of complexity and the possibilities multiply: by using both noisy and clean versions of an image during training, autoencoders can remove noise from visual data like images, video, or medical scans to improve picture quality.

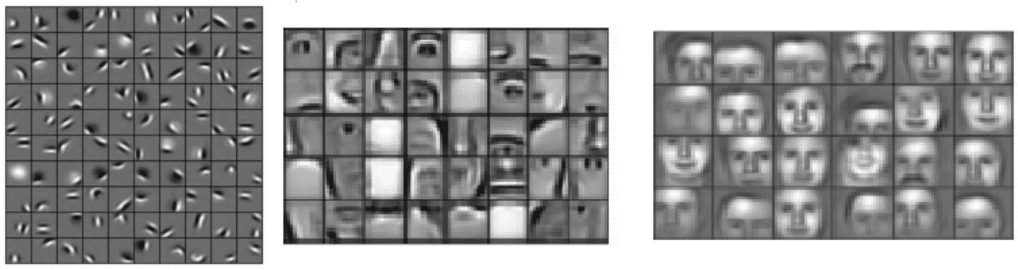


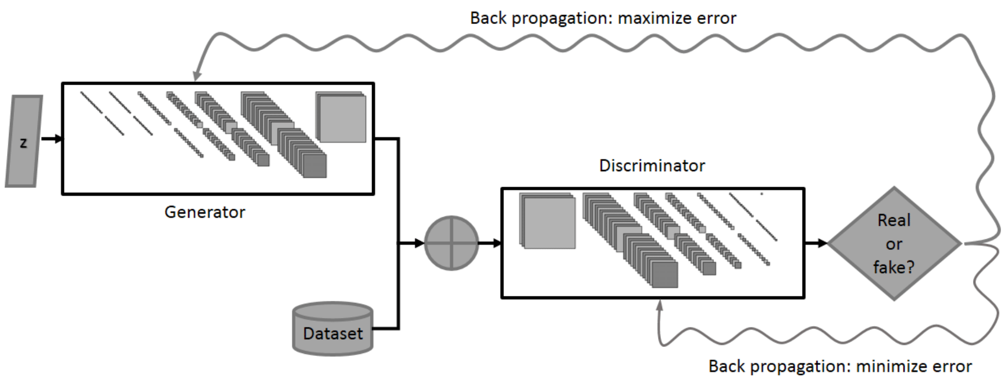
Figure 8: Feature Extraction on several layers of the neural network in Unsupervised Learning [14]

- *Semi-Supervised Learning*

The input for Semi-supervised learning is, usually comprised of both labeled and unlabeled data. This method is particularly useful when extracting relevant features from the data is difficult, and labeling examples is a time-intensive task for experts.

One of the most popular training algorithms that relies on a fairly small set of labeled data is general adversarial networks, or GANs. These kinds of networks and its diverse architectures has contributed to several applications recently in the AEC industry.

GANs comprises 2 deep neural networks instead of just 1. The first is called the Generator. It tries to generate data that looks like the training data. While the other network is called the Discriminator. Its main role is to detect whether the data it is inspecting is real data from the original dataset or a fake data created by the generator. At the beginning of training, both the Generator and the Discriminator are still very poor in doing their jobs.


Back propagation: maximize error
z
Generator
Discriminator
Real or fake?
Dataset
Back propagation: minimize error
Figure 9: GANs workflow [14]

By time both of them begin to improve. The generator tries to outsmart the Discriminator while the Discriminator become more experienced in capturing the fake data provided by the Generator. In the end, the generator improves its ability to create convincing fakes. These kinds of fakes, especially when the data are in the form of images, are regarded as a form of machine creativity, being able to create images that has never existed before.

- *Reinforcement Learning*

Reinforcement learning operates on the principle of earning rewards for good decision and penalties for bad decisions. The entity receiving this reward or penalty is usually called an AI Agent. The goal of this AI agent is generally related to finishing a certain task in a correct way without being given proper instructions or steps on how the goal could be achieved.

To make its choices, the agent relies both on learnings from past feedback and exploration of new tactics that may present a larger payoff. This involves a long-term strategy, just as the best immediate move in a chess game may not help you win in the long run, the agent tries to maximize the cumulative reward.

It’s an iterative process: the more rounds of feedback, the better the agent’s strategy becomes. This technique is especially useful for training robots, and it has also gained huge popularity in achieving noticeable results in RTS and Turn-based Games as well.

1.2. Deep Learning in AEC

In this sub-section, a few examples of deep learning applications are provided from literature. The main focus would be on the diverse applications of Generative Adversarial Networks (GANs) with its varying architectures. It is noted that in the past years, there is a growing interest in the AEC industry in using GANs with a special focus on the Design phase of the projects [16]. This extends from conceptual design automation to design evaluation and detailed plans or layouts generation [17]. However, GANs architectures were also witnessed throughout the spectrum of Construction processes as well as early trials of design and construction processes integration [18].

Starting with the conceptual phase, deep learning has been adopted to assist in developing conceptual designs. For instance, following the statement of Louis Sullivan that “form follows function” [19] , researchers in [20] has decided to evaluate existing buildings through the analysis and development of interrelated relationships between its building elements.

This was achieved using a graph-based representation instead of analysis of images and texts. The main goal of the research was to extract the subgraphs of significant building blocks in order to use them as a tool to create new conceptual compositions. The research followed Deep Neural Network (DNN) approach, in order to come up with conceptual function-driven design using a graph-based approach.

Since this research adopted a graph-based rather than the traditional raster-based approach adopted in several GANs architectures, their decision was to use InfoGAN architecture that has the ability to learn latent codes (see Figure 10). This process can be controlled to restrict the Generator to only generate variations of samples that correspond to the specified code.

This research has demonstrated that a system based on DNNs can use graphs to generate function-based conceptual designs. The GAN architecture used to evaluate graph-based samples was able to generate new results not seen in the training set. However, the research scope focused solely on functional relationships with no considerations of other parameters like aesthetics, geometry, environment, or structure.

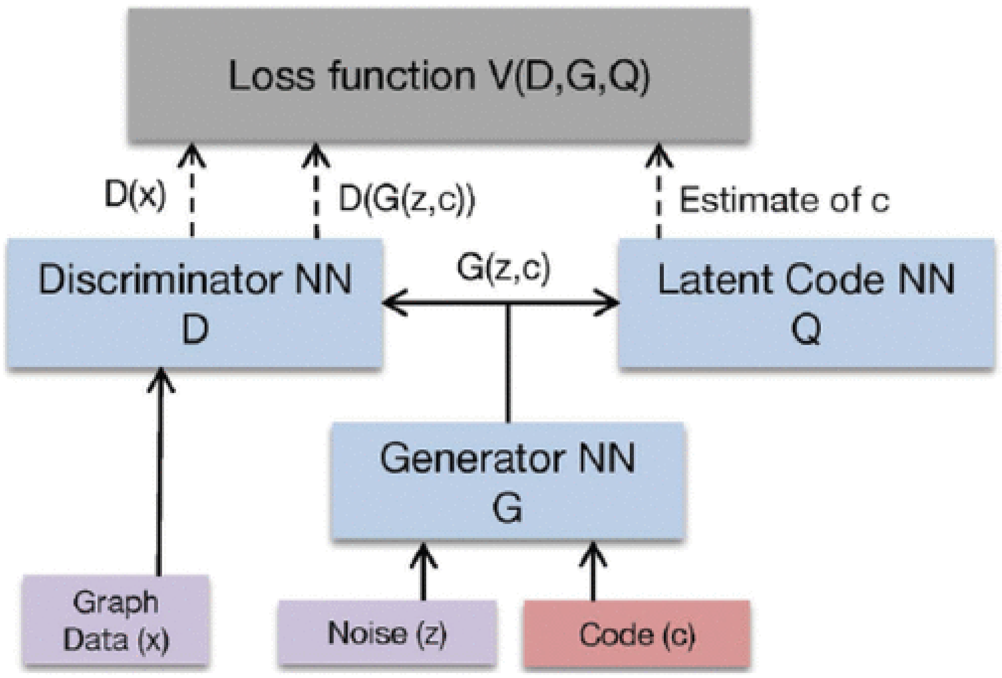
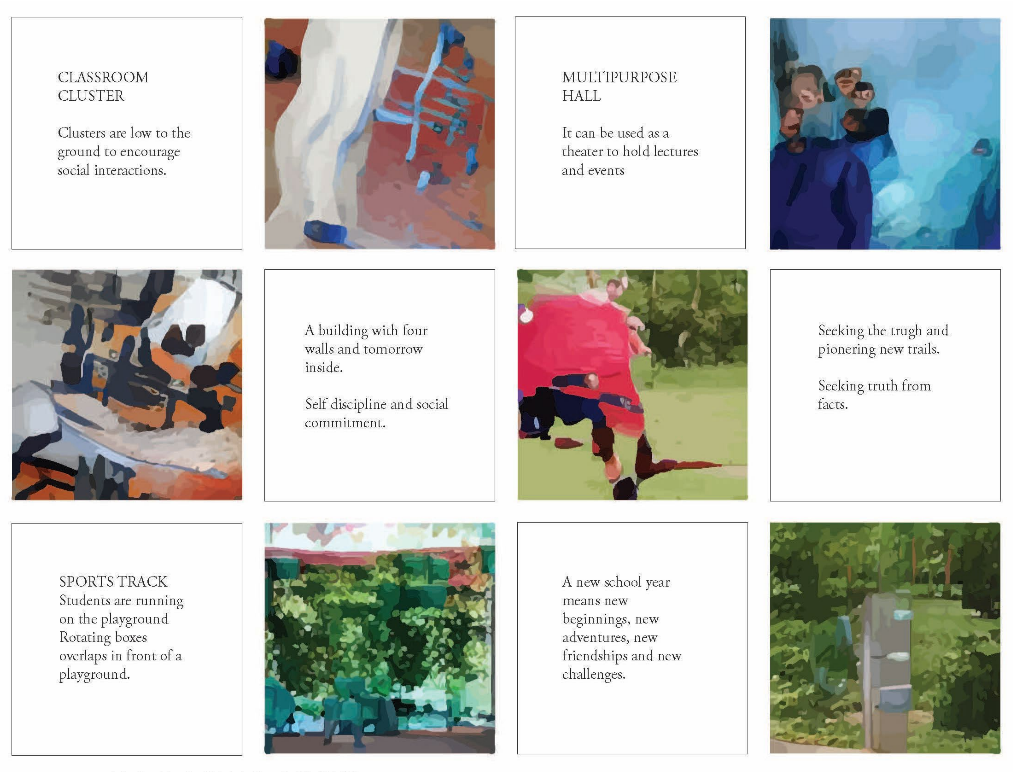


Figure 10: InfoGAN architecture – training three DNNs (D, G and Q) simultaneously [20].

Adding a new layer of "visual and auditory perception" to graph-based approaches, [21] explored the use of Attentional Generative Adversarial Networks (AttnGAN) as a design technique in architecture. The goal was to experiment with Spoken language and how it can be used to generate inspirational images for conceptual design purposes. Instead of depicting the interdependent relationships between building elements, AttnGANs allow attention driven, multi-stage refinement for fine-grained text-to-image generation.

The network initially starts from a global sentence vector and uses it to produce low generation image. Afterwards, the attention layer comes into action by using the image vector in each sub-region to query word vectors forming a word-context vector. The regional image vector is then combined with the corresponding word-context vector to form a multimodal context vector. These form the basis on which the model generates new image features in the surrounding sub-regions, which results in higher resolution pictures with more details at each stage (see Figure 11).


CLASSROOM CLUSTER

Clusters are low to the ground to encourage social interactions.

MULTIPURPOSE HALL

It can be used as a theater to hold lectures and events

A building with four walls and tomorrow inside.

Self discipline and social commitment.

Seeking the trugh and pionering new trails.

Seeking truth from facts.

SPORTS TRACK
Students are running on the playground
Rotating boxes overlaps in front of a playground.

A new school year means new beginnings, new adventures, new friendships and new challenges.

Figure 11: Examples of results of the AttnGANs [21]

Accordingly, despite that such a technique is relatively new in architectural context, it provides a promising tool for transforming language into shapes and design. This could be considered a new unique method for design inspiration with more than just visual senses.

Moving from Conceptual to Detailed Design, GANs architectures have been successfully implemented in several processes like floor plans [22], elevations [23] [24] [25], and inspirational layouts generation [26]. For instance, [22] has implemented a modified version of GANs called Pix2PixHD, to eventually produce floor plan images based on a coloring scheme.

Pix2Pix architecture is composed of 2 Convolutional Neural Networks since both the generator and discriminator are dealing with images. The role of the discriminator is no longer stating whether an image is fake or not but is actually checking a pair of images for conformity. The authors applied this method in recognizing and generating architectural plans.

The generation process involves feeding the trained network with a colored image where each color fills the boundary of a certain room or space, and each room is given a specific color code. The network then produces an image of a floor plan based on the boundaries provided by the input image. Each room in the produced floor plan is furnished based on the color code included in the input image (see Figure 12). Similarly, regarding floor plan generation, authors in [25] have used a hybrid method of GANs and case-based reasoning, for creation of possible evolutions of the current design based on the most similar previous designs.

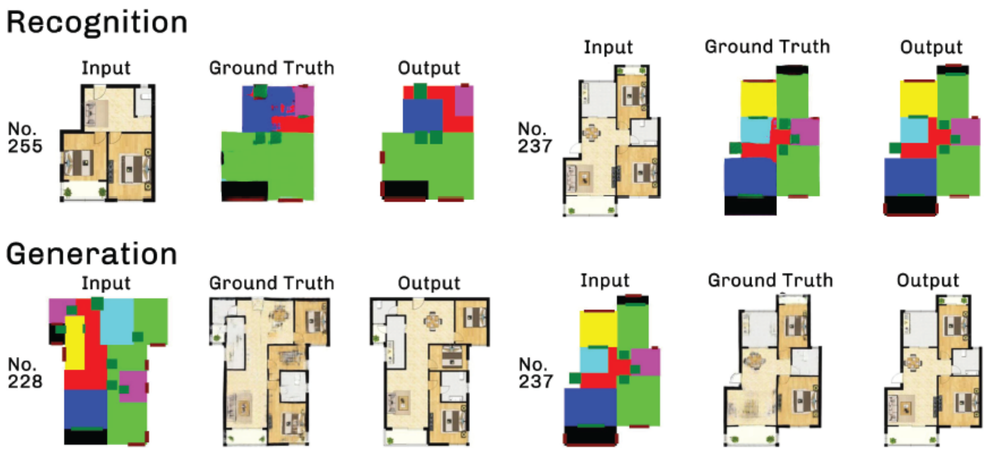


Figure 12: Apartment floor plan: recognition and generation using Pix2Pix in [22]

A more artistic approach can be seen in [26], where the authors used a variation of GAN called StyleGAN2 [27]. This network architecture was trained on a data set of plans from the baroque era as well as modern floor plans. The output floor plans "hallucinations" of the algorithm contained a kind of fusion between these 2 styles. The produced new plans contained hybrid features of the 2 styles producing images of floor plans that had never existed before.

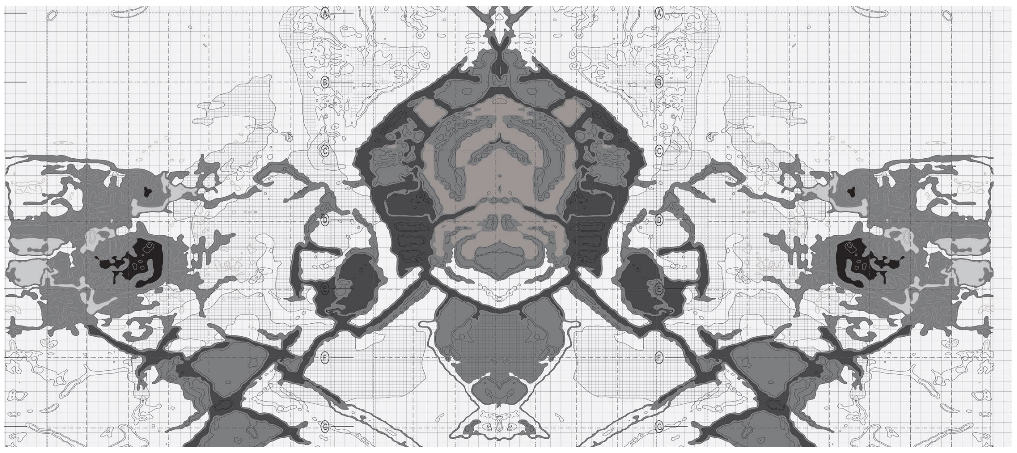


Figure 13: Close-up of one of the resulting plans based on the StyleGAN2 process between Baroque and Modern plans [26].

On the other hand, [26] pinpointed some doubts that could be generalized among all the outputs by different alternatives of GANs that are generated in the detailed design phase. This doubt lies in the absence of a direct method that can bridge the gap between the output images and the 2D CAD-generated architectural drawings. Despite our visual recognition of the produced images, the networks don't really understand the detailed semantic information of building elements. Perhaps a possible path could be a hybrid model that relies on graph-based information regarding relationships between building components and at the same time can link these relationships with each image vector containing these elements.

Furthermore, the use of GANs in architecture was not only constrained to 2D design inspirations, but also 3D GANs were experimented to produce a voxel-based 3D Preliminary Designs [28], [29]. Despite the tremendous amount of progress being made in the field of deep learning, voxel-based 3D GANs can be still considered in its infancy stage. Once issues regarding practical constraints on model resolution and training time are resolved or at least reduced, this technology would create a major disruption in the design process.

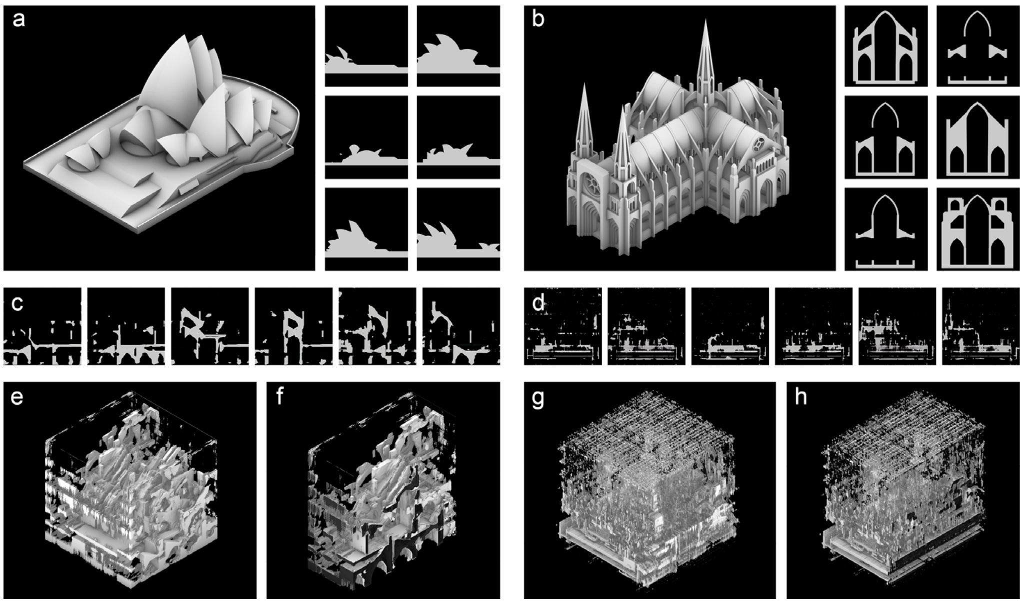


Figure 14: 3D style transfer results through CycleGAN as in [29]

Moving to the contribution of deep learning in the optimization of construction processes, there are several applications that spans to include several activities like construction robots path planning [30] and detection of building defects [31], [32].

One of the most promising applications is the integration of material properties with architectural design tools. An example of which is the experimentation with irregular wood-logs that are always counted as waste materials. The process thereby gives an example of how the natural forms and properties of sawlogs can be directly used to generate new structures and spatial conditions [33], [34].

DNNs are implemented in these processes in order to find proper properties of sticks to improve selection mechanism of construction process. Such systems could also be inspired from nature, since animals in nature like beavers for instance use irregular sticks to build dams [33].

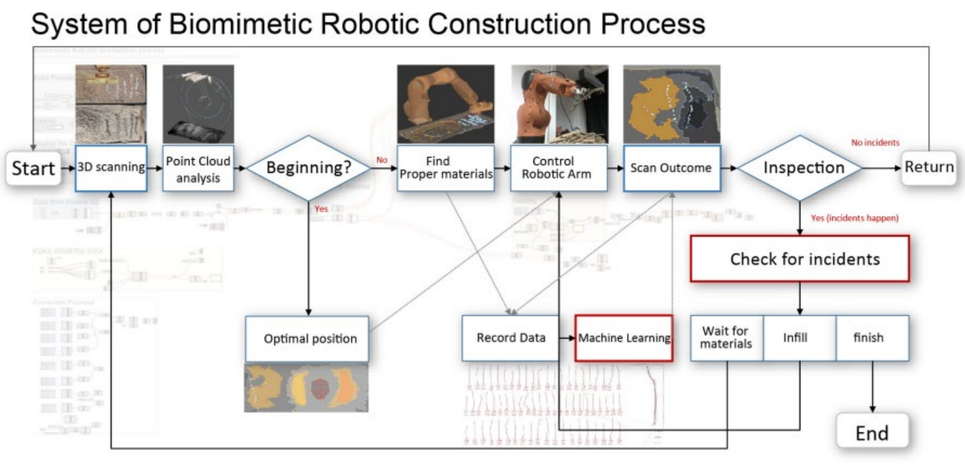


Figure 15: Systems of biomimetic robotic construction process [33]

Furthermore, authors in [18] have used deep learning to automate the pattern generation of robotically assembled bricks. The main role of the semi-supervised Pix2Pix algorithm is to generate brick patterns using image pairs where the input is just a (wall boundary) and target image is the (brick wall).

As explained before, in order to achieve this output, the network has to learn first through a set of image pairs containing a wall boundary and a solution of brick wall pattern group. Once learnt, the output 2D image of the network (brick wall pattern) is then converted into spatial positions for automatic robotic assembly simulation. This is carried out by an image processing algorithm.

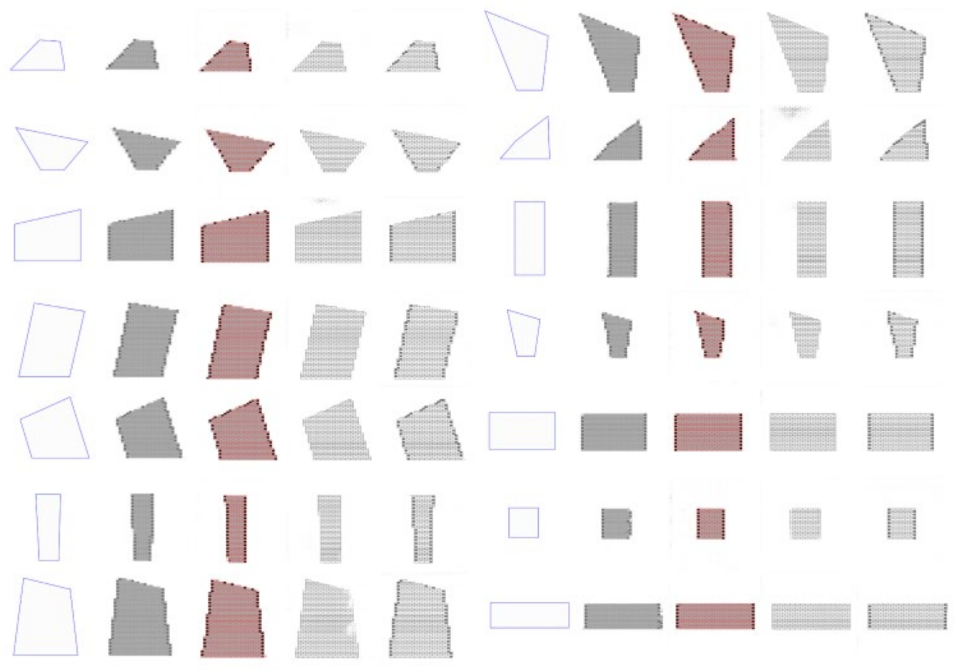


Figure 16: Input and output images for the four dataset styles in [18]

1.3. Reinforcement Learning in AEC

As discussed before in subsection 2.1.3 (Deep) Reinforcement Learning (RL) is one of the main branches of deep learning. This sub-section is dedicated to the sate-of-the-art research regarding the deployment of Deep RL algorithms in AEC industry.

Generally, Deep RL applications are not yet implemented on a wide scale in AEC industry. In this sub-section, we will start by mentioning some examples of RL deployment in conceptual design phase, then ideas regarding detailed design or design simulations are mentioned. Afterwards, robotic assembly tasks that benefited from RL are presented. Finally, we will indicate some trials of combining design and construction goals altogether for an agent to solve simultaneously using RL algorithms.

To start with, [35] proposed an AI agent that can produce 3D abstraction representing artificial design habitats. They decided to adopt a RL algorithm known as Spiral. The decision was to go for Spiral instead of a GAN algorithm because, unlike GANs, it can autonomously decide the number and characteristics of the features to reproduce in a synthetic visual abstraction.

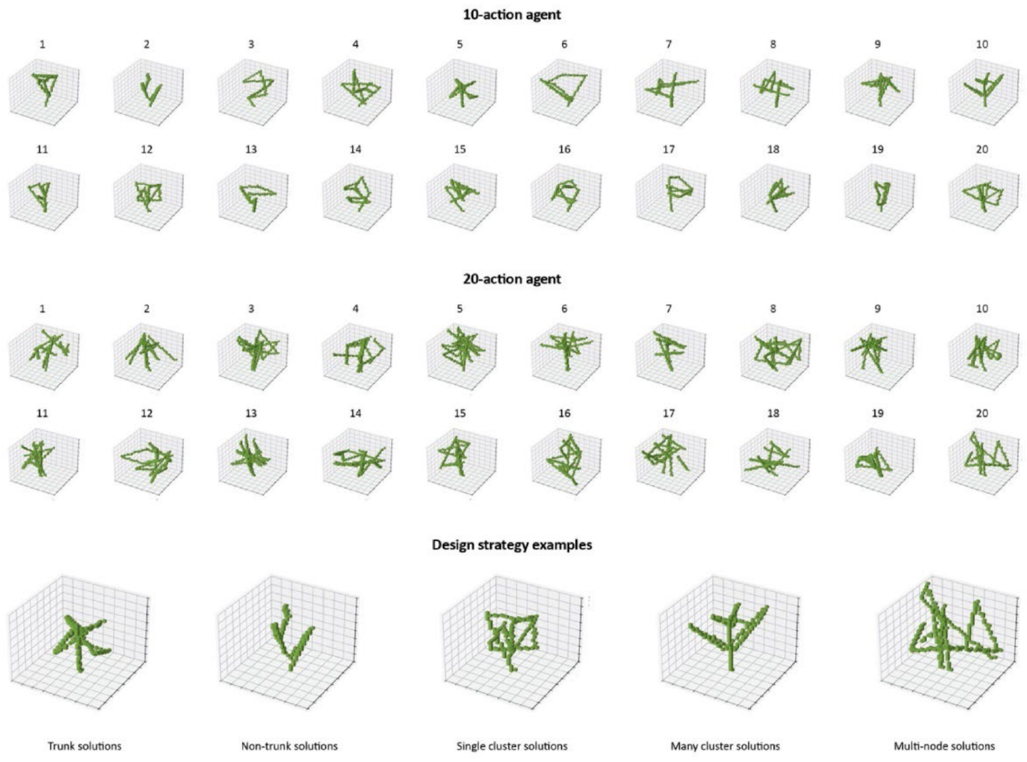


Figure 17: Visual abstractions produced by the 10-actions agent and the 20-actions agent during the last training iterations using SPIRAL algorithm in[35].

The adopted AI agent can be manipulated through a reward system to support the design of human habitat that is inspired from natural habitat structures. This is achieved through the assessment of the AI agent final results of its actions, which mainly involves the placement of 3D voxels in digital space, by comparing its geometry with 3D abstract representation of natural trees.

This article primarily investigates the potential of AI in supporting the design of structures by providing design suggestions aligned with the main goals set by the designer, which in this case is to capture the essence of arboreal wildlife. This becomes handy in many situations where incomplete knowledge about complex natural forms can constrain the design and performance of human-made artefacts.

Similarly, [36] demonstrated in a 3D voxel representation design environment the meaningful impact of RL on augmenting generative design approaches with intuitive capacity and sophisticated control. The agent is defined as a mesh graph that consists of certain fixed number of vertices and predefined topology. At the beginning of each episode the vertices are randomly generated in the 3d voxel environment (see Figure 18)

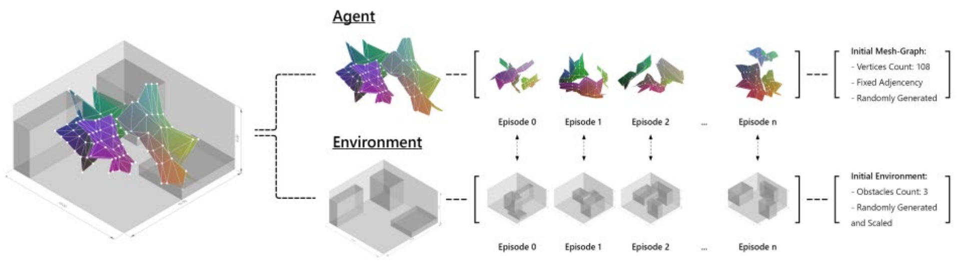


Figure 18: Strategies of Initializing RL Agent and Environment as described in [36]

The agent actions basically include the possibility of movement of each vertex one step to one of the adjacent voxels, However, to prevent intersection of the meshes or falling of more than one vertex in the same voxel, spatial separation and tension cohesion is controls the available actions at each step. Thus, individual decisions of agents are also subject to interactive behaviors to abide by the general behavior of an emergent complex adaptive system.

The RL algorithm used is Proximal policy Optimization (PPO). The environment was developed in Unity Platform with ML-Agents toolkit. The demonstrated RL-based generative training experiment has been conducted with a total episode of 20,000 (500 steps each). The generative training outcomes are recorded every 200 episodes, as well as the mean-reward recorded every 500/2000 episodes.

Furthermore, research focusing on implementing RL in generative design approaches can also be evident in [37] with similar voxel-based 3D environment. It has to be noted that, the general impression given by these different implementations of RL algorithms could give a reflection on the interactive correlation between designers and computational intelligence. It also provides insights of human-machine collaboration during early design stages.

On the other hand, moving from the abstract building/pavilion conceptual designs to generative design on the urban scale, authors in [38] incorporate deep reinforcement learning (DRL) and computer vision for urban planning through a case study to generate an urban block based on its direct sunlight hours, solar heat gains as well as the aesthetics of the layout.

The Deep RL algorithm adopted was deep deterministic policy gradient (DDPG). It was trained to guide the generation of the urban schemes. In each episode, the agent could only arrange one building (see Figure 19). The actions involved controlling the building XY coordinates as well as its Length, Width, and Height.

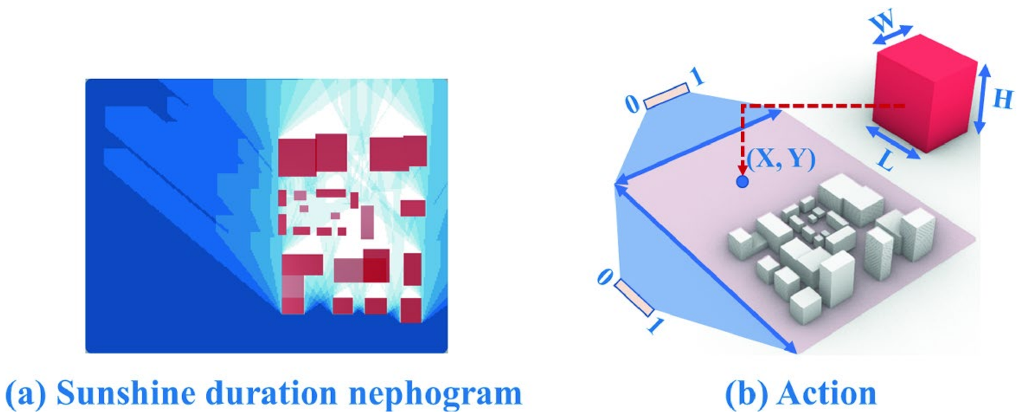


Figure 19: The diagram of observation, action and reward as depicted in [38]

Moreover, Deep RL algorithms has been also experimented in generating spatial configurations as in [39]. In this research, multi-agents deep reinforcement learning (MADRL) were used to control spatial partitions and interact in a 2D grid environment. This approach used double deep Q-network (DDQN) combined with a dynamic convolutional neural-network (DCNN).

env. grid	16 x 16 (w/padding)	**loss function**	Smooth L1 Loss
action grid	5 x 5	**optimizer**	RMSProp
actions space	26	**batch size**	64
n of agents	6	**learning rate (α)**	0.0001
adacencies (k)	3	**τ**	0.02
neighbors (m)	3	**capacity of D**	10000
episodes	43350	**initial ε**	0.9
steps per episode	64	**minimum ε**	0.1
γ	0.92	**ε reduction**	custom schedule

Figure 20: Training Configuration used in [39]

During the experiment, the trained agents has managed successfully to generalize their knowledge to different settings, consistently explore good spatial configurations, and quickly recover from perturbations in the action selection.

Deep RL algorithms were also used in the development of an environment prototype that is that learns from Electroencephalogram (EEG) feedback in real-time [40]. Unlike previously mentioned episodic tasks, this is considered a continuous one. PPO algorithm was adopted for the training of the agent. The agent's goal was to keep the subject's alpha wave stable or decline, which indicated a more calming state, by intelligent decision of illumination state according to subject's EEG.

Moving from design applications to task promoting autonomous construction, RL algorithms has been deployed in several applications in that context. For instance, to avoid collision between drones participating in a construction task, the master's thesis developed by [41], has provided an RL system that can control a number of drones for autonomous construction in a dynamic continuous environment. This system was successfully deployed in 2 experiments regarding brick laying and façade coating.

Furthermore, researchers at ETH Zurich [42] have applied Deep RL on timber assembly tasks using industrial robots. This solution was introduced to overcome problems associated with Robotic Assembly in Architectural context such as small tolerances and complex contact situations, especially in assembly of elements with form-closure such as timber structures with integral joints. The researchers have adopted an adapted Ape-X DDPG algorithm to train the agent in as simulated environment.

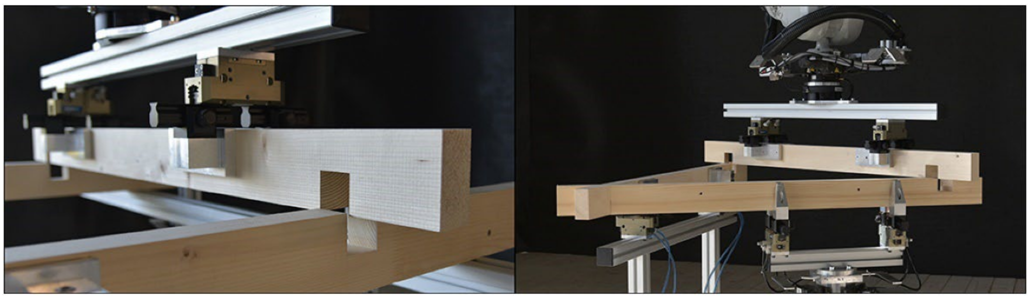


Figure 21: Stills from an assembly of an H-shaped (left) and Δ-shaped (right) double-lap tasks by the real robot as proposed in [42]

Despite that the control policy was trained entirely in simulation, it managed to overcome tolerances and shape variations that didn't occur in the simulation and was witnessed only in real world. This generalization behavior of the agents is considered to be one of the main powerful points of strength that training of RL Agents could provide.

Depending also on the proper assumptions of the observations and letting the agent encounter different situations during training, the Agent can then learn how to overcome these situations and can use this experience to overcome real-world situations that may have never occurred during training. This is achieved by linking the actions taken during learning in similar situations and applying them on the current real case.

Lastly, we discuss a type of Deep RL applications that had sought to integrate some design and construction processes together into one problem translated to a Markov Decision Process (MDP). For instance, authors in [43] developed a framework for designing and implementing effective autonomous architecture defined by three key properties: situated and embodied agency, facilitated variation, and intelligence. PPO algorithm was also used to train an agent to learn adaptable behaviors related to autonomous mobility, self-structuring, self-balancing, and spatial reconfiguration.

Physical properties and degrees of freedom were applied as constraints in a simulated physics-based environment in Unity. ML-Agents toolkit was used to implement the Deep RL framework. Both single and multi-agent setups were provided. Topological rules of tensegrity were applied to develop assemblies with actuated tensile members. After simulation, Physical robotic prototypes were built and actuated to test simulated results.

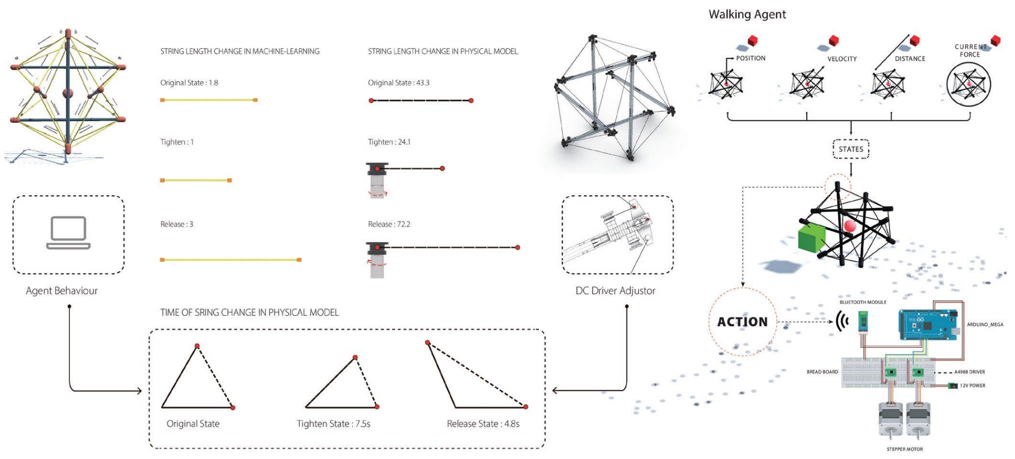


Figure 22: Diagrams: Interelationality between physical robots, simulated robots and reinforcement learning models as developed in [43]

Moreover, research proposed in [44] discusses the capabilities of reinforcement learning in game engine for integrating design, and construction processes. The main motivation comes from promoting a circular strategy by reusing scrap elements as building elements. To proof this concept, the author developed an application in Unity to train an AI Agent using PPO algorithm on the proper placement of scanned geometry of wasted plastic chips objects in a way that can create a stable structure based on certain criteria set by the designer.

Six major types of data are observed to train ML-Agents to assemble according to the performance of such priorities. These parameters are lighting variety, floor area, symmetry, structural stability, and thermal dynamic variety. The data are obtained via the cross-platform synchronization of shell model and analysis.

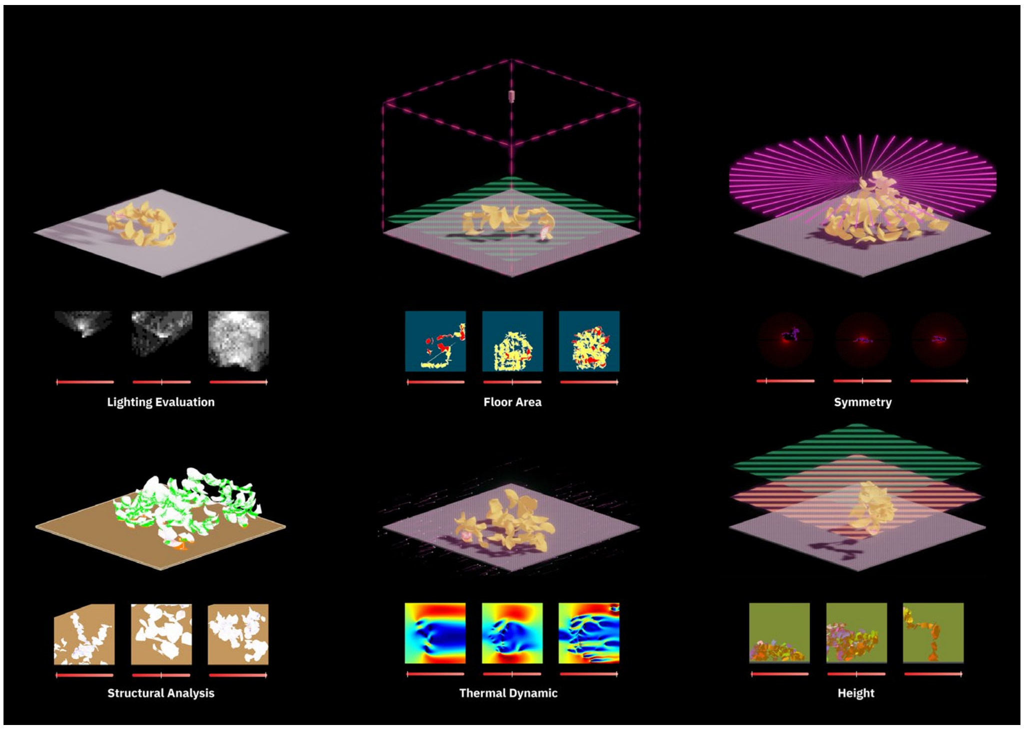


Figure 23: Six major types of data are observed by the agent

The geometries were streamed from Unity to Grasshopper for Karamba structural analysis. Karamba is a Grasshopper plugin for finite element analysis that can provide real-time feedback. The analysis result is then streamed back to Unity through UDP for rewards stating. Other observation data, such as floor areas and thermal dynamics, are obtained using a camera within the game and RGB distribution analysis.

Although the research target was to include both design and building constraints, it was more oriented towards providing design insights on how to utilize these scrap materials in reconstruction. For instance, the simulation, despite being developed on a game engine, it didn't benefit directly from physics capabilities provided by Unity platform. It rather used an external structural design check for checking an already placed static version of the elements without considering the kinematics of the assembly process itself.

The physics capabilities provided by Unity could help in simulating the actual impact of adding each new scrap element to the rest of the structure. This could show for example the effect of weight and gravity in the simulation environment in an accurate way that is close to what could happen in real world. However, this research could be an encouraging starting point to explore this concept provided that it could be integrated in a more elaborative way to expand its applicability.

1.4. Conclusion

In this section, we presented an introduction to artificial intelligence, machine learning and deep learning. Then, an explanation of different deep learning algorithms was provided. Afterwards, we demonstrated several applications of deep learning in the AEC industry. However, reinforcement learning was discussed as a separate sub-section apart form the other deep learning algorithms since we wanted to highlight its potential in the AEC industry.

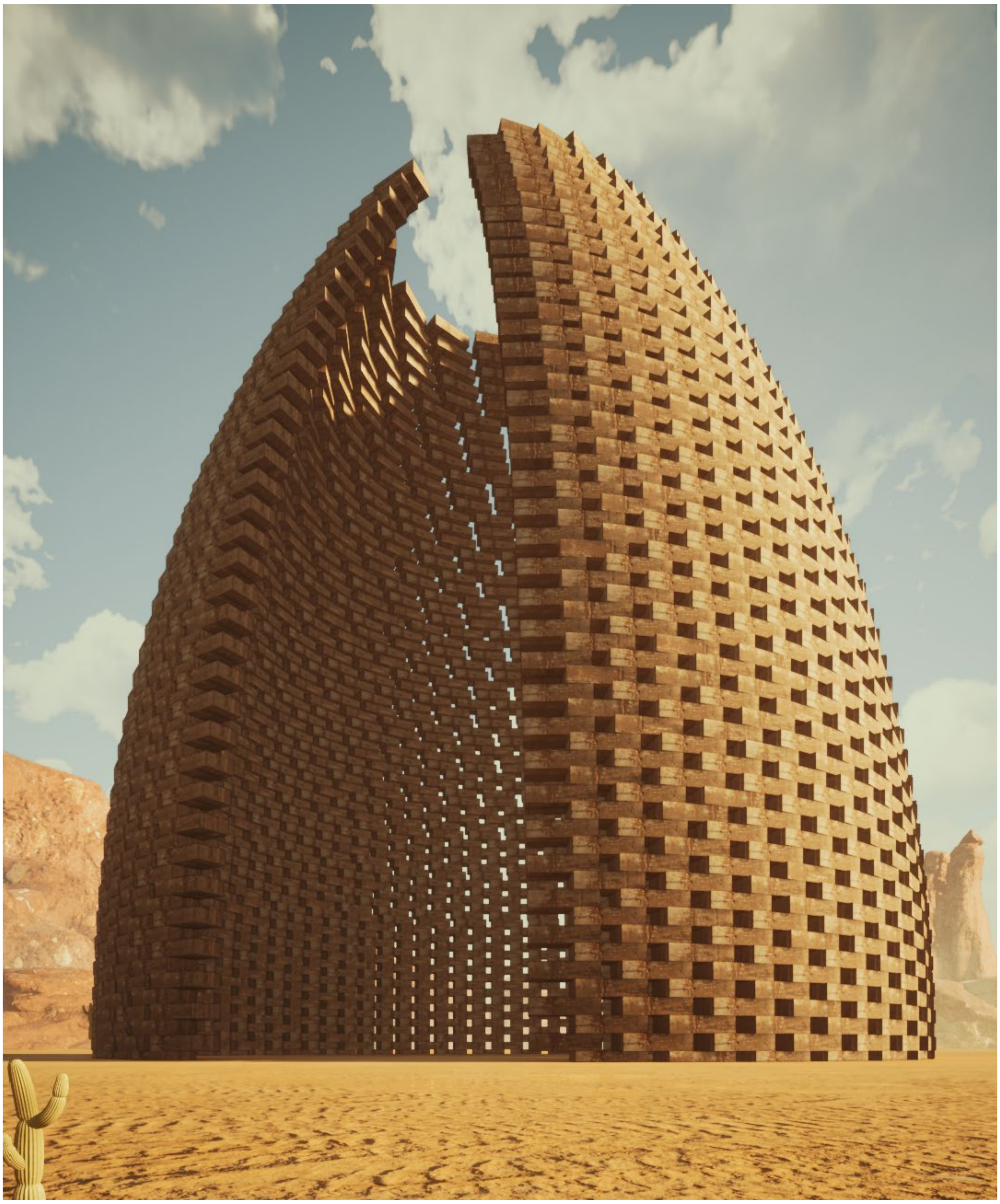
The state-of-the-art applications of reinforcement learning provided has shown several experiments that benefited from the success achieved through RL algorithms in different domains away from the AEC sector. Due to the complexity and unstructured nature of this sector, several external solutions usually fail to provide similar results when adopted.

However, it was noticed that game engines like Unity and the possibility of simulating real case scenarios can favor the success of AI agents in learning design and construction goals and then be implemented by execution robots in real life. One of the main challenge remains in finding the silver lining between creating a very abstract version of real life that renders the simulated environment incompatible with real applications and developing a very detailed environment that exponentially grows the level of game complexity to the extent that AI Agents would need a massive amount of time and computational resources to be trained.

Finally, GANs and AI Agents are regarded as promising tools to help create a paradigm shift in AEC Industry. Designers could then get assisted throughout the process and get inspired by innovative solutions. This new paradigm might also release some daunting work off the shoulders of designers and builders through automating different processes. The role of designers and builders however would shift more towards fine tuning the proper incentives and reward systems for the agents to help reach the goals required from them.

Chapter 3

Original Contribution



2.1. Introduction

This section represents the “original contribution” that this thesis postulates. It starts with the explanation of how AI challenges in Real Time Strategy (RTS) games have an influence in the better implementation of AI in AEC Industry. Then, the new framework is introduced that revives the ancient term “Master Builder” in a rather compatible way to the digital tools available nowadays.

Afterwards, the section continues with explaining the game that was developed as a proof-of concept to the framework proposed. Both human version and Robot version of the game are exposed. The way agents learn the design and construction goals are then explained thoroughly. In the end, the results achieved by the agents are demonstrated and discussed.

2.2. Real Time Strategy Games vs Design and Construction Projects

In this section we try answering the following questions:

- What are RTS games and what differs them from other games?
- What are the characteristics of RTS games that makes them a perfect candidate to be compared to the process of Design and Construction of a building?
- Why an algorithm succeeding in achieving high results in RTS games can be a good candidate to be applied in construction industry?

2.2.1. RTS Games definition and its Characteristics

- *RTS Games Definition*

Real Time Strategy Games (RTS) is a sub-genre of strategy games in which a player needs to build an economy (gathering resources, building and training units, and researching technologies), and sometimes need to also develop military power to defeat an opponent [45] . The difference between RTS games and Turn-Based Strategy games is that in the latter, each player has enough time to carefully think and consider the next move while the opponent would not be able to take any actions during this period. However, in RTS games, players must attempt to work on all aspects of the game simultaneously while knowing that the opponent is expected to be doing the same thing. Even in RTS games where the focus is only on the prosperity of the player’s people or “citizens”, the player should always be able to be “multi-task” and take diverse actions simultaneously, to ensure all the needs required by the “citizens” are being met. Therefore, time is a crucial factor to be considered in RTS games, thus an additional level of complexity is being introduced [46].

- *RTS Games Characteristics*

Accordingly, RTS games are characterized by the following [45]:

- They are simultaneous move games, where several players can perform actions at the same time. These actions are mostly durative, i.e., a series of actions is required in order to be completed or to witness some sort of impact on the game.
- RTS games are "real time", which means that the time to take an action is very limited. For example, in the game Starcraft, the game is executed at a framerate of 24 frames/sec. Thus, theoretically, a player can take an action every 42 *ms* before the game state is changed. However, for a "human" player this rate is practically impossible.
- RTS games are partially observable, so players can only the part of the map they actually have units in it. This is known as "fog of war".
- RTS games are nondeterministic. Thus, there is never only "one way" to succeed or win in an episode, and a serious of actions might not yield the exactly the same results every time they are executed.
- The game complexity is quite large in terms of state-space size and number of actions available at each state. For instance, the state space of Starcraft is estimated to be many numbers of magnitude larger than a typical Turn-Based Games like GO (10^{170}) or Texas Holdem Poker (10^{80}).

Figure 24: A screenshot of an ongoing episode of StarCraft with a battle occurring between Protoss and Terran

Based on these characteristics, the traditional methods that were used in the past for solving classic board games such as Game-Tree Search for example, wouldn't be as effective if applied to RTS games, without introducing a sufficient level of abstraction to the game. The application of AI techniques and specifically Deep RL has proven to yield very promising results in both genres of games [47]. The complexity of RTS games still enforced the engagement of some level of abstraction and hierarchy during the learning process. However, results obtained by applying deep RL algorithms were able to beat records reached by top human players in famous RTS games (like StarCraft) without any game restrictions [48].

2.2.2. Challenges in applying AI in RTS games

Unlike other genres of games, RTS games are a fertilized land for AI research due to the challenges they impose. These challenges were early summarized by [49] as the following:

- Resource management
- Decision making under uncertainty
- Spatial and temporal reasoning
- Collaboration
- Opponent modeling
- Adversarial real-time planning

Later on, [45] built on the previously mentioned challenges, but regrouped them under six different areas namely:

- ***Planning:*** Given the extremely large state-space in RTS games, planning for moves (series of actions) during the game should be executed with multiple levels of abstraction. At the macro level, the agent (player) needs to plan on the long-term to develop strong economy. At a micro-level, short term planning could be evident in optimizing the movement and positioning of units in coordination with each other to win over an opponent in a certain battle that may arise. Or in case of gathering of resources, "gathering units" should be deployed in an efficient way. Such a deployment has also an impact on the long-term as well (Figure 25). Thus a hierarchical decomposition could be a solution to address such planning issues as in [50].

- ***Learning:*** Learning could be divided into 3 types:
    - ***Prior learning:*** This learning relies on existing replays of previous games, recently this term can be referred to as "Imitation Learning". An example could be that of "Alpha Star" [48] which is now ranked above 99.8% of active players on Battle.net (the online platform for playing StarCraft). It has initially learnt from previous games recorded for top players to accelerate the learning process.
    - ***In-game Learning:*** In RL terms, it could also be referred to as "Online learning". In RL, some algorithms rely on such a method directly for learning while other relies on an in-between approach in which an experience replay for example

can be used to store experiences necessary to fully train an agent.

- ***Inter-game Learning:*** It refers to applying what an agent learnt in one environment into another environment. This could increase the chance of victory in the next game.

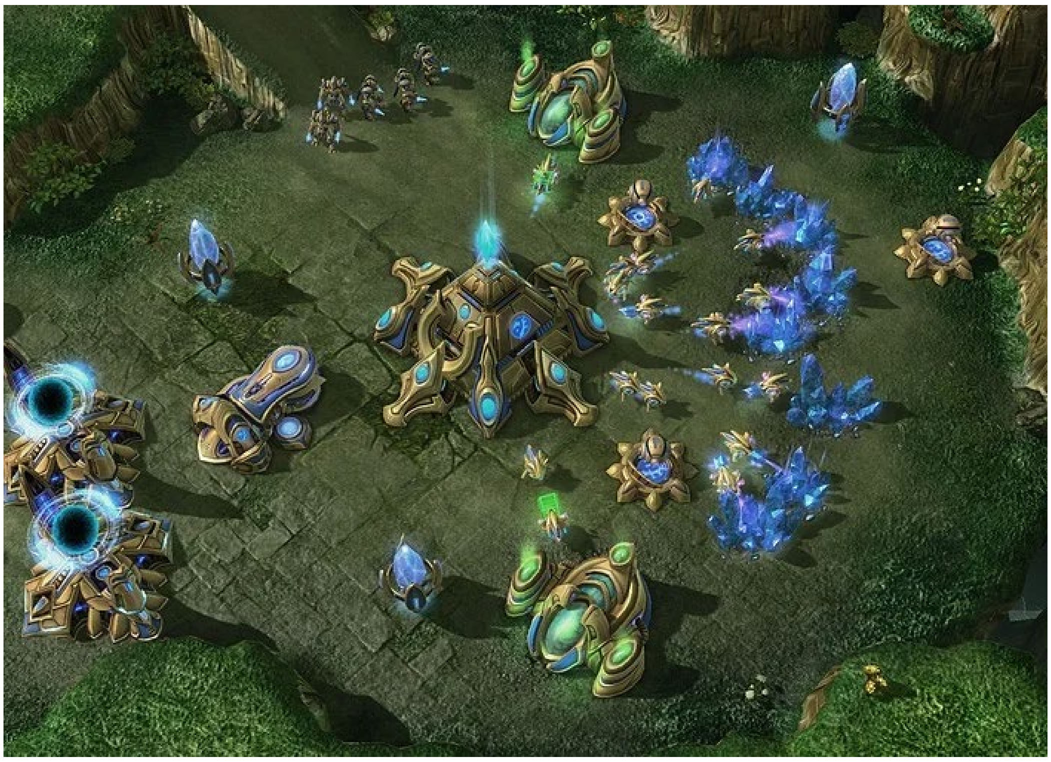


Figure 25: Protoss Command Center and gathering units are situated as close as possible to the resources

- ***Uncertainty:*** Adversarial Planning under uncertainty is a complex challenge in RTS games. Two main examples of uncertainty can be found in the partially observed map of the game and in predicting what action the opponent can take at a certain point in time.
- ***Spatial and temporal Learning:***
  - ***Spatial learning*** in RTS games is related to terrain exploration. For example, the positioning of building units should be done in a way that from one hand can form a defensive mechanism that protects more important building units through forming a fake "wall" around them, while on the other hand the position of buildings shall not be an obstacle for a player's own units, so it shall secure a smooth movement of large tactical units in between them.
  - ***Temporal Learning*** could be evident in RTS games, for instance, in the adjustment of the timing of an attack on an opponent by trying to enforce this attack when the opponent is not giving full attention to defensive mechanisms or focusing

too much on economic prosperity. Another example for economy based RTS games could be on a higher strategic level, where a player needs to decide how to plan, in the long-term, the priorities for resources investment. This shall be done in a way that secures the appropriate deployment of services to the “citizens” and that the funding should be properly distributed between fast-track projects that render direct impact on the citizens and long-term projects that need a continuous flow of investment, and their results shall appear after a relatively longer period of time.

- ***Domain knowledge Exploitation:*** Domain knowledge could be acquired by two methods. The first is to rely on experts in the domain that somehow could fine-tune the reward system to be used as an interim-guide for the agents during the learning process to follow. The second method could be “Imitation Learning” as discussed before. The latter approach was early proposed by [51] and [52] even before the exploitation of this option by recent algorithms like the ones adopted by AlphaStar.
- ***Task Decomposition:*** Given all the previous challenges, a reasonable approach would be to decompose the problem of winning an episode in an RTS game into a set of smaller sub-problems. A common sub-division suggested in [45] could be to subdivide the big task into 5 sub-divisions namely: Strategy, Tactics, Reactive Control, and Terrain Analysis.

It has to be noted that there are many forms for categorizing these challenges apart from the one adopted above. However, those different categorizations share together one general idea, namely, an AI perspective of challenges in RTS games should be addressed in a completely different and much more detailed manner with respect to how a human player could perceive and categorize them. The other observation is that any AI supported solution adopted should consider the interrelations between each category and try to detect patterns that can help generalize the problem and reduce the required time for exploring the state-space as much as possible.

2.2.3. Challenges in applying AI in Design and Construction Projects

Building on the applications of AI in design and construction already discussed in (sub-sections 2.2 and 2.3), this section concludes the challenges mentioned in literature that face the proper exploitation of AI. It also demonstrates probable suggestions for better reaping the benefits of AI.

Given today's practices in AEC industry, [53] have imposed that direct application of AI might even increase uncertainties, unreliable predictions and poor management decisions. The researchers owed such a negative assessment to the lack of proper understanding the true benefit of AI tools and inability to translate their impact into real dimensions. Research of [54] has focused on the reason behind the uncertainties in these AI applications in AEC industry.

However, the research only focused on 5 types of AI algorithms namely: Primary Component Analysis, Multilayer Perceptron, Fuzzy Logic, Support Vector Machine and Genetic Algorithm. Causes of uncertainty mentioned varied according to the type of algorithm under investigation. Reasons for uncertainties included:

- Subjective assumption that the relationship between different input (observed) features is linear.
- Applying AI for optimizing a specific task in AEC industry might sometimes yield a limited data set for training. In case of neural networks, a limited dataset could lead to an unreliable output and perhaps overfitting of the problem in case of inability to expand the input to samples from different situations.
- Applying AI for solving larger complex problems, on the contrary to point 2, may be too challenging if not properly generalized, and in case of a neural network, underfitting might occur.
- Subjective assignment of values to an expert system might yield inconsistent systems where each one is based on the expert's personal judgment of the problem.

To mitigate such uncertainties, several assumptions and considerations could be used according to [54]. These assumptions were inspired by scholars from other disciplines apart from AEC industry and included: the consideration that there is a "black box" and the fusion of more than one system together. The first one meant the contribution of designers and engineers' intuition where such a tacit knowledge could be considered a "black box" where algorithms need to take that into consideration.

While an example of the second could be the use of a least square support vector machine optimized by particle swarm optimization (PSO-LSSVM) or the use of a hybridized fuzzy logic with supported neural network. Despite not being directly mentioned, Deep RL algorithms could also be considered an improved “hybrid” model.

Eventually, the authors have reached an important conclusion. This conclusion is considered one of the important pillars this research is built on. They have mentioned that given the particularity of AEC industry, where not only theoretical knowledge is required, but also design, engineering experience and intuition, the uncertainty of AI can’t only be solved by a single certain algorithm.

However, the solution must include the proper capturing of the tacit knowledge; hence a technique similar to fuzzy logic should be introduced to reduce the uncertainty of algorithms. In addition, due to the complexity of AEC projects, game playing techniques could be a solution to capture contributions from different stakeholders. Also, interdependent relationships between different technologies/phases in AEC industry should be taken into consideration. Lastly, the authors highlighted the absence of a detailed framework for the employment of heuristic algorithms in the context of AEC industry.

Similarly, other scholars in [55] have stated some conclusions when it comes to the implementation of deep neural networks applications in AEC industry. They explicitly indicated that there is a lack of using Deep learning architectures that implement reinforcement learning like deep Q-networks. [56] added that RL-applications were identified in AEC, but they are either still in experimentation phase or isolated into specific tasks that still need to be integrated into BIM workflow.

Moreover, [55] also mentioned that adopting AI in AEC industry requires *“embracement of change” and “re-engineering”* of the processes in order to reap the full benefits of AI on a proper scale. These changes should be applied on all possible levels such as organizational, technological, mindset and even cultural. To reach this, AEC organizations should invest in establishing AI methodologies and prioritize education and training of their employees on AI and the subsequent change in the work environment and tasks needed.

Finally, the same research also draws the attention that *a large portion of AI research in AEC tends not to implement ideas and theories from other research fields* but tends to build up on AI-related work within their own specific area of expertise.

2.2.4. Applying RL success in RTS games into AEC applications

The previous 2 sections have concluded, separately, the challenges towards the application of AI in RTS games and AEC industry respectively. It has been evident that most of the challenges derive from the complexity of both fields. Given the similar characteristics and challenges involved in the 2 fields, this research will benefit from the success achieved in adopting RL algorithms in RTS games and try applying it, in a similar approach, in a design and construction application.

This process shall also benefit from the advancements in game engines and the ability to integrate on-site characteristics into a simulated environment through IoT infrastructure. Therefore, RL-training can be accurately executed without any effect or damage in the real world, and once the training has reached a satisfactory level, it can easily be employed to real "robots" or other tools on site for actual implementation.

In order to apply this idea, the traditional "human" perspective in which the design and construction processes are regarded as 2 discrete processes shall be rethought. Now it is needed to wear an oculus but jump in a world where we put ourselves in the shoes of an AI agent. Once we do so some questions would pop out like:

- Could lessons learnt from AI in RTS games be implemented in AEC industry, thus introducing a new framework?
- Do we need then to "rethink" the whole design building problem not from the perspective of a "human Player" but rather from a perspective of an "AI Agent"?
- In that case, do we still need the segregation of the design-construction process into 2 separate stages from an AI perspective?
- Are the currently available computational tools, digital fabrication methods, and construction robotics capable of supporting this paradigm-shift and realizing it in real word?

In the following sections we will start to find answers to those preceding questions, but first we need to find answers from history.

2.3. Revival of the Master Builder

2.3.1. History behind the master builder

According to University of Colorado's design-build glossary, the definition of master builder is [57]:

"A term historically applied to an "Individual" who was responsible for both the design and construction of a project. During the Renaissance, a divergence appeared between the individual who prepared the project's design, and the individual who was responsible for its construction. With the rise of design professionals in the late 19th century, the term fell out of favor and is used infrequently today in reference to design/build firms."

Therefore, before this "separation", a Master builder was the designer and constructor at the same time. In antiquity, unskilled labor used to haul the materials, which could be large blocks of stones to the construction site. The skilled craftsmen would then transform them into artistic and structural components of the structure. The direction that they had to follow was controlled by the individual knowledgeable person who was responsible for the building design. At that time, the design would not be more than a mental "image" in the designer's mind.

Accordingly, the designer essentially had to be the builder in order to be able to direct his skilled craftsmen during actual construction on what they are supposed to do. This means that, the design and construction of a project were basically inseparable, and so the Master Builder would think during the design of a certain element, how would this element be realized on-site. If there is a difficulty in achieving that, he would also have to design the tools that could facilitate the construction of his masterpiece.

From the point during the Renaissance Period where the Architect and Engineer Filippo Brunelleschi has succeeded in using the camera obscura to copy architectural details from classical ruins, what we know now as "blueprints" was officially born [58]. Consequently, a "master builder" wouldn't need to travel from Florence to Rome for instance just to be present every single moment in the construction site. With the help of well explained drawings, he can then delegate his on-site duty to someone else.

Thus, by time, the designer's main aim became focused on the ability to express his ideas coherently in paper, and then a builder can take charge of the construction process. Therefore, the designer has no longer need to worry about the task of finding the convenient tools and methods for building realization. Instead, this task became completely assigned to the builder [59]. Nowadays, the term "designer" and "builder" refer to groups of hierarchical teams that collaborate together.

Recently, the term “Master Builder” was seemingly revived in the term “design build” [60]. This term defines a method of project delivery where one entity is responsible for both design and construction of a project. However, if we notice the internal process itself, it remained the same: 2 separated teams, one for design who delivers the work first and then hand it over to the Construction team that finds the optimum ways to properly execute the project on-site. Therefore, the “design-build” process hardly coincides with the core concepts of a “master builder”.

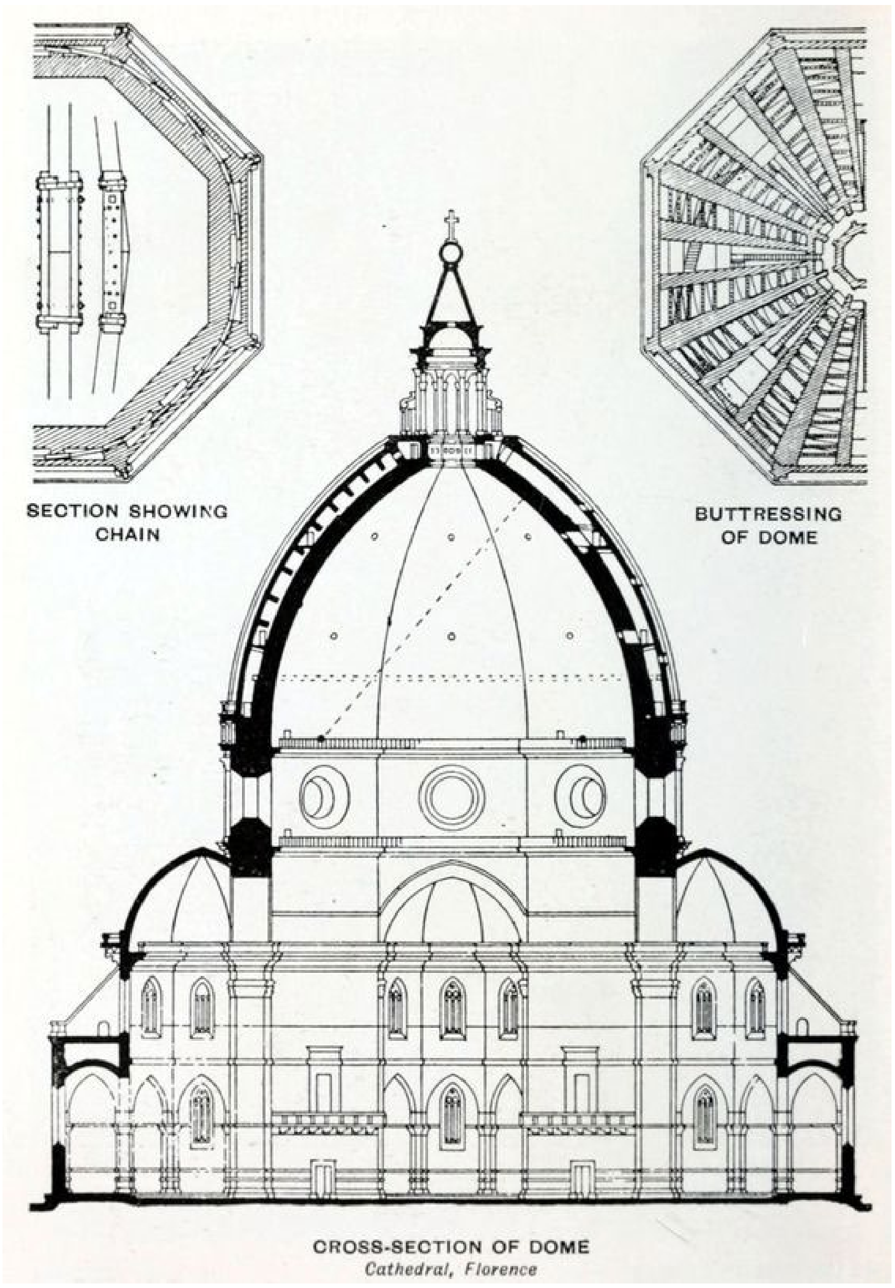


Figure 26: Section “blueprint” of Brunelleschi's dome of Florence Cathedral [61]

2.3.2. Introducing the new concept for the Master Builder

Inspired from the old term “Master Builder”, and by the inclusion of AI, the new version of “Master Builder” could finally see the light. Instead of discretizing the design and construction process, they are now united into one process. The resemblance of the new and old version can be explained further in the following lines.

As for the main “Master Builder”, it could refer to the team that possess design and construction knowledge. The team could be seen as a “puppeteer” who has in hands the tools necessary to control or guide multi-agents to achieve

the final goal; namely the proper design and construction of a building (Figure 27).

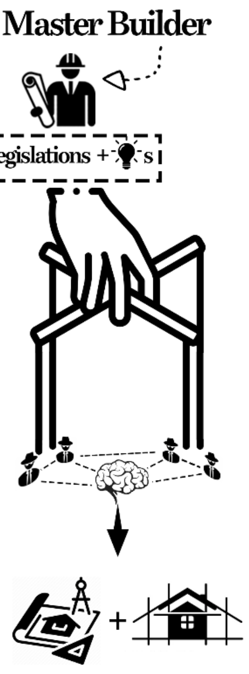


Figure 27: Scheme showing the main idea behind the new concept of "Master Builder"

The “Master builder” can start a project by an idea or a sketch of what he/she wants to achieve. Afterwards, it is required to come up with certain rules/guidelines that enlightens the “agents” during their learning journey. Thus as in old times, the “Master Builder” would act as a “Supervisor” making sure that his craftsmen undersood the design he has in mind. With proper insturctions from the “Master Builder”, these “Agents” can receive proper training and can then transfrom a simple idea in the head of the Designer into an actual functional building in real world. This can be done by the actual deployment of robots, the real world agents, that can carry out what they learnt in the simulated environment. Thus robots are practically our modern day “workforce”.

2.4. Proposed Framework for the design and build of structures using RL Agents (AI- Supported Agents)

In this section, the proposed framework is introduced, where AI, construction robotics and digital fabrication are situated in the core of the proposed workflow. As explained earlier, this framework revives the concept of "Master Builder" and initially speculates that an AI-centered design can inspire a new paradigm shift that redefines how we design and construct our buildings.

In Figure 28, the proposed framework is composed of 5 phases: Design Intent, Site Data Acquisition, Legislation, Learning and finally Robotic Execution. These phases can be explained as the following:

2.4.1. Design Intent

This is simply the initial phase of the project where a designer starts to brainstorm about an initial idea to start the project design. As a source of inspiration, the framework proposes some methods:

- Sketches – Main Contributor: Designer

  In this traditional solution, the designer holds a pencil in his/her hand and starts to brainstorm an idea and think about an inspiration after studying project details.

- Machine Hallucinations – Main Contributor: Designer + Agent (Machine):
  - In this solution, the machine, or an "AI Agent", acts as a catalyst for inspiration. Part of the idea evolution will be initiated by the designer and then by the help of one of the forms of machine "Hallucination", the designer can get inspired by what a machine can produce. For instance, style GANs (Generative Adversarial Networks) can be used to come up with "new" images based on a certain architectural style, or a mix of styles, that the designer initially trains the machine on. After proper training the Network can produce images that have never existed before yet follow in their form the input "style" or mix of "styles".
  - This solution is thus considered one of the forms of human – machine collaboration in the early phases of design. The AI agent is thus acting as a designer assistant.
- Only functional Structure – Main Contributor: AI Agent
  - In case the focus is more on a functional structure more than the aesthetics or uniqueness of the building, the solution could be carried out entirely from an AI Agent database. In other

words, the design intent can be directly formulated in the succeeding legislations phase set by the designer and followed by the AI Agent to achieve the building requirements.
    - This solution relies, more than the others mentioned, on the AI Agent so the design intent mentioned in the framework from a human designer perspective could be skipped.

2.4.2. Site Data Acquisition:

This phase is the pivot to jump from the real world into the simulated one. The accuracy of the data retrieved on-site is crucial to ensure a proper training of the agents on the “actual” site conditions. There are many advanced data acquisition technologies nowadays that can capture site details with high level of precision. Examples of site data acquisition types could be:

- Site Topology:
    - The forms and feature of site surfaces are variables that needs to be captured with a considerable number of details. Later, in execution phase, when actual robots can be deployed, if an error has occurred from the beginning in the site topology, misalignment of elements could occur relative to the proposed position in the simulated environment. In all cases, there is always a room for error, however this precision error should be carefully controlled so that on a larger perspective it won’t affect the project realization.
    - This site Parameter can be considered, a variable parameter that is never similar from one site to another. It can just be considered as a constant parameter if the focus is the site itself. This means that before any intervention on site, the parameter values remain almost the same. However, during execution, site modifications occurred but due to the actual activities done on-site and not because of an external parameter.
    - In most cases, site topological data is captured using laser scanners. These can vary from mobile scanners mounted on devices to fixed scanners mounted on tripods on site.
    - Similar parameters could include: Soil Layering and Scanning the surrounding elements around the construction (deployment) site.
- Prevailing Wind:
    - Prevailing wind is wind that blows consistently in a given direction over a particular region on Earth [62]. Due to factors such as uneven heating from the Sun and the Earth's rotation, this wind varies at different latitudes on Earth.
    - In dense urban areas, wind blowing on site becomes less reliant on prevailing wind direction. The parameter that becomes

more crucial is the distribution of the buildings surrounding our site. This building configuration determines how the wind flows in and out of the construction site.

- Therefore, prevailing wind data needs to be combined with accurate data on the surrounding structure in case of constructing in a dense urban area.

2.4.3. Design and Construction Legislation:

The phase is considered the starting point for revolutionizing the traditional workflow currently adopted. First, there is no more separation between design and construction, so both are regarded as one complete objective that is completed once an AI Agent succeeds in building an entire structure. The main responsible for this phase is the "Master Builder", and the AI Agents are considered as the students or the "Craftsmen". Thus, the process can be divided into the following:

- Simulation Environment Check:

  The designer has to initially make sure that the simulated environment parameters are well aligned with the actual site conditions.

- MDP Definition and Rewards/Penalties Assignment:

  This is considered the core of the legislation phase. The designer should first define how the "Real" Design and Construction Problem will be translated into a Markov Decision Process (MDP). MDP is the way we could describe the environment in RL terms for the agent to Understand [63].

  Then the designer needs to set the rules through which he/she can find it appropriate for helping the Agents succeed in the learning process and completing the tasks required from them.

  The rules can be translated in Reinforcement Learning terms as Rewards and Penalties; rewards being positive incentives given when the agent accomplishes a certain task and penalties being a form of punishment that indirectly informs the agent that it did something wrong or failed to achieve a certain task. Both rewards and penalties are considered a form of translating the tacit knowledge that the designer "Human" possess into certain criteria that the agent uses as its main guide during the learning process.

- RL Algorithm Selection:
  - The designer needs to consider the RL Algorithm(s) convenient for the tasks required to be solved by the agent. The proper selection of the learning algorithm, as well as the

hyperparameters, could help in speeding up the learning process of the Agents. This process requires some knowledge to be acquired by future architects/designers in order to easily adapt to this new paradigm shift.

- The process of RL Algorithm selection implicitly includes the whole planning process that is accompanied by the selection process. For instance, issues to be discussed and planned includes, but not limited to:
    - Hierarchical division of the RL-Problem into sub problems to facilitate better learning process for the Agents.
    - Combination of more than one algorithm to solve discrete parts of the problem.
    - Selection and Adjustment of the type of the algorithm to fit the MDP definition and the type of Agents involved (single vs Multi-Agent, Competitive vs Collaborative Agents ..etc.)

- Hyper Parameters Adjustment:

  This task is executed in parallel with the Agent Learning phase (the succeeding phase). Since the relation between Legislation and Learning phase is iterative (as shown in Figure 28), there is a need to monitor the learning process and the ability of the agents to pick up the lessons "quickly". Based on the performance of the agents in the simulated environment, the designer needs to fine tune, every now and then, the "hyperparameters" to make sure that the learning process is running smoothly.

- *3D GANs*:

  An additional feature that can be deployed in the future is 3D GANS. The actual application in the workflow remains flexible depending on how quick the advancement in this branch would evolve.

  3D GANs could eventually contribute to not just a conceptual "machine hallucination" but it could create a 3d-voxel-based detailed design. Although such a refined level of outcome is not yet reached, there could be a future potential in the development of 3D GANs. The main obstacle remains the massive number of voxels required for an architectural product and the complex interdependent relationships involved between different building elements that need to be respected in the design.

2.4.4. Design and Construction Learning:

This phase represents the time required for the agents to proceed from “Zero” to “Hero” in terms of experience and knowledge of performing the task allocated to them. To ensure the effective training, several procedures could be adopted, this includes:

- Initializing multiple environments with Agents learning in parallel. This procedure helps speed up the learning process.
- Variation of Environment Parameters to ensure that the agent can face several circumstances during the learning process. This procedure ensures that the agent, or the executing robot in real world, would be able to adapt to variations in the environment and can be familiar with these changes and be equipped with enough knowledge in the learning process to be able to take the proper action in real world.

2.4.5. Robotic Execution:

Once the designer has seen that the agents have learned enough and that they can perform the tasks required in a satisfactory way, the time comes to deploy the learnt lessons in real world. The executing robots would then be equipped with the “agents’ brain”. This would help translate the data perceived by the robot sensors into the values understood by the agents in the “simulated environment”. The agent would use the embedded Neural network model to feed the right action to the robot based on the observed input data. The robot would finally translate the perceived order from the agent’s brain into an actual step or an action in the real world

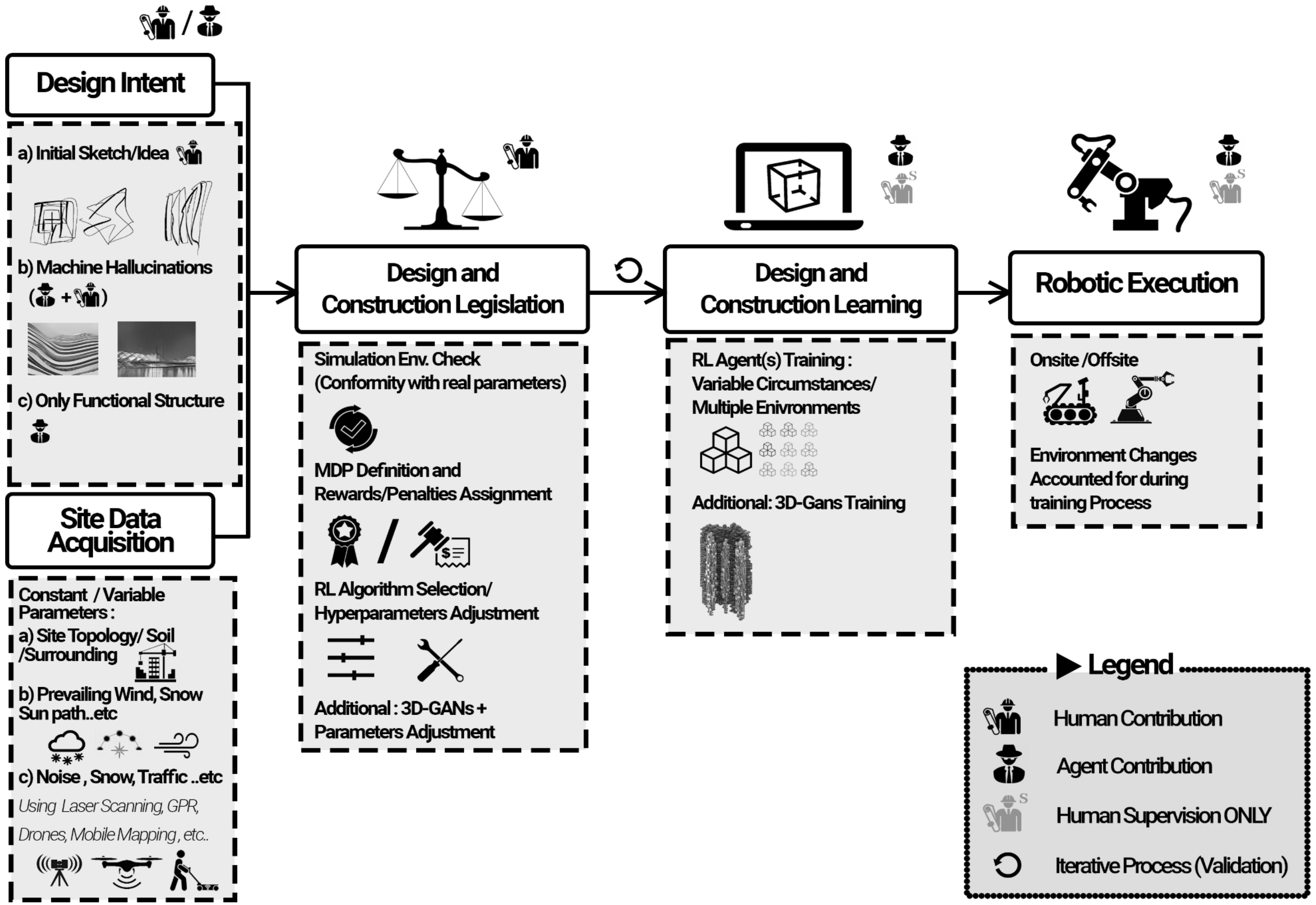

Design Intent
a) Initial Sketch/Idea
b) Machine Hallucinations
(+)
c) Only Functional Structure
Site Data Acquisition
Constant / Variable Parameters :
a) Site Topology/ Soil /Surrounding
b) Prevailing Wind, Snow Sun path..etc
c) Noise , Snow, Traffic ..etc
Using Laser Scanning, GPR, Drones, Mobile Mapping , etc..
Design and Construction Legislation
Simulation Env. Check (Conformity with real parameters)
MDP Definition and Rewards/Penalties Assignment
RL Algorithm Selection/ Hyperparameters Adjustment
Additional : 3D-GANs + Parameters Adjustment
Design and Construction Learning
RL Agent(s) Training : Variable Circumstances/ Multiple Enivronments
Additional: 3D-Gans Training
Robotic Execution
Onsite /Offsite
Environment Changes Accounted for during training Process
Legend
Human Contribution
Agent Contribution
Human Supervision ONLY
Iterative Process (Validation)

Figure 28:Proposed framework for design and Construction using AI Agents and Robots

2.5. RoBuilDeR – Proof of concept

"The Circle implies an idea of movement, and symbolizes the cycle of time, the perpetual motion of everything that moves, the planets' journey around the sun (the circle of the zodiac), the great rhythm of the universe. The circle is also zero in our system of numbering, and symbolizes potential, or the embryo. It has a magical value as a protective agent, … and indicates the end of the process of individuation, of striving towards a psychic wholeness and self-realization" (Julien, 71).

RoBuilDeR is a developed application that works as a proof of concept for the framework proposed. The application is developed in 2 versions namely: "Human Version" (UV) and "Robot Version" (RV). The first is concerned with providing an experimentation playground for the designers to evaluate different game parameters as well as assessing the design and construction rewards and penalties. On the other hand, the Robot Version represents the learning environment for the Agents.

In accordance with the framework, the scope of the application is focused on the Design and Construction Legislation phase and the Learning of Agents phase. The Robotics Execution phase is regarded out of this research scope, however there are many complimentary Literature that has investigated Robotic Execution whether on or off-site. Accordingly, RoBuilder focuses on the 2 phases that mainly differentiate the proposed framework from traditional workflow and are considered the real paradigm shift.

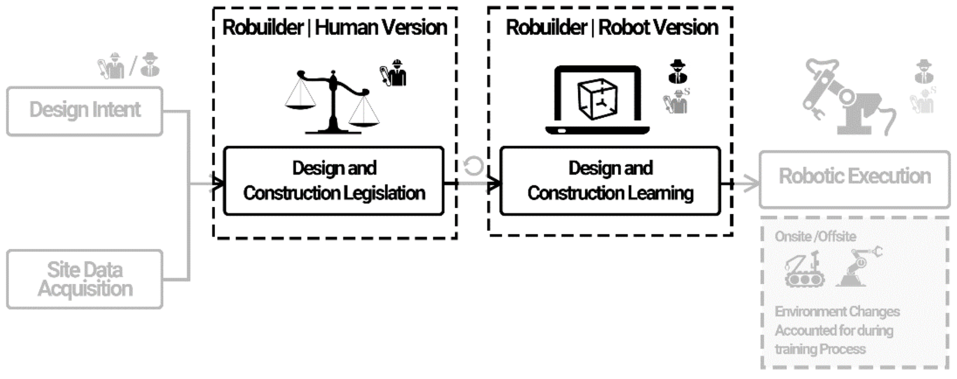

Robuilder | Human Version
Robuilder | Robot Version
Design Intent
Site Data Acquisition
Design and Construction Legislation
Design and Construction Learning
Robotic Execution
Onsite /Offsite
Environment Changes Accounted for during training Process

Figure 29: Scope of Application of RoBuilDeR Versions with respect to the Proposed Framework

2.5.1. Design Intent and Inspiration for RoBuilDeR

The initial design idea that will be developed in this research is inspired by a Roman miracle structure and one of the most conserved monuments of this era [64], namely the Pantheon. The pantheon was initially constructed as a temple in Rome dedicated to the Twelve Gods and to the Living Sovran [65]. Its present form is rebuilt by the emperor Hadrian (between 120-124 AD)

[64]. The structure can be divided into 2 parts: the portico and the rotunda (dome) (Figure 30). The dome of what is now serving as a church is the part of interest in this research.

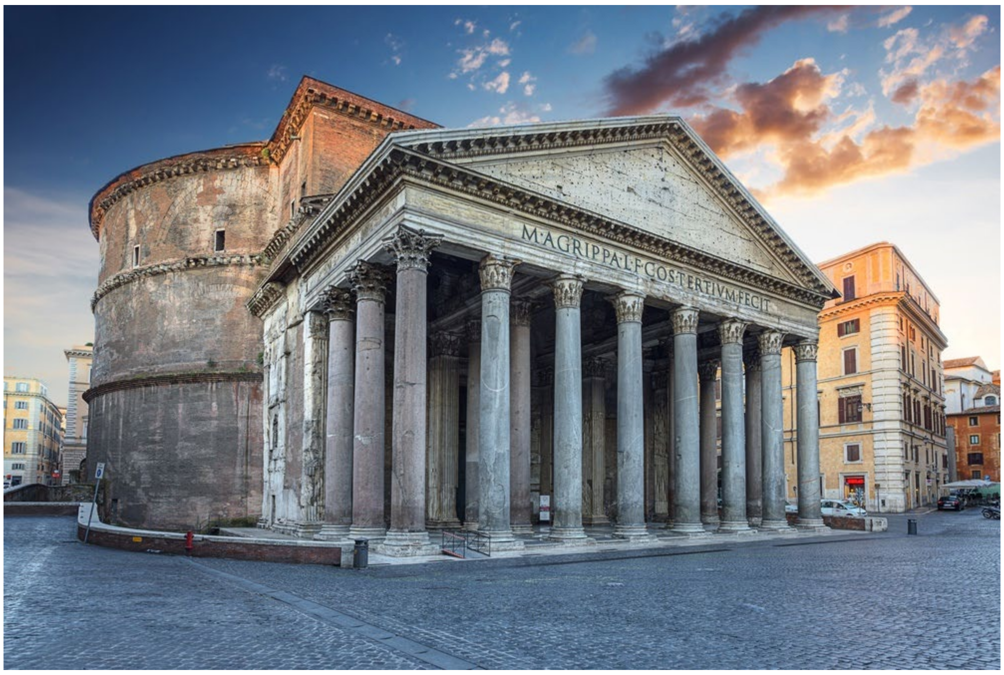


Figure 30: The Pantheon in Rome from the outside [66]

- *The Rotunda (Dome) - Pantheon*

The Rotunda is the spherical shaped dome structure. The height to the oculus and the diameter of the interior circle are the same, 43.3 meters (Figure 31) [67]. Almost two thousand years after it was built, the Pantheon's dome is still the world's largest unreinforced concrete dome [68].

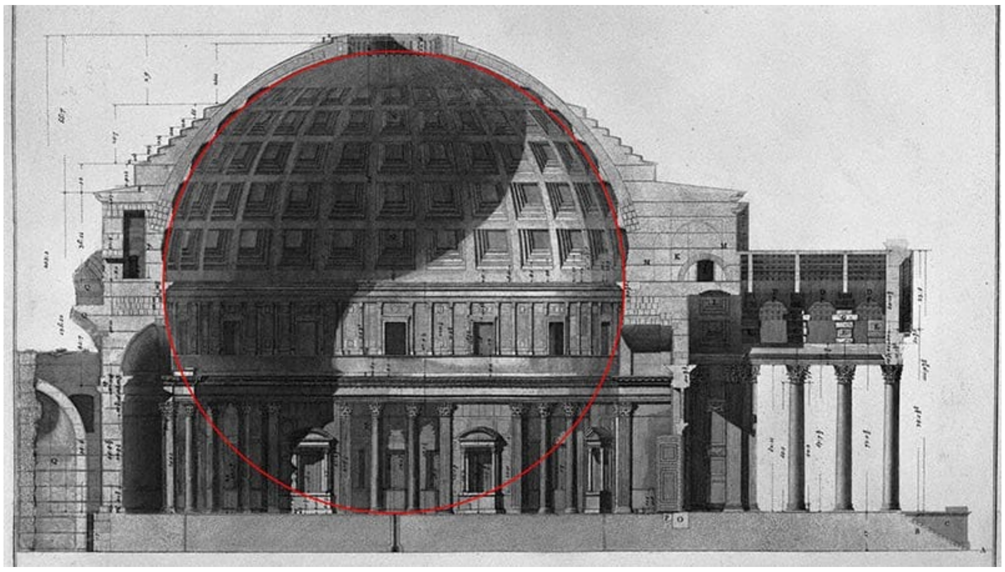


Figure 31: The Perfect Circle Inscribed inside the dome of the Pantheon Rotunda [69]

The uniqueness of the Pantheon in Roman Architecture as well as being highly conserved due to its continuous use, yielded this building as a standard exemplar of classical style revival. Hence, it has been copied several times in modern times. The central hall of the U.S. Capitol in Washington, D.C., and the rotunda of the General Grant National Memorial (Grant's Tomb) in New York City are examples of modern architects replication of the Pantheon [70].

There are still ongoing studies regarding the mathematical, astronomical and structural miracles conveyed in this monument. For instance, the structural integrity of the dome was secured by a series of massive, concentric stepped rings and the lightening of the dome by coffering and gradated, light-weight aggregates [71].

At the very top, where the dome would be at its weakest and vulnerable to collapse, the oculus actually lightens the load [72]. The oculus at the dome's apex and the entry door are the only natural sources of light in the interior. Throughout the day, the light from the oculus moves around this space in a reverse sundial effect [73]. The oculus also serves as a cooling and ventilation method. During storms, a drainage system below the floor handles the rain that falls through the oculus (Figure 32).

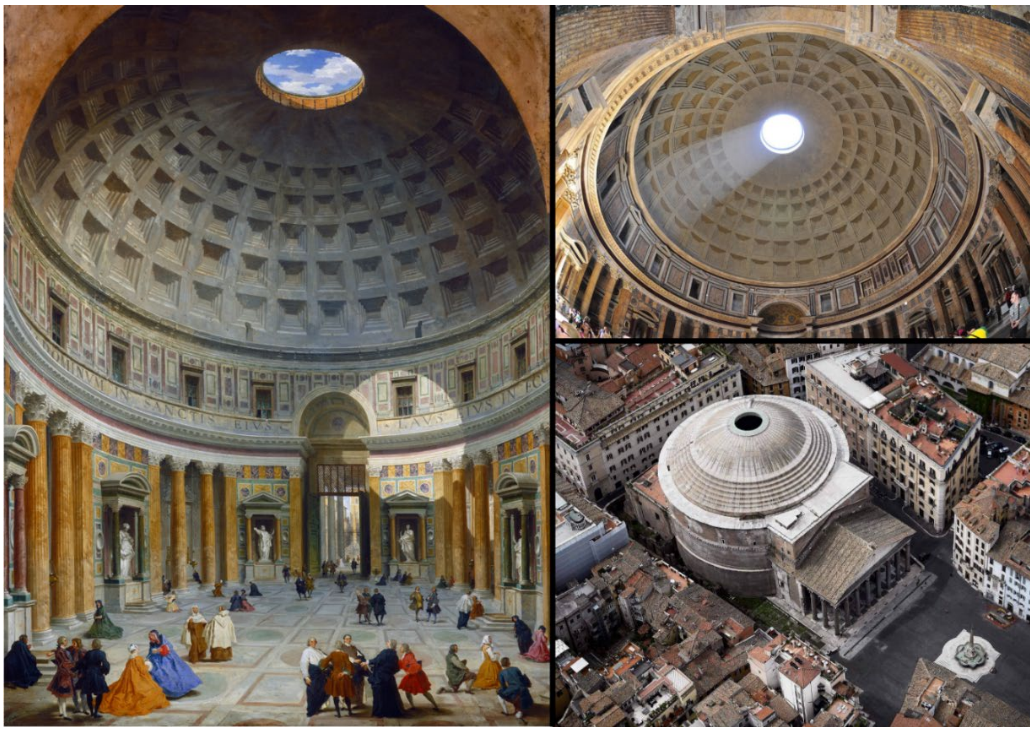


Figure 32: (Left) A painting showing how the oculus and the entry door provided light in ancient times vs a recent picture (top right) showing the sun light entering from the oculus and reflecting in on the inside of the dome. (Bottom right) is an aerial view of the pantheon within the surrounding urban context showing the portion of the dome left open which represents the oculus.

- *The Endless Wall – Gramazio Kohler Research*

On the other hand, this research project is also inspired, from the execution side, by the project of Gramazio Kohler Research named “The Endless Wall” and realized in 2011 [74]. This project is regarded among the very early investigations of the capability of industrial robots to be mounted and deployed in construction sites for building purposes. The mounted robot was equipped with an end effector that can perform “pick and place” of concrete bricks. The initial position of the brick supplies as well as the final position of each brick in the “circular wall” (Figure 33) was designed and checked by the responsible architects prior to execution by the help of a simulated environment. This environment replicated the actual site where the robot had been assigned to.

The main challenge in this task was to develop a “cognitive” language between the real “imperfect” world and the counterpart “perfect” simulated environment. A feedback loop was continuously checking the data retrieved from sensors and laser scanners. Given the complexity and unpredictability of a real construction environment the robot must be able to recognize its own position, the surroundings, and its components with regard to the material tolerances. Consequently, the robotic system has to respond to actual tolerances and be able to adapt to changing conditions autonomously [74]. The robot unit employed an innovative scanning system which enables it to orient itself and at the same time to process the information gained from different materials and surrounding environment.

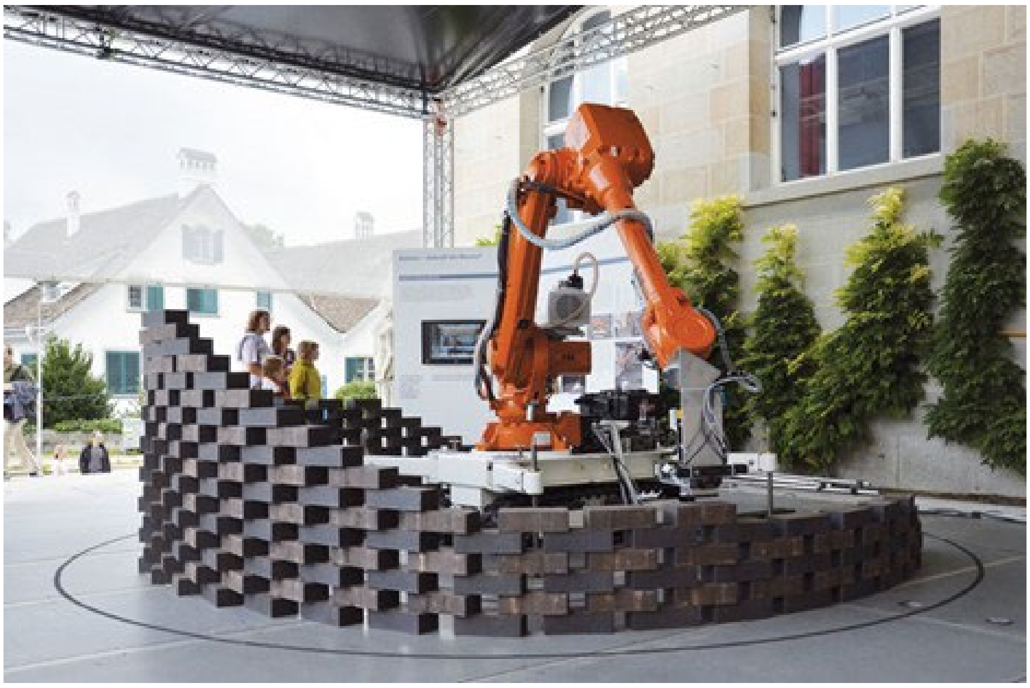


Figure 33: The Endless Wall, ETH Zurich, 2011 [74]

Accordingly, the 2 inspirational examples, combined, had a significant contribution to the initial preliminary design of the proposed Brick Pavilion.

Where the pavilion takes the “brick” nature of the “the Endless Wall” project, backed up by the success in robotic execution, and combines it with the oculus-dome inspired by the Pantheon’s “Rotunda” design.

This design also fits perfectly with the nature of bricks assembly with little to none mortar or adhesive involved. The presence of an open-ended dome would reduce the risk of falling bricks compared to the situation where the dome has been completely closed with mounted bricks. This benefit is also added to natural sunlight (Figure 34), stack ventilation and all the other benefits previously mentioned regarding the role of the oculus in the Pantheon’s design.

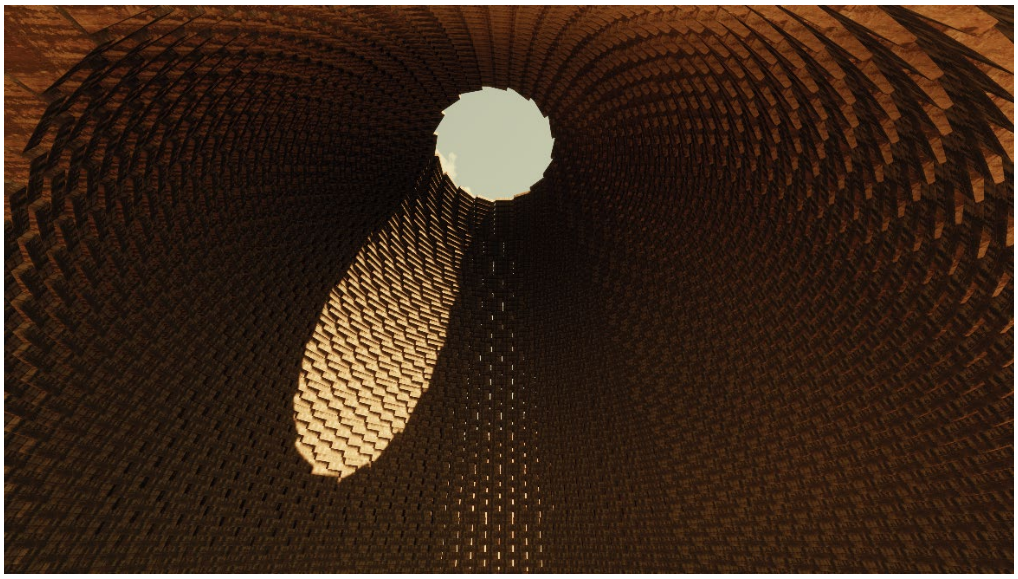


Figure 34: Imaginary Visualization Inside of the Brick Pavilion showing the Sunlight entering from the "oculus" and projecting on the bricks inside

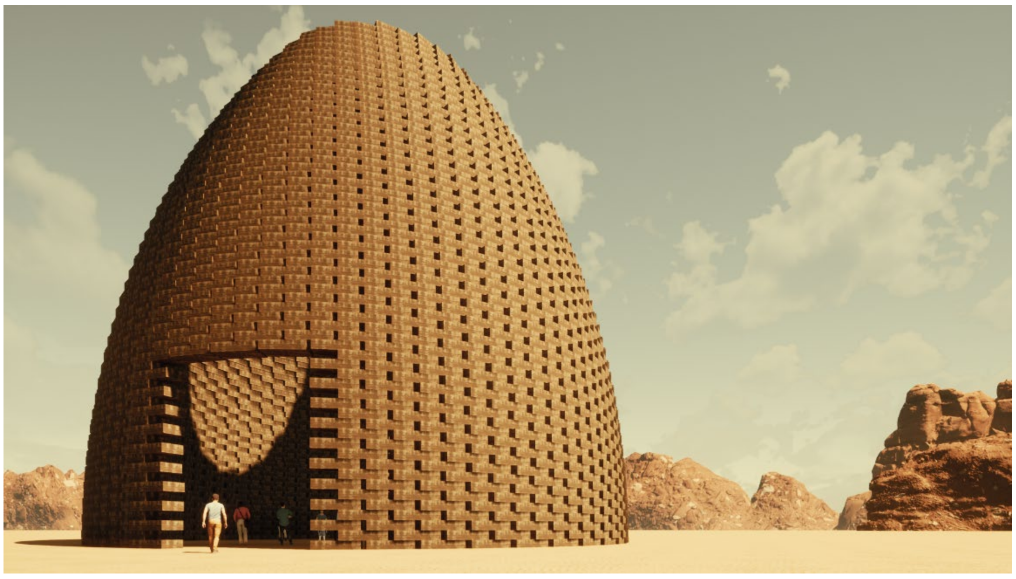


Figure 35:Imaginary Render of how the Brick Pavilion would look like

Starting from a circle, the idea of one of the most remarkable monuments in ancient times was born, and recently, from the simplest building unit; brick, placed accumulated in a circular manner, one of the earliest modern experiments of full robotic execution was rendered successful. Consequently, the robotic brick placement and the design of curved oculus dome, combined, are regarded as a perfect inspiration to the Brick Pavilion. This pavilion is sought to one of the earliest trials for the design and construction of a building executed by RL-Agents.

In the following sections, more details will be discussed regarding the design parameters involved in the Brick Pavilion, as well as how these parameters were employed within the game mechanics. In addition, a focus on the human version of the game (UVersion) will highlight how the designer can visualize the impact of changing one of the design parameters directly on the execution of the pavilion.

2.6. Game Development in Unity

Figure 36: Unity Logo

This section provides a brief explanation for the Choice of Unity platform for the application development. It also provides an explained diagram of the components and classes implemented in RoBuilDeR and the connections involved between them. This diagram was initially based on the Human Version (UV) of the game. The Robot Version (RV), however, has been tweaked to accommodate the different methods and classes provided by ML-Agents to allow for the proper external connection to the RL Algorithms necessary for the training of the Agents.

2.6.1. Why Unity?

To answer this question, first it is needed to briefly explain what **Unity** is. Unity is a cross-platform game engine developed by Unity Technologies. The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces [75].

Given the successful simulations carried out on this engine in the AEC industry, this game engine has also specific characteristics that is compatible with the nature of simulation required by this research. These characteristics could be summarized to the following:

- Physics-enabled environment.
    - Unity helps in simulating physics in a proper way to correctly accelerate and respond to collisions, gravity, and many other forces [76]. Regarding the 3D physics, Unity integrates and relies on Nvidia Physx engine. This allows for an efficient quasi

realistic response to all physics behaviors for an object. This allows for the automatic linking of an object response based on the material it was assigned to. In this research, the building bricks were modelled in Concrete. However, by changing the material type and properties the physics engine will handle all the changes implicitly and there would be no need to change any other thing from the user side.

- Nvidia Physx helps as well in simulating gravity for when the bricks fall. The gravity effect can also be witnessed when stacking a high pile of bricks on top of each other, where some tilting and slight variation in the original position could occur. These variations greatly help the agent in understanding some of the imperfections that occur in the real world.
- It also detects and simulates collisions between bricks and even the effect heavy wind blowing on the stability of binder-free stacked bricks.

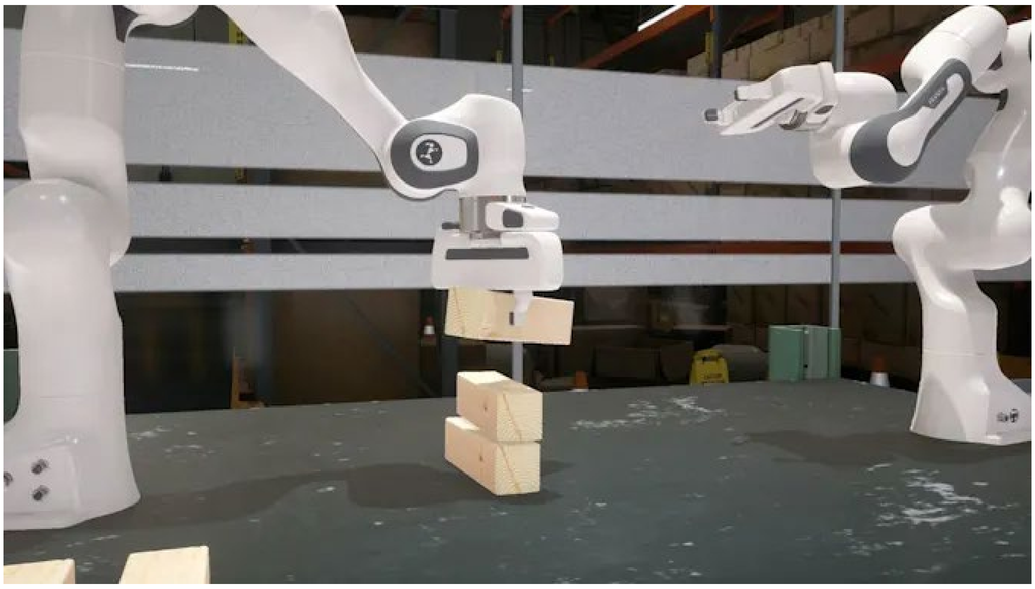


Figure 37: Nvidia Physx-supported Unity 3D environment can simulate complex behavior robotic tasks with high accuracy [77]

- Reinforcement learning compatible environment (ML-Agents):
  - The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents. The toolkit provides implementations of the of state-of-the-art algorithms to train intelligent agents for 2D, 3D and VR/AR games. Toolkit is mutually beneficial for both game developers and AI researchers as it provides a central platform where advances in AI can be evaluated on Unity’s rich environments and then made accessible to the wider research and game developer communities [78].

- This facilitates the direct transformation of an application manipulated by human users into a learning environment for Reinforcement Learning (RL) Agents to train. In addition, imitation learning could also be easily integrated by recording human players' moves in the game and providing these recordings for the agents to learn from.

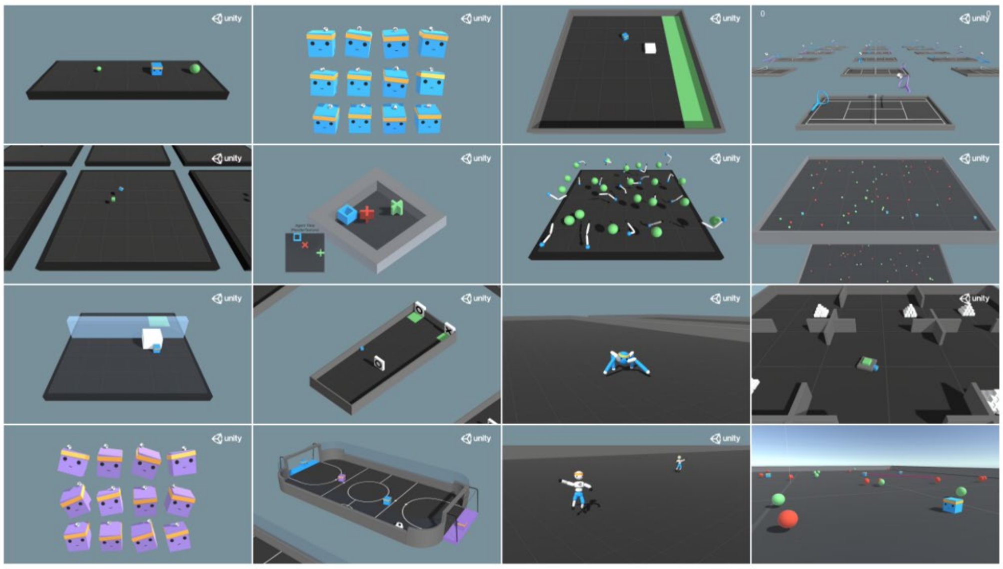


Figure 38: Different ML-Agents sample environments [79]

- Time Manipulation:
    - Along with the previously mentioned 2 factors, time or frame manipulation of physics effects are a crucial factor in the learning of the agents. For instance, given that the time speed during the learning process could learn 10x to 20x the normal time speed in real life, the effect of physics simulation should also be configurable to cope with such a change in the speed.
    - For instance, if a brick takes 1 second to fall from a certain height in real world and in normal times, such an effect should be reduced to 0.1-0.05 seconds during the agent learning process depending exactly on the value of "time scale" in which such simulation is running on. Otherwise, bricks falling, or any other PhysX process will be too slow compared to the current time speed the simulation is running on and hence there will be a major discrepancy in the agents' behavior when applied in the normal time scale and in the real actual situations.

- Interoperability
  - This feature allows the smooth transfer of data of any source and integrating it with the model developed in Unity. Examples of interest could include point cloud data from laser scanners, data from different sensors, and geometry from CAD environments like (Rhino, Autodesk Revit, ArchiCAD..etc.).
- Supporting almost all available platforms.
  - The developed application especially for the UV can be easily exported and launched on any platform like IOs, Android, Windows Desktop, Mac OS, Linux, and WebGL [80].

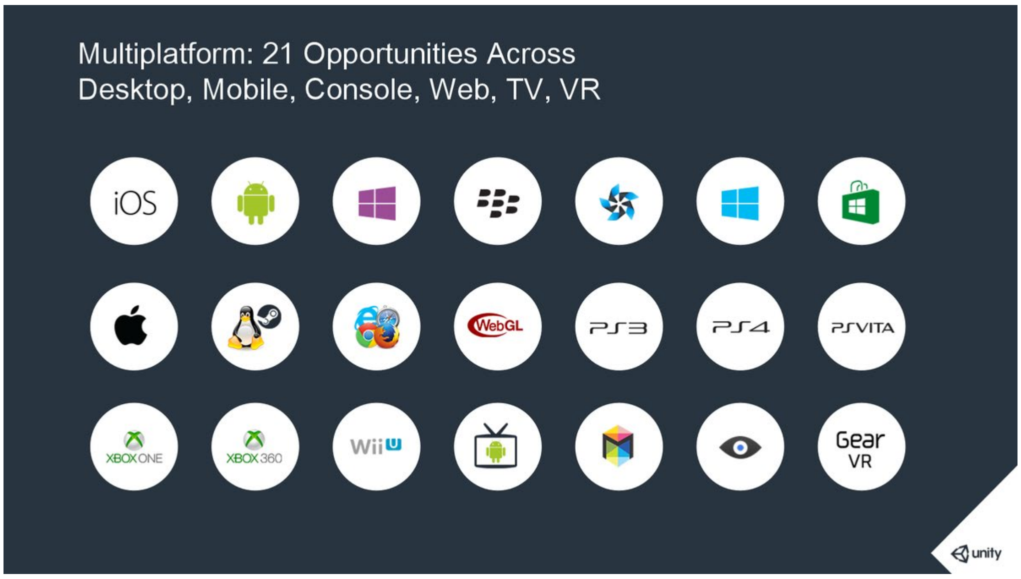

Multiplatform: 21 Opportunities Across
Desktop, Mobile, Console, Web, TV, VR
iOS
WebGL
PS3
PS4
PSVITA
XBOX ONE
XBOX 360
WiiU
Gear VR
unity

Figure 39: Unity 3D supported platforms [81]

2.6.2. Component and classes Diagram

Given the explanation of the reasons behind the development of the application in Unity, the following sub-section demonstrates how the game, in its 2 versions, is divided in terms of components (game objects) and classes attached to them. This is explained in the form of Components and Classes diagram as in (Figure 40).

This diagram identifies the main Game objects which are the term used in Unity for referring to elements or units deployed in a game scene in the application. For each Game object there could be a script class attached to it. This C# script defines a certain behavior to the attached Game Object. Unity allows the attachment of the same script to multiple Game Objects and allows the attachment of several scripts to one Game Object as well.

Therefore, the visualization of such complex relations in an abstract way like in a diagram could easily help understand the main scripts and game objects involved in the application. Such a demonstration tool helps also during the actual development itself of the game.

RoBuilDeR Components and Classes Diagram

Legend

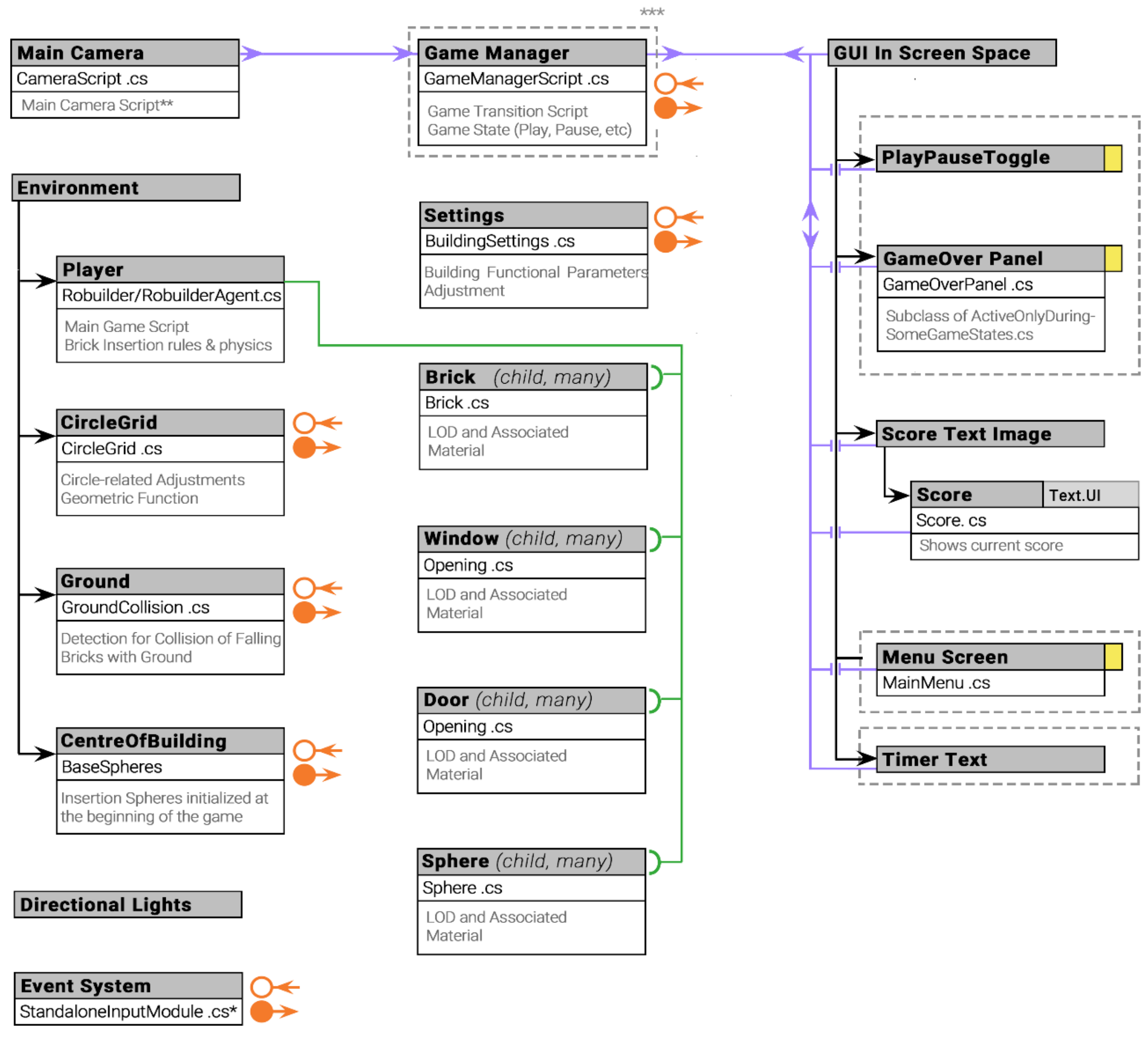
↳	A solid black line denotes a parent/child relationship in the Unity Heirarchy pane.
↳(	A solid line with a semi-circle on the end represents one GameObject instantiating another GameObjects as needed.
↳	A light purple arrow denotes a class pulling info from or utilizing another class.
▤	Boxes stacked together or touching each other are multiple Componenets on the same Game Object.
○ ●	The Robuilder.cs/RobuilderAgent.cs script has relationships with so many classes that are denoted with these lines. The open cirlce is data going from Robuilder script, and the closed circle is data going to it.
⬚	Scripts/GameObjects that are only available for UV only.

Independent Scripts

BrickLogic .cs
Helper Static Class : Snapping to right orientation,Methods for different Insertion Points

ScoreUtil. cs
Helper Static Class : Methods for score Calculations

ActiveOnlyDuringSomeGameStates. cs
Sets GameObjet.active based on GameState



Notes

* This module determines some paramteres like Input Actions per Second. From one point it is necessary to adjust the game Mechanics for Uversion (user version), it's preferred to maximize it for (Rversion) to speed up the computation process (i.e: Increases the number of actions taken per second)
** Main Camera script has several features for (UV), this helps navigate in different perspectives to help explore the game terrain in various situations before taking decisions.
*** In (RV), GameManagerScript gets merged with RobuilderAgent.cs

Figure 40: Components and Classes Diagram RoBuilDeR

In the following table (Table 1), a more detailed explanation of the important Game Objects and Scripts is demonstrated as well as the indication of whether this class is only available in UV of the game or is also present in the RV.

Table 1: Explanation of Main Game Objects and their attached scripts (if any)

Game object	***Explanation***	***Attached Script***	***Explanation***	***Available in***	
				UV	***RV***
Main Camera	Controls the main Camera deployed in the Scene	CameraController.cs	In UV, keyboard and mouse controls are enabled to allow for the ease navigation of the scene and visibility of all sides of the pavilion during construction.	Y	Y
Building Area (Environment)	Each Instance of a "Building Area" GameObject includes all Active Gameobjects required during gameplay. In case of the instantiation of bricks, windows, or doors, they will also be inserted in the hierarchy under the “Building Area” they were instantiated in. In case of UV, there is generally only 1 Environment, however in RV, multiple environments are instantiated in the same scene, during training process, to allow for parallel learning of the agents in different environments simultaneously. Thus, this configuration yields a more effective learning.	None		Y	Y
Player	**Parent -> Environment(Building Area)** The GameObject has no physical existence. However, it is considered as the player that manipulates the design and building process with	Robuilder.cs	Main script in the game that takes control of the game mechanics during "Game Play Mode" and handles all the processes like: Instantiating bricks, Windows, or Doors Instantiation, and the update of "Insertion Spheres". These	Y	N

	the help of the script attached		spheres are potential points for inserting bricks, doors or windows later on in the game. Handling the change of decision between inserting brick, door or window. Handling the physics behavior of the bricks and controlling when to disable this parameter. Checking the success or failure of a game episode and reporting to the game manager Updating the score with the help of methods from external static classes.		
		RobuilderAgent.cs	This script is only used in the RV of the game and in addition to all the processes already executed by the Robuilder.cs script, it also acts as the GameManager and implies all the methods needed to be implemented by ML-Agents toolkit in order to be able to convert the problem into a Markov Decision Process (MDP) necessary for training the Agents.	N	Y
Circle Gird	**Parent -> Environment(Building Area)** This GameObject includes the "building ring territory" attached to it. This ring represents the available initial circular boundary available for brick insertion at the beginning of each episode.	CircleGrid.cs	This script has all the functions needed to calculate the initial "base sphere points". These are the points that are instantiated only at the beginning of each episode to represent the allowable insertion points for new bricks or door. It also contains several static functions that are concerned with the translation from one point on the circle to another, dealing with angles, and vectors at various heights.	Y	Y
Ground	**Parent -> Environment(Building Area)**	GroundCollision.cs	The script has only one function "OnCollisionEnter" that detects whenever a brick	Y	Y

	The available Ground "Land" in each environment instance.		has fallen and collided with the ground at any point. Thus, only when the collision has been detected, the method would report this event to the "Listeners" in order to take the appropriate actions (ending the game or showing the Game Over GUI in case of UV).		
Brick	**Actual Brick instantiated during game play**	Brick.cs	Brick-Specific methods and variables that differs from one brick to the other.	Y	Y
Window	**Actual Window Opening instantiated during game play**	Opening.cs	Depending on the type of the Opening and its insertion location, each instance would have its own specific values for different variables.	Y	Y
Door	**Actual Door Opening instantiated during game play**				

Moreover, Table 2 provides an explanation of the static independent classes that contain different methods used inside of the main scripts.

Table 2:Explanation of the Main Independent Static Classes

Independent Script	**Explanation**
ScoreUtil	Contains all the static methods for the calculation of rewards/penalties based on the current action taken. For example, if a brick is inserted, the "Robuilder.cs" script recalls the static functions from "ScoreUtil.cs" that would eventually calculate if this brick at this position generates any kind of reward or penalty. If yes, such a value would be reported to the score manager, cumulated to the current total score and appears on the screen (in case UV). In case RV, the score reported from a "ScoreUtil.cs" method is directly inserted in a method called "AddReward()" which is responsible for collecting and accumulating rewards and penalties for each agent in each episode.
BrickLogic	Contains all the static methods related to the insertion of brick and its relationship with the surroundings. It also contains methods for: - Generating insertion points above each new brick - Checking the possibility of clash if a brick would be inserted in one of these points later. - Generating Insertion points for bricks on the top of any Window or Door Openings and methods for performing same safety checks for regular bricks insertion.

2.7. RoBuilDeR – Human Version (UV)

In this section, UV is explained in more detail. It starts with a demonstration of the User Interface (UI) and the illustration of different GUI objects on the screen. Then, a brief walk-through of the several actions that could be taken during gameplay and possible consequences are presented. Afterwards, different scenarios leading to ending game or to game completion is discussed. Finally, the section ends with a list of all properties that can be controlled by the designer to adjust different design and construction parameters. These parameters are then utilized for the learning process of the RL-Agents.

2.7.1. User Interface

- *Start Menu*

In this sub-section, different canvases of the game will be displayed and briefly explained. The game starts with the preparation guide with optional tips regarding game mechanics is offered. Afterwards, the game play features start to appear on canvas once the actual game starts. Depending on the progress in the game, “Game Completion” or “Game Over” screen would eventually appear. In case of pausing the game, the pause menu appears.

To start with, once the HV game is open, the following screen appears:

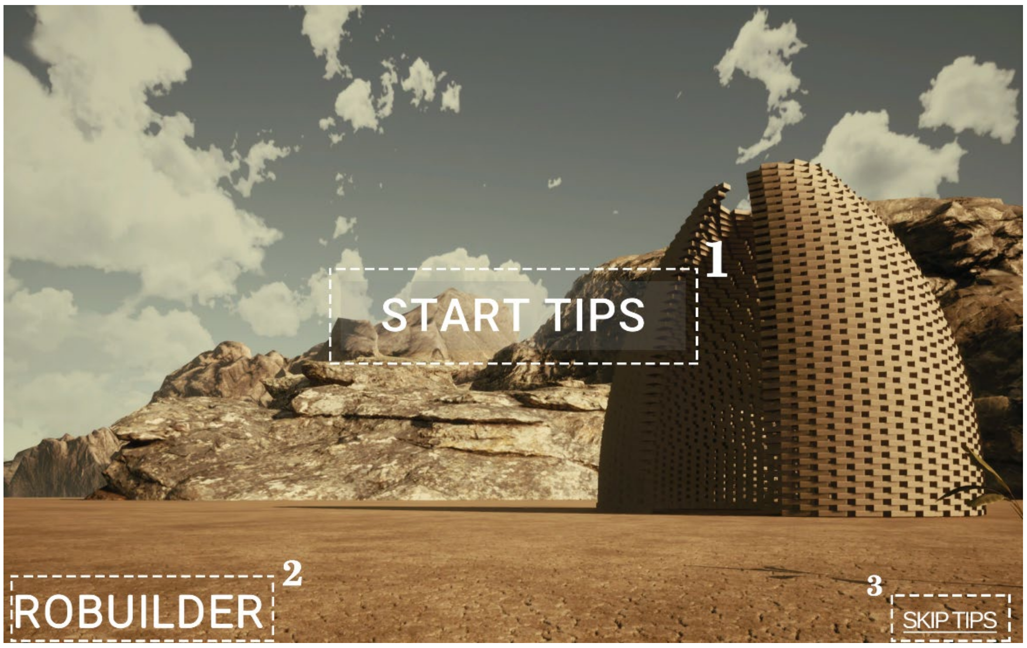
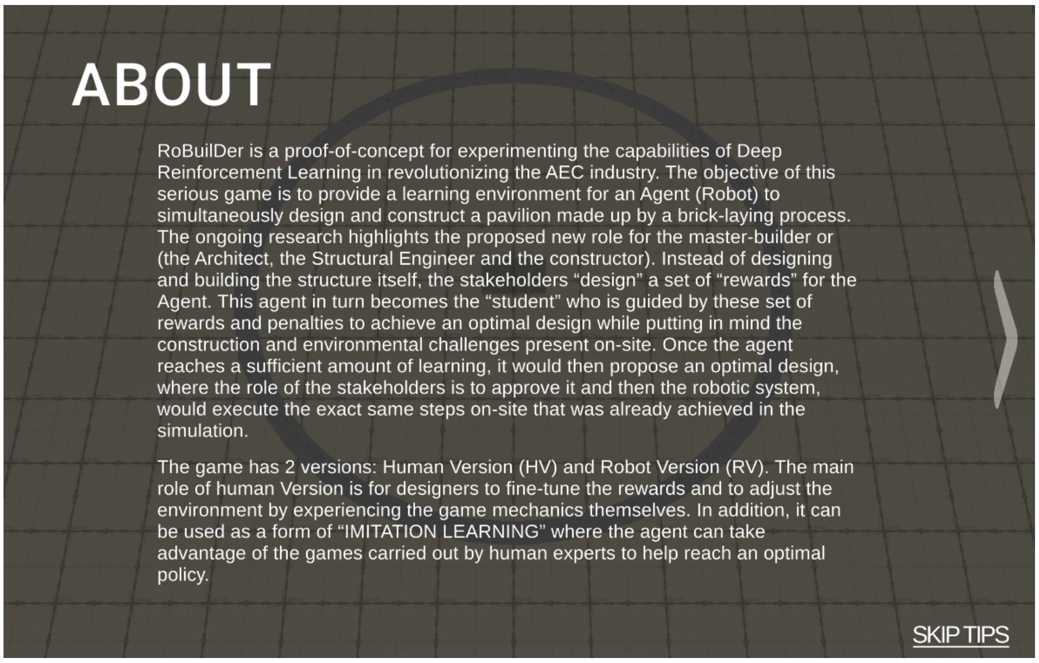


Figure 41: Start Screen in RoBuilDeR

Figure 41 is the start screen where it shows in the background an imaginary render of the Brick Pavilion in addition to the highlighted items, namely:

1. Start Tips (Button): On clicking, a series of game tips is shown.
2. The Game name is shown on the bottom left side of the screen
3. Skip Tips (Button): On clicking, the game moves immediately to game play canvas and the game starts.



ABOUT

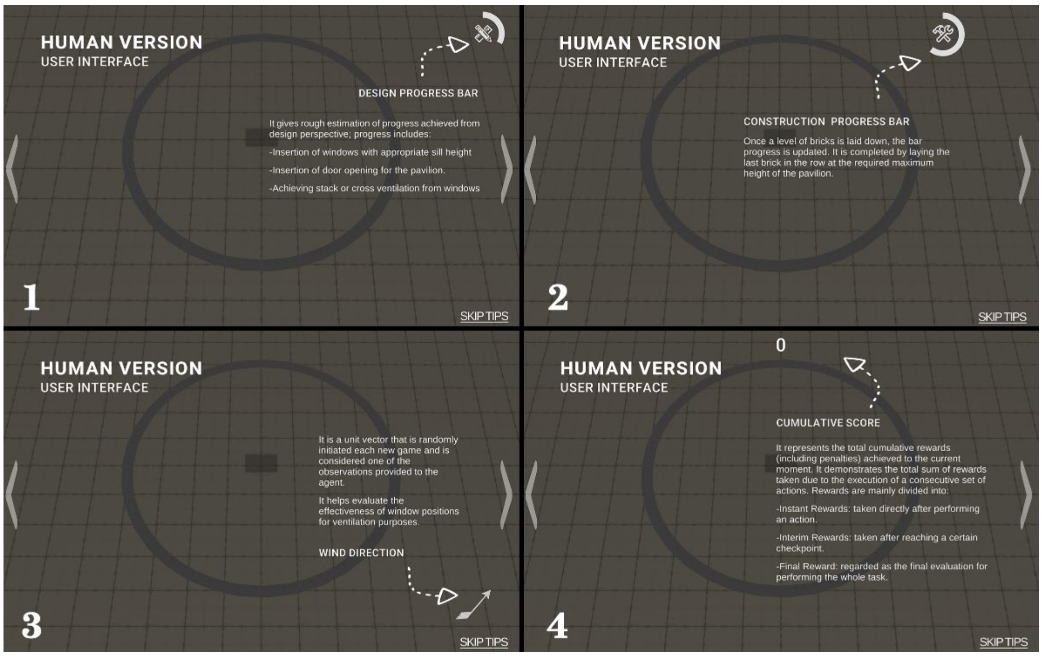
RoBuilDer is a proof-of-concept for experimenting the capabilities of Deep Reinforcement Learning in revolutionizing the AEC industry. The objective of this serious game is to provide a learning environment for an Agent (Robot) to simultaneously design and construct a pavilion made up by a brick-laying process. The ongoing research highlights the proposed new role for the master-builder or (the Architect, the Structural Engineer and the constructor). Instead of designing and building the structure itself, the stakeholders “design” a set of “rewards” for the Agent. This agent in turn becomes the “student” who is guided by these set of rewards and penalties to achieve an optimal design while putting in mind the construction and environmental challenges present on-site. Once the agent reaches a sufficient amount of learning, it would then propose an optimal design, where the role of the stakeholders is to approve it and then the robotic system, would execute the exact same steps on-site that was already achieved in the simulation.

The game has 2 versions: Human Version (HV) and Robot Version (RV). The main role of human Version is for designers to fine-tune the rewards and to adjust the environment by experiencing the game mechanics themselves. In addition, it can be used as a form of “IMITATION LEARNING” where the agent can take advantage of the games carried out by human experts to help reach an optimal policy.

Figure 42: First Screen Appearing in for Game Tips Provided. It gives brief explanation of RoBuilDeR

Figure 42 is the first screen appearing after clicking on “Start Tips” Button. This first screen provides a brief explanation of RoBuilDeR and the motivation behind developing this application.

Afterwards the following series of screens are shown consecutively (Figure 43). These screens explain each icon featured during game play, and how they get updated or changed based on the performance of the player and achieving certain milestones during game play.



HUMAN VERSION
USER INTERFACE

DESIGN PROGRESS BAR

It gives rough estimation of progress achieved from design perspective; progress includes:

-Insertion of windows with appropriate sill height

-Insertion of door opening for the pavilion.

-Achieving stack or cross ventilation from windows

1

SKIP TIPS

HUMAN VERSION
USER INTERFACE

CONSTRUCTION PROGRESS BAR

Once a level of bricks is laid down, the bar progress is updated. It is completed by laying the last brick in the row at the required maximum height of the pavilion.

2

SKIP TIPS

HUMAN VERSION
USER INTERFACE

It is a unit vector that is randomly initiated each new game and is considered one of the observations provided to the agent.

It helps evaluate the effectiveness of window positions for ventilation purposes.

WIND DIRECTION

3

SKIP TIPS

HUMAN VERSION
USER INTERFACE

0

CUMULATIVE SCORE

It represents the total cumulative rewards (including penalties) achieved to the current moment. It demonstrates the total sum of rewards taken due to the execution of a consecutive set of actions. Rewards are mainly divided into:

-Instant Rewards: taken directly after performing an action.

-Interim Rewards: taken after reaching a certain checkpoint.

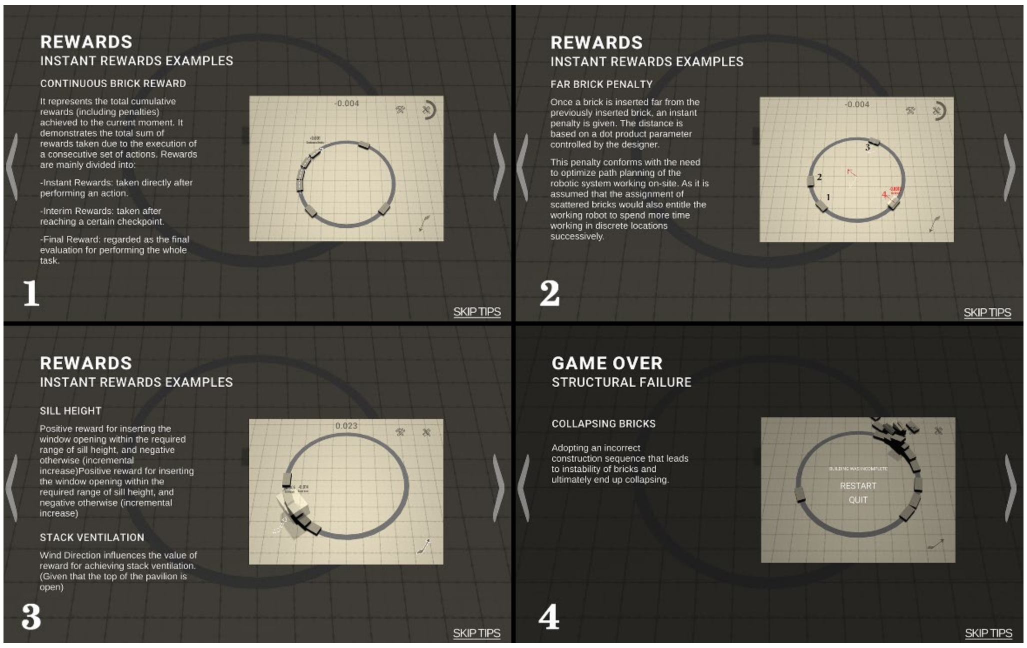
-Final Reward: regarded as the final evaluation for performing the whole task.

4

SKIP TIPS

Figure 43: Tips provided regarding Game Play Canvas Features

Following the explanation of different Game Canvas features, the last group of game tips are shown. These tips explain how rewards/penalties work during gameplay as well as demonstrating a game over scenario (see Figure 44).



REWARDS
INSTANT REWARDS EXAMPLES
CONTINUOUS BRICK REWARD
It represents the total cumulative rewards (including penalties) achieved to the current moment. It demonstrates the total sum of rewards taken due to the execution of a consecutive set of actions. Rewards are mainly divided into:
-Instant Rewards: taken directly after performing an action.
-Interim Rewards: taken after reaching a certain checkpoint.
-Final Reward: regarded as the final evaluation for performing the whole task.
1
SKIP TIPS

REWARDS
INSTANT REWARDS EXAMPLES
FAR BRICK PENALTY
Once a brick is inserted far from the previously inserted brick, an instant penalty is given. The distance is based on a dot product parameter controlled by the designer.
This penalty conforms with the need to optimize path planning of the robotic system working on-site. As it is assumed that the assignment of scattered bricks would also entitle the working robot to spend more time working in discrete locations successively.
2
SKIP TIPS

REWARDS
INSTANT REWARDS EXAMPLES
SILL HEIGHT
Positive reward for inserting the window opening within the required range of sill height, and negative otherwise (incremental increase)Positive reward for inserting the window opening within the required range of sill height, and negative otherwise (incremental increase)
STACK VENTILATION
Wind Direction influences the value of reward for achieving stack ventilation. (Given that the top of the pavilion is open)
3
SKIP TIPS

GAME OVER
STRUCTURAL FAILURE
COLLAPSING BRICKS
Adopting an incorrect construction sequence that leads to instability of bricks and ultimately end up collapsing.
4
SKIP TIPS

Figure 44: (1,2 and 3) Rewards Explanation. (4) Game Over scenario Demonstration

- *Pause Menu*

Finally, during gameplay, once the "Esc" button is clicked, the Pause Menu appears. It includes the following:

1. Resume (Button): Resumes the game
2. Restart (Button): Restarts the game turning back to the first screen.
3. Quit (Button): Exits the Application.

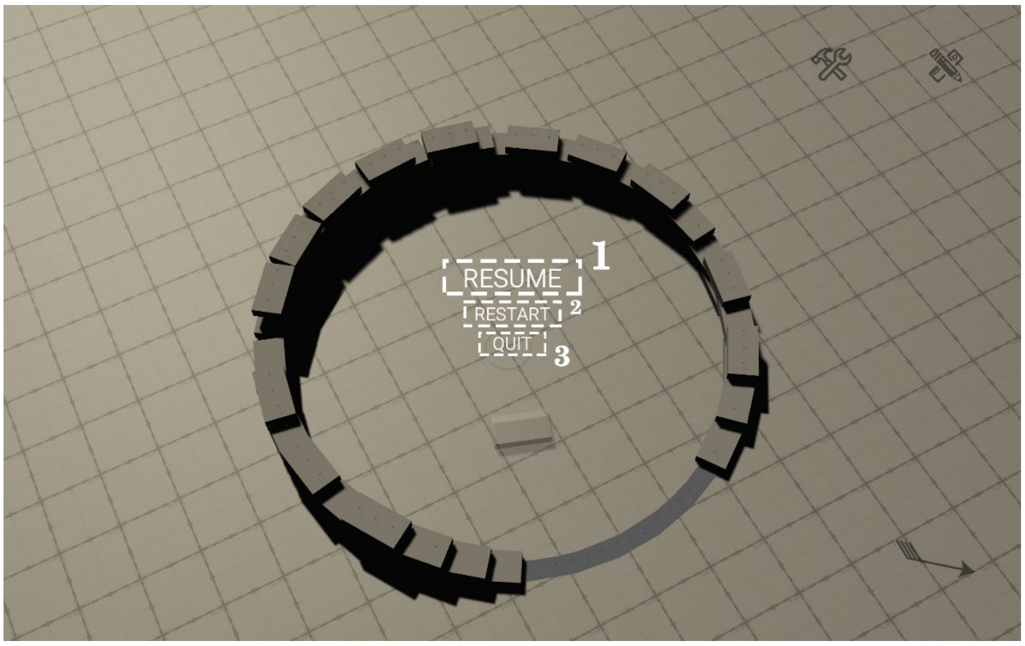


Figure 45: Pause Menu and its components

2.7.2. Game Play and Functions

In this sub-section, the game controllers and navigation keys are introduced, followed by a demonstration of possible actions taken during gameplay.

- *Game Play Navigation Keys*

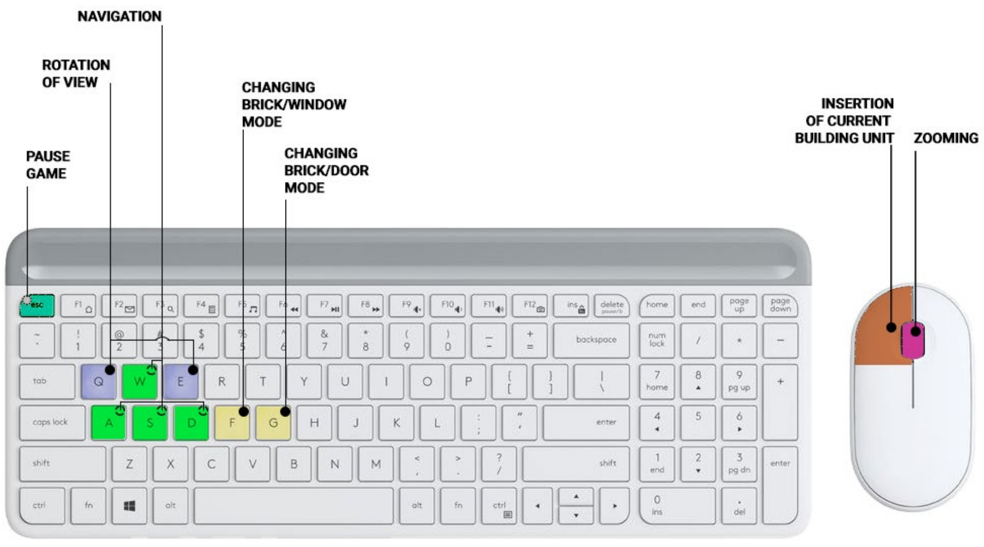


Figure 46: Keyboard and Mouse Controllers

Figure 46 shows the keyboard and mouse buttons highlighted in different colors. Their functions can also be summarized in the following table:

Table 3: Summary of different keys/buttons used in the game and their corresponding functions

Keyboard / Mouse Button	**Function**
Esc	Pausing / unpausing the game
W / A / S / D	Navigation Forward - Left - Backward - Right
Q / E	Rotation Clockwise/Anticlockwise
F	Changing between brick and window mode
G	Changing between brick and door mode
Left Mouse Button	Insertion of the Current Unit in the nearest possible position (if exists)
Scroll (Mouse)	Zooming

- *Game Play Actions:*

Since the goal of the game is to realize the Brick Pavilion, the main game actions can thus be summarized into: *Brick*, *Window* and *Door Placement.*

A. *Brick Placement:*

Brick is the fundamental building unit in the pavilion. Once a brick is inserted in one of the available positions it becomes instantly part of the structure, i.e.: the player can no longer move the brick from its place, and its color changes from *transparent* to its own *material (ex: concrete)*. Once the player clicks and the brick is inserted, another new "transparent brick" is instantiated at the cursor's current position on screen (see Figure 47).

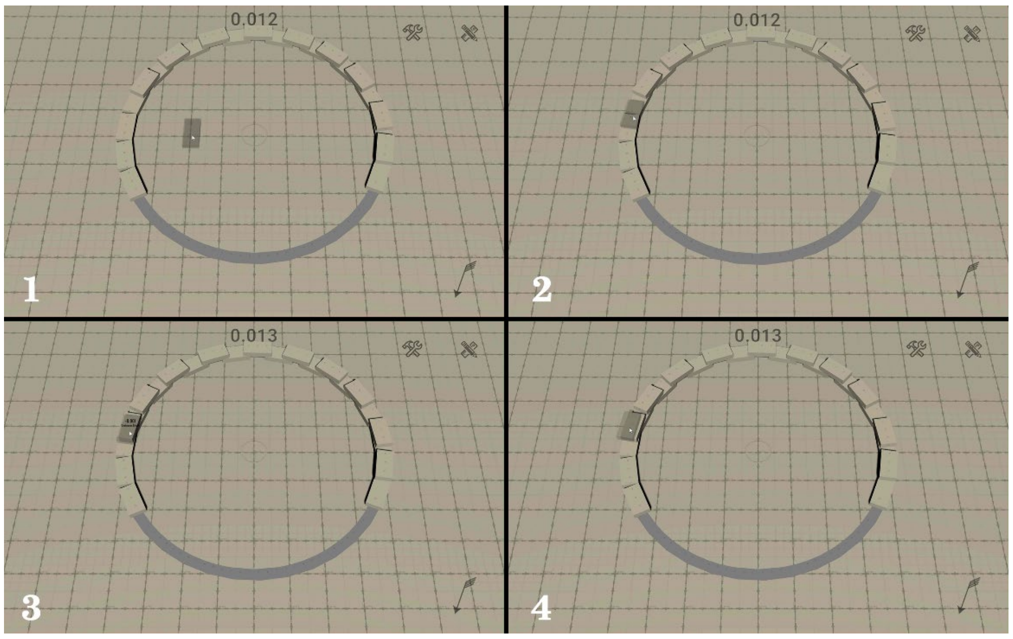


Figure 47: Steps of Insertion of brick: (1 - 2) Moving from random position towards the desired position, (3) Clicking and Placing, (4) Instantiation of new real brick in the nearest possible position and transparent dummy brick shift over the newly inserted brick.

This "fake" brick directly follows the point at which the cursor is pointing on the screen translated to actual 3d position in the environment. The motivation behind the transparent dummy brick is to give the player a chance to evaluate the possible placement positions for the next brick before actual placement takes place (see Figure 48).

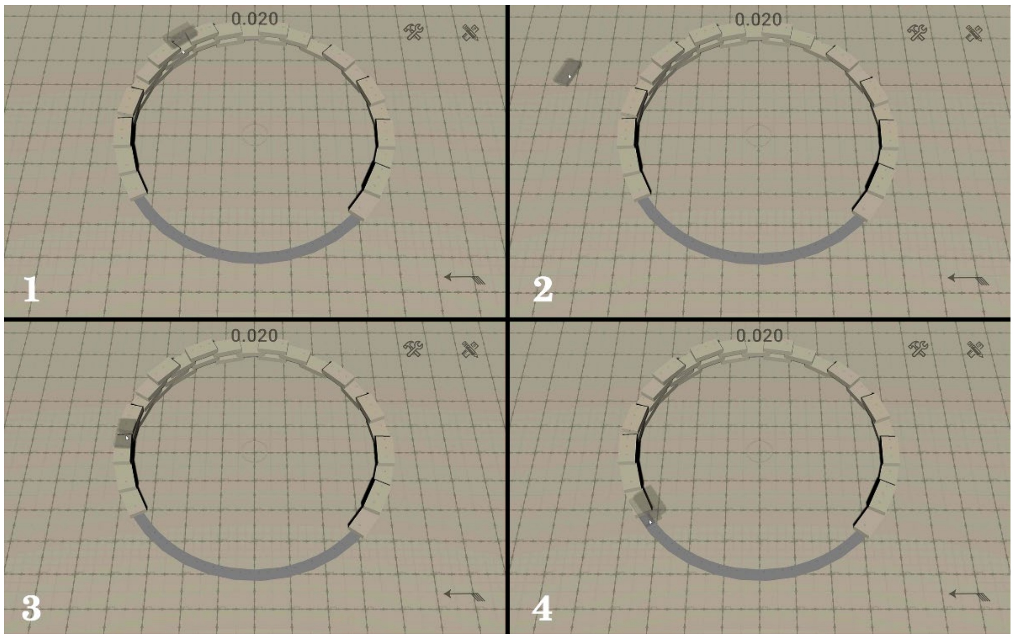


Figure 48: Dummy brick adjusts its position based on its location on the circular grid. This helps the player in visualising any "hypothetical" brick position before actual placement

Moreover, the action of placing a brick can be categorized into 3 scenarios:

- *Brick Placement on the base ground*: The bricks are based directly on the ground by selecting one of the available "base spheres" points. This term comes from the fact that these are insertion points lying on the base ground.

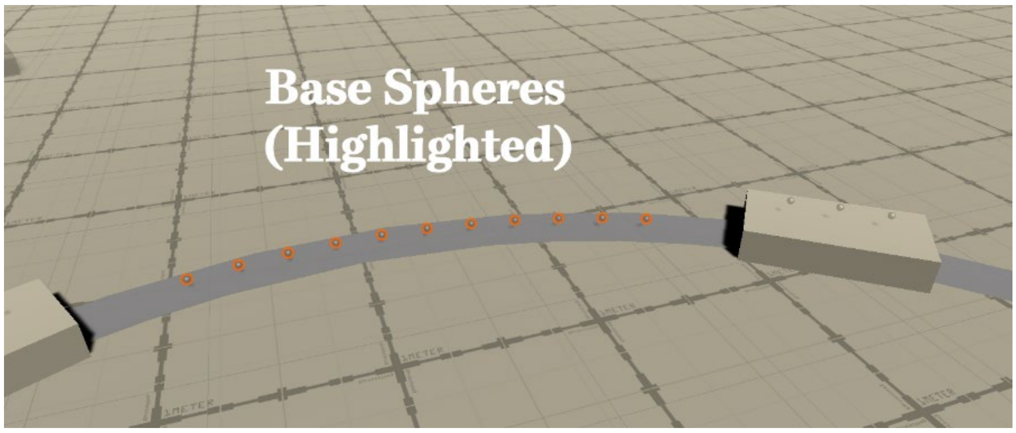


Figure 49: Base Sphere placed on the ground

- *Brick Placement on top of existing bricks*: These bricks are placed on safe positions that are available on the exposed face of the existing brick structure. These safe positions are represented by circular spheres as in Figure 50.

  The sphere points and their location depend on many factors. For example, one factor is the proximity and actual location of bricks, doors, and windows in the vicinity of this brick which could cause clash with the newly inserted brick. Another factor is the value of a parameter adjusted by the designer that controls the abundance of these points. Also, the current phase of construction, i.e., dome construction, base construction, windows insertion etc., could directly affect the number of points due to geometry variations occurring in the structure.

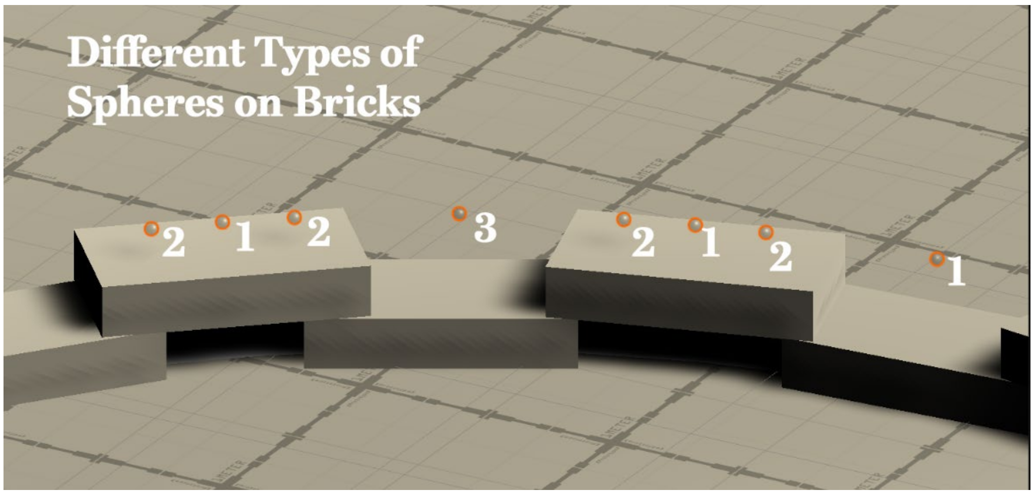


Figure 50: Insertion Spheres on top of existing bricks are highlighted, the numbers represent the type of each sphere, namely: Center Point (1), Wide Point (2), Star Point (3).

- *Brick Placement on top of window or door lintel*: Once the bricks confining the window or door opening on both sides, reach the upper level (lintel) of such opening, a new set of insertion points are enabled on the opening. These new points allow bricks to be built on top of the opening. As for their type, they fall under the category of "base points". Accordingly, these points follow the same rate of distribution and angle separation of base points initiated on the ground (see Figure 51).

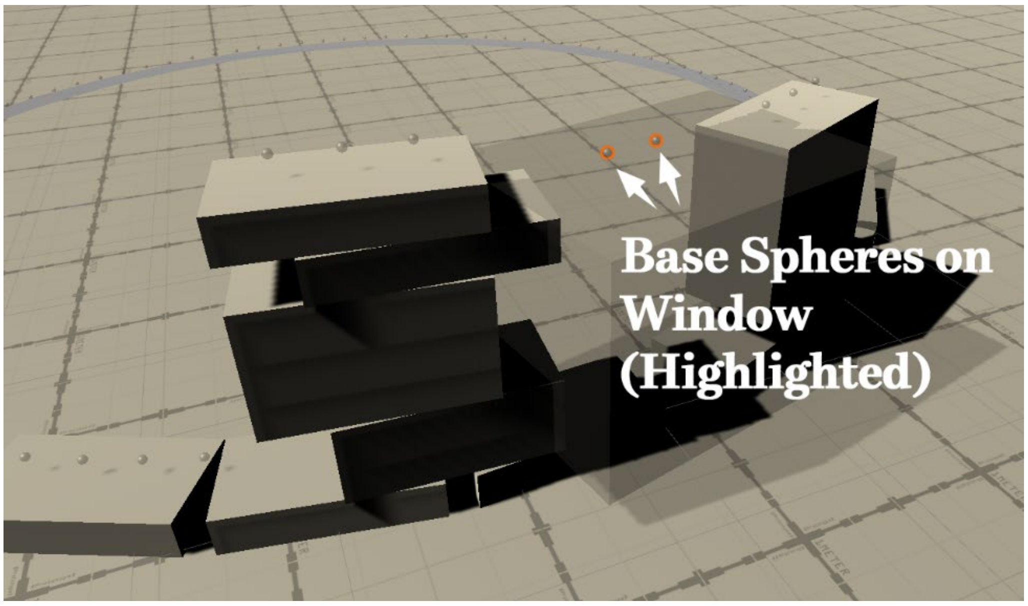


Figure 51: Base spheres inserted on top of Window

Independent of the type of insertion point, once a brick is placed, a “Virtual” brick collider captures all the existing insertion points being occupied by the physical volume of the new brick. It also captures and deletes “nearby insertion points that if, afterwards, a brick is to be inserted in one of them, it would coincide with the volume of the newly inserted brick. This would lead to the creation of a crash and perhaps a collapse in the brick structure if such an event occurred during gameplay. Therefore, the role of a “brick collider” is to prevent the occurrence of such an event before happening by performing this quick check every time a new brick is to be placed (Figure 52).

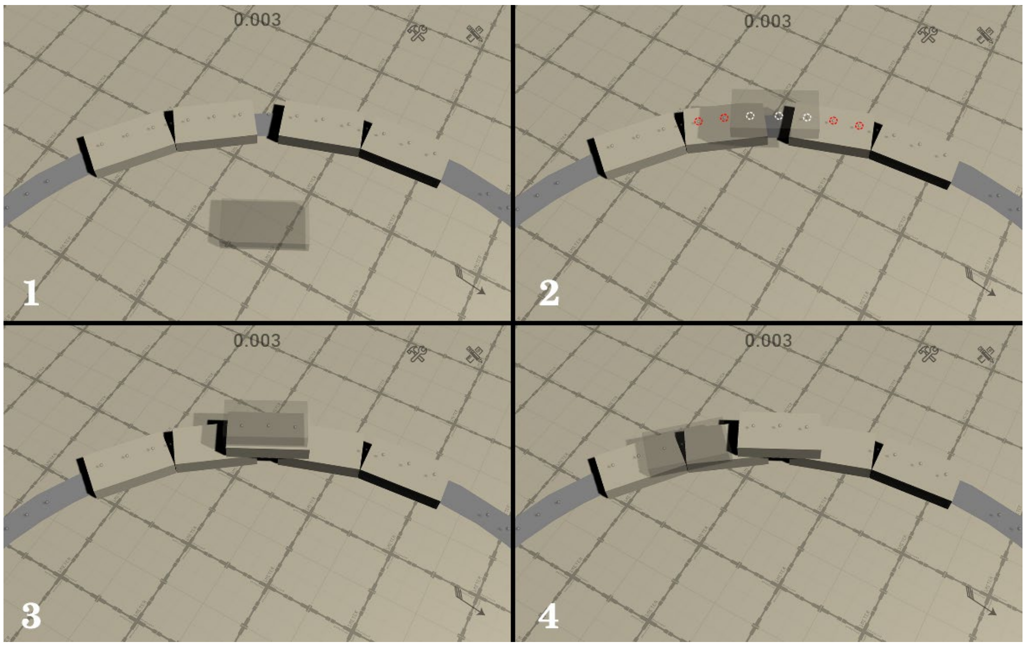


Figure 52: Steps of inserting a brick while focusing on the role of the invisible “Brick Collider”. It deletes not only white points but also red points that would lead to a clash of a new brick would be inserted in one of them.

This kind of assumption could be replicated in real world with the help of visual sensors. These sensors could give instant feedback to the simulator about the current “geometry” of the structure and then the “brick collider” could perform a “clash detection” check on the real current structure information that were transformed to the virtual world.

To sum up, there are 4 types of insertion points that may appear:

Table 4: Insertion Sphere Points Summary

Insertion Sphere	**Position**	**When it is generated**
Center Point	Exactly on top of the centroid of a newly placed brick (see Figure 50)	*On new Brick Insertions*

Wide Point	On top of a newly inserted brick on both sides of Centre Point (if possible) (see Figure 50)	*On new Brick Insertions*
Star Point	In between two Bricks, being one of them the newly inserted brick, on condition that there is enough "support" that could be provided on both sides for any brick to be later inserted in that point (see Figure 50)	*On new Brick Insertions on the condition that there exists a brick next to it, that is close enough to provide a simply supported beam effect on any future brick inserted in that point.*
Base Point	On the ground / On top of Window/ Door Openings (see Figure 49	*At the beginning of the Game / When 2 sides of bricks surrounding an Opening reach the same level as the top of the opening.*

B. *Window Placement:*

Same as the conceptual sequence for adding a brick, the window can be added in the same manner. However, the window “Box”, differs from the placed brick since it is not considered as “Physical Element”, rather than a “Void” space that is open to the outside. Inside this “Void” it is not possible to have any brick insertion points.

A ‘Window Collider” is equipped with some sequential rules to define the time at which the bricks from both sides of the “Window” have reached the top height of the box. In this case the cursor can hover over the top of the box and additional insertion points become available on top. This procedure allows the closure of the Window Void by bricks from top. It approximates the presence of a “Window Lintel” in real world.

In this version of the game, windows are programmed to only be inserted on “Centre Points” or “Star Points”. Therefore, it is not possible to place windows on “Base Points”. This prevents the occurrence of situations like the insertion of a window on the ground or the placement of a window opening exactly on top of another one. However, prevention of inserting windows on “Wide Points” is a design choice. Therefore, it is an option that can be altered by the designer based on the needs and has only an aesthetic effect.

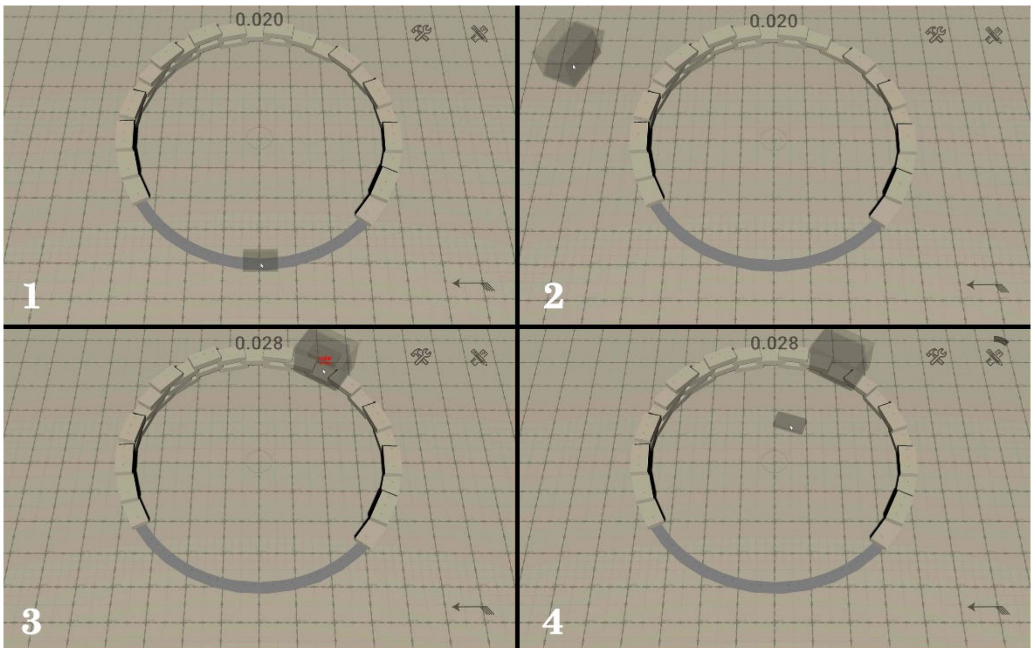


Figure 53: Window Insertion Steps. (1) All insertion Points are Visible. (2) Converting to window mode and disappearance of inadequate points. (3) Placement. (4) Brick mode is automatically recalled

C. *Door Placement*

Placing a door has almost similar properties as that of windows, in terms of possibility of adding new "base points" on top of the Door Opening once bricks from both sides reach that level. However, door insertion differs in the logical functional property as an "opening" for people to enter the pavilion. Accordingly, door insertion is allowed to be inserted *only* on the ground. Thus, it can be deployed on "Base Points" that lie on the ground.

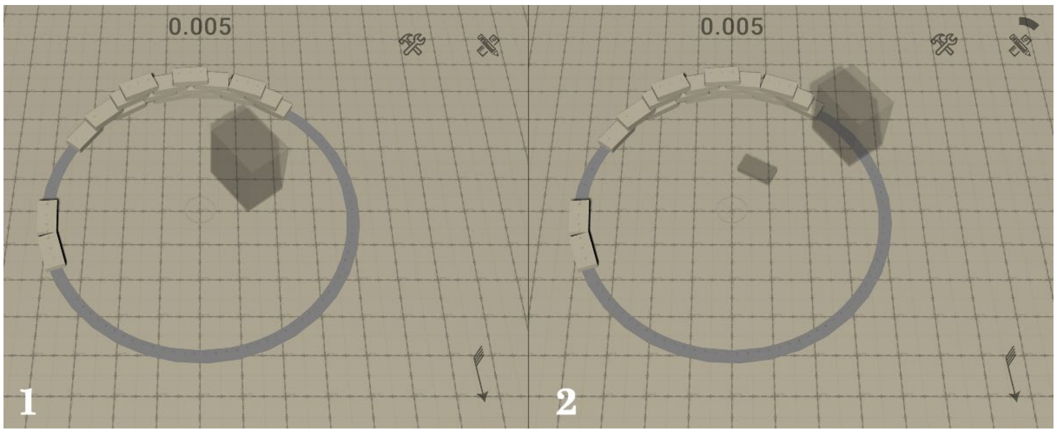


Figure 54: Door Placement Process

2.7.3. Score and Progress Tracking

Since the first aim of UV is to let the designers test the conformity of the game against the real process, once the functional properties are approved, the second objective comes into action. This second objective focuses on 2 main iterative tasks. The first one is adjusting and fine-tuning the rewards the agent relies on for a smooth learning path. The second is adjusting the methodology of progress evaluation.

This later process could help in the appropriate division of the complex goal of designing and constructing the pavilion into sub-tasks, making it easier for the agents to learn. In case of using multiple robots, each having a different capability on-site, this task division could also help in distributing these tasks among them, hence, working with a multi-agent environment.

Accordingly, UV provides 4 "visual" metrics to track the Score and Progress instantly during game play. These metrics are: Total Score, Instant Achievement, Design Progress and Construction Progress. They are visually tracked through a score counter, an achievement pop-up, progress bar design and another for construction respectively.

- *Cumulative Score Counter*

It is a counter located on the top center part of the game screen that accumulates the rewards received during a certain episode of the game. The score can also be negative if the total sum of penalties are more than the rewards achieved during an episode.

Example of rewards experimented:

Table 5: Examples of Rewards/Penalties experimented in UV

Unit	Reward / Penalty	Explanation
Brick	Consecutive Bricks	*Inserting bricks in series on the same level*
	Far Bricks	*Inserting bricks far away from each other*
	High Bricks	*Inserting bricks at a higher level from the current lowest incomplete level*
Window	Sill Height	*Placement of window on an appropriate height from the ground*
	Cross Ventilation Stack Ventilation	*Benefit from wind direction in a way to maximize this effect*
Door	Minimize Wind Capture	*Placing the door away from prevailing wind direction*

In the following images from the game, there are some examples of rewards achieved or penalties received among the ones included in Table 5. The rewards are eventually summed up on the Score counter on the top of the screen.

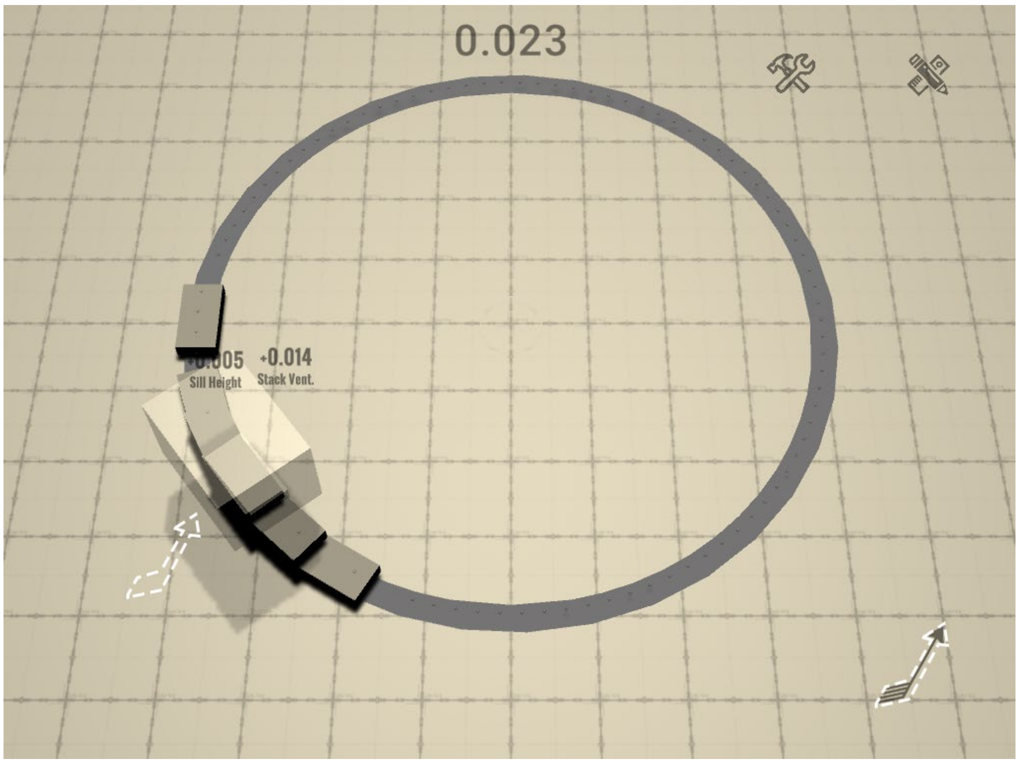


Figure 55: Window Placement achieving 2 positive rewards for proper sill height and stack ventilation achievement.

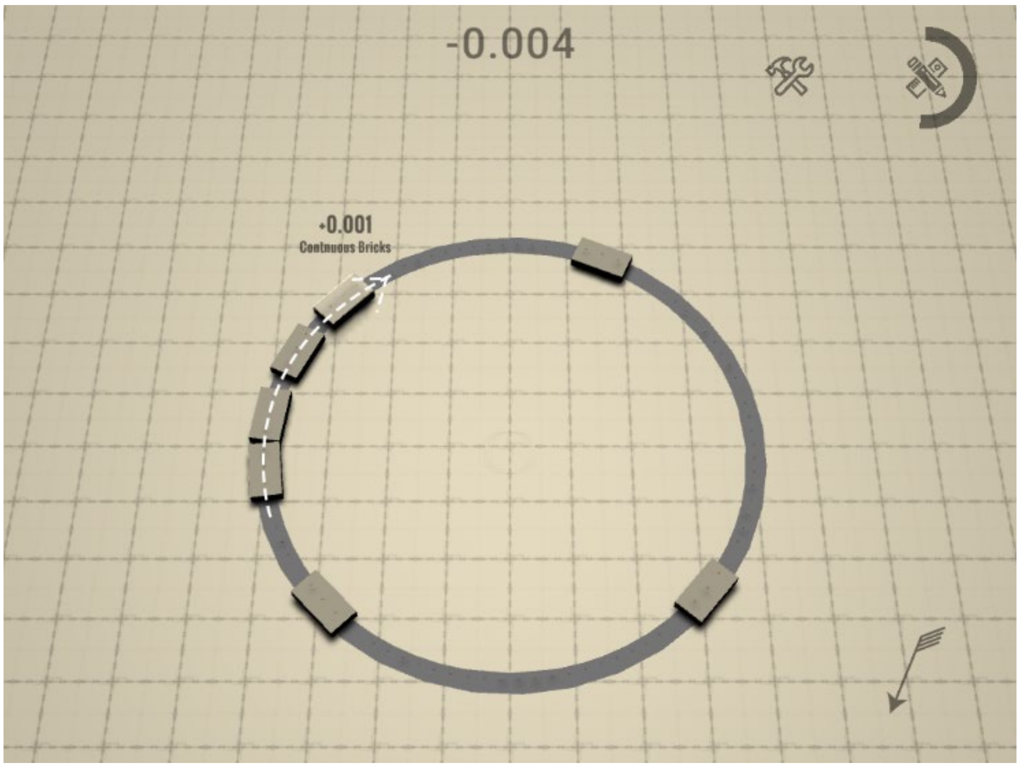


Figure 56: Reward for consecutive bricks placement

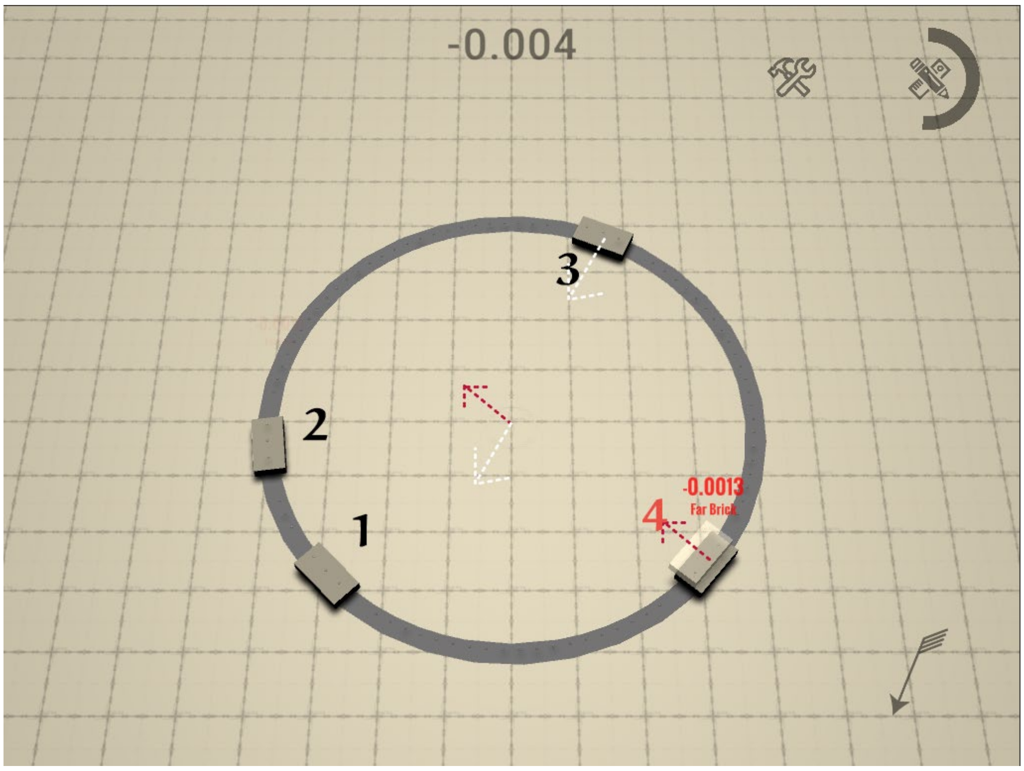


Figure 57: Penalty for placement of far brick

- *Achievements Pop-Up*

Achievements Pop-ups gives instant feedback to the user if a certain action, or set of actions, led to a positive reward or a negative penalty. Each pop-up is composed of 2 parts: a text representing the kind of reward/penalty given and a numerical value for such a reward/penalty. Positive rewards are shown in black color while negative rewards are shown in a red color, as shown in Figure 58.

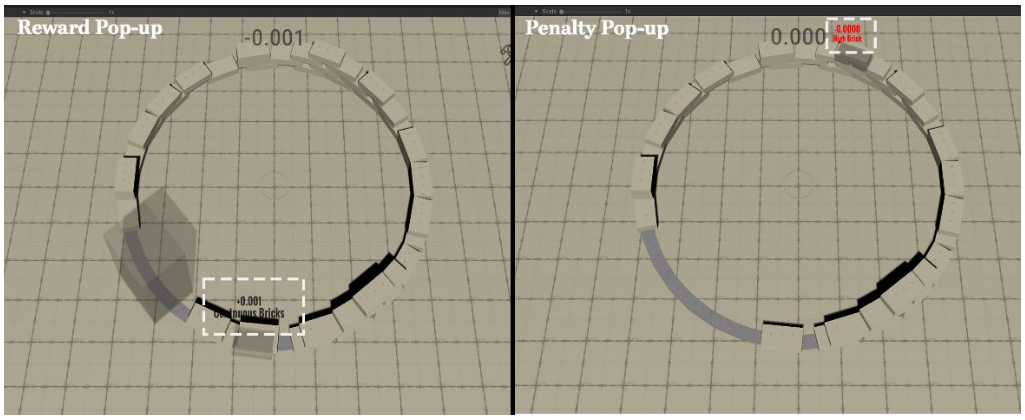


Figure 58: 2 types of Pop-up text are shown. (left) is a reward pop-up while (right) is a penalty pop-up.

- *Design and Construction Progress bars*

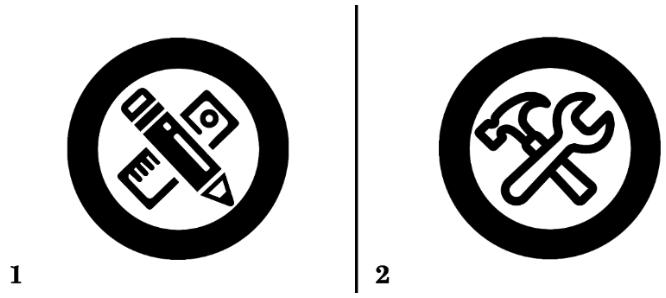


Figure 59: (1) Design Progress Bar (2) Construction Progress Bar

Design and Construction Progress Bars can be considered as a visual tool to highlight benchmarks achieved during gameplay. These benchmarks could be related to carrying out a certain number of actions that eventually leads to the completion of a design or a construction task (see Table 6, Figure 60, and Figure 61).

Table 6: Examples of tasks that contribute to making progress in their relevant Design or Construction progress bars

Bar Type	Achievements contributing to the Progress
Design	Stack Ventilation Cross Ventilation Windows having adequate Sill height Adequate Placement of Door Opening Consistent Brick Perforations
Construction	Each completion of a full level of bricks

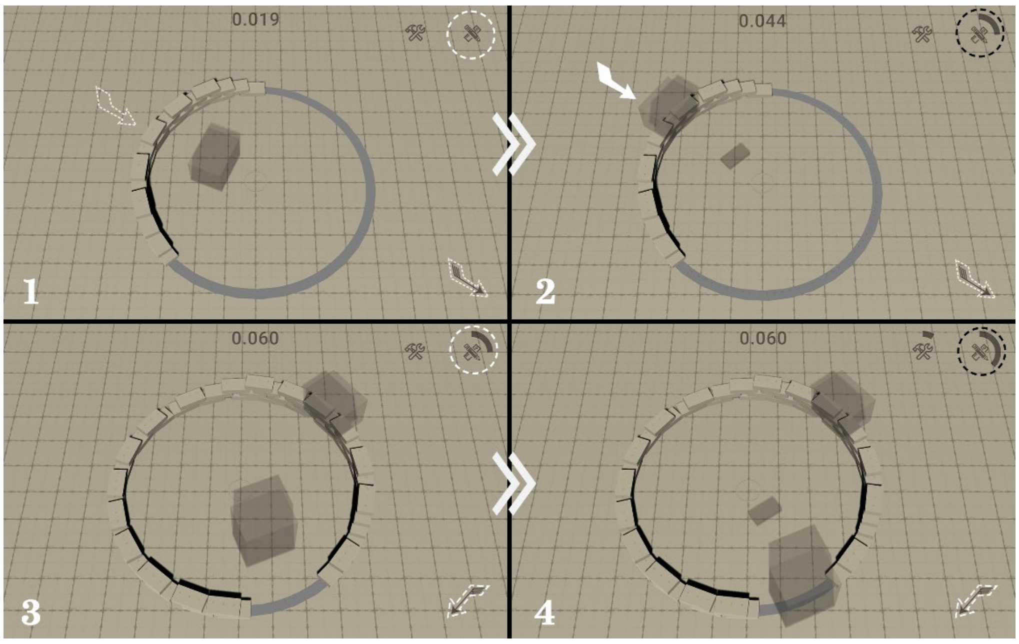


Figure 60: (1 -> 2) & (3 -> 4) Snapshot of before and after completing a Design Task and Monitoring the increase of progress on the Design Progress Bar

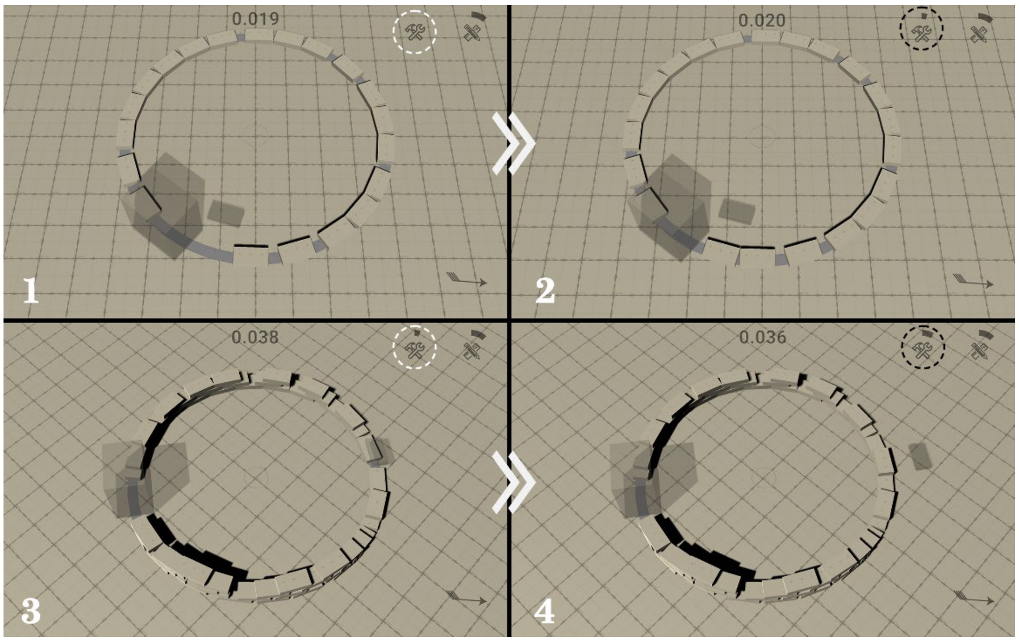


Figure 61: (1 -> 2) & (3 -> 4) Snapshot of before and after completing a Construction Task and Monitoring the increase of progress on the Construction Progress Bar

2.7.4. Completion and Game Over Scenarios

Succeeding in reaching the required height of the building while achieving satisfactory results in design and construction tasks during game play are the main indicators of "Success". This achievement signals the entire completion of the Brick Pavilion. To reach this goal, converted to numerical values, the player must have a positive total cumulative score and with no more "Insertion Points" available in the game while having reached already the required final height of the "Oculus".

On the other hand, failure could occur for many reasons, some of which were already mentioned in the previous sub-sections. The following table summarizes the different scenarios, that are present in the current version of the game, that leads to failure in finishing the game successfully. Each scenario is classified based on whether it is related to an inadequate design aspect or improper construction execution sequence.

Table 7: Summary of Probable Game Over Scenarios

Type of Failure	Cause of Failure	Possible occurrence Scenario
Design	No Door Placement	Consuming all sphere-insertion points available on the ground level in placing bricks, which makes it no more possible to place a door opening on the ground in the game

		Another Scenario could be the insertion of windows in a way that doesn't give a space anymore for inserting a door. *
	Inadequate Door Placement	Placing the entrance door to be facing prevailing wind direction.
	No Window Placement	Reaching the starting height of the dome (point at which the radius of placement circle starts to shrink), with no windows inserted yet.
	Inadequate Sill Height for Windows	Placement of windows either very close to the ground or higher than the average height at which a normal window would be placed
	Poor Cross Ventilation	Rearrangement of Windows in a poor way in terms of benefiting from Cross Ventilation Effect.
	Poor Stack Ventilation	None of the Windows inserted in the game are facing the prevailing wind direction.
	Uneven large gaps in the design	Having uneven, discrete, large gaps in the overall shape of the pavilion.
Construction	***Poor Brick Placement Technique*****	Getting a consecutive penalty due to the insertion of bricks in a sequence that has fewer consecutive bricks and more "far" or "distant" bricks. Far and distant bricks were explained in Section 3.7.3.
	Brick Collapse	Falling of bricks on the ground or even the tilting of a brick more than a certain threshold after its placement. This threshold is part of the parameters defined by the designer before the start of a game episode.

Notes

* *During the placement of a window or a door opening, the game is programmed to leave an obligatory space between this opening and any future openings. This obligatory distance is one of the parameters that the designer shall decide on its value before the game starts. The reason for leaving this space is to allow for adequate brick separation between any 2 openings and provides a structural support as well for the*

placement of the lintel for different openings, and for the support of the upper dome.

** *Placement of a sequence of bricks in such a way that consumes more "energy" by the executing robot or may increase the obstacles in the robot paths caused by the placed bricks themselves. First, in this game it was assumed that there is only one "hypothetical" mobile robot equipped with a crane system that allows the robot base to be elevated upwards when needed (see* Figure 62*).*

Therefore, the optimum goal is to insert the bricks in a way that reduces the practical distance between the supply and demand points and to always try to provide more "freed-space" for the robot arm to move. Since this tool path planning is out of the scope of this research and many efficient solutions to solve this issue were already provided and tested in both research and practice, it was still necessary to include its "Core concept" in the execution process. So, to translate this issue into tangible simple values, 3 parameters were assessed: Consecutive placement of Bricks, High Placement and Far Placement.

Optimum placement strategies were regarded as the one in which bricks are inserted consecutively one after the other and avoiding as much as possible placement of bricks at much higher levels than the lowest incomplete current one. However, this aspect was overlooked in the dome construction phase. During that phase, much priority was given to inserting the bricks in whichever way that maintains stability of the structure during and after placement.

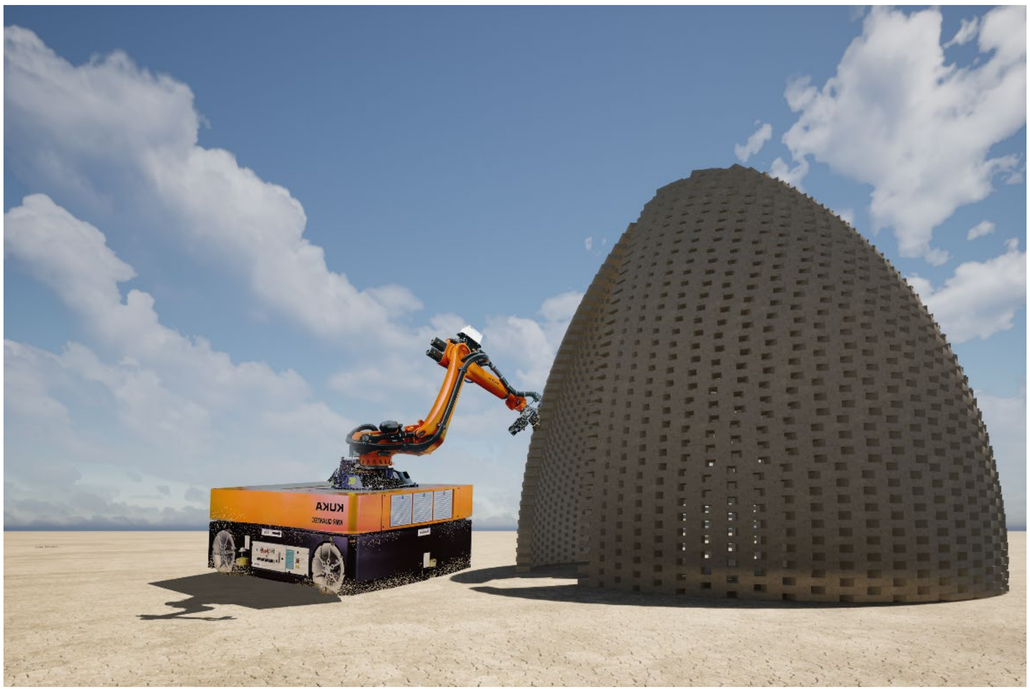


Figure 62: Imaginary render of KMR QUANTEC [82] working on site

The following 2 figures provides actual screenshots from the game where “Game Over” menu is present, as the gameplay was rendered unsuccessful.

The reason of failure in Figure 63 is due to the collapsing of bricks due to the consecutive placement of bricks vertically over each other in one column. This series of actions has led to instability in the structure and eventual collapse of top bricks due to gravitational forces.

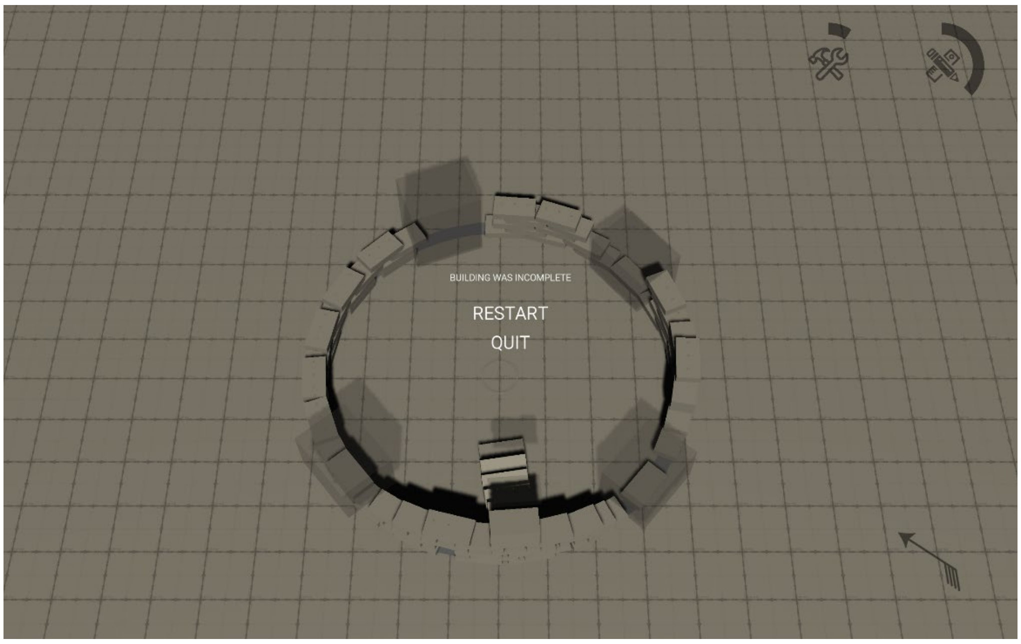


Figure 63: Game was over due to collapsing of bricks

In Figure 64, failure in completing the game was due to the consumption of all ground points in placing bricks, that there is no more room for adding a door.

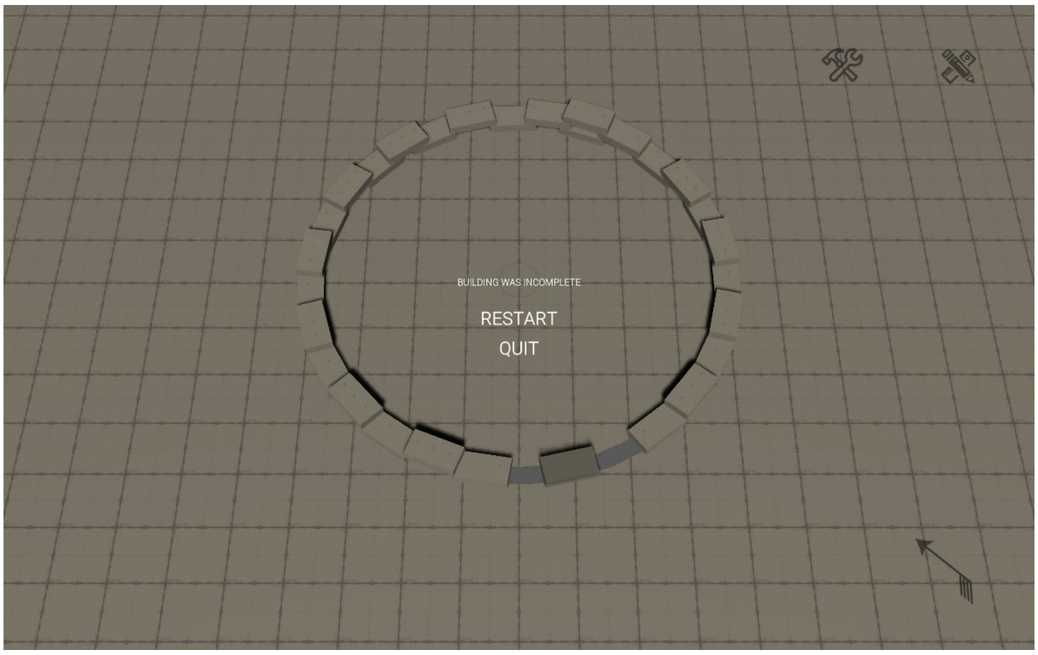


Figure 64: Game was over due to absence of a placed door on the ground

2.7.5. Design and Functional Parameters Influencing the Game

This sub-section provides an overview of some of the main parameters included in the application. These parameters are classified according to their type of contribution into: Angular-Driven Parameters, Dimension-related, Functionality-driven, Design-driven, and Physics-related respectively.

- *Angular-Driven Parameters*

Since the pavilion is built on a “circular” concept at its core, all geometric values need to be “translated” to values easily interpreted in a “circular environment”. For instance, space occupied by bricks and void left between them can be easily expressed using angular values referencing the center of the building; the point where everything revolves around. Such a translation helps reducing the complexity of comparing different measurements and provides a common ground for all terms.

The following figure (Figure 65) shows an example of different angles calculated for voids and spaces on one level of bricks.

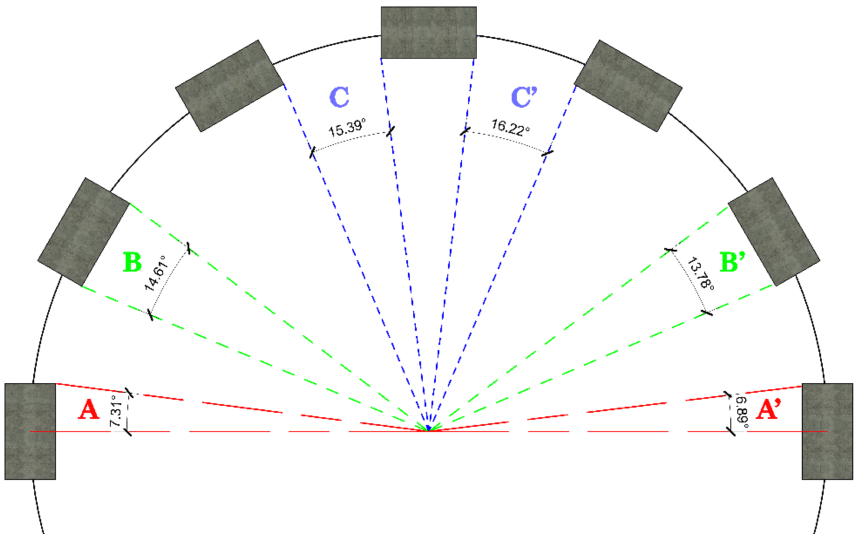


Figure 65: Different variables presenting identifying bricks and void positions from angular perspective

These angles and their corresponding variable names are presented in the following table. These variables are thoroughly used in game-calculations.

Table 8: Angular-driven Parameters Definition

Letter	Variable Name	Value*
A	Half Brick Angle**	7.31
A'	Half Brick Angle on Center Line***	6.89
B	(Full) Brick Angle	14.61
B'	(Full) Brick Angle	13.78

C	A Space between 2 bricks	-
C'	A Space between 2 bricks on the Center Line	-
*	This value is based on the default circle radius. Any reduction in the radius of the circle at later stages of dome creation, results in an increase in all these values.	
**	Any letter ending without (') means that this value is calculated based on the end tip of the unit in consideration	
***	Any letter ending with (') means that this value is calculated on the circle passing through the centroid of the examined unit.	

- *Dimension-related Parameters:*

They represent the dimensions of the main units involved in the design, namely: Brick, Window and Door Openings. These parameters give flexibility to the Designer in visualizing how different options would look like.

The following table represents the actual dimensions used in both UV and RV.

Table 9: Dimensions of building units

Unit	Length	Width	Height
Brick	1.2	0.64	0.24
Door	2	1.5	2.4
Window	2	1.5	1.44

- *Functionality-driven Parameters*

These group of parameters are more driven towards adjusting game-mechanics and making sure that the logic behinds the game itself runs smoothly. They are related to different logical aspects and may also contribute to changing how the pavilion would look like. However, their impact on functionality of the game could be seen as more dominant than their other contributions.

These parameters are sub-classified based on their specific role they contribute to. Some examples of these parameters are presented. Each sub-group of parameters is shown in a separate table, where each parameter is defined and explained.

Table 10: Group 1: Brick Related Parameters

Brick-related Parameters

Parameter	Definition	Function	Default Value	Range

Offset Distance Percentage of Brick Factor	A percentage multiplied by half extents of a certain brick collider	Factor that contributes to the detection of sphere points needed to be deleted once a new brick is inserted. This value is adjusted based on other design parameters in order to capture all points needed to be deleted. This would avoid a clash potential if a point was missed, and another brick was inserted in that point afterwards (see Figure 52).	1.04	[1.03-1.3]
Search Factor	It is a factor multiplied by half brick angle at the horizontal level under study.	Factor that contributes to detecting nearby bricks on both sides and on the same level of the brick inserted. The overall process eventually helps in: 1. Assignment of star points 2. Detection of gaps in bricks for Design Evaluation.	6	[1-10]

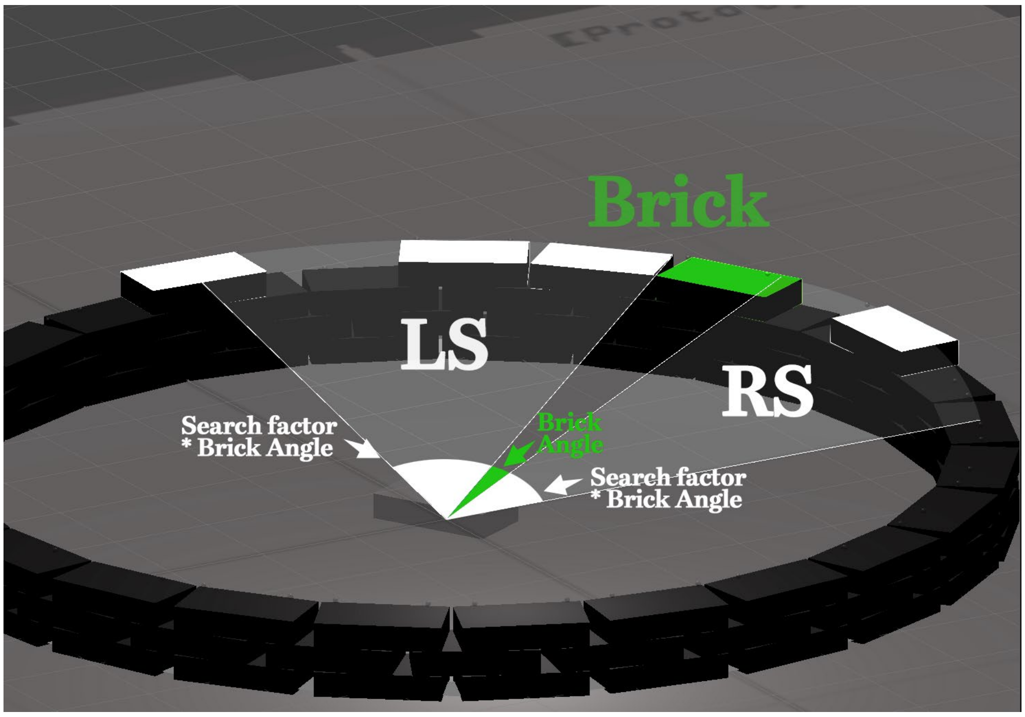


Figure 66: Search Factor Demonstration

Table 11: Early Collapse Detection Parameters

Early Collapse Detection

Parameter	Definition	Function	Default Value	Range
Tilting Limit Factor of Safety	Minimum Value that the dot product between the normalized y-vector of a specific brick and the normalized global y-vector (0,1,0) should not exceed. ***(Note: Y is the Upper direction in unit, instead of the commonly used Z)***	This Crucial factor determines the tilting of any brick by comparing its upwards unit vector to the normalized Y axis vector of the game using dot product. The more the brick tilts, the less the value of dot product shall be. Therefore, this *Tilting Limit* is an early alarm in which once exceeded, there is no need to wait till a fallen brick touches the ground to end the game. This factor is also crucial in putting more constraints on the instability of bricks. This would encourage the player to follow a more conservative approach when placing bricks, to avoid any kind of instabilities even if they might not lead to an event of brick collapse.	0.9936	[0.97-0.995]

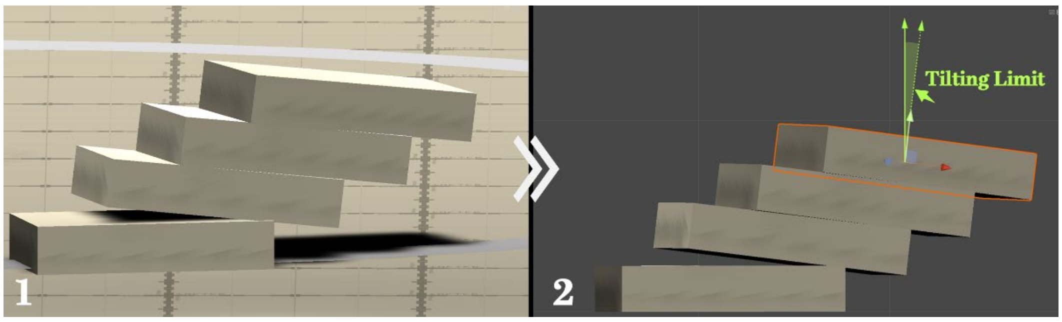


Figure 67: Tilting Value Limit Demonstration

Table 12: Opening Related Parameters

Openings related Parameters

Parameter	Definition	Function	Default Value	Range
xFactor Half Brick	Factor greater than 1 that is multiplied by the length and width of the original size of an opening collider.	Factor that vertically expands the collider attached to an Opening after it is placed. This would help in any upcoming step in capturing the bricks on both sides of such an opening. This process is crucial to keep track of bricks that surround openings. For instance, capturing those bricks helps in determining when the 2 brick-sides of an Opening have been fully completed This would allow for the addition of basepoints on top of the Opening lentil for brick placement. Therefore, this factor helps in creating a geometric trigger for any brick insertion next to Openings.	1.1	[1.1-1.5]

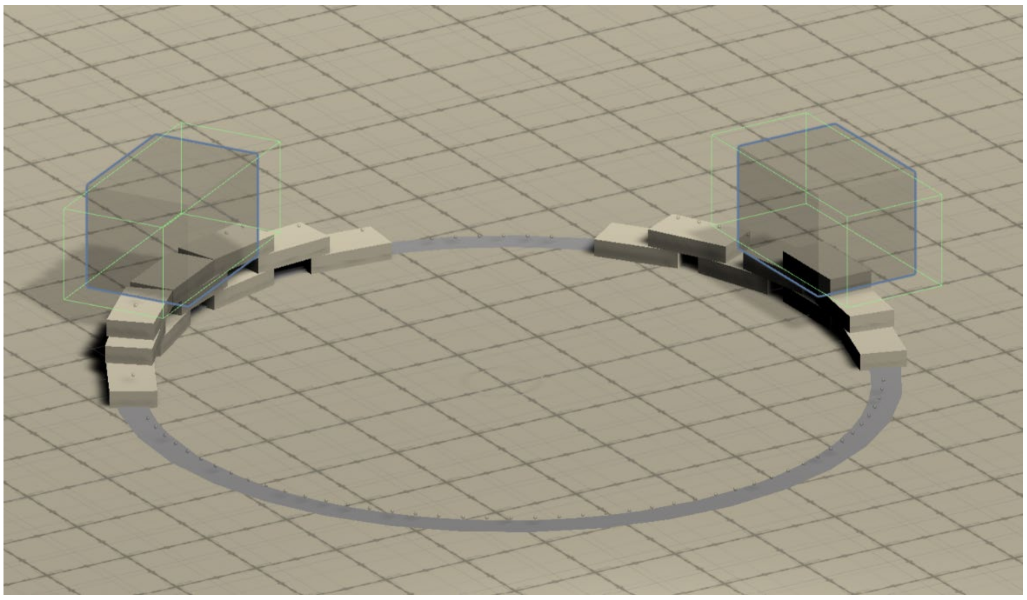


Figure 68: Opening Colliders (in green) expanding after Insertion

- *Physics Related Parameters:*

These parameters are mainly concerned with the periodic check of the stability of the placed bricks. Since Nvidia PhysX engine is enabled, all bricks will always have a “live instance” in the scene. Increasing the number of bricks placed, is coupled with an increase in the computational power required to keep track of the behavior of each brick.

To overcome this issue, a process named “Stability Check” is carried out every “Late Update”. This test includes all bricks inserted and then several incremental tests are enforced on each brick. If a brick is proven stable for a “sufficient” amount of time. It will no longer be “alive” in the scene and will considered static in its place.

Proper fine tuning of these parameters guarantees an evident boost in the performance of the game with very limited effect on its real dimensionality. In the following table, an example of these parameters is introduced.

Table 13: Physics-related Parameters

Parameter	Definition	Function	Default Value	Range
Stability Counter Threshold	Minimum Integer threshold against which each brick stability counter is compared to. If the number exceeds this threshold, the brick is considered stable.	Factor contributes to assessing the stability of a brick for a continuous period after it was already placed. It eventually determines the number of counts in which the up unitary vector of a brick remains constant or within a very narrow range of movement.	10	[2-20]
Factor Brick Length Physx Vicinity Below	A Factor multplied by brick length to be compared against the horizontal projected distance between 2 bricks above each other	Factor contributing to one of the stability assessment checks carried out during the game. It would compare the shifting distance of the centroid of the tested brick to the centroid of the brick below. This determines if they can be considered almost below each other or way shifted and hence high	0.1	[0.01-0.2]

eccentricity could exist. Other checks are executed afterwards to explore other factors that could improve the tested brick stability if they existed.

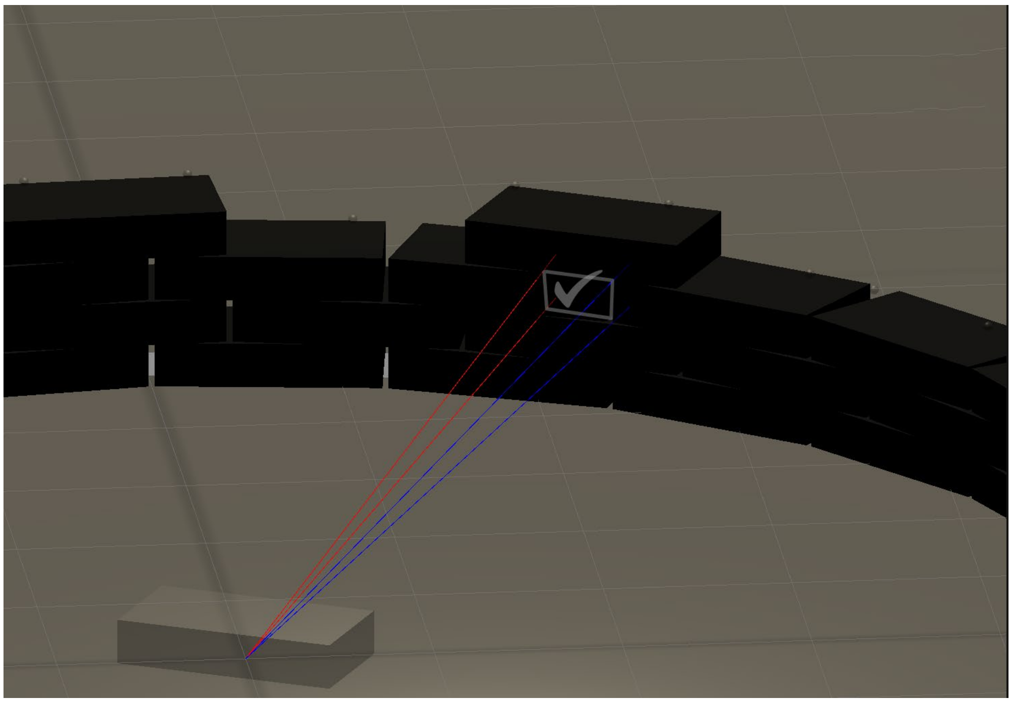


Figure 69: Demonstration of the factor "Factor Brick Length Physx Vicinity Below"

- *Design-driven Parameters:*

This cluster of parameters directly affects the final design aspects of the pavilion. The designer should decide, based on experimenting in the UV environment, definite values to help train the agents at a later stage.

Table 14: Building Height Parameters and Ground Base Points Parameters

Building Height

Parameter	Definition	Function	Default Value	Range
TotalHeight	Absolute total height of the pavilion required	-	8.4	[7.2-9.12]

Ground Base Points Parameters:

Parameter	Definition	Function	Default Value	Range
Multiplyer Factor	A factor that is multiplied by the default number of base points that gets initialized at the beginning of the game	Factor that controls the number of points available at the the ground base level for bricks/door insertion. Increasing this factor, increases the amount of placement choices available at the ground level.	1	[0.25-2]

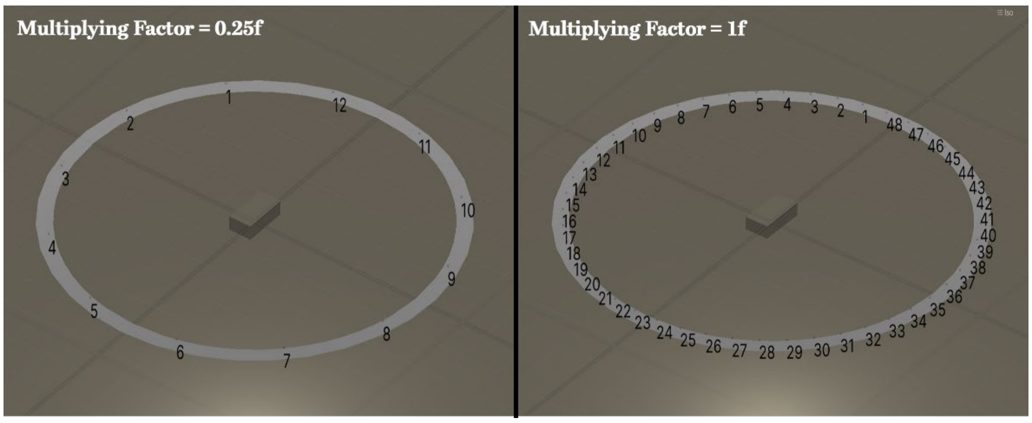


Figure 70: Comparison of the effect of different values of "Multiplying Factor" on the count of ground base points

Table 15: Wide Angles Positioning Parameters

Wide Angles Positioning

Parameter	Definition	Function	Default Value	Range
Percent of half Brick Wide Point Insertion (not used in RV)	A percentage multiplied by half of brick angle	Factor used to determine the position of wide angles on any new inserted brick. Increasing this value widens the angle, calculated from the center of the circle, between the center Point and any of the wide points. Maximum limit is assigned to ensure that the centroid of any new brick inserted at that point would be a sufficient distance away from the tip of the current brick. Since there is a	0.5	[0.4-0.8]

		chance that the new brick will be only supported on the current brick only, this limit prevents the new brick from falling if its centroid is almost on the edge of the lower (current) brick.	

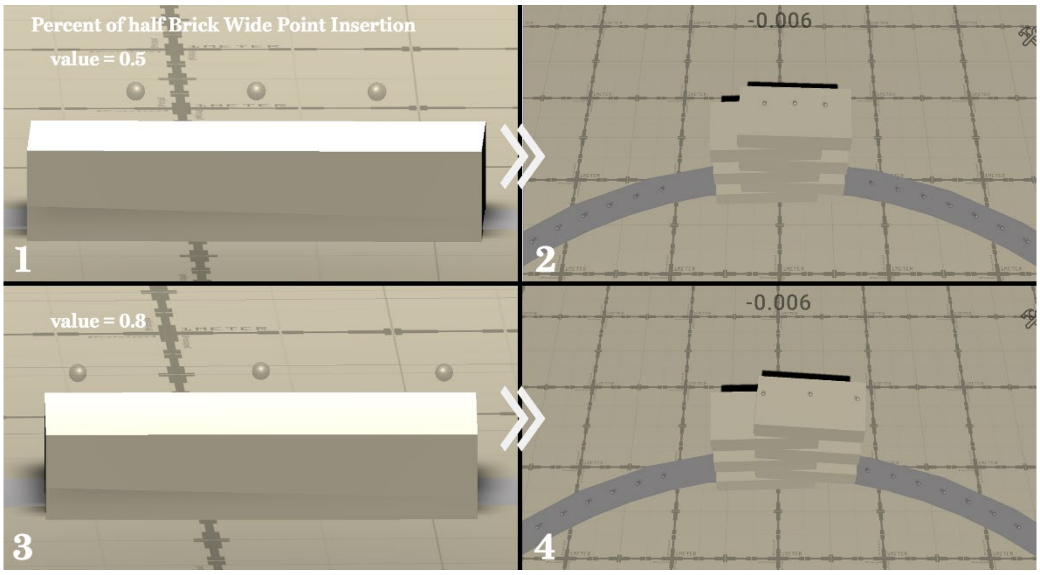


Figure 71: Effect of changing the percent of half brick factor for wide points insertion on the position of wide points (1 & 3). (3 & 4) The effect of this factor can be significantly witnessed on the "Geometry" of a stack of bricks.

Table 16: Sill Height-related Parameters

Sill Height Related Parameters

Parameter	Definition	Function	Range
Eligible Sill Height (Upper and Lower Bounds)	2 values in which the sill height of a window is compared to.	If the sill height of window is within these bounds, the player is then eligible for a positive reward, or at least it is considered that this design parameter has been addressed properly. However, the more the actual sill height value would be further away from the bounds (towards the center value), the more rewarding and acceptable this process would be.	[0.48-1.44]

Table 17: Dome Design-Related Parameters

Dome Design-Related Parameters

Parameter	Definition	Function	Range
Inward step (Upper and Lower Bound)	2 values lower than half of Brick Width, that gets multiplied by this width. This results in upper and lower bounds function in the brick width.	These upper and lower bounds are used in a "slerp" function to determine the distance needed to be moved towards the center of the building once a new level in the closure dome has been reached (see Equation below).	[0.1-0.4]

The following Equation demonstrates how the Inward Step Factor is calculated depending on the height of the current brick:

$$ISF = St_{lb} \times \left(\frac{1}{St_{lb}} St_{ub}\right)^{t_{rel}}$$

Where:
ISF: Inward Step Factor
St_{lb}*: Inward Step Value Lower bound (designer parameter)*
St_{ub}*: Inward Step Value Upper bound (designer parameter)*
t_{rel}*: Ratio between relative height from start of dome to the maximum dome height*

$$t_{rel} = \frac{h_{abs,current} - h_{startDome}}{h_{abs,max} - h_{startDome}}$$

Where:
$h_{abs,current}$*: Absolute Current Brick Height*
$h_{startDome}$*: Absolute starting height of the dome*
$h_{abs,max}$*: Absolute maximum height of the pavilion*

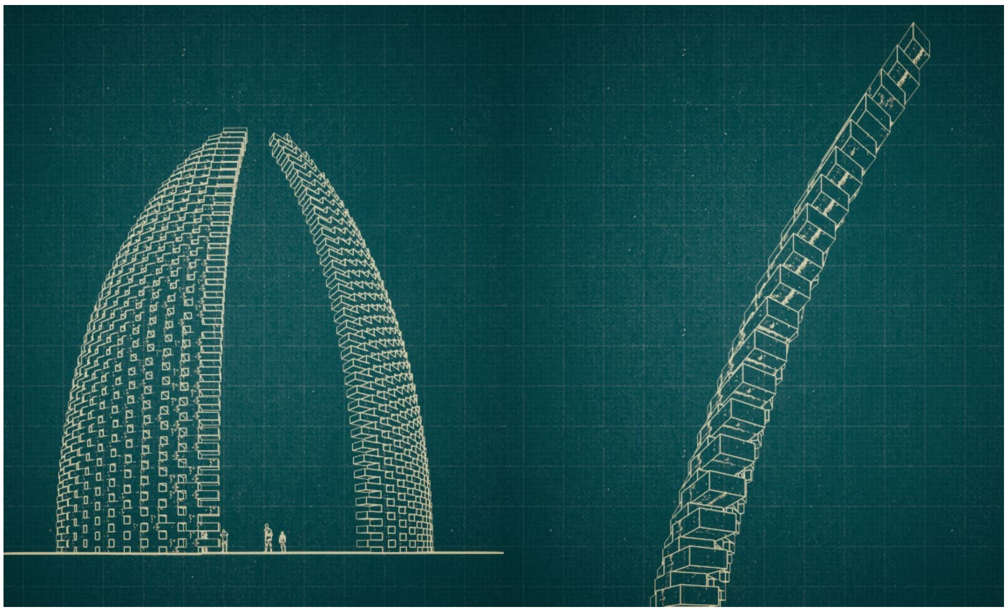


Figure 72: Evidence of the effect of the exponential equation, used to design the dome, on the incremental evolution of the section curve of the dome.

2.8. Robot Version (RV) and ML-Agents

In this section, the Robot Version (RV) of the game is introduced. As discussed earlier in the framework, this version of the game covers the “learning” part of the agents and how such an environment could be created. The section starts with a brief explanation of the MDP (Markov Decision Process) that the agents need to be deployed within and train on solving. This is demonstrated along with the translation of MDP into C# code, using the methods and classes provided by ML-Agents. Then, a review is provided on how the problem was decomposed into simpler problems “phases” to overcome the game complexity

This is followed by a detailed explanation of how each different process included in the MDP is being implemented using ML-Agents in the unity environment. These processes start by stating the *Actions* taken in the environment and how “*Masking of Actions*” was utilized. Afterwards, *Observations* captured in each phase are demonstrated, along with observables included in the changing-sized *Buffer Sensor*. The Reward System for the agents is then provided for each phase.

Once the MDP key components are covered, a sub-section including all the hyper-parameters used for training is provided. The hyperparameters for each learning phase are presented separately. Finally, the section ends with showcasing the Inference Mode of the game. This mode represents a compilation of all the “separate” experiences gained by the agents in each phase combined to form one complete structure.

2.8.1. Choice of ML-Agents for Implementing the Game MDP

In a typical Reinforcement Learning (RL) problem, there is a learner and a decision maker called ***agent*** and the surrounding with which it interacts is called ***environment***. The environment, in return, provides ***rewards*** and ***a new state based on the actions of the agent*** [83].

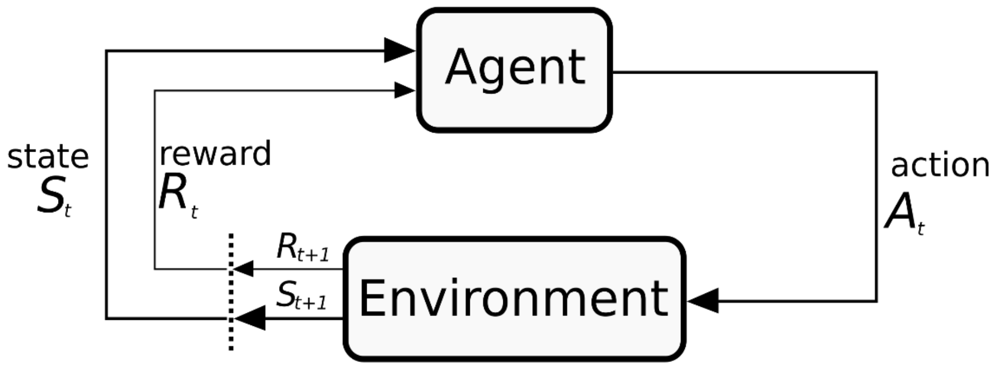


Figure 73: Markov Decision Process (MDP) [84]

So, in reinforcement learning there is no direct orders given to the agent on how to solve the problem, but rather, rewards are presented to the agent based on the actions or series of actions taken throughout the learning process. These rewards, whether positive or negative, help guide the agents on figuring out what is the optimum “policy” that could be adopted in order to maximize the cumulative rewards earned during gameplay.

To help achieve this optimum policy, we used the Proximal Policy Optimization Algorithm (PPO) . PPO is an on-policy reinforcement learning algorithm that uses a neural network to approximate the ideal function that maps an agent's observations to the best action an agent can take in a given state. It is motivated by the question: how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance collapse?

There are several reasons behind the adoption of PPO, for instance:

- ML-Agents provides an implementation of PPO out-of-the-box. It is considered the default algorithm in this toolkit. The ML-Agents PPO algorithm is implemented in Pytorch and runs in a separate Python process (communicating with the running Unity application over a socket) [78] (see Figure 74).
- It is a method that has been shown to be more general purpose and stable than many other RL algorithms [78].
- PPO can be used for environments with either discrete or continuous action spaces [85]. This gave us the flexibility to experiment both action spaces, however in the presented version of the game a “discrete” action space is eventually adopted.

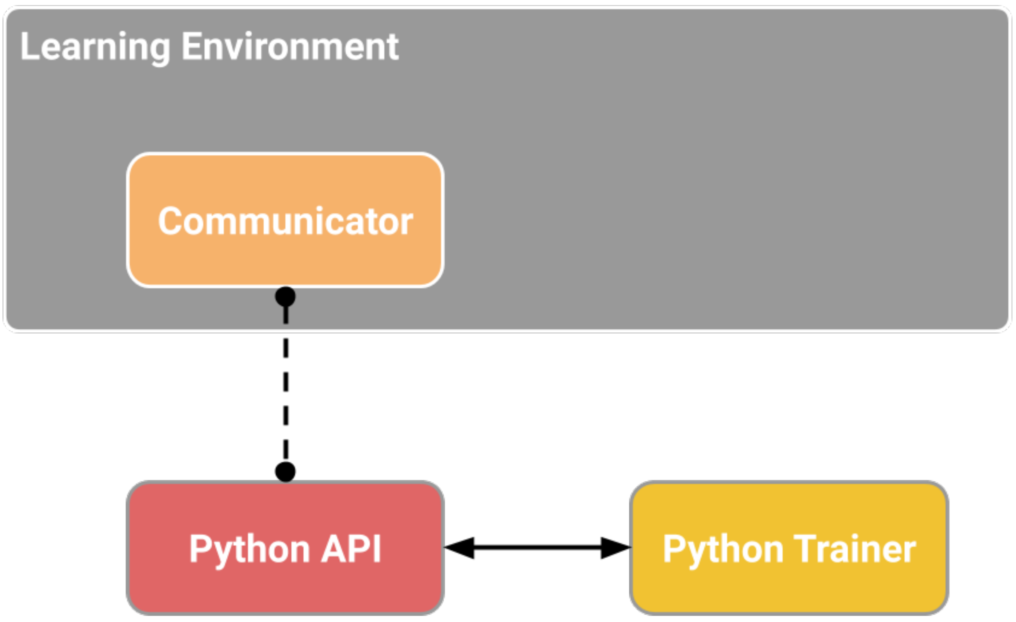


Figure 74: ML-Agents Toolkit high-level components

2.8.2. MDP expressed in ML-Agents terms

As discussed in Figure 73, to formulate an MDP, we need to define the required processes in terms of actual components interpreted from the game. These components are then translated into a functional MDP through the implementation of ML-Agents' methods and classes.

- *Environment*

Our Environment in RV is considered the "Building Area" in which all the "actions" take place. This "Building Area" is expressed in Unity as a game object that contains all the other "Children" game objects involved in the design and construction process. To maximize the training process, several instances of "Building Area" were instantiated before training (see Figure 75).

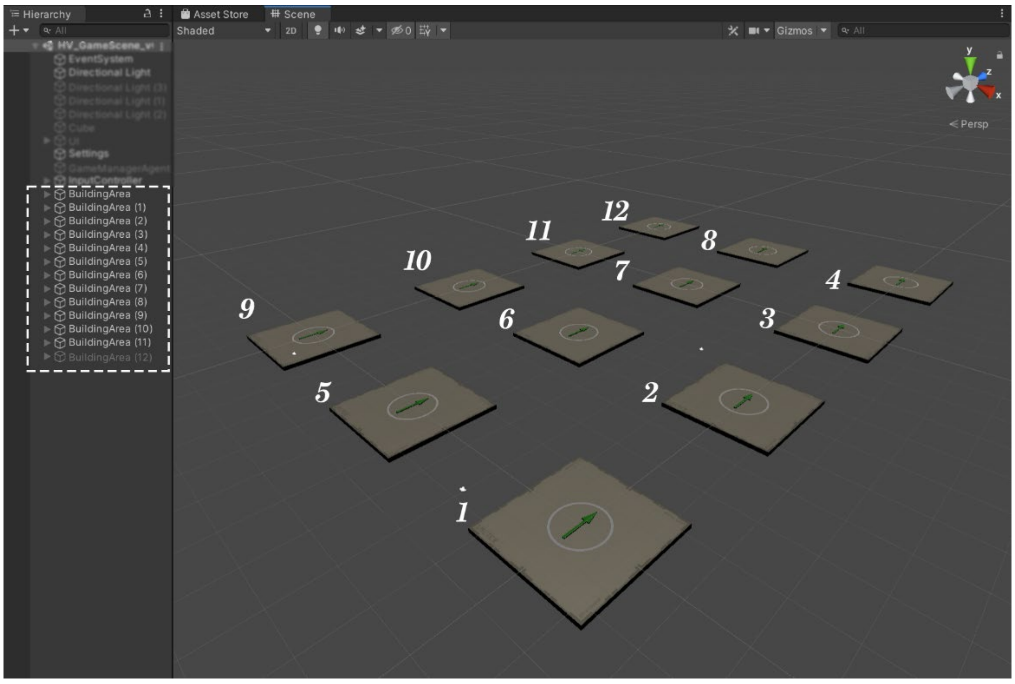


Figure 75: Multiple Instances of "Building Area in the current scene of RoBuilDeR

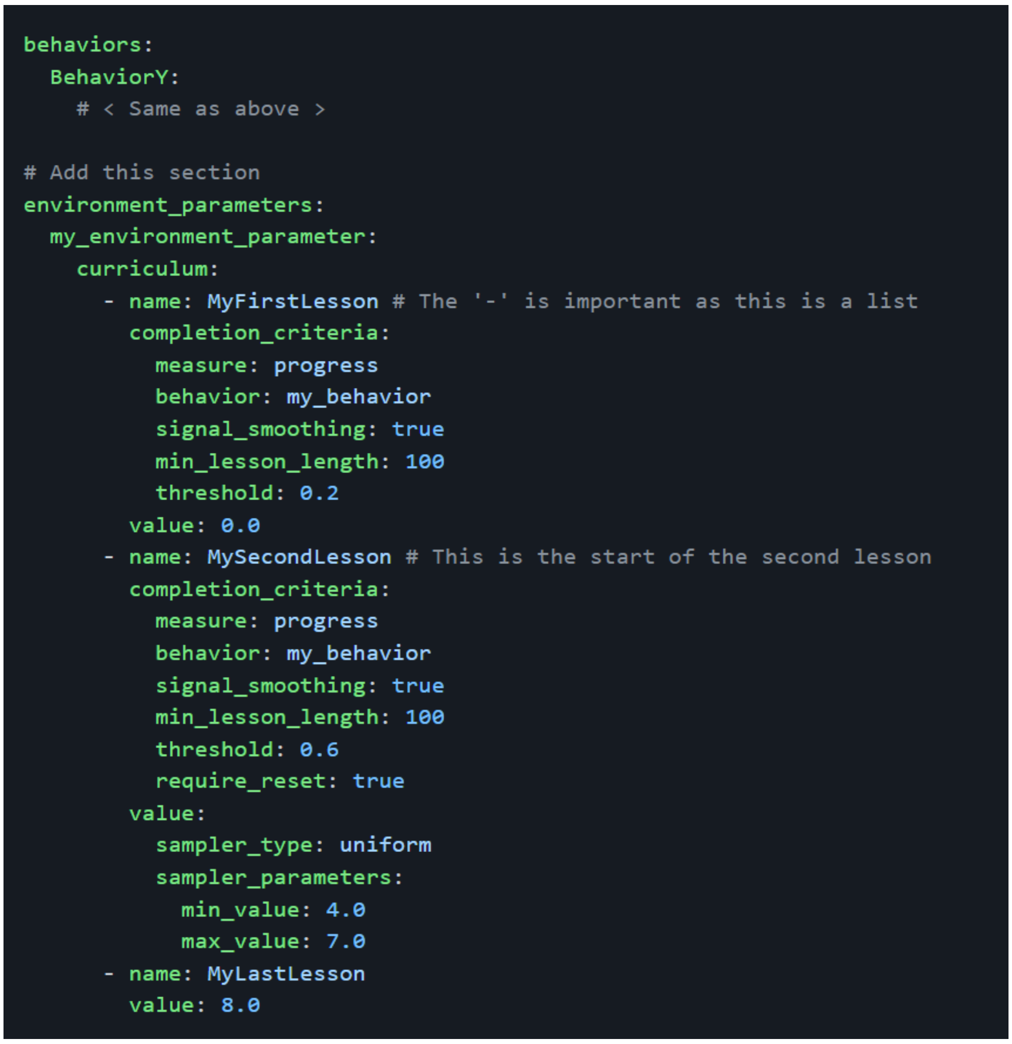
Training using Concurrent Unity Instances is one of the features provided by ML-Agents. Instantiating several environment instances allows the parallel training of agents [86]. This process enhances the learning process in two ways. First, it increases the learning speed, and second, it allows for better encounter of different game scenarios based on some varying parameters in each environment. This eventually yields an agent capable of adapting itself to more diverse situations than an agent being trained only in one environment.

However, regarding the acceleration of the learning process, there is a necessity to find the optimum maximum number of instances at which the maximum learning speed limit is reached. This number depends on the performance of the device on which the learning process is running on.

Therefore, if the number of instances increases more than a certain threshold, the speed is hindered, and the learning pace starts to slow down.

On the other hand, in ML-Agents, the word "Environment" has a larger scope. The "Learning Environment" actually refers to the whole Unity Scene that contains all game characters. This environment is linked to an external Python Trainer that deploys the machine learning algorithm needed. This is done through a communicator on the "Game Side" and the python low level API from the Trainer side (see Figure 74).

The communicator on the game side is managed by `Class Academy`. This is a singleton that manages agent training and decision making. On the other hand, since we chose PPO as the learning algorithm, the access to the "built-in" code settings, is accessible through the adjustment of the algorithm's "Hyper Parameters". This adjustment takes place in an external configuration file. This file specifies the hyperparameters used during training and it can be edited with a text editor to add a specific configuration for each Agent's "Brain". This file serves also as a container for the "Environment Parameters" that might need to be modified during training (see Figure 76).



```yaml
behaviors:
  BehaviorY:
    # < Same as above >

# Add this section
environment_parameters:
  my_environment_parameter:
    curriculum:
      - name: MyFirstLesson # The '-' is important as this is a list
        completion_criteria:
          measure: progress
          behavior: my_behavior
          signal_smoothing: true
          min_lesson_length: 100
          threshold: 0.2
        value: 0.0
      - name: MySecondLesson # This is the start of the second lesson
        completion_criteria:
          measure: progress
          behavior: my_behavior
          signal_smoothing: true
          min_lesson_length: 100
          threshold: 0.6
          require_reset: true
        value:
          sampler_type: uniform
          sampler_parameters:
            min_value: 4.0
            max_value: 7.0
      - name: MyLastLesson
        value: 8.0
```

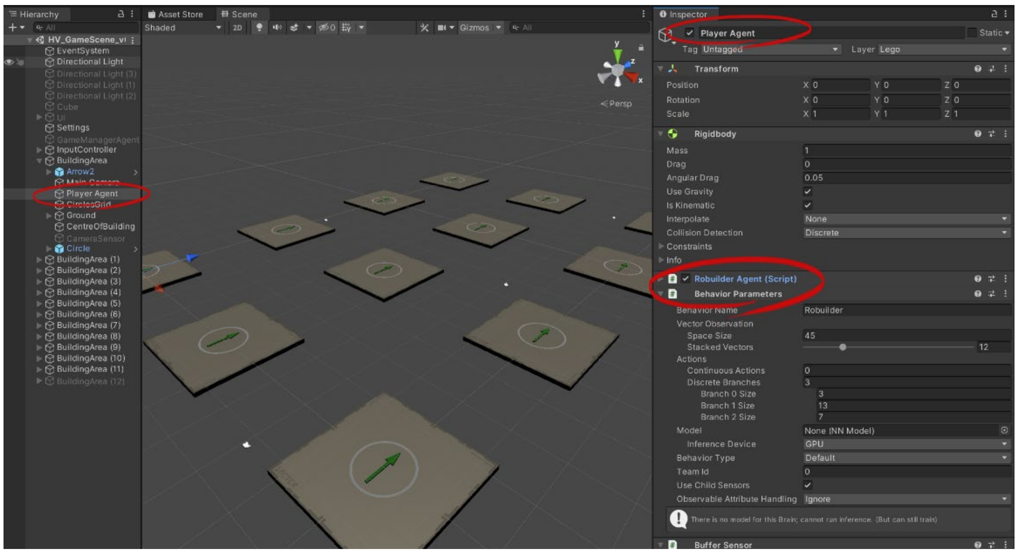
Figure 76: Example of configuration file (.yaml)

- *Agent(s)*

In MDPs, the agent is the one who takes the decisions inside of the environment. In other words, the agent uses the currently adopted policy to decide on what next step it shall take. Therefore, during the learning process, the Agent is the "student" that takes notice of the state it is in and starts to learn based on the rewards and penalties received.

Similarly, in ML-Agents terms, an agent is an actor that can observe its environment, decide on the best course of action using those observations, and execute those actions within the environment [87]. The Agent Implementation in Unity is accomplished by adding an implemented version of the subclass (`Class Agent`) to a game object inside Unity.

In RV, the game object to which the (`Class Agent`) script will be attached to is the "Player". This Game object was previously receiving the input from the Human User and handling the execution of these "actions" internally. However, in order to be qualified as an agent, the "Player" has to implement the `Class Agent` in one of its attached scripts and consequently the "Behavior Parameters" component would be automatically added to the game object. In our case, the class inheriting from `Class Agent` is "Robuilder Agent" (see Figure 77).



```
Player Agent                                   Static
Tag Untagged                  Layer Lego
Transform
Position        X 0      Y 0      Z 0
Rotation        X 0      Y 0      Z 0
Scale           X 1      Y 1      Z 1
Rigidbody
Mass                        1
Drag                        0
Angular Drag                0.05
Use Gravity                 ✓
Is Kinematic                ✓
Interpolate                 None
Collision Detection         Discrete
Robuilder Agent (Script)
Behavior Parameters
Behavior Name               Robuilder
Vector Observation
  Space Size                45
  Stacked Vectors                       12
Actions
  Continuous Actions        0
  Discrete Branches         3
    Branch 0 Size           3
    Branch 1 Size           13
    Branch 2 Size           7
Model                       None (NN Model)
  Inference Device          GPU
Behavior Type               Default
Team Id                     0
Use Child Sensors           ✓
Observable Attribute Handling  Ignore
There is no model for this Brain; cannot run inference. (But can still train)
```

Figure 77: "Player Agent" Game object Implementing "Agent" Sub Class and "Behavior Parameters" as a prerequisite for qualifying as an agent

Regarding the behavior parameters, they are specific attributes of the agent such as the number of actions that an agent can take, whether they are discrete or continuous, the number of observations and their stacking, etc. A behavior can be thought of as the function that receives the observation and rewards and then decides which "Brain" will be fed by this information to come up with an action [78].

Accordingly, there 3 types of brains or "Inference Devices", namely, Heuristic, Learning or Inference.

- *Heuristic Behavior*: Is a behavior defined by hard coded rules, these rules could also be adapted to receive input from the Human User. The implementation function that should be filled with such a code is called " `Heuristic(in ActionBuffers actionsOut)` ". It monitors the Observations and Rewards and feeds the `ActionBuffers` with the final actions. In our case, this function was primarily used as a debugging tool, where mouse and keyboard inputs were allowed in order to test and debug the new implementation of `Class Agent` before the actual starting of the learning process.
- *Learning Behavior: It* is one that is not, yet, defined but about to be trained. So, it means that once there is a valid external connection with the Python Trainer, the Neural Network of the algorithm will be the one in charge of the actions.
- *Inference Behavior:* Once we are satisfied with the learning behavior of the agents, the training could be stopped. Then, the neural network file (.onnx) containing the most updated weights can be placed in the "Model Parameter" and thus the Behavior is inferred directly to this Neural Network File without any training involved in the process.

To sum up, this sub-section provided an overall explanation of how MDP can be implemented in Unity using ML-Agent's toolkit. The transformation from a UV to RV with the main "brain" being the neural network instead of the human player has been briefly discussed.

2.8.3. Phases

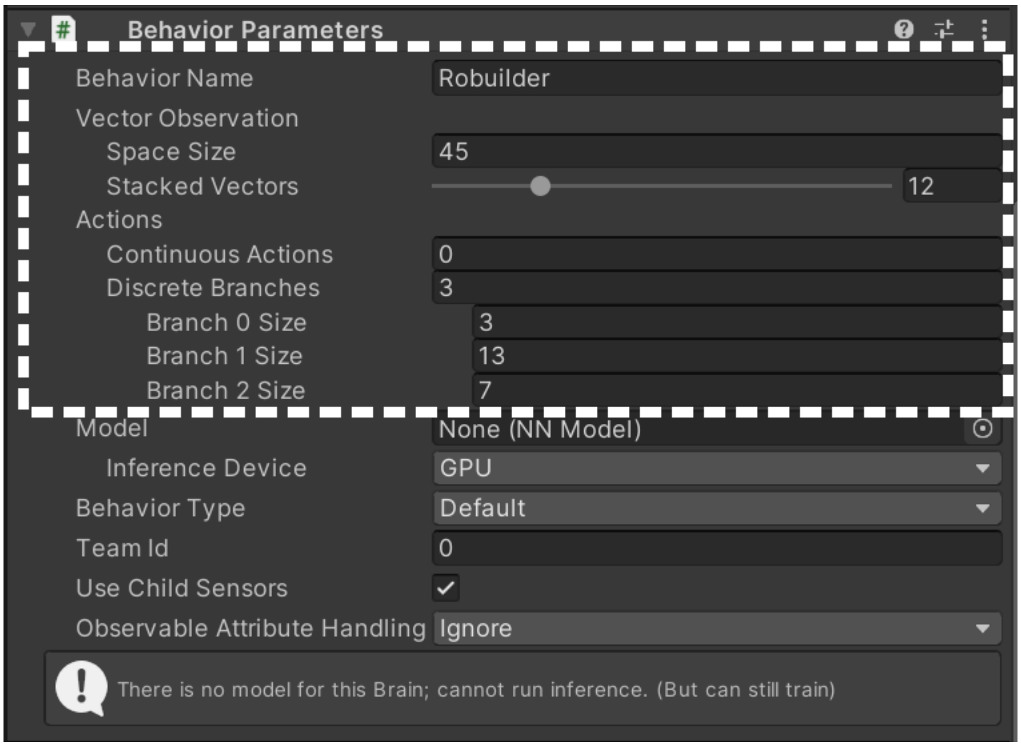
In order to demonstrate how the Rewards, Observations and Actions of a MDP has been formulated in our game and how they were implemented in the language of "ML-Agents", we need first to define what is the episode or episodes that the agent will train on.

Due to the game complexity and to accelerate the learning process of the agents, the decision was to break down the main objective into a set of consecutive goals. These goals, or phases, are considered as a division and separation of the design and construction of the brick pavilion into smaller tasks.

The agent is exploring each of these tasks separately without any interference with the other tasks or phases. Therefore, these phases are considered "stand-alone" episodes that has its own rewards, observations, and actions. The main challenge relies in 2 aspects: *constraining the number of observations and actions to the same number along all phases and finding a way to assemble these discrete phases in inference mode in order to be able to build the pavilion.*

In this sub-section we will be confronting the first challenge, while the second challenge regarding the way of combining the phases together after learning is discussed in sub-section 3.8.8.

As for the first challenge, the reason we needed to maintain a constant number of actions and observations in each phase is mainly because it is not possible to change these parameters "automatically" when changing from one phase to the other according to ML-Agents. Therefore, this concept would not be violated by the proposed "phases" solution as long as each phase maintains the same number of observations and the same "type", branch sizes and number of branches of actions as in Figure 78.


Behavior Parameters

Behavior Name	Robuilder
Vector Observation	
Space Size	45
Stacked Vectors	12
Actions	
Continuous Actions	0
Discrete Branches	3
Branch 0 Size	3
Branch 1 Size	13
Branch 2 Size	7
Model	None (NN Model)
Inference Device	GPU
Behavior Type	Default
Team Id	0
Use Child Sensors	✓
Observable Attribute Handling	Ignore

There is no model for this Brain; cannot run inference. (But can still train)

Figure 78: Behavior Parameters that it is required to be constant during the learning and inference process

Regarding the training phase, an integer input parameter was added, within the *settings* script, ranging between [0,4]. These values represent the 5 phases that we decided to split the game into for training purposes. Before the training of a certain phase, the gauge is set up by the human "Supervisor" on the specific phase the agents are needed to learn. However, during the inference phase these values are internally controlled to ensure an automatic sequential transition of phases with no human interference. In the following figure, a scheme of what kind of learning is involved in each phase is demonstrated.

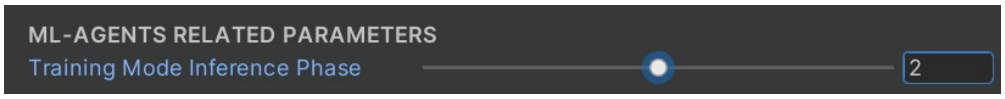

ML-AGENTS RELATED PARAMETERS
Training Mode Inference Phase 2

Figure 79: Phases Selection "Slider

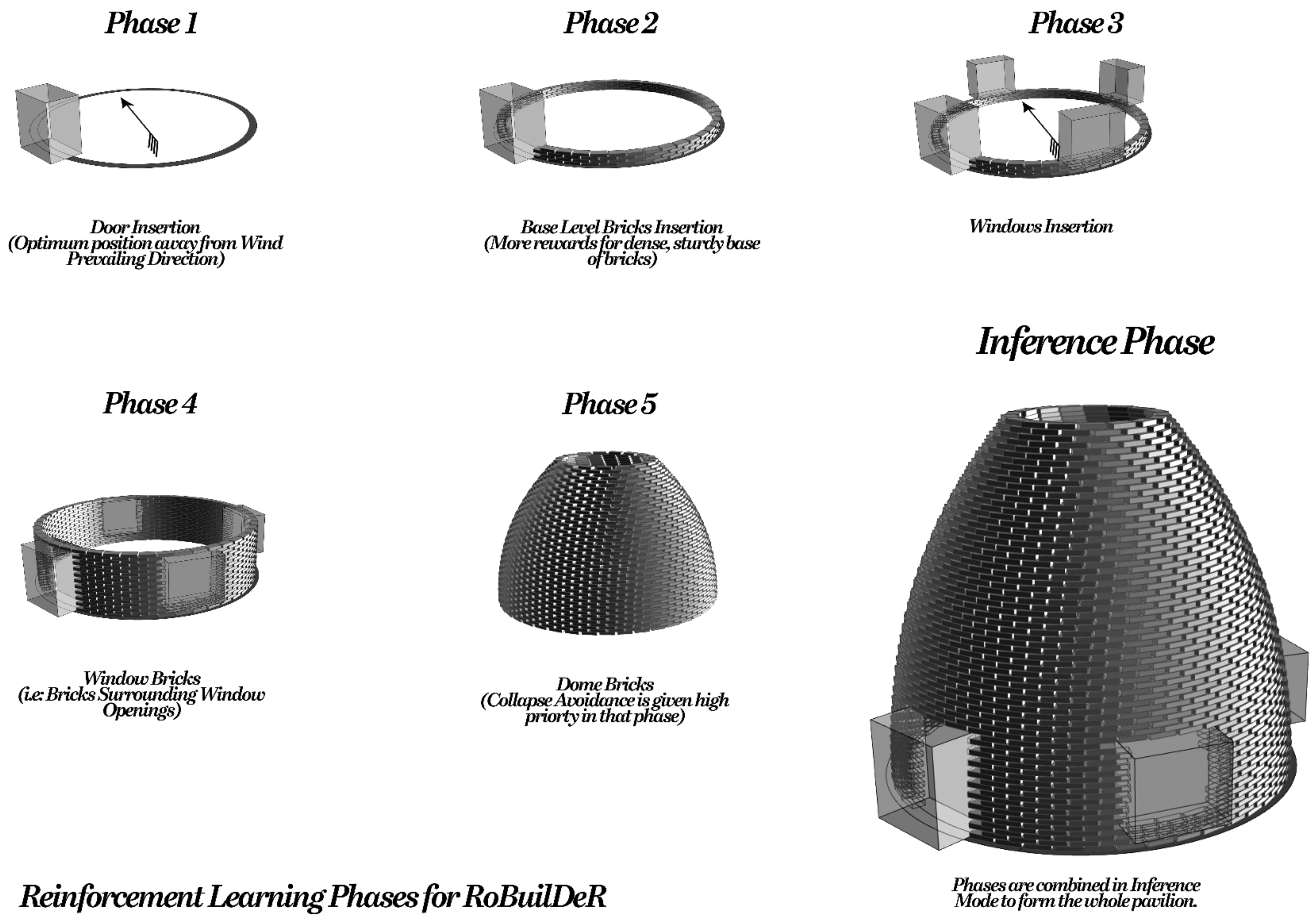

Phase 1
Door Insertion
(Optimum position away from Wind Prevailing Direction)
Phase 2
Base Level Bricks Insertion
(More rewards for dense, sturdy base of bricks)
Phase 3
Windows Insertion
Inference Phase
Phase 4
Window Bricks
(i.e: Bricks Surrounding Window Openings)
Phase 5
Dome Bricks
(Collapse Avoidance is given high priorty in that phase)
Phases are combined in Inference Mode to form the whole pavilion.
Reinforcement Learning Phases for RoBuilDeR

Figure 80: Phases Demonstration for RoBuilDeR RV

Based on the scheme provided in Figure 80, the phases provided can be perceived and explained as follows:

- *Door Insertion Phase:*

In this phase, the main concern of the agent is the proper placement of the door opening. The optimum placement of the door is to insert it as far away from prevailing wind direction as it was previously explained in Table 5.

- *Base Bricks Phase:*

This phase is concerned with laying the bricks from the ground till reaching the sill height for inserting windows. The main goal for the agent in these phases is to keep the structure as sturdy as possible. This could be achieved by the proper packing of bricks as close as possible to each other. This is considered the “foundation” upon which all loads form the upper bricks will be transferred to.

- *Window Insertion Phase:*

This phase involved the proper placement of window openings in such a way that optimizes the natural ventilation strategies. This would include the proper achievement of both cross and natural ventilation. Cross ventilation would be realized through the alignment of 2 windows: one directly facing the wind direction and the other one on the opposite side of it. On the other hand, stack ventilation can be best captured by the placement of only one window facing the wind direction and benefiting from the original dome design in which there is an oculus (opening) that permits air ventilation.

- *Window Bricks Phase:*

After the placement of the “openings” for windows, the aim is to strengthen around the surrounding area around them. This would be done using the same methodology as in the first brick placement phase, namely: trying to pack the bricks as close as possible to support the bricks on the dome above and to mitigate the impact of the void openings present in the structure.

Therefore, we would eventually observe that training an agent on the first phase of bricks can be helpful in speeding up the learning process in the window bricks phase. This could be done, by initializing the learning process, from where the neural network of the previous brick placement phase has stopped. This could only be realized since both phases opt for the same “general” goals to be achieved.

- *Dome Bricks Phase*

This represents the final phase to be realized in order to achieve the full structure of the building when combining all phases. The main priority in this phase is to abide by the design rules in terms of limited unnecessary gaps between bricks while at the same time ensure the proper placement of bricks to avoid collapse. This is considered the most critical phase among the 5 phases included in the project.

2.8.4. Actions and Masking

In this sub-section we will first visualize the general action strategy adopted for inserting a unit (brick, window or door). Then, the general procedure for action masking will be explained. Afterwards, phase-specific requirements that would lead to masking of additional actions shall be provided.

- *General Actions Strategy*

Actions taken can either be discrete or continuous, depending on the nature of the environment [78]. In RoBuilDeR, the action needed could be briefly explained as the process of selecting an appropriate position for a "Unit" to be inserted in one of the available positions along the circular ring. This unit could be a door or window opening or the Brick itself. In the current UV that was previously demonstrated, the decision was to go for a specified number of "insertion points" available for placing the appropriate unit.

Therefore, the decision was to use a *discrete* number of insertion points rather than a more complicated continuous area. The reason for this approach, is to reduce the significant amount of learning needed in such an environment if it was regarded as a continuous canvas for unit insertion. This was realized from the early trials on the use of continuous actions in the game.

One of these early trials involved the use of 1 continuous action branch with 1 variable. This float variable generally generates an output from the PPO algorithm pre-clamped between [-1,1]. This range was remapped into [0, 360] which represents the degrees in a circle. A selected angle by the agent was then transformed into a unit vector that point towards a location on the circle gird in which the Agent has decided to place the brick on. This approach required a massive amount of learning time for the Agent to make a simple progress.

Accordingly, it was then decided to opt for a discrete set of actions. The current action schemes rely on 3-level-action branches. However, any unit placement activity would need the execution of this action scheme in 3 separate consecutive steps before actually being inserted in the required position.

The first set of actions is focused on selecting the appropriate level for the unit placement, the second involves dividing the circle into 12 segments, hence, selecting one of them. Finally, the last action branch contains 6 indices representing the division of the selected segment into 6 equally sized portions.

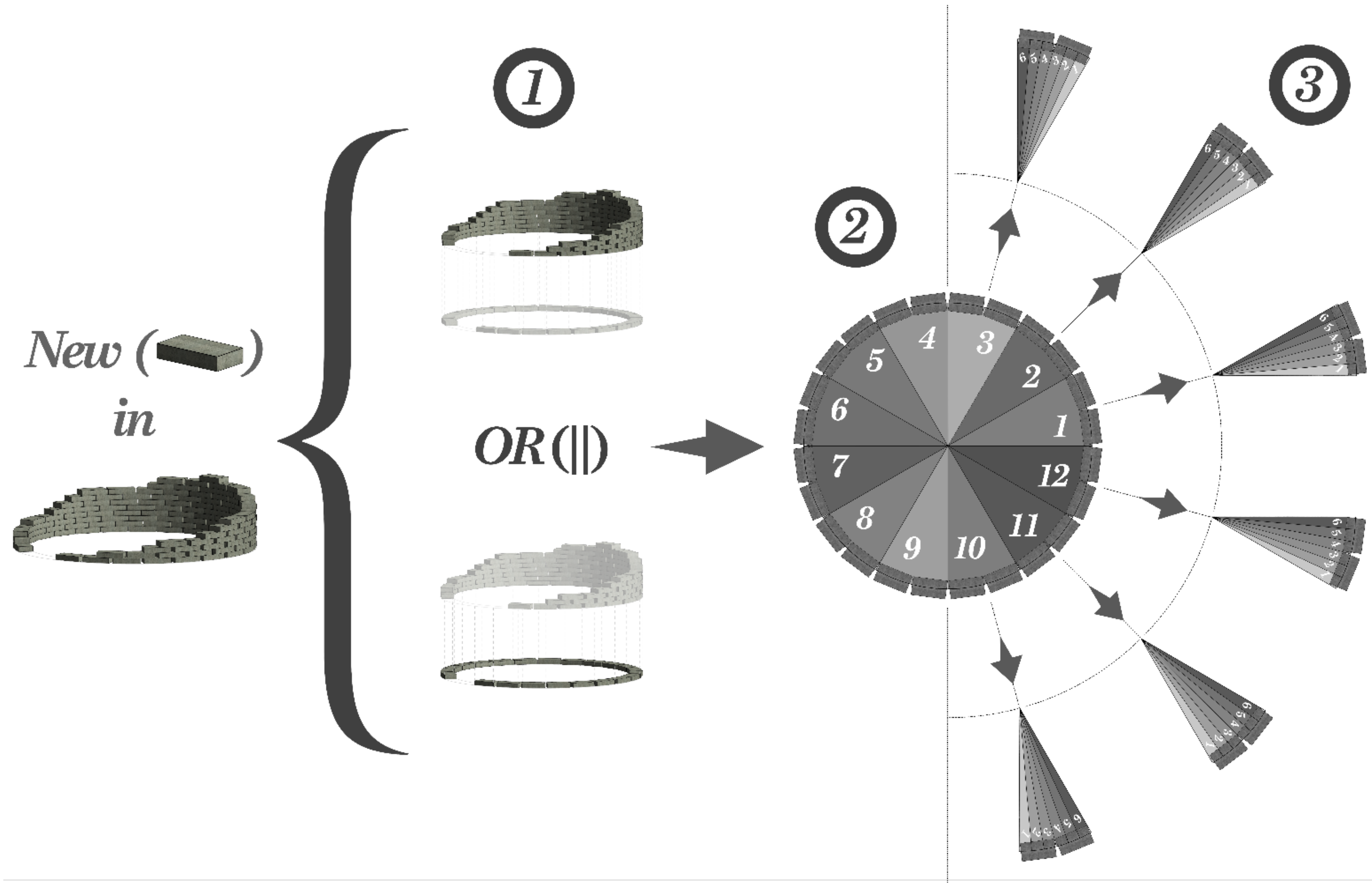


Figure 81: General Action Strategy Scheme for placing a brick

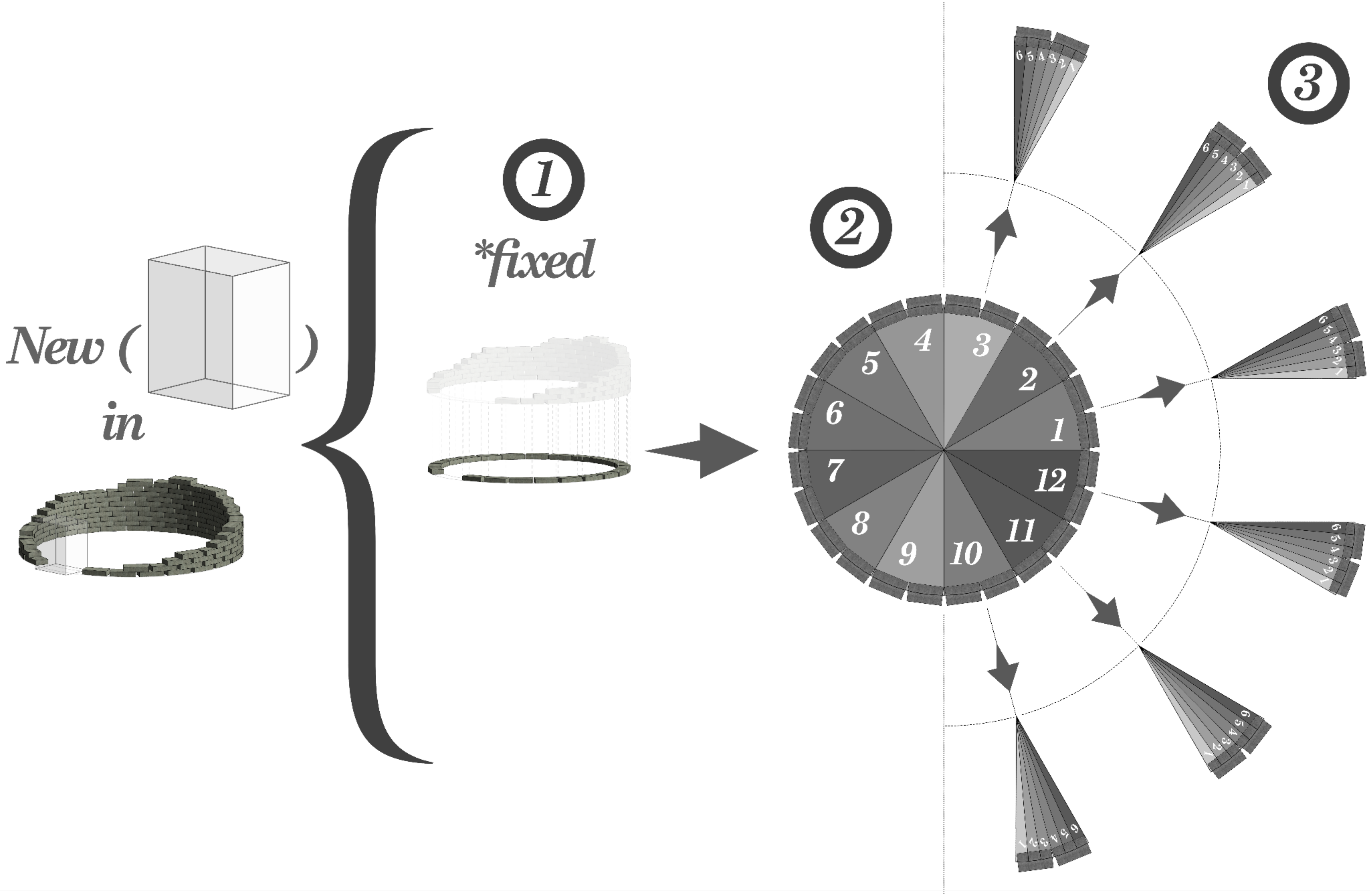


Figure 82: General Action Strategy Scheme for placing a Door Opening

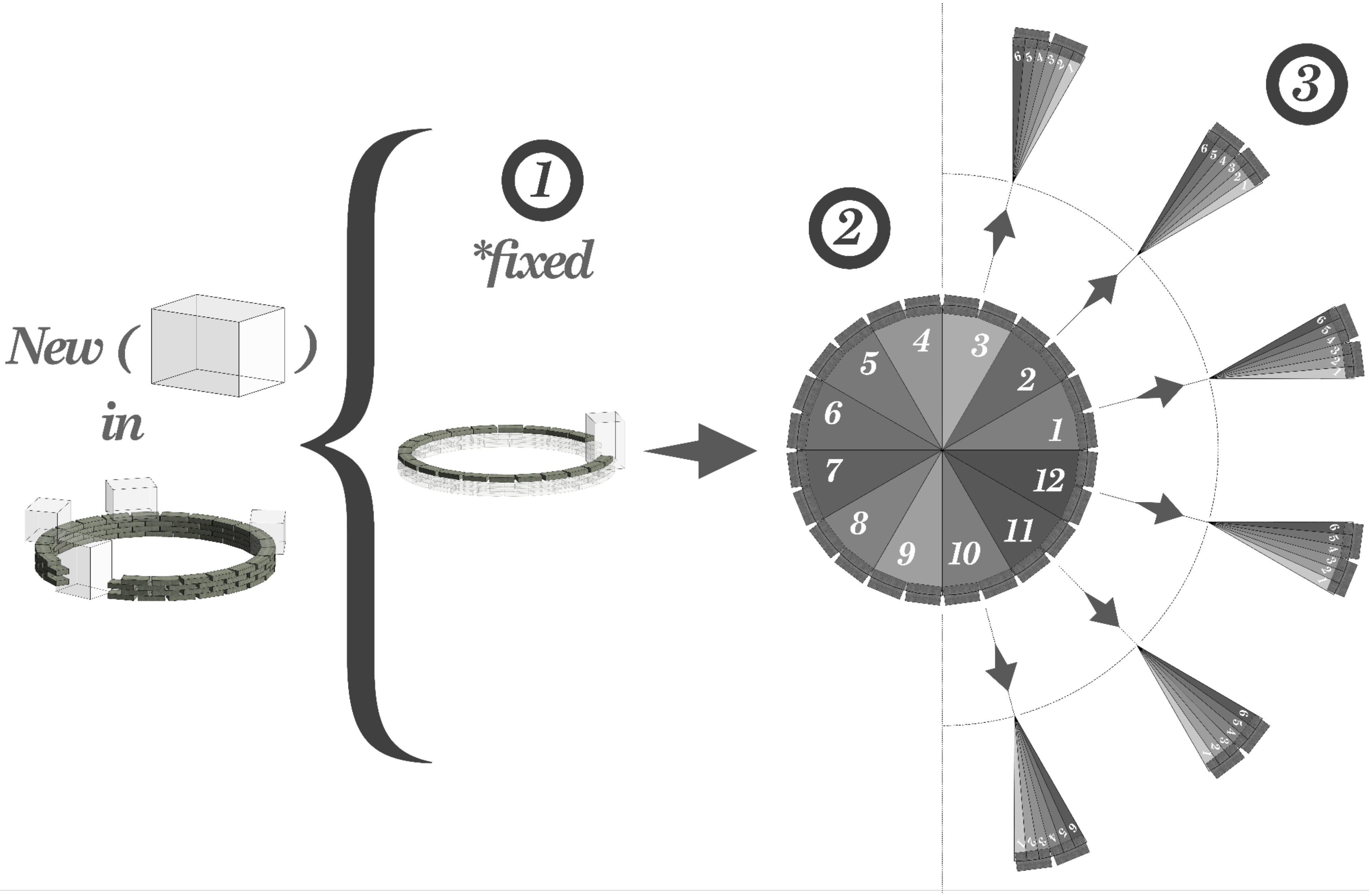


Figure 83: General Action Strategy Scheme for placing a Window Opening

As anticipated from Figure 81, Figure 82, and Figure 83 respectively, the action scheme for placement of brick, door or window unit passes by a 3-branch-action levels. Since the first action level is concerned with selecting the appropriate "height" for unit placement, there is a slight difference between brick placement versus window or door placement.

This difference comes from the way the phases were divided. For instance, for door placement, the insertion is constrained only to the ground floor. Therefore, there is this *fixed annotation in the scheme in Figure 82. Similarly, for windows phase windows placement are tied to a specific level that is determined by the designer for deciding on the appropriate sill height. This also led to "fixing" the placement to just one level as in Figure 83.

Theoretically, this difference in number of actions in the first action branch should not be allowed as per ML-Agents' procedures, otherwise it would not be possible to "assemble" the 5 phases together if they don't have identical action-branch size. However, using the ability to mask actions provided by ML-Agents, it would be possible to "pretend" that a level selection option does exist in the first action branch, for door and window phases.

This would mean that the neural network shall always pick the option of inserting the brick on the lowest incomplete level in both window and door placement phases. So, we would just "mask" the upper levels from the action choices for the agent. This procedure will be explained in more details in the next sub-section.

- *General and Phase-Specific Masking Schemes*

To implement the proposed scheme shown for discrete action branches, we need to understand the general concept of masking provided by ML-Agents. This procedure secures a smooth flow from one branch level to the other.

For the General scheme, in the first action branch there were 2 actions which involved choosing between lowest incomplete level or upper levels. Once selected, it leads to the second branch that contains 12 segments dividing the circle and once a segment is selected the third branch is activated and a max of 6 indices appear to select from. Selecting one of those indices then leads to picking up the insertion point available in this index and the unit could finally be placed as part of the pavilion structure.

However, by observing the values inserted in "Behavior Parameters" for the sizes of the 3 discrete action branches (Figure 84), it is noticed that there is always an additional (+1) action added in each branch. This additional action option is added to represent the "do nothing" option. This could be optional in case of 1 discrete action branches, however, the addition of this option is obligatory when dealing with multiple branches, as it is required that for each action taken, the neural network needs to have at least one option provided in each level to select from.

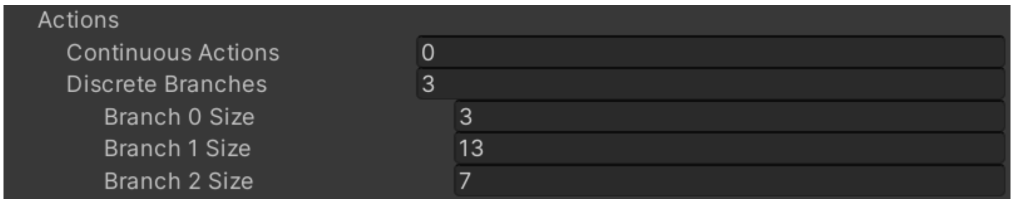

Actions
Continuous Actions 0
Discrete Branches 3
Branch 0 Size 3
Branch 1 Size 13
Branch 2 Size 7

Figure 84: Discrete Action Branches Values inserted in Behavior Parameters Component

For example, if we are now at the point of selection of the action of whether to build on the lowest available incomplete level (indicated by 1) or any other higher levels (indicated by 2), the first thing that we have to do is to mask the "do nothing" option at that level. This makes sense, as we don't want the neural network to "do nothing" at that current level.

On the other hand, during that same decision, there is a need to "mask" all the segments and indices in the branches below. In case this is not done, the neural network would select a segment from second branch and an index from the third. This would not be accurate as we still don't know whether there is an actual point at this segment or not.

In case no point exists, the network would still choose a segment and then an index where there is no point in. This inaccurate behavior would later on affect the learning process and makes it harder for the agent to link the actions taken with the consequences provided. This chaotic scenario can be avoided by the "do nothing" option available at each branch.

Turning back to our example, if the agent decides, based on its policy, to place a brick on the lowest incomplete level (indicated by 1), the final result of the first step taken could be :

First Branch: 1 = Lowest Incomplete Level
Second Branch: 0 = do nothing
Third Branch 0 = do nothing

This indicates that the Agent has executed an actual step in the environment independent of whether a brick was placed or not. Based on the output given from that step, it is now possible to observe the environment, and be ready for the next action to take. Inside of the game mechanics, the masking function is also utilized for another purpose, namely, blocking the segments in which there are no points in them, since we are now considering the selection of the appropriate segment to place the brick in.

Intuitively, this procedure wouldn't have been possible if we didn't know from the beginning on which level the agent would place the brick, perhaps if it had chosen to build on upper levels, a segment in which there are available points on the lowest level, may happen that its indices doesn't contain any points on the upper levels and vice versa.

Consequently, based on the masked actions fed to the network, it would select a certain index based on its policy currently adopted, so let's say the final action branches for step 2 would be as follows:

First Branch: 0 = do nothing
Second Branch: 5 = Segment 5 (arbitrary number just for demonstration)
Third Branch 0 = do nothing

The latter would be the final output of step 2. Finally, to prepare for step 3, the masking method again becomes handy, blocking the indices that doesn't contain any points and showing only the ones that does. So, eventually the output of the neural network could be as follows:

First Branch: 0 = do nothing
Second Branch: 0 = do nothing
Third Branch: 3 = Index 3 (arbitrary number just for demonstration)

After executing these 3 steps, the actions received may now be sufficient for placing a desired unit. From the index number, the point lying in that index is selected and then the unit is finally inserted at that point. It has to be noted that, before the selection of an action when the game as at Step 1, there is also a procedural check that is carried out to makes sure that there are points available in both options provided.

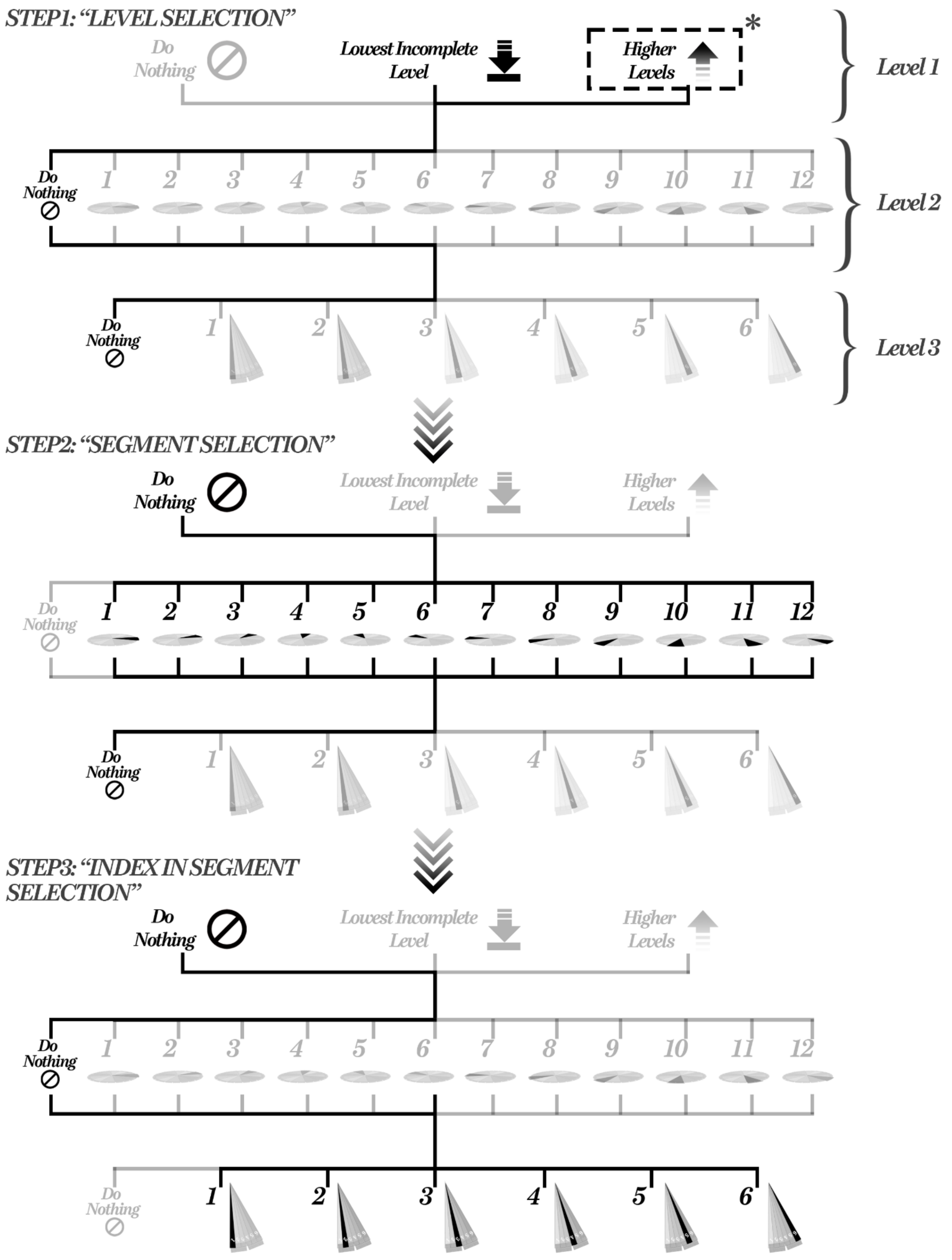
Logically, in all cases and unless the game is finished, there is always at least one insertion point available at the lowest incomplete level. However, at some circumstance, like when there is only one level remaining for episode completion, the availability of points on upper levels may not be present and hence the network would be already blocked from picking this option from the beginning and practically has only 1 option in Step 1. This is actually the case for door or window insertion phases, since the placement is only allowed in one single level.

Accordingly, provides a schematic explanation of the 3 steps for placing a brick along with the action masking role in blocking the other "actual" options on the other levels except the current required one. The scheme notes that the in door and window insertion phases, the option of "upper levels" is always masked. Thus, at Step 1 the neural network always choses the "Lowest incomplete level" option only.

Eventually, masking of actions is regarded to be beneficial in 3 aspects:

1. General masking of "actual" options in all branches other than the one we are currently choosing from.
2. Specific masking of options on the current branch. This could turn off all options that doesn't contain any available insertion points.
3. Specific masking of options throughout the whole phase. This is the case for window and door insertion phases that are deployed only on one level.

MASKING OF ACTIONS



* *This action is "Masked" in "Door Insertion" and "Windows Insertion" phases*

Figure 85: Masking of actions throughout the 3 steps leading to an actual Unit placement

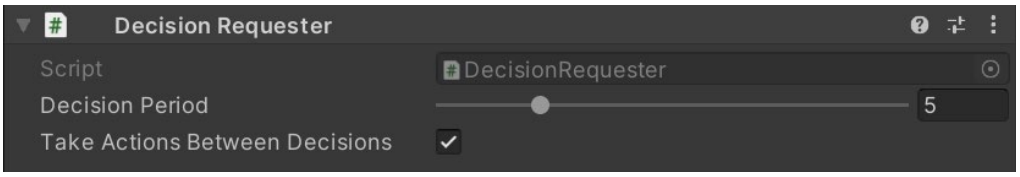
It has to be noted that, for ML-Agents, the method needed to be overridden in order to disallow an action(s) is called `Agent.WriteDiscreteActionMask()`. Inside of which the method `SetActionEnabled()` is called on the provided `IDiscreteActionMask` as follows:

```
public   override   void   WriteDiscreteActionMask(IDiscreteActionMask
actionMask)
{
   actionMask.SetActionEnabled(branch, actionIndex, isEnabled);
}
```

Therefore, in order to disable an action we need to specify in the `SetActionEnabled()` method: the action branch, its index and in the third parameter we need to specify "false". This indicates that we want this action to be disabled.

- *Decision Request Adjustment*

The last piece in the action scheme configuration is the "decision requester". This component is generally added to the game object acting as the agent in Unity along with the other components mentioned before. This component implements (`Class DecisionRequester`). Its main objective is to automatically request decisions for an Agent instance at regular intervals (see Figure 86)[88].



Decision Requester

Script: DecisionRequester

Decision Period: 5

Take Actions Between Decisions: ✓

Figure 86: Decision Requester Component in Unity

However, due to the nature of our game, there is no possible way to define the time at which the agent should take a decision only by setting a time interval. Therefore, the automatic request of decisions was disabled. In return, the method itself responsible for taking decisions namely `RequestDecision()` was directly deployed within "RobuilderAgent" script, itself.

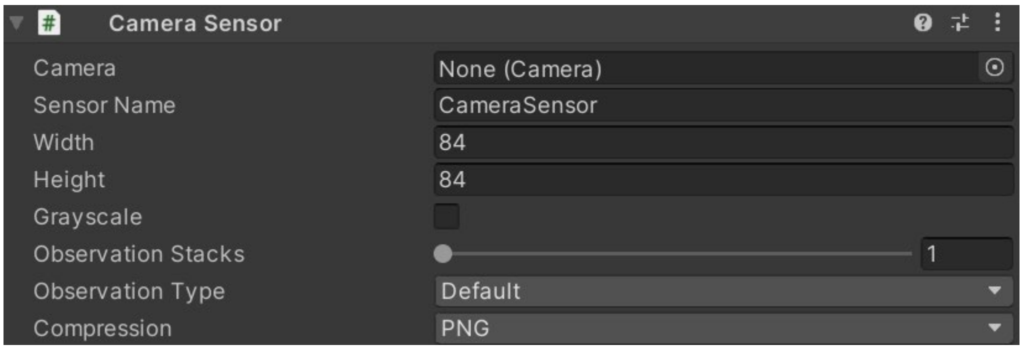
This solution has provided us with a control on when to allow the agent to make a decision. This function was linked to a Boolean parameter, that makes sure that all the procedures needed after taking a decision is completed before allowing the agent to take a new action. For instance, at branch 3 action level, once the agent takes an action and decides on a certain index in a Segment, the agent is kept blocked from taking any decision till the unit is placed in the designated point, and all insertion points are properly updated. Afterwards, the agent is allowed to take a new decision for Step 1. This important update ensures the presence of reliable data about the actual insertion points available in the scene. This would eventually help in masking empty branches that has no points inside of them.

2.8.5. Observations

Observations are the most crucial factor for proper training of agents [78]. The observations have to include all the information needed by an agent to accomplish a task. In case of insufficient, irrelevant, or inaccurate information, the agent may take longer time to learn or may not learn at all. In order to include the necessary information, the human "designer" should think thoroughly about all the information he/she would need to solve such a problem and try to enable the agent access to such information in each step taken in the environment.

To achieve this, ML-Agents has provided several tools to convey information to the agents. The main method used for that purpose is `Agent.CollectObservations()` where all observations are passed inside of its implementation. It is considered the best used way for aspects of the environment which are numerical and non-visual. In our case, we have translated all "visual" and non-visual observations into numerical values. This decision was taken due to the complexity accompanied by feeding the network with different kinds of observations at the same time.

For instance, the use of visual observations was experimented in RoBuilDeR. One of the early trials has involved the use of static camera in each "Building Area" that captures an orthogonal picture of progress on-site at each step. The picture resolution can be adjusted before training through the "Camera Sensor" component attached to the Agent game object in Unity. Based on the width and height of the picture in pixels, the RGB values of each pixel or only a Grayscale value in case selected, will be reported as observations to the network. Therefore, a picture of 84 x 84 pixels would report 7056 observations in case gray scale and tripe this number in case of RGB (see Figure 87).


Camera Sensor

Camera	None (Camera)
Sensor Name	CameraSensor
Width	84
Height	84
Grayscale	☐
Observation Stacks	1
Observation Type	Default
Compression	PNG

Figure 87: Camera Sensor Component Added to the Agent

The main issue results from the massive number of pixels that convey useless information to the network. These pixels represent all the empty fields away from the circular ring patch in which all bricks and other units are being placed. This was evident in the slow and at some points no learning acquired by the agents using this method for capturing observations.

Accordingly, it was regarded that the optimum solution is convey all geometric values, locations, and other seemingly visual aspects into numerical values without the need to capture visual images of the scene.

- *Maintaining Vector Observation Size throughout all phases*

As previously discussed in earlier sub-sections, once of the main challenges encountered when dividing the complex problem into sub-tasks that an agent with a single behavior is working on, is that the number of observations should remain constant throughout all phases of the project. To tackle this issue the following procedures has been adopted:

- Analyzing the observations needed in each phase and separating common parameters among various phases that would contain the same number of elements in each phase.
- Working on minimizing the gap between the number of observations in the phase with the maximum number of them, and the phase with the lowest number.
- Once the difference is not that significant, the number of observations will be governed by the phase with the highest number of them.
- The observations will be ordered in an order that reports the common parameters first to the neural network, followed by phase-specific observations. The values of these observations will be reported based on the current active phase.
- The phases with total number of observations lower than the maximum governing one, will be padded with zeros in place of the missing entities.

- *Vector Observations*

`Agent.CollectObservations()` produces *vector observations*, which are represented at lists of floats. Despite that the final output reported to the neural network is in the form of floats, it is essential to differentiate the initial type of an observation that is being reported. For instance, if the value of the observation is Boolean, this means that the network would always receive either 0 or 1. In that case it would be translated to floats but still it would always receive just to values from this observation parameter.

One-hot encoding categorical information

On the other hand, to report categorical observations or enumerations, especially when there are several values, one-hot style can be used. That is, adding an element to the feature vector for each element in the categorical or enumeration list, setting the element representing the observed member to one and set the rest to zero. For example, in case of reporting the index of an already placed brick, there are 6 indices. If the brick's index is 3, then we would add to the observation a one-hot observation of 6 possibilities with 1 at the third index and zeros in the rest of the indices. So, this list would be something like [0, 0, 10, 0, 0, 0].

Normalization

Furthermore, for the best results when training, the components in the vector observation are normalized to the range [-1, +1] or [0, 1]. It is regarded that when the values are normalized in this manner, the PPO neural network can often converge to a solution faster. Since the greater the variation in ranges between the components of observation, the more likely that training will be affected.

Stacking of vectors

Stacking refers to repeating observations from previous steps as part of a larger observation. In our case, this feurture is og great impoertance, since it acts as a “memory” of past actions and observations and include them in the observations provided to the Agent in the current state. This shall help the neural network in developing a sequential relationship between several steps and relate between a series of different actions and their combinations.

However, the drawback of such a parameter is the additional amounts of observations that needs to be fed into the neural network. Increasing this value significantly may lead to a slower learning rate and an adverse effect to that intended from the beginning from using such a feature.

Accordingly, in RoBuilDeR (RV), the value that was selected for stack vectors is 12. Practically, it means tracking the last 3 units (bricks, door, or windows) that were placed in the game, since 3 steps are required for each unit placement. On the other hand, the total number of observations are set to 45 different variables (Figure 88). As discussed earlier, this is the maximum value that is only reached in the phases with maximum number of observations compared to the other ones.

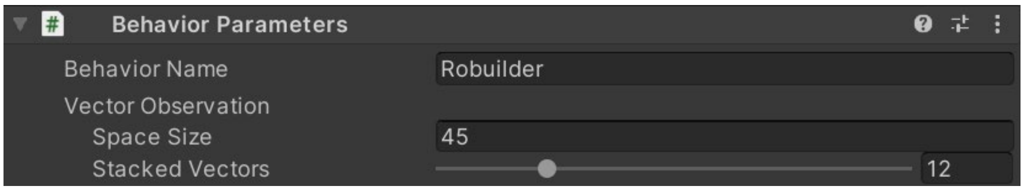

Behavior Parameters
Behavior Name: Robuilder
Vector Observation
Space Size: 45
Stacked Vectors: 12

Figure 88: Vector Observations and Stacked Vectors in RV

The following tables provide a detailed explanation of each observation included in all 5 phases. The tables start by showing common parameters that were observed among all phases. Afterwards, parameters that are common in all 3 brick placement phases are brick placement. Finally, phase-specific observations are introduced.

It has to be noted that, in order to better capture observations related to variable game units that gets instantiated during gameplay, an additional method of observation collection was also integrated. This method is called “buffer Sensor” and the observation captured through this component is also combined with other vector observations in the same `Agent.CollectObservations()` method. More detailed information regarding this method is explained afterwards separately.

Table 18: General Observations included in All Phases

General for ALL Phases

Observation	Explanation	Norm-alize	*Type*
Type of Point Array Normalized	*Only activated at Step 3** *An Array of Point Types*		array
Point Type	Type of the insertion point available in a specific index of a segment	Eq1	float
Angle of Points Array Normalized	*Only activated at Step 3** *An Array of Point Angles*		array
Point Angle	Difference between 2 angles: 1. Angle between a vector from the center of the building to the insertion point that is projected on the XZ plane and the global X-axis 2. The Starting angle of the segment in which this point is present. (Calculated from the X-axis) See Figure 89	Eq 2	float

Notes

* Step 3 was previously demonstrated in the Actions and Masking sub-section. It indicates the third step before inserting a unit. This step involves the selection of one of the available indices in a certain segment.

Eq 1
$$\frac{\text{Point Type}^{**}}{\text{Count of Point Types in the Game } (4-5)}$$

** Point Types is of type enumeration that contains all types of points. Therefore, the used value in the normalization process is the integer value that represents the order of that point in the enumeration.

Eq 2
$$\frac{\text{Point Angle}}{\text{Angle of 1 segment } (= 360/\text{Total Number of Segments})}$$

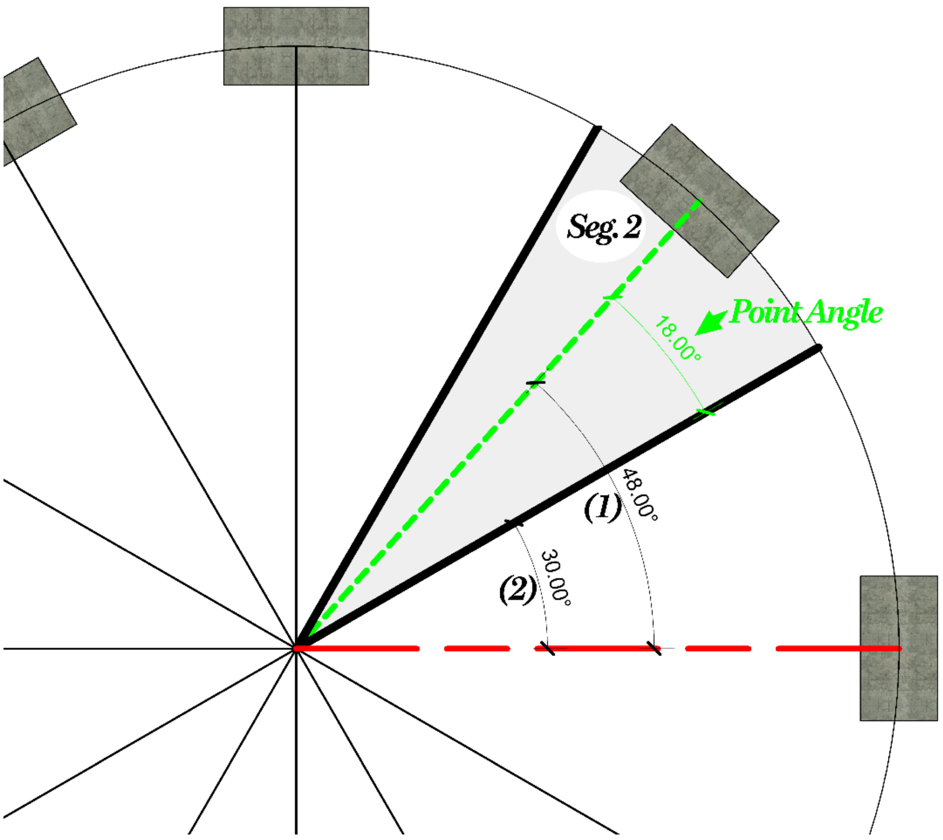


Figure 89: Point Angle Calculation as explained in Table 18

Table 19: Common Observations among all 3 Brick placement Phases

General for All Brick Phases

Observation	Explanation	Norm.	*Type*
Lowest Height In segment			array
Lowest Height In Each Segment	Detects the lowest insertion point available in a given segment	Eq 3	float
Highest Height In segments			array
Highest Height In Each Segment	Detects the highest insertion point available in a given segment	Eq 4	float
Height of Points Array Normalized	*Only activated at Step 3*		*array*
Height of point at index of a Segment	Detects in a certain index in a segment the relative height of the point situated within this index (if a point is available)	Eq 5	*float*

Notes

Eq 3	$\frac{(Lowest\ Height\ In\ a\ Segment)_{abs} - (Height\ at\ phase\ start)_{abs}}{(Max\ possible\ Height\ during\ current\ phase)_{relative\ to\ phase\ start}}$
Eq 4	$\frac{(Highest\ Height\ In\ a\ Segment)_{abs} - (Height\ at\ phase\ start)_{abs}}{(Max\ possible\ Height\ during\ current\ phase)_{relative\ to\ phase\ start}}$
Eq 5	$\frac{(Height\ of\ pt\ at\ index\ of\ a\ Segment\ )_{abs} - (height\ at\ phase\ start\ )_{abs}}{(Max\ possible\ Height\ during\ current\ phase)_{relative\ to\ phase\ start}}$

Table 20: Phase-Specific Observation Parameters for Brick Placement

Phase 1 Base Bricks / Phase 3 Window Bricks

Observation		Explanation	Norm.	*Type*
Consecutive Bricks Array				array
	Consecutive Bricks Count	Counts the number of consecutive bricks inserted in sequence after each other on the same level. If the streak is interrupted the counter turns to 0 and it starts counting from the beginning. If the level is full, the counter value remains constant and continues to count again if the bricks were to be inserted in the same manner on the next level (the lowest incomplete level).	Eq 6	float
	Consecutive Bricks On Top of Each Other Counter	Counts the number of consecutive bricks that are inserted on top of each other. The counter restarts if this behavior is interrupted.	Eq 7	float

Notes

Eq 6	$\frac{\text{ConsecutiveBricksCount}}{\text{Constant } (= 0.014667\text{f})}$
Eq 7	Case Counter == 0: 1 Case Counter > 0: Counter $*$ factor (-1)

Table 21: Phase 5 (Dome Creation) specific observation parameters

Phase 5 Dome

Observation		Explanation	Norm.	*Type*
Consecutive Bricks Array				array
	Consecutive Bricks Count	Counter that adds +1 if the most recently placed brick is inserted directly adjacent to existing bricks, on either side of them on the same level. If the streak is interrupted the counter turns to 0 and it starts counting from the beginning. If the current level is full, the counter value remains constant and is increased if the bricks were continued to be inserted in the same manner on the next level (the lowest incomplete level).	Eq 8	float
	Consecutive Bricks On Top of Each Other Counter	Counts the number of consecutive bricks that are inserted on top of each other. The counter restarts if this behavior is interrupted.	Eq 9	float

Notes

Eq 8

$$\frac{\text{ConsecutiveBricksCount}}{\text{Constant }\ (= 0.01\text{f})}$$

Eq 9

$$-1 * \frac{\text{Brick Counter}}{\text{ConsecutiveBricsdOnTopEchOtherThreshold}^{*}}$$

** Consecutive Bricks on Top of Each Other Threshold (integer) is a fixed number (adjusted before learning starts) that when reached, the game is over (this function could be disabled so as not to contribute to game over scenarios). However, if game is allowed to continue, the maximum value remains at (-1) so as not to disrupt the range of observation values inserted to the Neural Network for training* [78], *even if the number of consecutive bricks over each other exceeds this threshold.*

Table 22: Common Observation Parameters in both Door and Window Placement Phases

Common Observation Parameters for Phase 0 and 2 in general (Windows / Door)

Observation	Explanation	Norm.	*Type*

"All Wind Info"				array
	Wind Segment No. (Continuous)	Segment in which the direction from which the prevailing wind is blowing is situated inside it.	Eq 10	float
	Wind Segment No. (Discrete)		Bool	One-Hot
	Wind Index in Segment No. (Continuous)	The closer Index of the Segment that best represents the direction from which the prevailing wind is blowing	Eq 11	float
	Wind Index in Segment No. (Discrete)		Bool	One-hot
	Wind Unit Vector Angle	Angle in degrees of the Unit Vector representing wind direction. The angle is calculated from the X-Axis	Eq 12	float
	Wind Unit Vector (X value)	X - component of the Wind Unit Vector	Already Norm. [0, 1]	float
	Wind Unit Vector (Z value)	Z - component of the Wind Unit Vector	[0, 1]	float
	Wind Unit Vector Dot X	Dot product between the wind unit vector and the right* vector (unit vector in X direction)	[-1, 1]	float

Notes

*	Right side refers to the red axis in Unity which is a unitary vector in the X-axis direction (see Figure 91).
Eq 10	$\frac{Wind\ Segment\ No.}{Total\ Number\ of\ Segments\ (12)}$
Eq 11	$\frac{Wind\ Index\ No.}{Total\ Number\ of\ Indeces\ in\ Segment\ (6)}$
Eq 12	$\frac{Wind\ Unit\ Vector\ Angle}{360}$

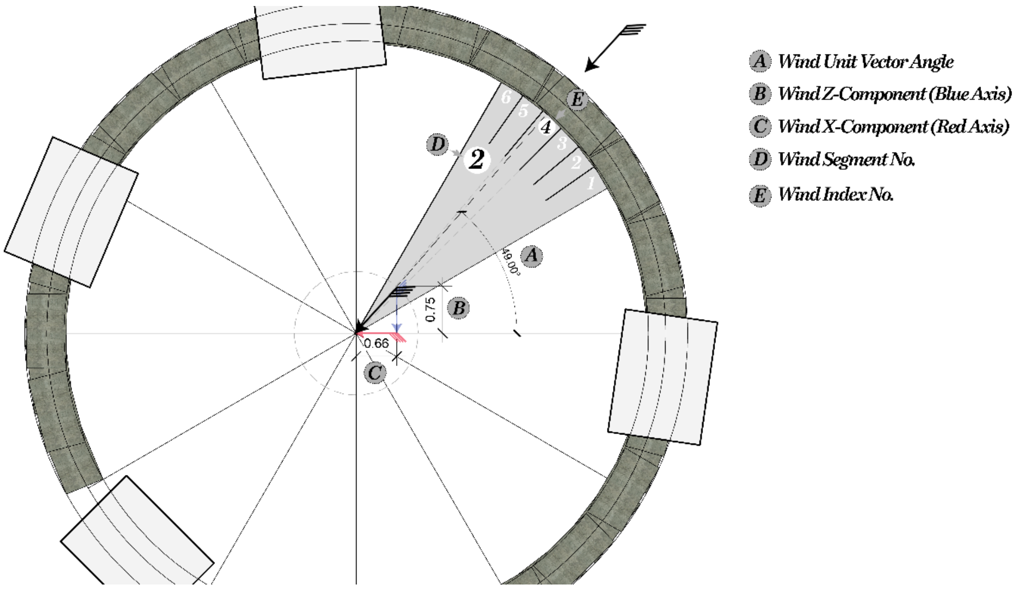


Figure 90: Different Wind Properties included in the Observation as indicated in Table 23

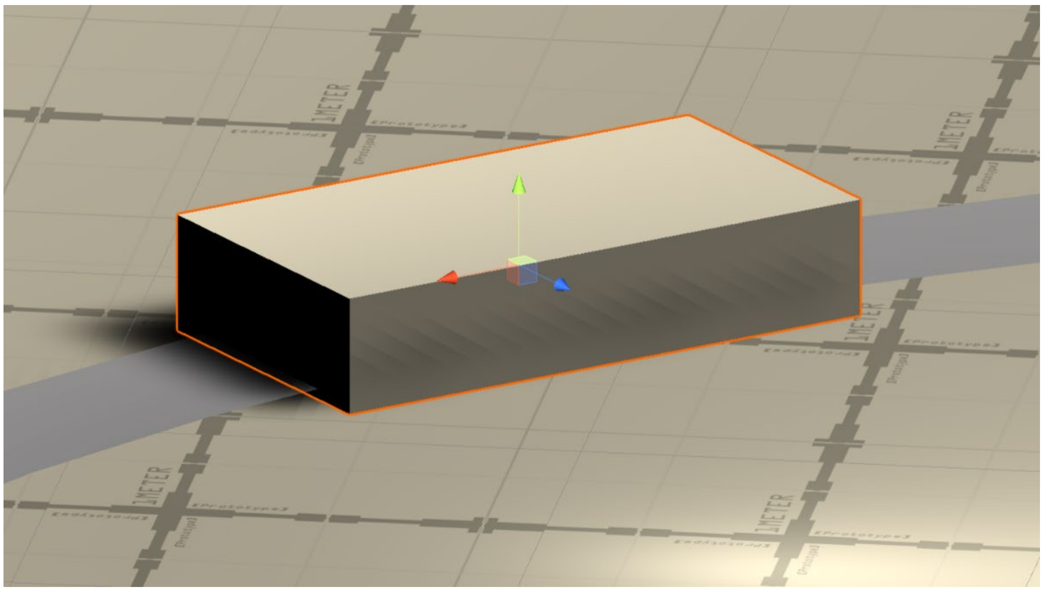


Figure 91: A brick with the local coordinates shown in Unity. Red or "right" Axis is the X-axis, Green or up vector is indicating the Y-direction, and Blue or forward vector is in the Z-direction

Table 23: Specific Observation Paramters for Windows and Door placement respectivley

Windows and Doors Specific Observations

Observation	Explanation	Norm.	*Type*
Windows:			

	Windows Count	Number of Window Openings Inserted	Eq 13	float
	Window Wind Cross Vent Product	Value of Success achieved so far in Cross Ventilation Assessment.	[-1, 1]	float
	Max Window Wind Dot Product	Dot Product between (normalized vector from center of the building to the insertion point of the Window projected on the horizontal plane) and the unit Wind Vector.	[-1, 1]	float
Doors:				
	Door Count	Checks if the door is inserted or not	Bool	int

Notes

Eq 13

$$\frac{Windows\ Count}{Total\ num.windows\ required}$$

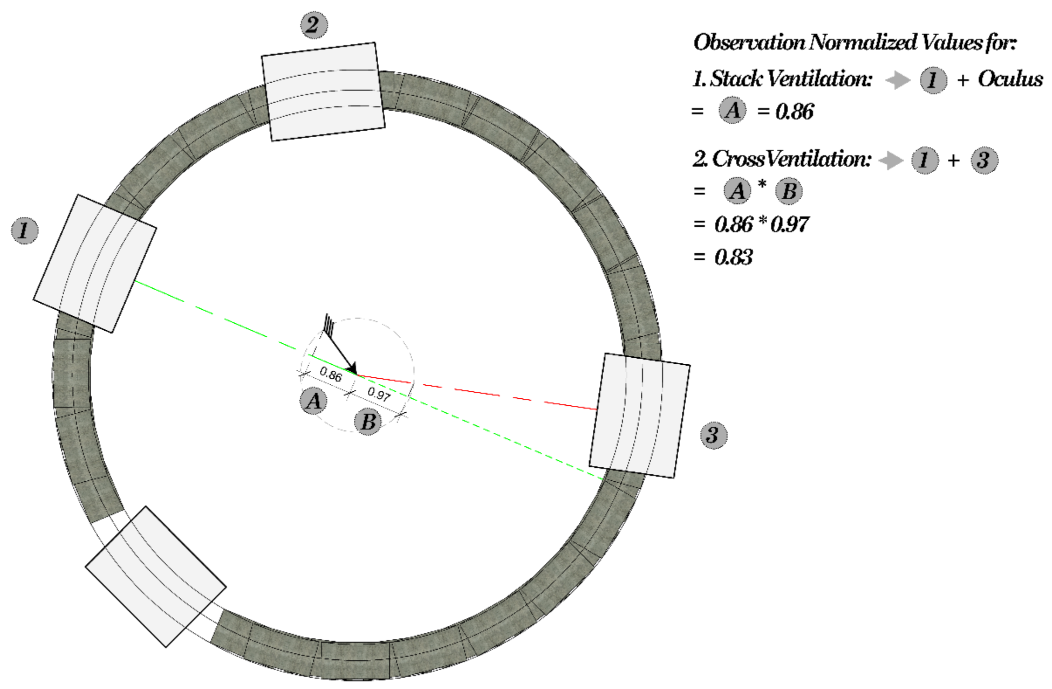

2
1
3
0.86
0.97
A
B
Observation Normalized Values for:
1. Stack Ventilation: 1 + Oculus
= A = 0.86
2. Cross Ventilation: 1 + 3
= A * B
= 0.86 * 0.97
= 0.83

Figure 92: Example of a prevailing wind scenario for the calculation of Stack and Cross Ventilation Normalized Values. Values indicated are used as observation parameters.

- *Buffer Sensor*

Buffer sensors enables the collection of observation data from a varying number of Game Objects. It differs from traditional vector observations in cases when it is not possible to determine the actual number of observables that will be present in the scene. On the trainer side, the BufferSensor is processed using an attention module.

The BufferSensor can be useful in situations in which the Agent must pay attention to a varying number of entities. For instance, in games industry, this could be evident in observing enemies that are spawn during gameplay and their number does not remain constant nor can be determined prior to the beginning of the episode. Since some might be killed, others would be instantiated and so on.

Similarly, in RoBuilDeR the game object that perfectly fits with these conditions is the brick game-object. First, no bricks are instantiated at the beginning of the game. Then, during game play, several bricks are instantiated. Based on the behavior of the policy adopted, the structure can be stable or may collapse and fall. Therefore, in each episode it is not possible to determine the actual maximum number of bricks that will be instantiated.

In addition, if enemies could be killed during game play and their game object gets destroyed, we can compare this situation to the “sleeping” of stable bricks. These bricks are the ones that are proven to be stable for enough time that they get fixed to their position and no more physics check are carried on them. So, conceptually speaking they could be counted as not participating anymore in the actual game events and hence could be considered like a killed enemy.

Moreover, one of the great advantages of attention modules architecture is that attention layers are invariant to the order of the entities, so there is no need to properly "order" the entities before feeding them into the BufferSensor [78]. However, for implementing them 2 challenges arise. First, training or doing inference with variable length observations can be slower than using a flat vector observation. Second, even though the BufferSensor can process a variable number of entities, it is still needed to define a maximum number of entities.

This is because the network architecture implemented by ML-Agents requires to know what the shape of the observations will be. If fewer entities are observed than the maximum, the observation will be padded with zeros and the trainer will ignore the padded observations.

To overcome both challenges, and to reduce the minimum number of observables, we introduced the concept of “naked bricks”. These bricks are regarded as the most “active” bricks in the game, as they are bricks that it is possible to place a new brick directly above them (see Figure 93). Among those group of bricks lies the most unstable, less supported ones. However, it was decided to increase the number of observables more than the maximum number of naked bricks a game could witness. This slight increase in the maximum number of observables is intended to capture also vulnerable bricks on the lower fully covered levels. This additional layer of information is sought to give more insights to the neural network on the quick evolution of instability among bricks.

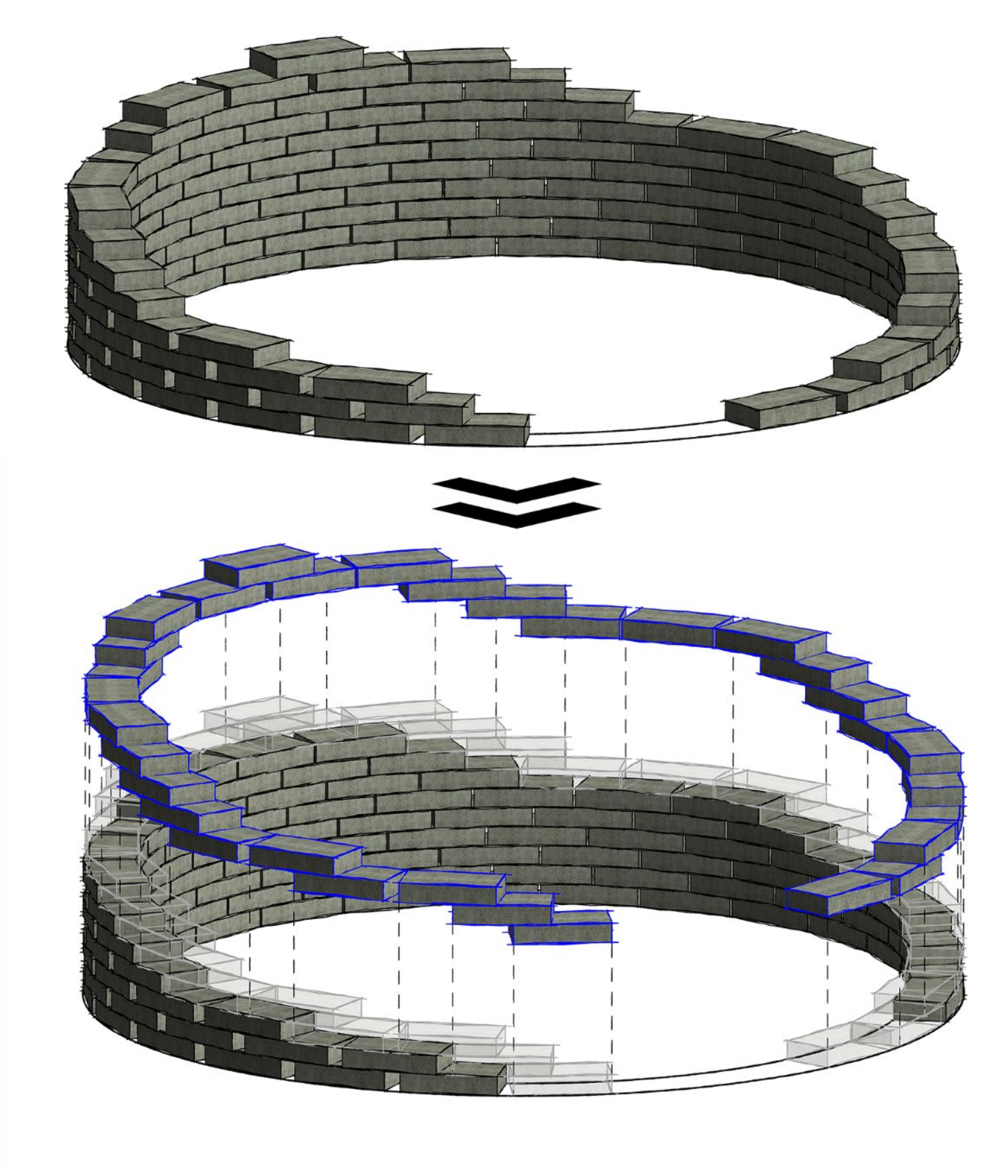
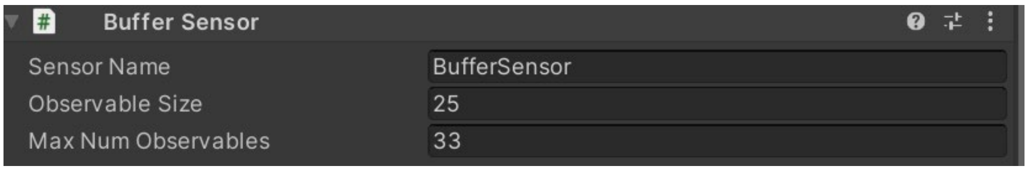


Figure 93: Naked Bricks Demonstration

Accordingly, Figure 94 demonstrates the maximum number of observables used during training for the 5 phases. The figure also shows another parameter called Observable size. This parameter indicates the length of the vector observation that each observable would contribute to. To optimize this number, we adopted the same strategy we mentioned before regarding the trails to reduce the difference between the observable with maximum features and the one with less features needed to be reported.


Buffer Sensor

Sensor Name	BufferSensor
Observable Size	25
Max Num Observables	33

Figure 94: Buffer Sensor Component Parameters and Values used in RoBuilDeR

Logically, the brick is the observable game object during all brick placement phases while the door and window openings are the observables in door and window placement phases respectively.

In the following table the features linked with each observable is provided. Similar to the tables showing vector observations earlier, the presentation of the features here are also classified with respect to common features among different phases first and then regarding specific features found in specific phases.

Table 24: General Observations included in All Phases and for all types of observable-game objects

General for ALL Phases

Observation		Explanation	Normalization	*Type*
Variables ($\sum = 12$)	Segment number	It is the segment of the insertion point on which the observable unit was placed.	Bool	One-hot
Variable ($\sum = 6$)	Index in a Segment of the Observable unit (Brick/Door/Window)	It is the index number (inside a segment) of the insertion point on which the observable unit was placed. (Brick/Door/Window)	Bool	One-hot

General for All Brick Phases

Observation		Explanation	Norm.	*Type*
Variable 1	*Relative Height*	Relative Height of the observable brick with respect to the starting height of the current phase	Eq 14	float
Variable 2	*Point Type*	The insertion points on which the observable brick was placed. It is one of the variables	Eq 15	float

		that each brick has and it is set to a value once a brick is placed in the structure of the pavilion.		
Variable 3	*Normalized Tilting Value (Dot product)*	Measure of how much a brick is "Tilting". It is a brick property. If the "tilting value" decreases, it means that the instability of the brick increases, therefore the observation value reported to the neural network shall always be negative in case of tilting.	Eq 16	int/ float
Variable 4	*Is Last Brick?*	Checks whether this brick observable is the last brick placed in the game or not.	Bool	int

Notes

Eq 14

$$\frac{(\text{Current Brick Height})_{abs} - (\text{Height at phase start})_{abs}}{(\text{Max possible Height during current phase})_{relative\ to\ phase\ start}}$$

Eq 15

$$\frac{Point\ Type}{Total\ Number\ of\ Point\ types}$$

Eq 16

Case No tilting: 0

Case Tilting Value < Threshold: -1*

$$\text{Case Tilting Value > Threshold: } \left\{(-1)\left[\frac{1}{1+e^{\frac{1-Tilting\ Value}{1-Threshold}}}\right]\right\}^{**}$$

** Tilting Threshold is a factor with a value less than one. If tilting value gets lower than this value, it indicates that brick instability has reached its maximum allowable value and hence game is terminated (see* Figure 67*)*

*** The reason for using a "sigmoid function" is to increase the negative value fed to the neural network once half of the allowable threshold has been reached. This is expected to give signals to the neural network that we are in a critical situation, given that suddenly there is a sudden increase in negative value input.*

Table 25: Common Parameters for Brick Observables in Phases 1 and 3

Phase 1 Base Bricks / Phase 3 Window Bricks

Observation		Explanation	Norm.	*Type*
Variable 5	Is Opening brick	Checks whether this brick is directly placed next to an	Bool	int

		Opening (Door or Window) or not.		
Variable 6	Angle between Opening Brick and the Opening	Only has a value if the Brick is an Opening Brick. It calculates the angle between the tip of the brick and the tip of the Opening on the Center Line. If the angle is negative, it means that the brick tip is inside the volume of the Opening.	Eq 17	int/ float
Variable 7	Angle between Consecutive Bricks	*Consecutive Brick Definition in Lower Brick Insertion Phases: Any 2 bricks placed consecutively next each other on the same level, given that the void space between them doesn't allow the placement of a third brick at the same level.* Case the observable brick is consecutive: This parameter will measure the angle between the observable brick and the closest brick to it from the consecutive series. The more the angle increase, the more negative value this observation will report to the neural network. This approach is adopted to encourage the bricks at the lower levels to be closer to each other as much as possible. Case the observable brick is not consecutive: This parameter will be 0 as there is no consecutive brick already	Eq 18	float

Notes

Eq 17

Case Angle < 0: 1

$$\text{Case Angle} \geq 0\text{:}\ \frac{\text{Angle between Brick and Opening}}{\text{Opening full Angle on the Center Line}}$$

Eq 18

$$\text{Case Consecutive Bricks: } (-1)\left[\frac{\text{Angle between Consective Bricks}}{180}\right]$$

Case Non-Consecutive Bricks: 0

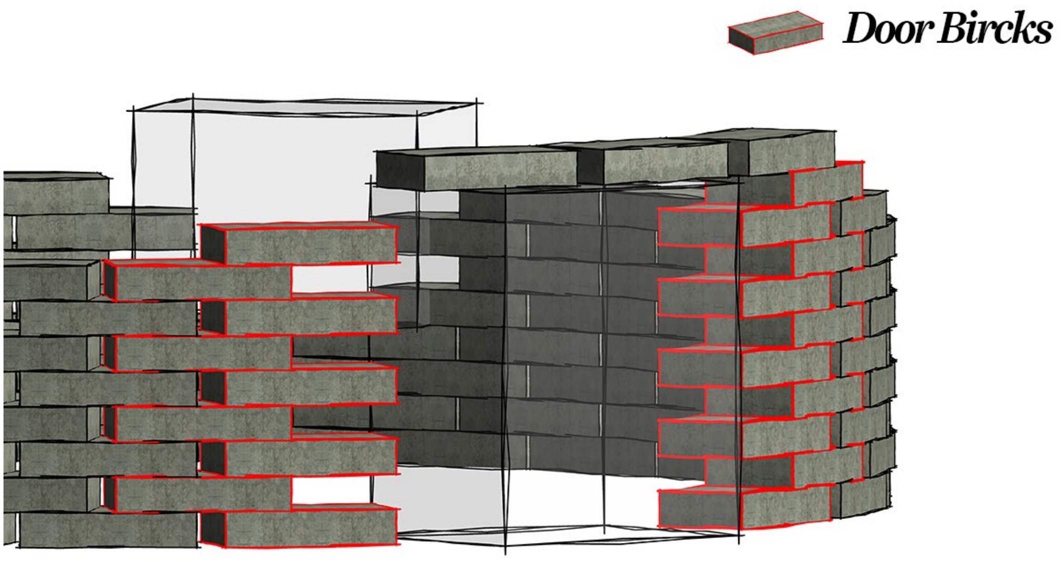


Figure 95: Door Bricks Demonstration

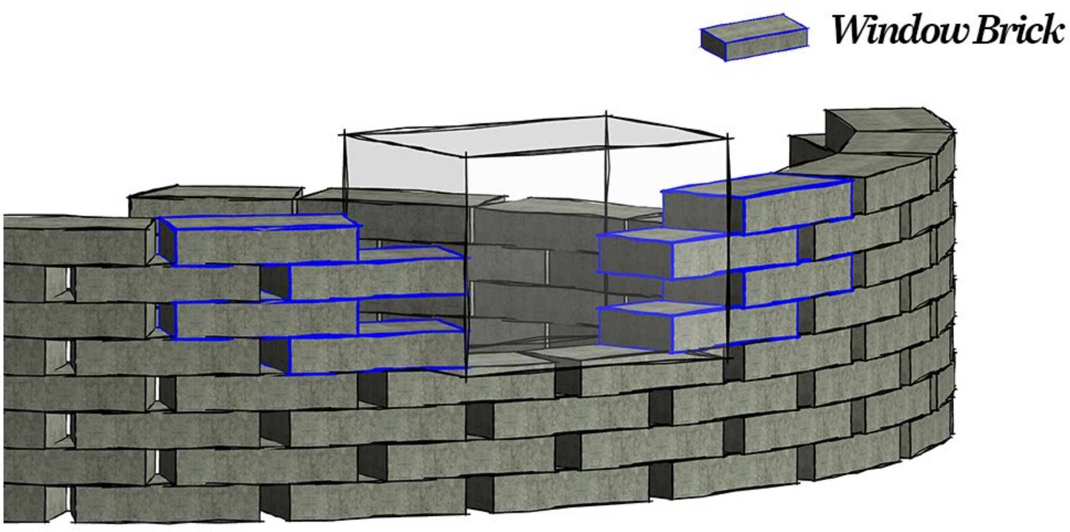


Figure 96: Window Brick Demonstration

Table 26: Brick Observable specific parameters for Dome Phase

Phase 5 Dome

Observation		Explanation	Norm.	*Type*
Variable 5	*Brick Radius*	Current Radius of the level on which this observable brick was placed.	Eq 19	float
Variable 6	*Adjacent Angle*	The angle between the observable brick and the closest brick on the "right" side of it, given that there are	Eq 20	float /int

		no insertion points between them. This angle is used to check if there are unnecessary gaps between a brick and its adjacent one.		
Variable 7	*Angle between Consecutive Bricks*	Consecutive Brick Definition in Dome phase: Any brick placed on the right or left side of an existing brick. Case the observable brick is consecutive: This parameter will measure the angle between the observable brick and the closest brick to it from the consecutive series. Case the observable brick is not consecutive: This parameter will measure the angle between the observable brick and the previous brick inserted before it.	Eq 21	float

Notes

Eq 19

$$\frac{Current\ Radius}{Default\ Radius}$$

Eq 20

$$Case\ existence\ of\ Adjacent\ Brick: -1\left(\frac{Adjacent\ Angle}{Factor^{*}}\right)$$

Case non-existence of Adjacent Brick: 0

** Changeable factor based on the current level, due to the variation in radius.*

Eq 21

$$Case\ Consecutive\ Bricks: \frac{Angle\ between\ Consective\ Bricks}{Current\ full\ Brick\ Angle\ Factor}$$

$$Case\ non\text{-}consecutive\ Bricks: (-1)\left[\frac{Angle\ between\ Consective\ Bricks}{180}\right]$$

Table 27: Specific Parameters for Door and Windows game objects respectively

Windows and Doors Specific Observations

Observation		Explanation	Norm.	*Type*
Windows:				
Variable 1	Insertion Point Type	It is the type of the insertion point on which the Window was placed	Eq 22	float

Variable 2	Dot Product Window and Wind	The dot product value between: (unitized vector projected on the horizontal plane form the center of the building pointing at the insertion point of the current observable window) and (the prevailing wind unit vector)	[-1,1]	float
Variable 3+	Dot Product Window and other Windows			arra y
	Dot Product Window / Window	The dot product value between 2 unitized vectors projected on the horizontal plane form the center of the building, 1 is pointing at the insertion point of the current observable window while the other is pointing at the insertion point of the "Selected Window" from windows list	[-1,1]	float
Doors				
Variable 1	*Dot Product door and Wind*	The dot product value between: (unitized vector projected on the horizontal plane form the center of the building pointing at the insertion point of the door) and (the prevailing wind unit vector)	[-1,1]	float
Notes				
Eq 22	$\frac{\mathit{Insertion\ Point\ Type}\ (\mathit{enum\ to\ int})}{\text{Total Number of Point Types in the Game } (4-5)}$			

2.8.6. Goals and Rewards

In reinforcement learning, the reward is a signal that the agent has done something right. While a penalty, or a negative value reward, is a signal that the agent has done something wrong. The PPO reinforcement learning algorithm works by optimizing the choices an agent makes such that the agent earns the highest cumulative reward over time. The better the reward mechanism, the better the agent will learn [78].

As mentioned before in UV, the rewards are the main channel of communication between the teacher and the agent student that learns based on the rewards received based on its behavior. Therefore, UV provides the designer, which in this case is the teacher setting the rules, a trial environment for experimenting with different rewards and goals before settling on the appropriate reward system for optimizing the training process of the agents.

It is common practice that the agent receives a positive reward when it completes a certain task during the episode, and a maximum reward value on episode completion. The same concept goes for penalties, where the agent gets the maximum negative penalty value when it doesn't achieve the final goal. This event could happen after completion of all tasks but with a poor performance or when the agent does a fatal error that it becomes not possible to complete the episode till the end.

In RV, we identified 6 game termination cases where the game is ended, and the agent receives the maximum reward or penalty. Each time after the execution of step 3 Action, a function named `EndGameInspection()` is executed. This method checks whether any of the game ending scenarios has been reached or not. If one of these events was triggered, the method freezes the game and signals that the game is finished, otherwise the game is normally resumed till the next time it reaches step 3 action and the cycle is repeated.

On signaling that the game is finished, a new method named `ConstitutionLawApplication()`is being called. This method is named in that way because it conveys a rule-based reward system. These rules are enforced based on the game termination case it has ended with. Table xx provides a scheme for the regular end game check that is executed every time step 3 action is implemented.

This scheme generalizes the 6 cases into 4 main scenarios. The first 3 namely "Goal Accomplished", "All Building Completed" and "Brick Collapse" are referring to exact scenarios happening, while in the last scenario in the scheme "Fatal Design Error" conveys 3 possible scenarios. These cases are related to not accomplishing design goals in either Door Placement, Window Placement or Brick Placement.

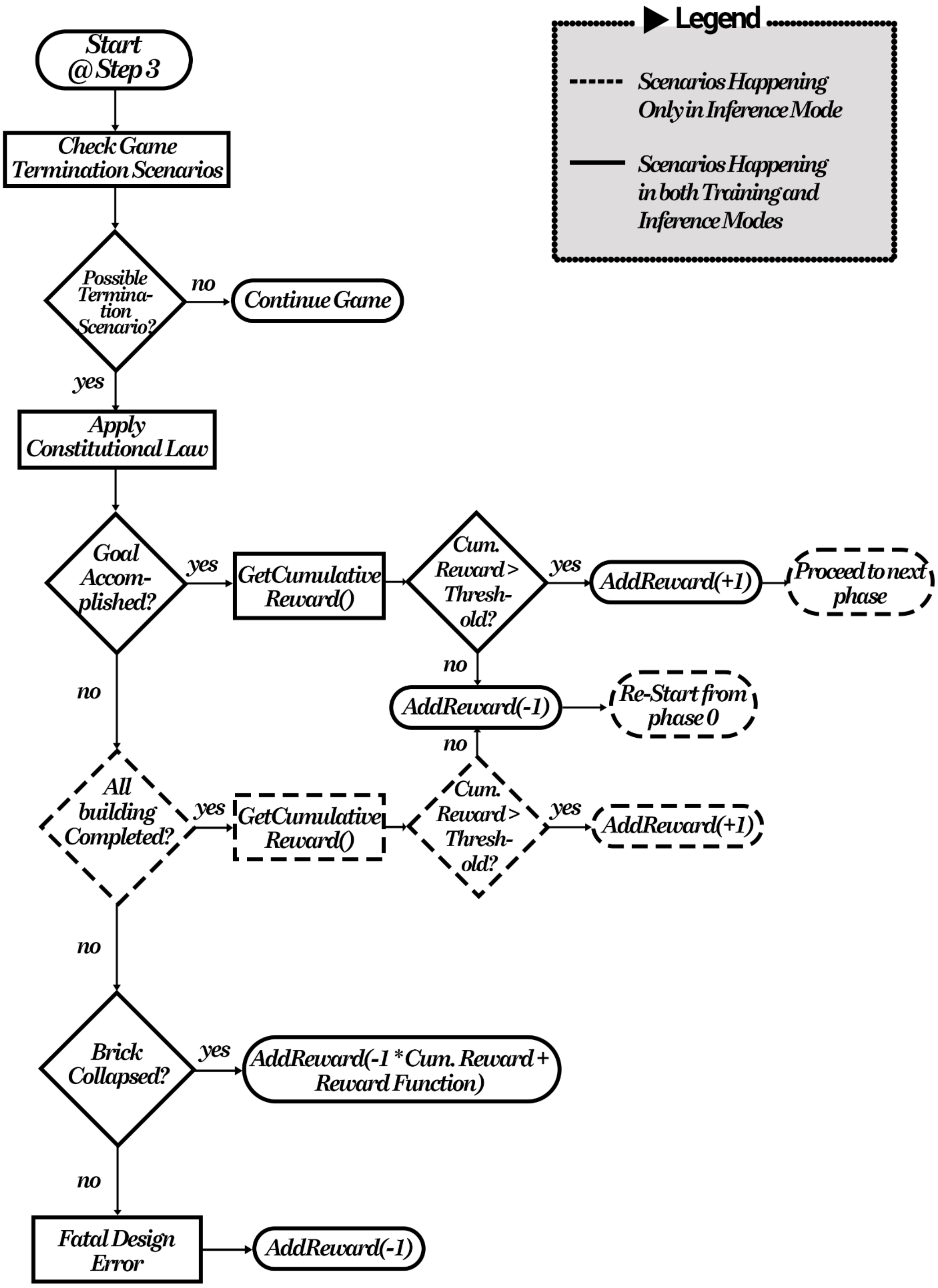


Figure 97: Flowchart for the game termination step executed every Step 3 Action

The following table explains each of these 6 game ending scenarios. It indicates how the game proceeds afterwards in both Training and Inference Modes. It also demonstrates the calculation of the final reward received by the agent as per each scenario.

Table 28: Game Termination Cases and the Corresponding Final Rewards

General Game Termination Cases:

Game Termination Case	Reward	*Explanation*
Goal Accomplished	Eq A	*It is achieved when the goal required from the Agent is achieved in a certain episode.* ***Inference Mode:*** *The Episode is completed, and the next phase starts* ***Training Mode:*** *The Episode is completed, and the agent is instantiated in the same phase again for training.* ***Deployed In Phase(s):*** *Phases (1,2,3,4)*
All Building Completed	Eq A	*It is achieved when the whole pavilion is completed. It is only applicable in Inference Phase as during training, each phase it trained separately.* ***Deployed In Phase(s):*** *Dome Creation Phase (Phase 5) and Only in Inference Mode.*
Brick Collapse	Eq B	*It occurs whenever any brick falls off the structure and hits the ground or when the tilting value of a certain brick falls below the stated threshold.* *The Episode is* ***Inference Mode:*** *The Episode is terminated and the whole structure is destroyed. Hence the game starts again from first phase.* ***Training Mode:*** *The Episode is terminated, and the agent is*

			instantiated in the same phase again for training. ***Deployed In Phase(s):*** *All Brick Phases (Phases 2, 4, 5)*
Design Error		*Eq C*	*It occurs whenever a criterion related to the design goals has been breached or poorly met. It may occur in 3 different scenarios.* ***Inference Mode:*** *The Episode is terminated and the whole structure is incomplete. Hence the game starts again from the first phase.* ***Training Mode:*** *The Episode is terminated, and the agent is instantiated in the same phase again for training.*
	Design Brick Goal Not Accomplished		*It occurs when a design criterion is not met during brick placement.* ***Deployed In Phase(s):*** *All Brick Phases (Phases 2, 4, 5)* *Example Case 1: Window Bricks (Phase 4)* *Number of bricks on the highest level has not exceeded a certain threshold** *Example Case 2: Dome Bricks (Phase 4)* *Number of large, void, unnecessary gaps between bricks exceeds a certain threshold**
	Design Door Goal Not Accomplished		*Occurs when final placement of the door is not far enough from prevailing wind direction.* ***Deployed In Phase(s):*** *Door Placement (Phase 1)*
	Design Window Goal Not Accomplished		*Occurs when final placement of the windows configuration is not entirely benefiting from Cross and Stack Ventilation*

	Deployed In Phase(s): *Window Placement (Phase 3)*

Notes

Eq A $\text{Reward} += 1$

Eq B *Step1: Calculate Excessive Height Penalty* $= -1 - 0.1 * \dfrac{Heighest\ Height\ Reached_{relative\ current\ phase} - Height\ of\ fully\ completed\ level}{Brick\ Height}$

Step2: Calculate Final Reward:

$$Reward += -1 * |Cumulative\ Reward| + Excessive\ height\ Penalty$$

Eq C Case Phase 5 and Gaps Count > Threshold:

$$Reward += -1 * Cumulative\ Reward$$

All other Cases:

$$Reward += -1$$

On the other hand, the following table provides all the possible rewards that the agent can get throughout each phase.

Table 29: Various Possible in Game Rewards

In Game Agent Reward Functions

Reward Trigger Behavior	Reward Value	*Notes*
Phase 1:		
Door Insertion Position Reward	Eq 1	See Table 5
Phase 2:		
Consecutive Brick Placement	Eq 2	See Table 20
Non-Continuous Brick	Eq 3	
Far Brick	Eq 4	See Table 5
Phase 3:		
Windows Instance Evaluation	Eq 5	Evaluation of a window placement directly

Windows Interim Evaluation	Eq 6	Evaluation of Cross Ventilation After the total number of Windows Required is Achieved.
No Available Position for Window Penalty	Table 30	When there is no more room for additional window opening and the existing number of windows hasn’t reached the total required target.
Phase 4:		
Consecutive Brick Placement	Eq 2	
Non-Continuous Brick	Eq 3	
Far Brick	Eq 4	
Counter Bricks on Recent Finished Level	Eq 7	Checks that the number of bricks at each level has surpassed a certain threshold. The function is called once a level is completed.
Phase 5:		
Consecutive Brick Placement	Eq 2	
Gap Between Bricks	Eq 8	
Far Brick	Eq 9	
High Brick	Eq 10	
Star Point	Table 30	optional reward that encourages the use of Bricks inserted at Star Points

Notes

Eq 1 $Step1: Dot\ Product\ \begin{pmatrix} Door\ Forward\ nit\ Vector \\ and\ Wind\ Unit\ Vector \end{pmatrix}$

Case Dot Product < Threshold:

$$Reward = (1 - dotProduct) * Door\ Away\ from\ Wind\ Effect\ Reward$$

Case Dot Product > Threshold:

$$Reward = (-1) * (dotProduct) * Door\ Away\ from\ Wind\ Effect\ Reward$$

Eq 2

$$Reward = Continuous\ Brick\ Reward * Count\ of\ Consecutive\ Bricks$$

Eq 3

$$Reward = Bricks\ Not\ In\ Sequence\ Counter * Non\ Continuous\ Brick\ Penalty$$

Eq 4

$$Reward = Lontana\ Penalty * 10 * \frac{1}{Angle\ between\ Bricks\ /\ 180}$$

Eq 5

$$Step1: Dot\ Product\ \left(\begin{matrix} Window\ Forward\ unit\ Vector \\ and\ Wind\ Unit\ Vector \end{matrix}\right)$$

$$Reward = (dotProduct) * Stack\ Ventilation\ Max\ Reward$$

$$Add\ Dot\ Product\ to\ windows\ \text{"}Dot\ List\text{"}$$

Eq 6 *Step1: From windows "Dot List": Pick the Maximum Value*

Step2: Pick furthest window from window with Max Value in Dot List

Step3: Calculate Dot Product between Both Windows

Step4:

$$Reward = Max\ Dot\ Product\ in\ List * Dot\ Product\ 2\ Windows * Cross\ Ventilation\ Max\ Reward$$

Eq 7 *Case Count of Bricks on Recent Finished Level < Threshold:*

Trigger a game termination Scenario.

Case Count of Bricks on Recent Finished Level ≥ Threshold:

$$Reward = Counter\ Bricks\ Exceeded\ Reward$$

Eq 8

$$Reward = Gap\ Between\ Bricks\ Penalty * \frac{gap\ angle}{full\ brick\ angle} * Counter\ for\ number\ of\ gaps\ currently\ in\ the\ game$$

Eq 9 *Case Angle Between Bricks > Threshold:*

$$Reward = \frac{Angle\ Between\ Bricks}{180} * lontanaPenalty * 2 * Count\ of\ Non\ Consecutive\ Bricks$$

Eq 10 *Case Brick height relative to current lowest incomplete level > Threshold:*

$$Reward = Count\ of\ Consecutive\ Bricks\ over\ Each\ other^2 * High\ Brick\ Penalty * 2$$

Finally, the previously mentioned rewards calculation, are calculated based on the input of the designer to the following rewards parameters:

Table 30: Rewards Parameters and the corresponding Values used for Training Agents in RV

Input Reward Parameters	Initial Reward Value
Continuous Brick Reward	+0.001
Non-Continuous Brick Penalty	-0.002
Gap Between Bricks Penalty	-0.1*
Star Point (optional)	+0.01
Lontana Penalty	-0.001
High Brick Penalty	-0.0005
Sturdy Brick Level	+0.1
Maximum Number of Gaps Count	10
Cross Ventilation Max Reward	+0.1
Stack Ventilation Max Reward	+0.1
Door Away from Wind Effect Reward	+0.1
Counter Bricks Exceeded Reward	+0.1

2.8.7. Curriculum Learning and Hyperparameters

- *Curriculum Learning*

Curriculum learning is a way of decomposing a difficult task into subtasks starting from a simple level and increasing difficulty till reaching exactly what is required in the main task. This idea has been around for a long time, and it is how we humans typically learn [78]. It is a form of prerequisites that a student needs to learn one after the other so he/she can reach the hardest lesson with all the previous courses in mind. The same principle can be applied to machine learning, where training on easier tasks can provide a scaffolding for harder tasks in the future.

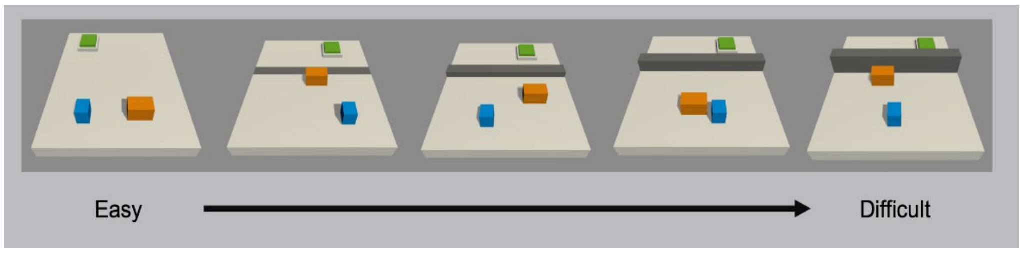


Figure 98: Demonstration of a hypothetical curriculum training scenario in which a progressively taller wall obstructs the path to the goal by ML-Agents [78].

The following figure shows a scheme of how the Phase of Dome Bricks has been divided into curriculum lessons.

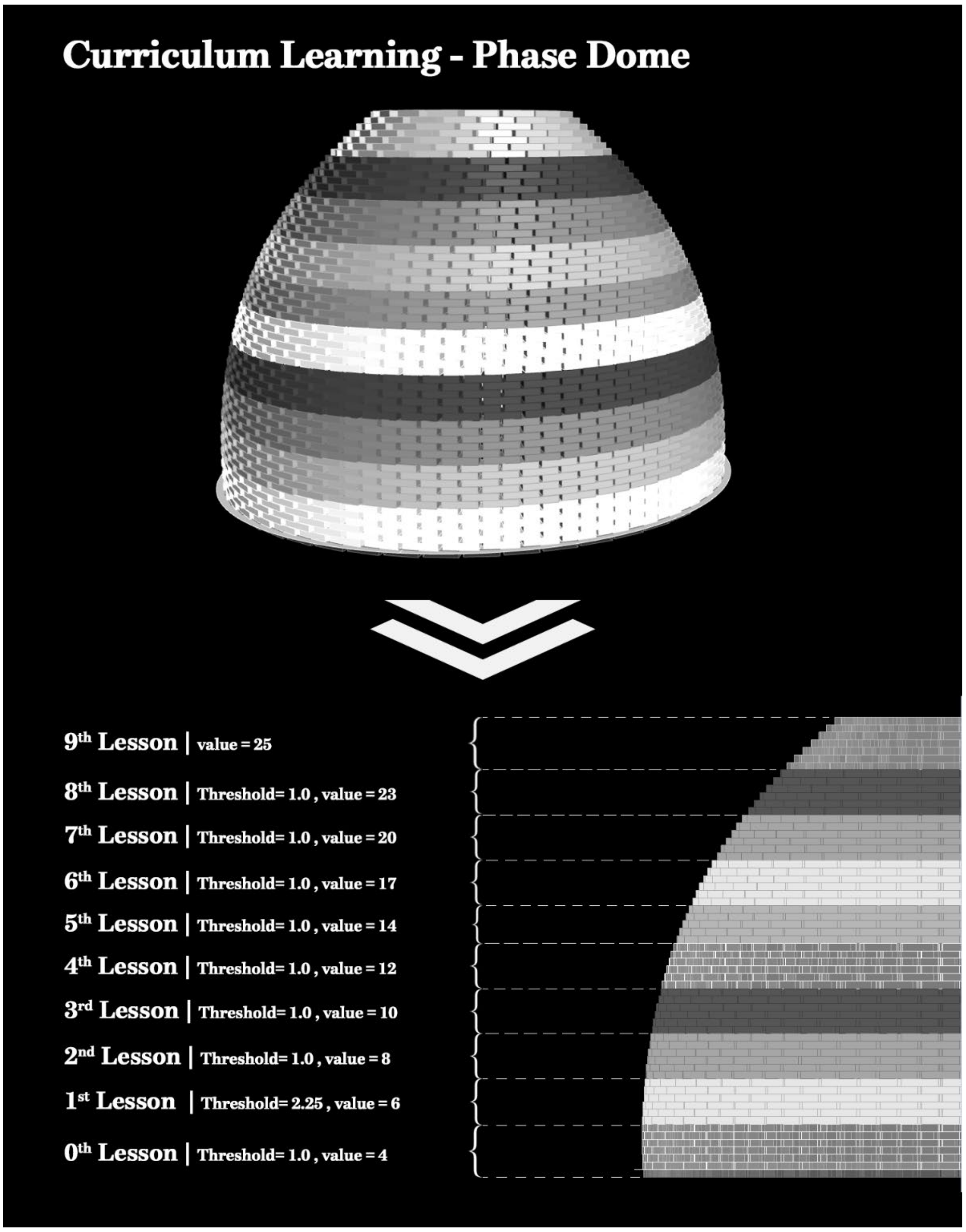

Curriculum Learning - Phase Dome
9th Lesson | value = 25
8th Lesson | Threshold= 1.0 , value = 23
7th Lesson | Threshold= 1.0 , value = 20
6th Lesson | Threshold= 1.0 , value = 17
5th Lesson | Threshold= 1.0 , value = 14
4th Lesson | Threshold= 1.0 , value = 12
3rd Lesson | Threshold= 1.0 , value = 10
2nd Lesson | Threshold= 1.0 , value = 8
1st Lesson | Threshold= 2.25 , value = 6
0th Lesson | Threshold= 1.0 , value = 4

Figure 99: Curriculum Learning in Phase 5 (Dome Creation)

For each "lesson" there are some parameters that needs to be defined, for example:

- `measure`: What to measure learning progress, and advancement in lessons by.
    - `reward`: Uses as a measure the received reward.
- `thresholds`: (float array) - Points in value of measure where lesson should be increased. In other words, since we decided that the measure

of success of a lesson is through calculating the reward, the threshold is the value that is needed to be reached so that the agent can be promoted to the next lesson.

- `min_lesson_length` (int) - The minimum number of episodes that should be completed before the lesson can change. If measure is set to `reward`, the average cumulative reward of the last `min_lesson_length` episodes will be used to determine if the lesson should change. Must be nonnegative.
- `Value:` Value of the environment parameter selected to have changing values based on the Lesson Number. In our case was the *maximum number of levels allowed* to be reached in each lesson.

Curriculum learning was also used in Window Bricks phase, where the phase was divided into 3 lessons. Each lesson had a maximum level constraint that increased by proceeding to the succeeding lesson. The first lesson had a maximum level (4) and reward threshold of 1.5, the second lesson had a maximum level of (7) with reward threshold of 2.8 and the final lesson had a maximum constraint on the level of bricks equals to 8. It has to be noted that in the final lesson there is normally no threshold needed since it is already the final lesson.

- *Hyperparameters*

The following table demonstrates the final hyperparameters used to train the agents in each of the 5 phases. These hyperparameters are the ones related to training configurations and doesn't include any environment parameters as we already discussed that part in the curriculum sub-section above.

Table 31: Comparison of Configuration file Hyperparameters for Different Phases

Hyper Parameter	Phases				
	Phase 1 (Door)	Phase 3 (Windows)	Phase 4*	Phase 5**	
				Session 1	Session 2
Behaviors:	Robuilder				
trainer_type:	PPO				
Hyperparameters:					
batch_size:	512	512	512	1024	1024
buffer_size	4096	4096	10240	20480	40960
learning_rate	0.0003				

beta	0.005	0.005	0.01	0.01	0.001
epsilon	0.2	0.2	0.2	0.2	0.2
lambd	0.9	0.9	0.9	0.9	0.9
num_epoch:	3	3	3	3	3
learning_rate_schedule:	linear	linear	linear	linear	linear
network_settings:					
normalize:	FALSE	FALSE	FALSE	FALSE	FALSE
hidden_units:	512	512	512	512	512
num_layers:	2	2	3	3	
vis_encode_type:	simple	simple	simple	simple	simple
reward_signals:					
extrinsic:					
gamma:	0.99	0.99	0.99	0.99	0.99
strength:	1	1	1	1	1
network settings					
normalize	FALSE	FALSE	FALSE	FALSE	FALSE
hidden_units	512	512	512	512	
num_layers	2	2	3	3	
max_steps:	150000	500000	1E+08	3.6E+08	10000000
time_horizon:	3	9	512	1024	
threaded	TRUE	TRUE	TRUE	TRUE	TRUE

Notes

Phase 4*	During Training of Phase 4 (Window Bricks), the trained outcome was working perfectly in Phase 2 (Base Bricks) as well. Therefore, the training outcome in this phase was used in inference mode of both phases. (See sub-section 3.9.4)
Phase 5**	Phase 5 was trained along 2 consecutive sessions, where session 2 was initialized from where session 1 has ended.

2.8.8. Training Vs Inference Mode

Once training concludes, the learned policy for each phase can be exported as a model file. Then during inference phase, the environment will still continue to generate observation, but instead of being sent to Python API, they will be fed directly into the (internal, embedded) model to generate the optimal action for each medic to take at every point in time.

Since the decision to divide the pavilion into 5 separate phases, it meant that we have 5 separate models that needs to be trained and the same number needed to be run consecutively during the inference phase. For training each episode, in theory, we need to have all the previous phases existing in the scene so that the bricks, or other units, placed in this phase will be placed on top of the previous phases.

However, to speed up the learning process, it was decided to create a sort of "fake" base for each phase that relies on having an existing structure to start from. The reason for doing this is to reduce the computation power and time wasted at the beginning of each episode to place the existing structure before the start of the current phase training.

Accordingly, the following pictures represent the differences between the training mode and inference mode in every phase that relied on the structure of previous phases. In most of the cases where the previous "Brick" structure was required, a single layer of bricks was placed at the beginning of each episode. This brick layer was altered in position randomly every episode start to ensure that the agent wouldn't get stuck if during inference mode the below layer of an existing structure doesn't exactly conform to the one it was used to train on.

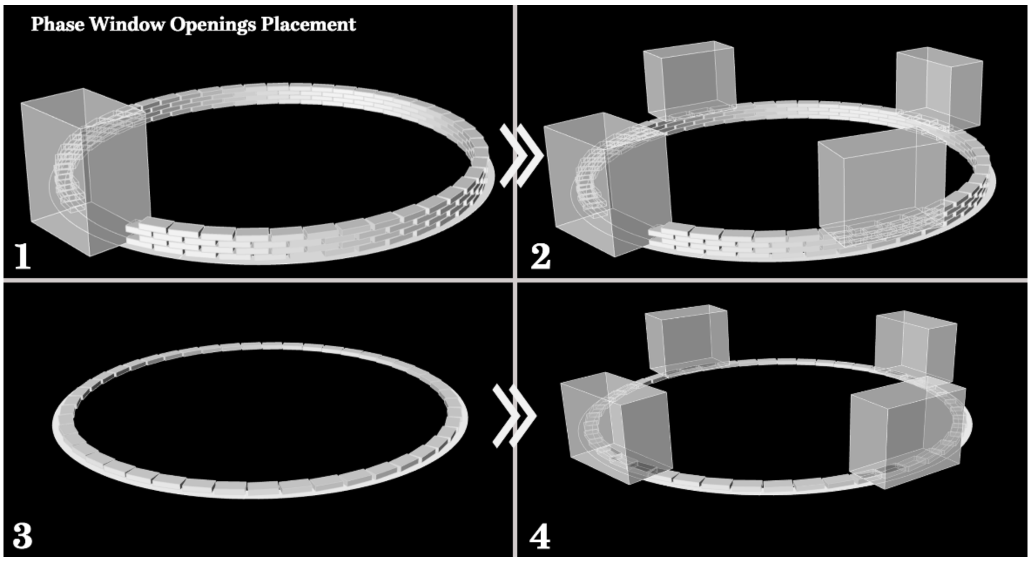


Figure 100: Phase Window Openings Placement, (1-2): Start and Finish in Inference Mode (3-4): Start and Finish in Training Mode

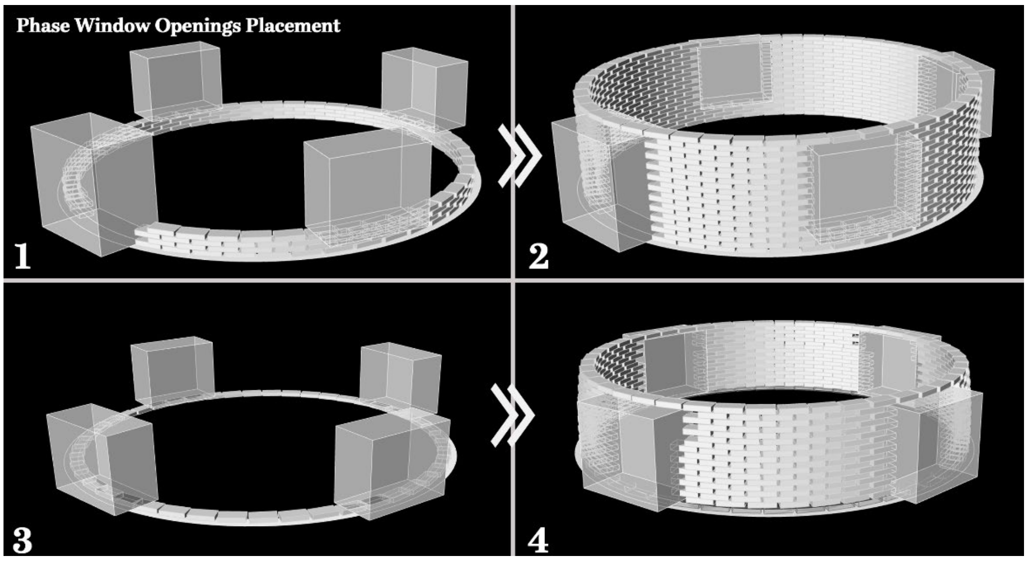


Figure 101: Phase Window Openings Placement, (1-2): Start and Finish in Inference Mode (3-4): Start and Finish in Training Mode

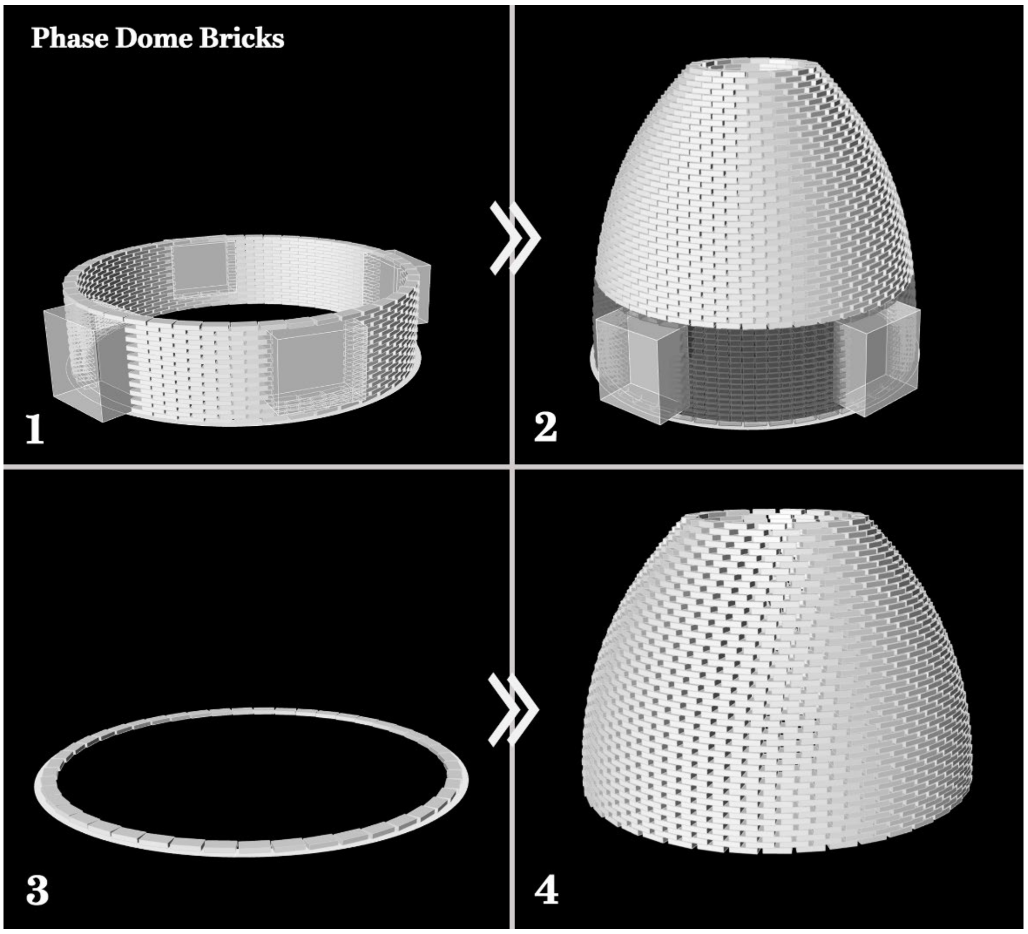


Figure 102: Phase Window Openings Placement, (1-2): Start and Finish in Inference Mode (3-4): Start and Finish in Training Mode

2.9. Discussion of Results

In this section we first present the results of the training sessions of the phases. These results are demonstrated using the training statistics and a brief discussion follows the statistics of each phase. It has to be noted that regarding Phase 2 (Base Bricks), its results were not mentioned in this section as eventually the trained model for phase 4 (Window Bricks), was proven to be efficient for being used as the main brain in inference mode for both phases.

After discussion of the training results and showing images of the training process, the section ends with illustrating the compilation of the trained models together in the inference phase. The last subsection, thus, shows several pictures from the final inference phase as well as discussing the final observations regarding the structure assembly.

2.9.1. General View of Statistics Results

Parameter	Explanation
Environment	
Cumulative Reward	The mean cumulative episode reward over all agents. Should increase during a successful training session.
Lesson	Only interesting when performing curriculum training. Provides at which number of steps a change in the lesson number occurs
Episode Length	The mean length of each episode in the environment for all agents.
Losses	
Policy Loss	The mean loss of the policy function update. Correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session. These values will oscillate during training. Generally, they should be less than 1.0.
Value Loss	The mean loss of the value function update. Correlates to how well the model is able to predict the value of each state. This should decrease during a successful training session. These values will increase as the reward increases, and then should decrease once reward becomes stable.

Policy	
Value Estimate	The mean value estimate for all states visited by the agent. Should increase during a successful training session. These values should increase as the cumulative reward increases. They correspond to how much future reward the agent predicts itself receiving at any given point.
Extrinsic Reward	Represents the rewards defined in the environment, and is enabled by default
Learning Rate	How large a step the training algorithm takes as it searches for the optimal policy. Should decrease over time.
Epsilon	corresponds to the acceptable threshold of divergence between the old and new policies during gradient descent updating. Setting this value small will result in more stable updates but will also slow the training process.
Entropy	How random the decisions of the model are? Should slowly decrease during a successful training process. If it decreases too quickly, the beta hyperparameter should be increased. This corresponds to how random the decisions of a Brain are. This should consistently decrease during training. If it decreases too soon or not at all, beta should be adjusted (when using discrete action space).
Beta	Corresponds to the strength of the entropy regularization, which makes the policy "more random." This ensures that agents properly explore the action space during training. Increasing this will ensure more random actions are taken. This should be adjusted such that the entropy (measurable from TensorBoard) slowly decreases alongside increases in reward. If entropy drops too quickly, increase beta. If entropy drops too slowly, decrease beta.

2.9.2. Phase Door Opening Placement

- *Graphical Statistics on Training behavior*

The following graphs represent summary statistics for "Phase Door Placement"

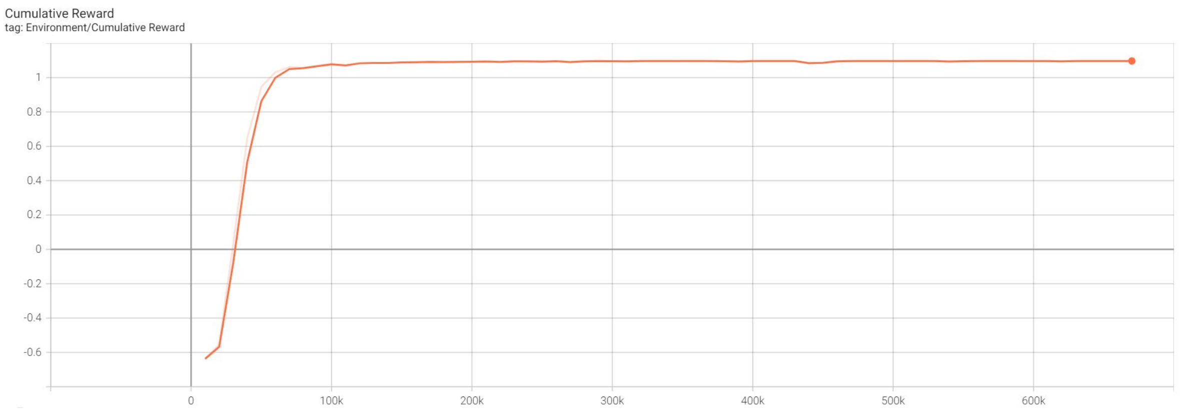


Figure 103: Cumulative Reward for Phase Door Opening Placement

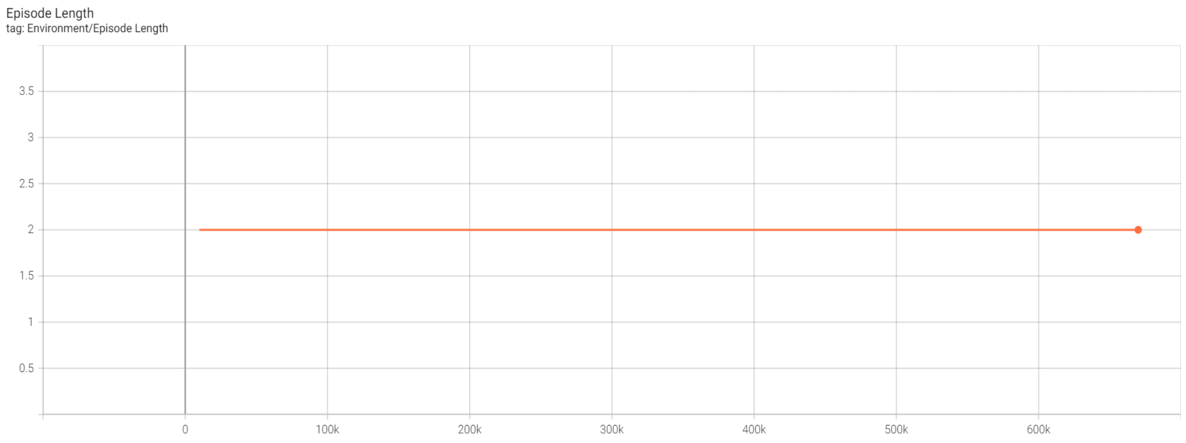
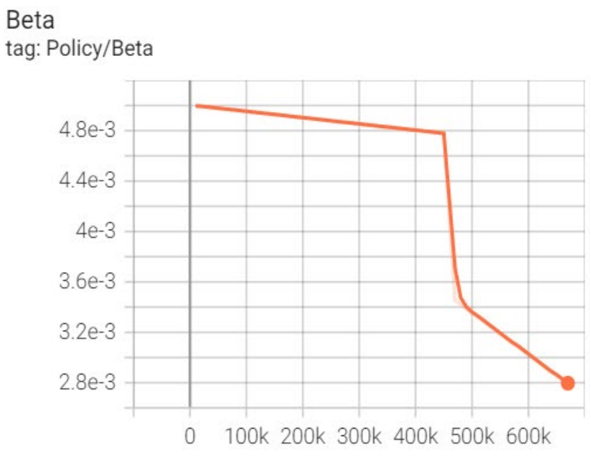


Figure 104: Episode Length per Step for Phase Door Opening Placement



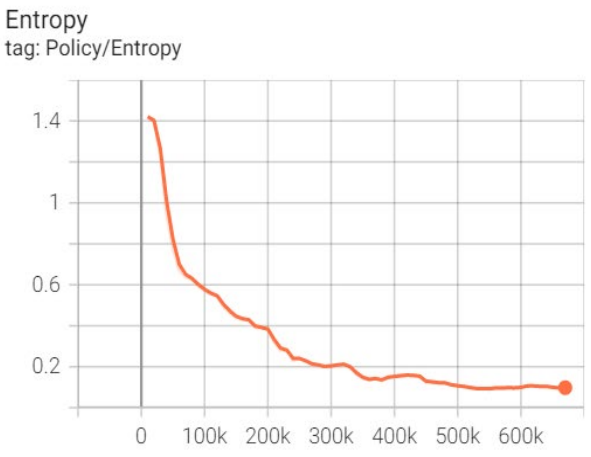


Figure 105: (left) Beta Evolution per Step- (right) Entropy Evolution per Step Phase Door Opening Placement

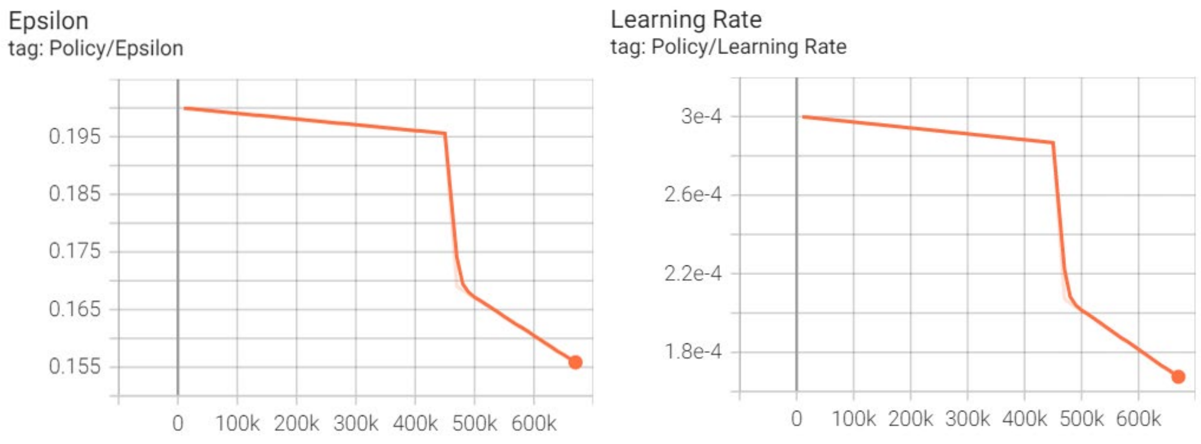


Figure 106: (left) Epsilon Evolution per Step- (right) Learning Rate Evolution per Step Phase Door Opening Placement

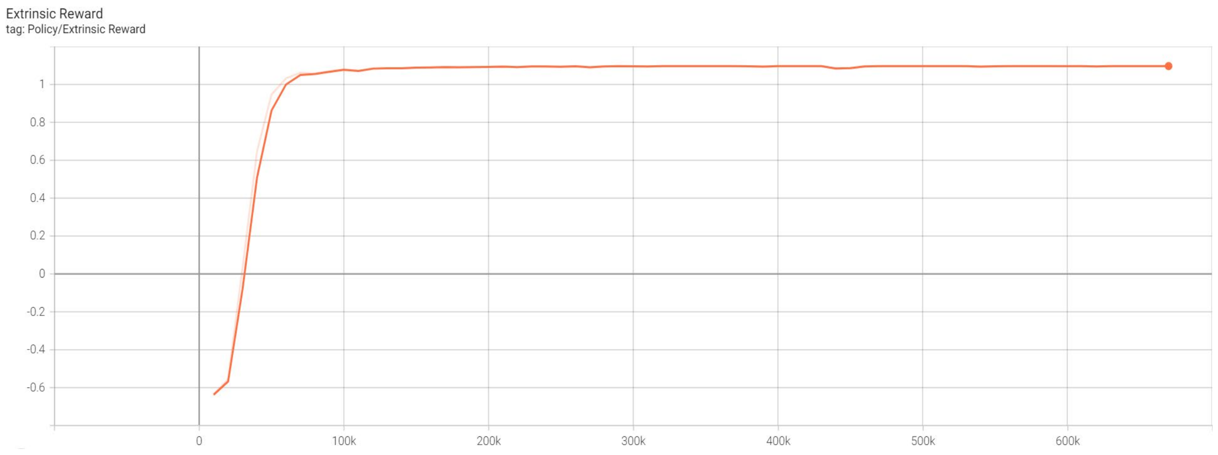


Figure 107: Extrinsic Reward per Step for Phase Door Opening Placement

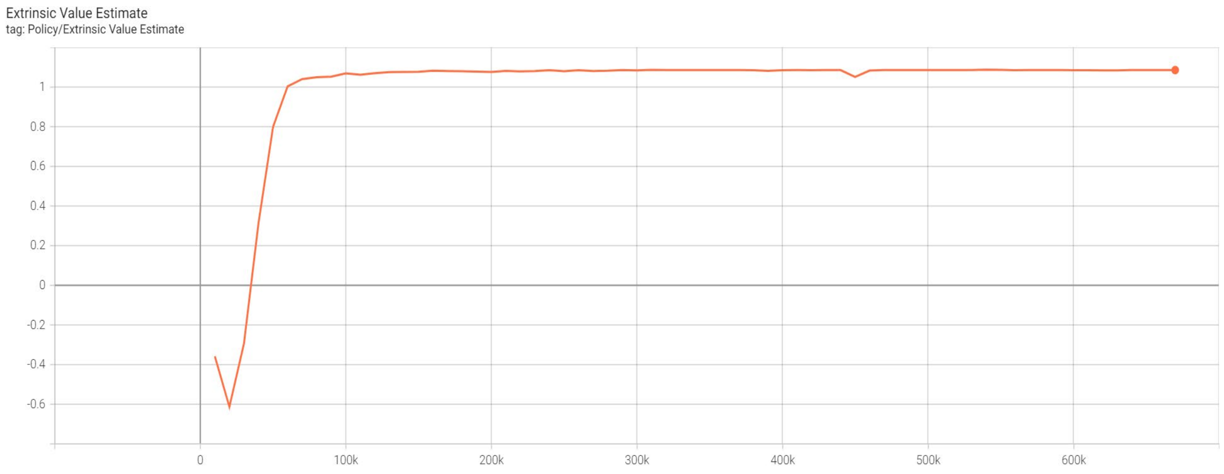


Figure 108: Extrinsic Value Estimate Per Step for Phase Door Opening Placement

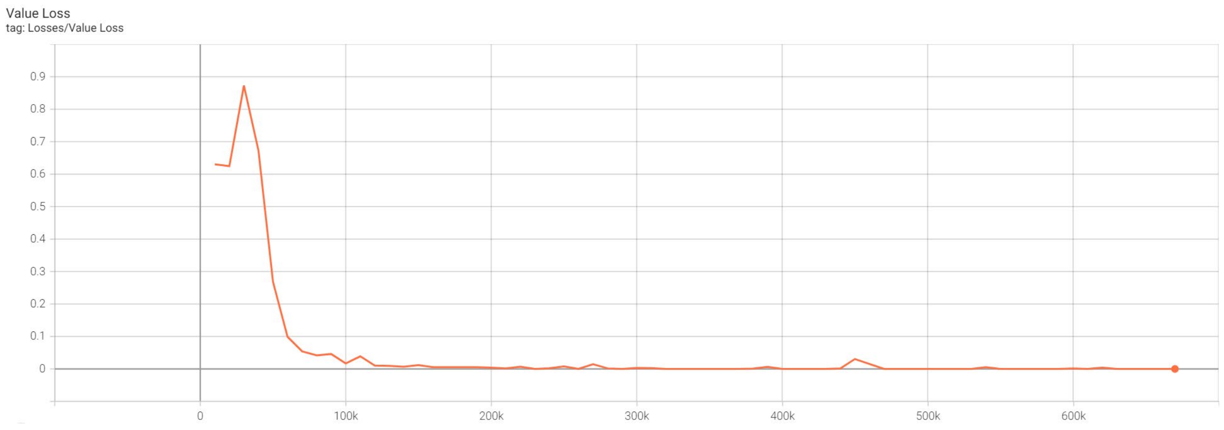


Figure 109: Value Loss per Step for Phase Door Opening Placement

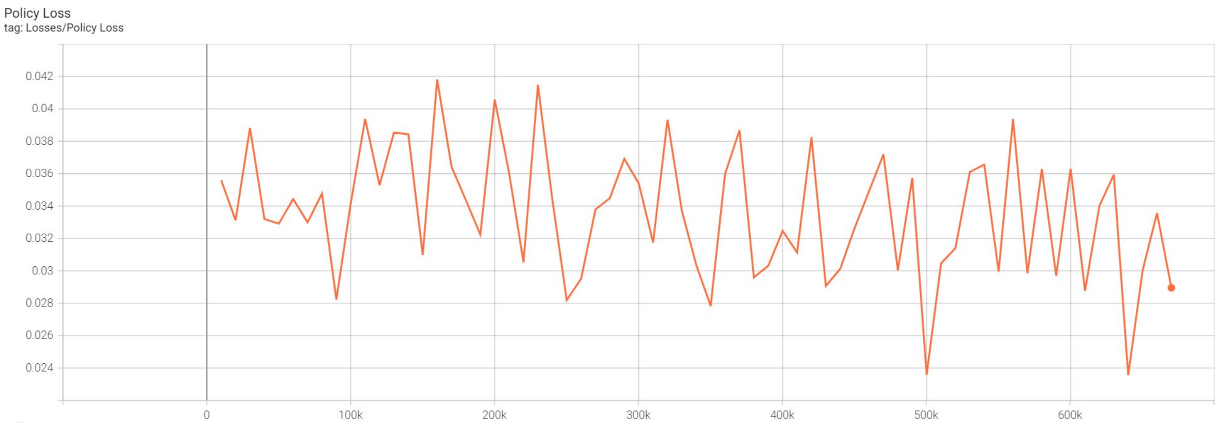


Figure 110: Policy Loss per Step for Phase Door Opening Placement

- *Summary Points of Statistics Output*

By observing the statistical results, it can be concluded the following main points:

- The training maximum cumulative reward reached has been *1.097* at step *670k in 55 minutes* and *3 seconds.*
- The average episode length was 2 steps overall.
- No curriculum learning was used therefore there was only one “lesson” involved with no environment parameters changing throughout the training process.
- Policy loss (the mean loss of the policy function update) values were oscillating throughout the training session with an overall “healthy” downward slope towards the end of training session.
- Entropy was consistently decreasing throughout the training session.
- The chosen option for “Learning rate” hyperparameter was selected to linearly decay on the span of the maximum steps.

- *Discussion of the Training Summary and hyperparameters*

Given the training time was approximately one hour, this is considered the fastest training session (along with windows placement phase). It makes sense as it is the simplest phase, comprising only of the placement of one unit (door opening).

The average range of 2 steps which didn't change throughout the training time, is in indicator that the 3-step action needed to place a unit was considered as a 2-step process by the network. This is because the last action was not counted among the calculated steps during the game since it led directly to game completion.

Regarding hyperparameters chosen, time horizon was set at 3 because the whole episode is technically made of 3 steps. We didn't need to reach the maximum steps of 1500000, as indicated above. Buffer size was set to 4096 which is on the low side of the typical range (2048 - 409600), as there was no need to collect more experiences before updating the model. However, Batch-size was chosen to be 512 (typical range for discrete actions is between 32 - 512) to capture, in each iteration of the gradient descent update, a diversity of game situations.

The use of 12 instances of the "Building Area", ensured that the agent has witnessed several values of wind direction vector. Even in each episode on the same "building area", there was a continuous change in the wind direction by the continuous random generation of its unit vector x and z components. This would ensure that the link between the door opening position and the wind direction is always linked in every possible wind direction.

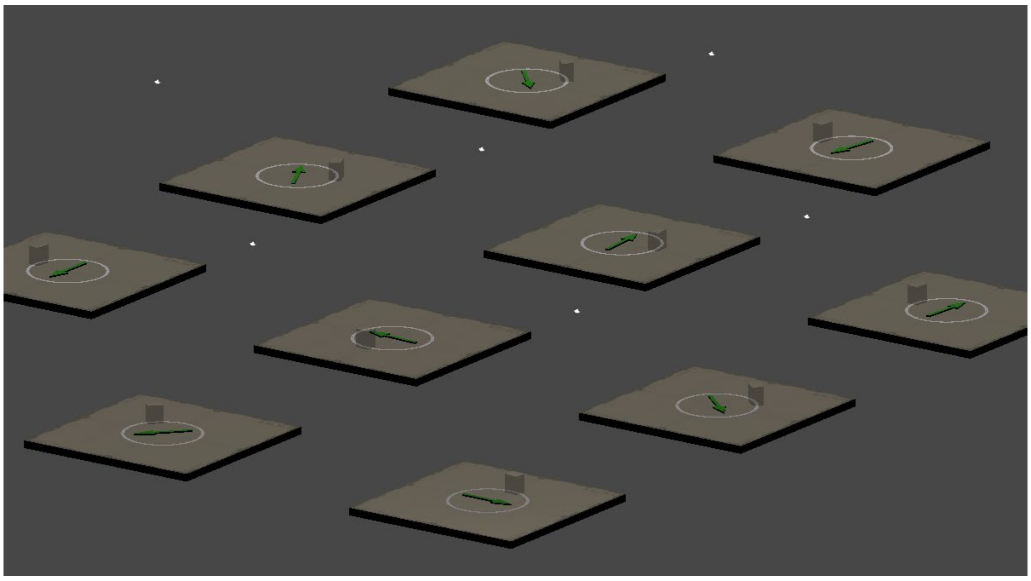


Figure 111: 12 Building Area Instances where in each one the agent is training to properly place a door opening.

2.9.3. Phase Window Openings Placement

The following graphs represent summary statistics for “Phase Windows Placement”

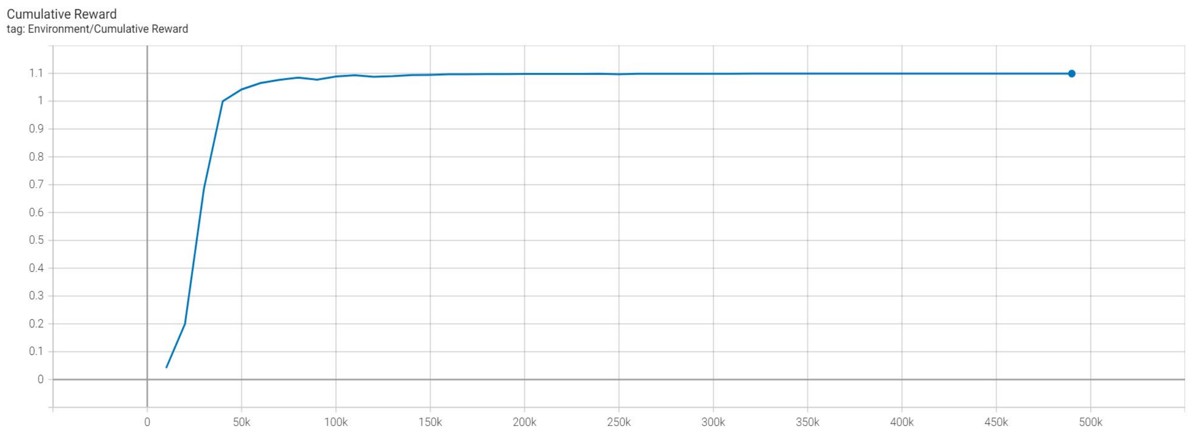


Figure 112: Cumulative Reward for Phase Window Openings Placement

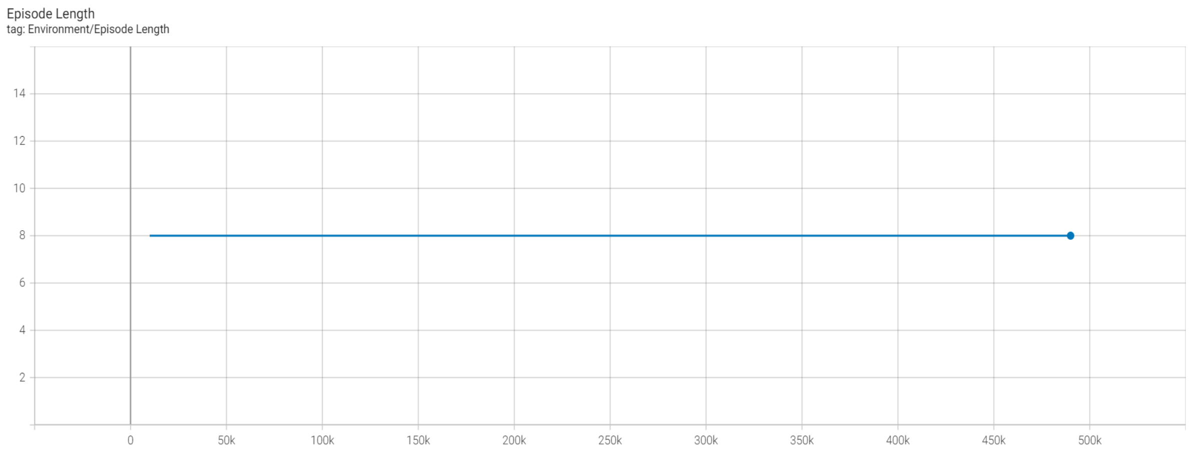


Figure 113: Episode Length per Step for Phase Window Openings Placement

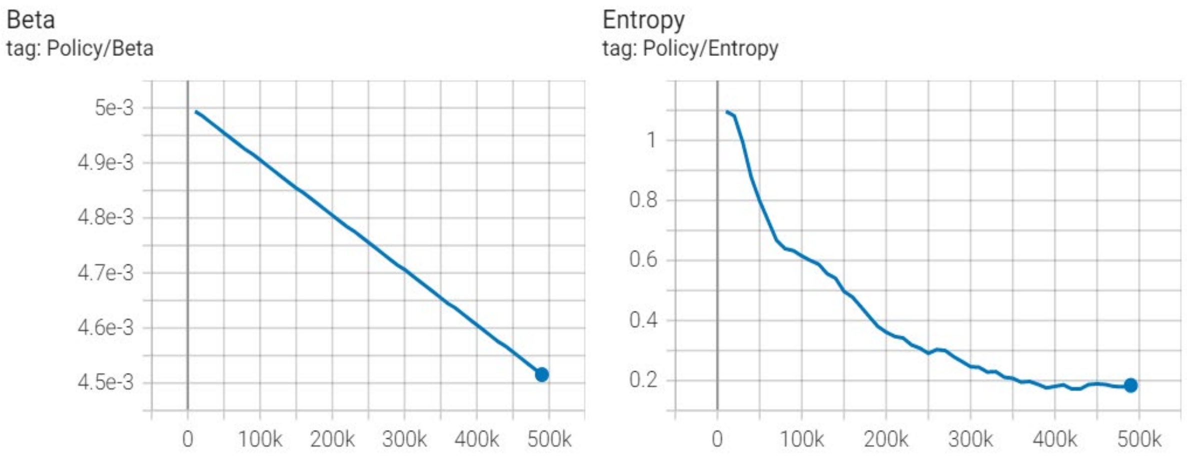
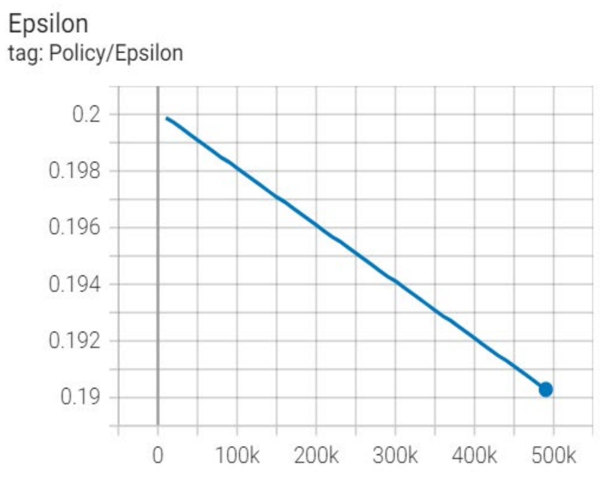


Figure 114: (left) Beta Evolution per Step- (right) Entropy Evolution per Step Phase Window Openings Placement



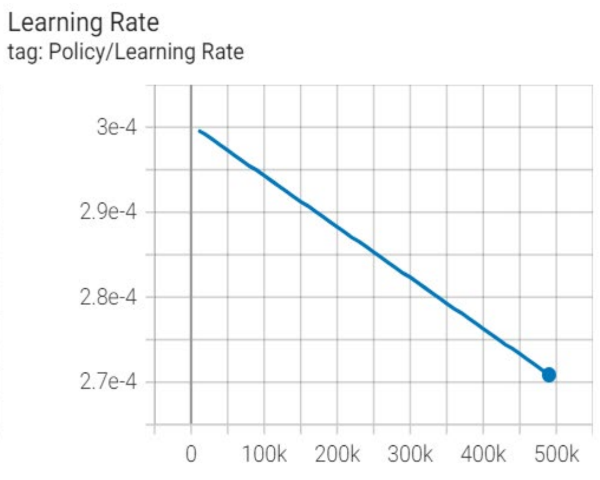


Figure 115: (left) Epsilon Evolution per Step- (right) Learning Rate Evolution per Step Phase Window Openings Placement

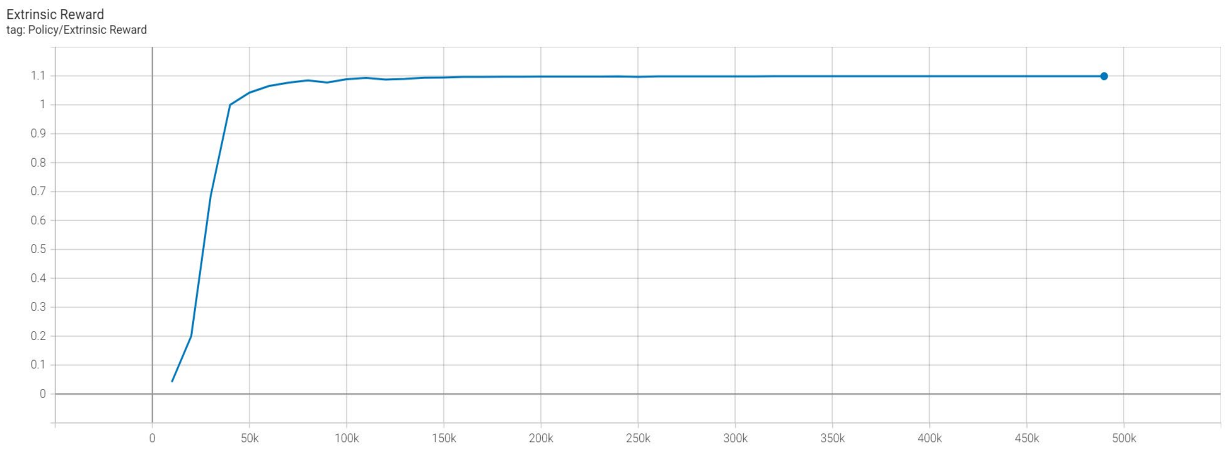


Figure 116: Extrinsic Reward per Step for Phase Window Openings Placement

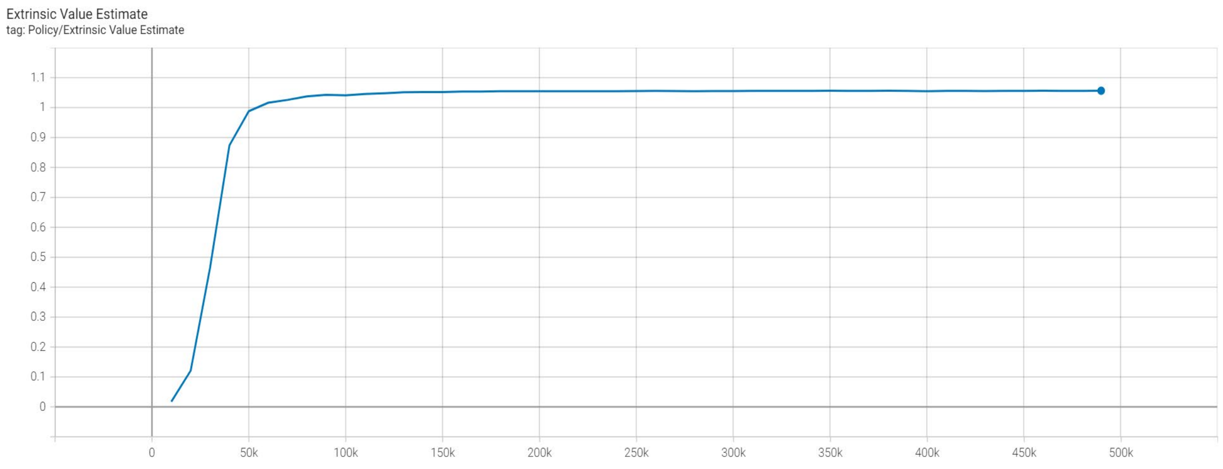


Figure 117: Extrinsic Value Estimate Per Step for Phase Window Openings Placement

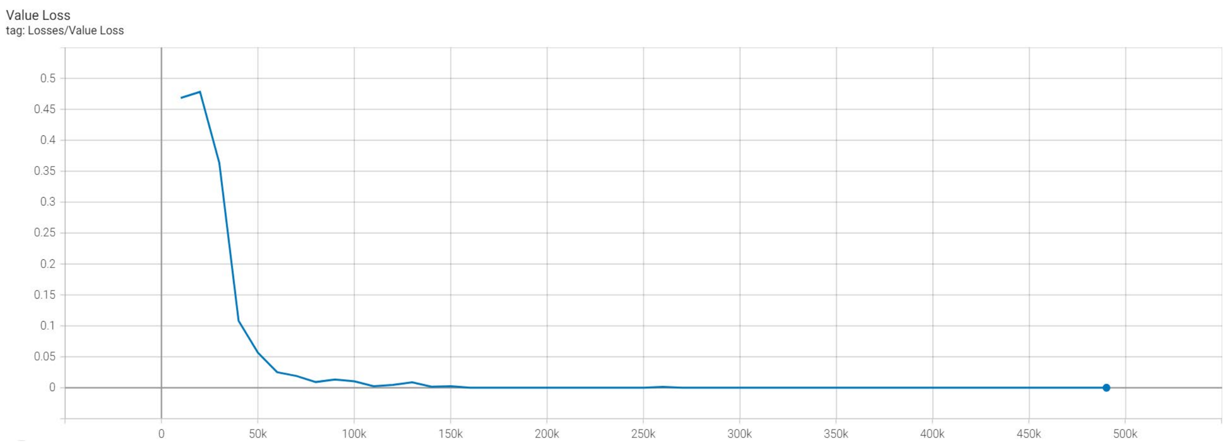


Figure 118: Value Loss per Step for Phase Window Openings Placement

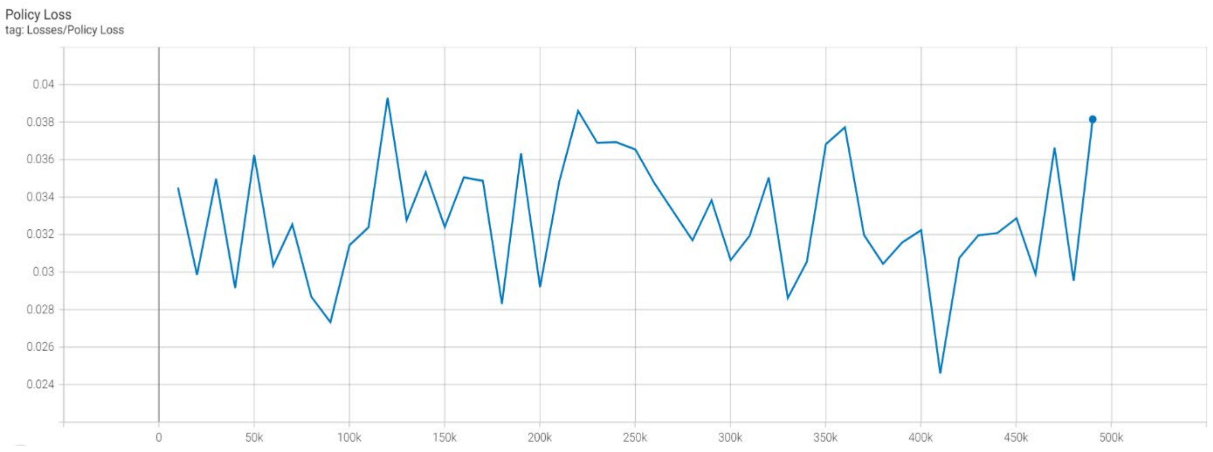


Figure 119: Policy Loss per Step for Phase Window Openings Placement

- *Summary Points of Statistics Output*

By observing the statistical results, it can be concluded the following main points:

- The training maximum cumulative reward reached has been *1.099* at step *490k in 50 minutes* and *12 seconds*.
- The average episode length was 8 steps overall.
- No curriculum learning was used therefore there was only one “lesson” involved with no environment parameters changing throughout the training process.
- Policy loss (the mean loss of the policy function update) values were oscillating throughout the training session with an overall “healthy” downward slope towards the end of training session.
- Entropy was consistently decreasing throughout the training session.
- The chosen option for “Learning rate” hyperparameter was selected to linearly decay on the span of the maximum steps.

- *Discussion of the Training Summary and hyperparameters*

The training for both phases of windows and door opening placement were executed in less than an hour. This is because for window openings the average number of steps was 8 and it involved only the placement of maximum 3 windows.

Both batch size (512) and buffer size (4096) were kept the same as the one used for door placement since the average steps per episode remains still below 10. Therefore, there is no significant change. Only the value for the hyper parameter “time horizon” was adjusted to account for 3 window instances per episode, i.e., 3 window placement * 3 action steps = 9.

The use of 12 instances of the “Building Area”, ensured that the agent has witnessed several values of wind direction vector. Even in each episode on the same “building area”, there was a continuous change in the wind direction by the continuous random generation of its unit vector x and z components. This would imply a better learning and adaptation of the agents to changing wind directions to achieve stack and cross ventilation requirements.

To sum up, both phases for window and door placement had achieved the same results with almost similar hyper parameters. There was no complexity in them that would require more advanced features on the neural network side to be adjusted. The padding of zeros in the vector observation, didn’t affect the performance of the neural network. This padding was done because the number of observations was governed by the brick phases that required a higher number compared to the ones essentially needed in both door and windows placement phases.

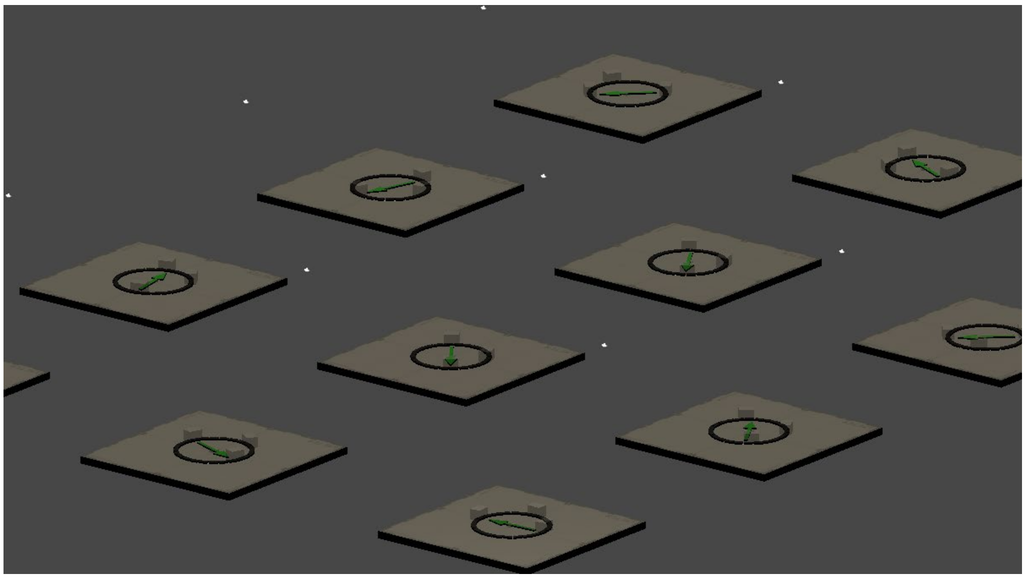


Figure 120: Building Area Instances with varying wind direction (green arrow) where in each one the agent is training to properly place 3 window openings

2.9.4. Phase Window Bricks

The following graphs represent summary statistics for “Phase Window Bricks”

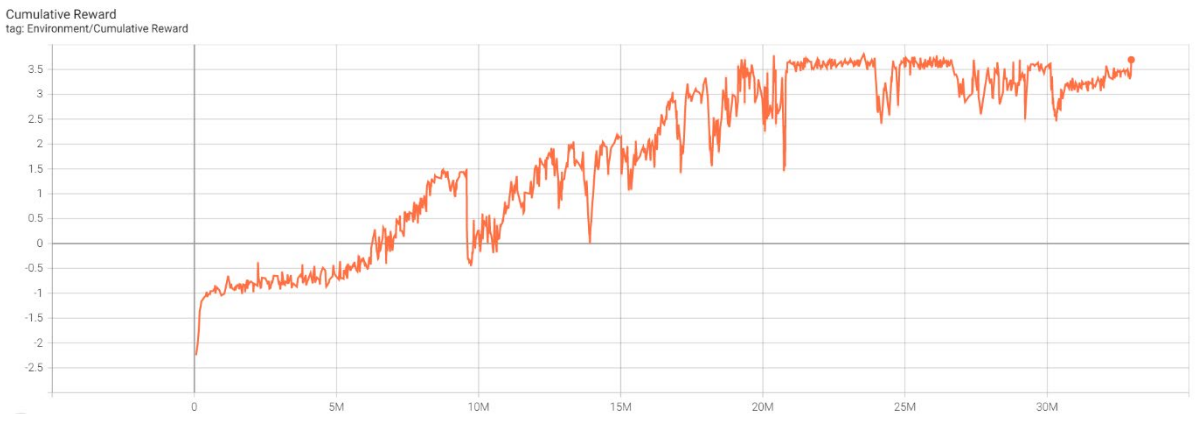


Figure 121: Cumulative Reward for Phase Window Bricks

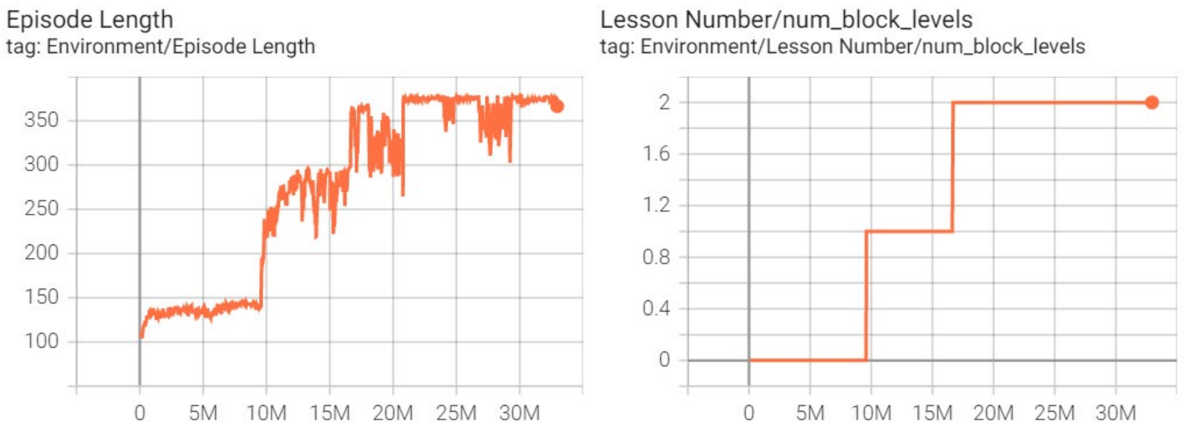


Figure 122: (left) Episode Length per Step – (right) Lesson number Per step for Number of Brick Levels for Phase Window Bricks

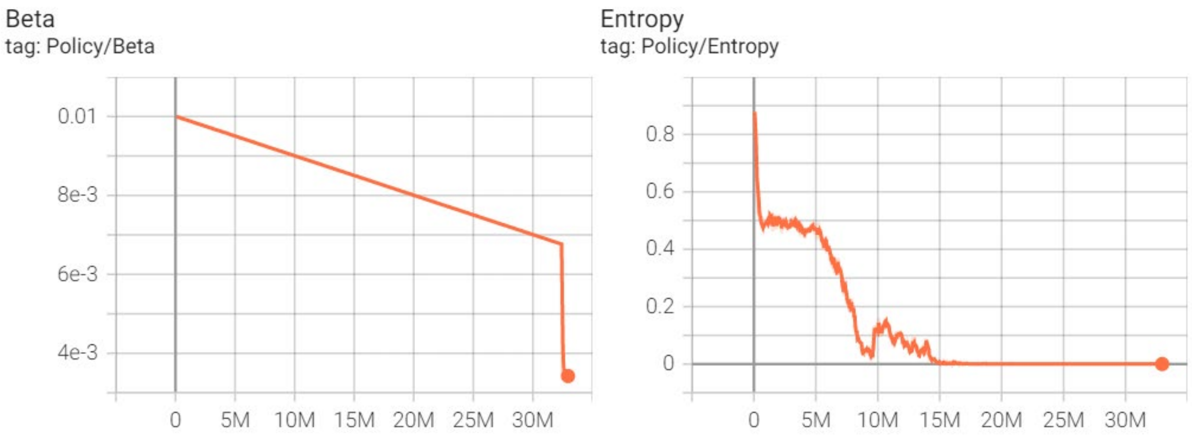


Figure 123: (left) Beta Evolution per Step- (right) Entropy Evolution per Step Phase Window Bricks

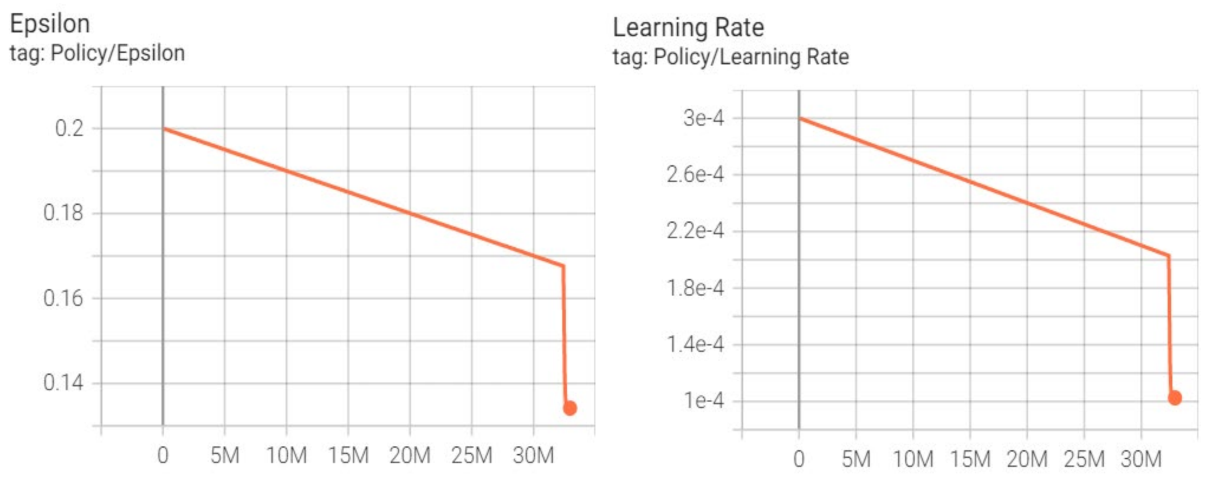


Figure 124: (left) Epsilon Evolution per Step- (right) Learning Rate Evolution per Step Phase Window Bricks

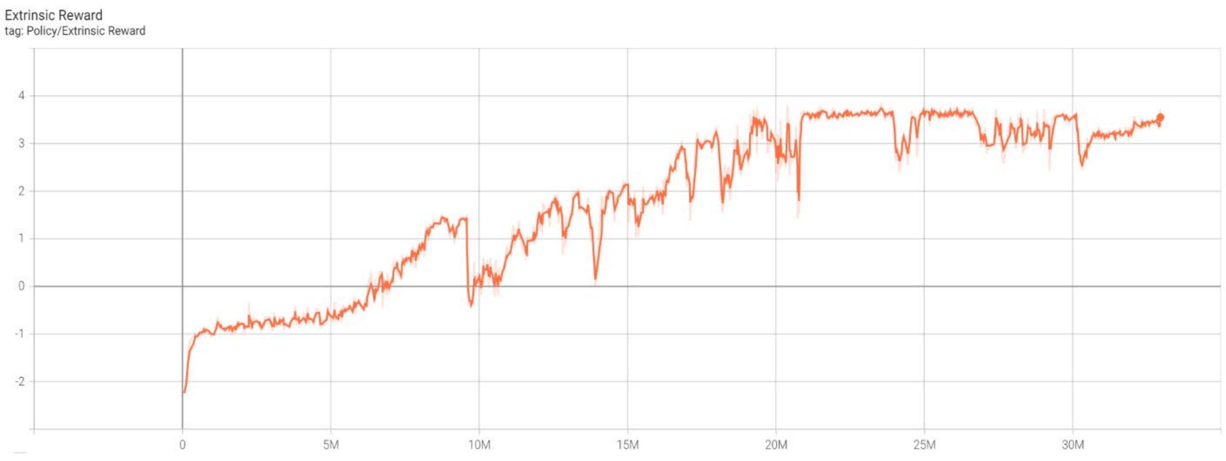


Figure 125: Extrinsic Reward per Step for Phase Window Bricks

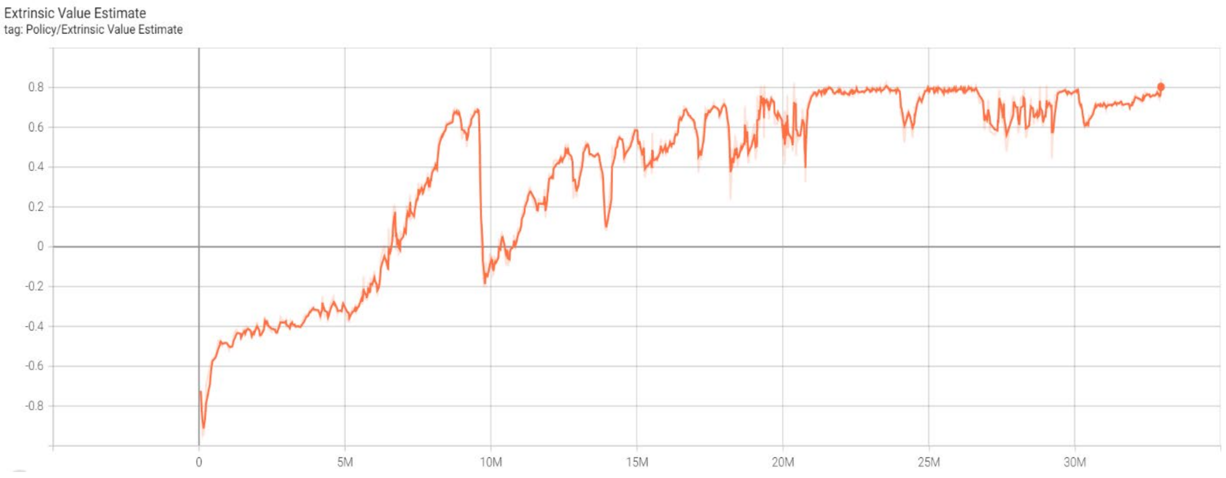


Figure 126: Extrinsic Value Estimate Per Step for Phase Window Bricks

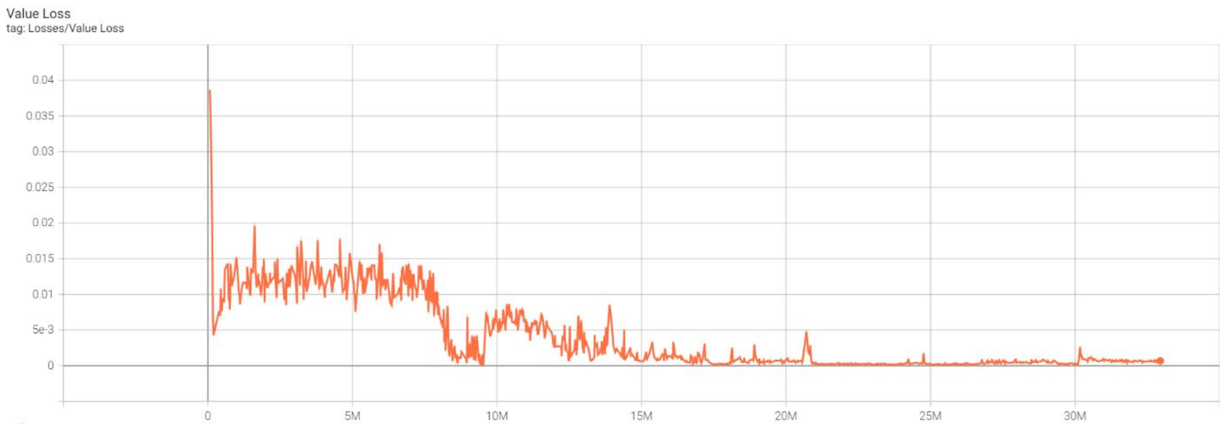


Figure 127: Value Loss per Step for Phase Window Bricks

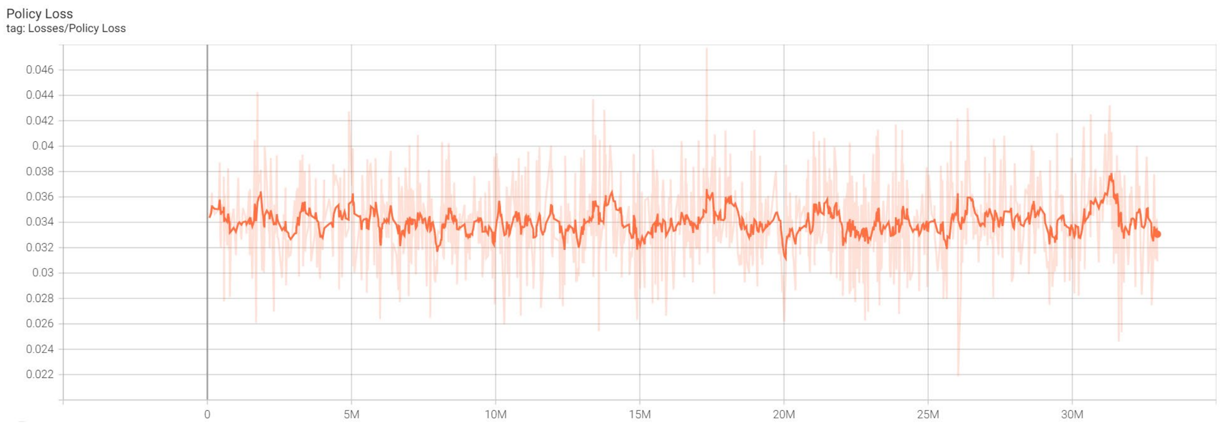


Figure 128: Policy Loss per Step for Phase Window Bricks

- *Summary Points of Statistics Output*

By observing the statistical results, it can be concluded the following main points:

- The training maximum cumulative reward reached has been *3.713* at step *32.96 million.*
- This phase was trained using curriculum learning of 3 lessons as discussed before. In each lesson the number of allowable level maximum level to be reached is elevated. It started at 4 levels then 6 and finally 7 levels.
- Policy loss (the mean loss of the policy function update) values were oscillating throughout the training session with an overall "slight" downward slope towards the end of training session.
- Entropy was consistently decreasing throughout the training session, apart from a minor period that coincides with the shifting from Lesson 1 to Lesson 2.
- The chosen option for "Learning rate" hyperparameter was selected to linearly decay on the span of the maximum steps.

- Epsilon, Beta and learning rate values were decreasing at a fixed rate until step 32.37 million, a sudden drop occurred then they continued to decrease at a slope steeper than the period before the sudden drop.
- Time horizon was raised from window bricks phase to be 1024 instead of 512. This value remained constant for both sessions 1 and 2.

- *Discussion of the Training Summary and hyperparameters*

This phase is the first one to adopt curriculum learning in its training routine. Practically, Phase "Base Bricks" was supposed to include also curriculum training but given that the trained model of this phase was eligible for direct use in the inference phase of Phase "Base Bricks", we neglected the training results of that earlier phase.

The curriculum, despite being divided into 3 lessons in this phase, basically has 2 different types of levels if they are to be classified according to the maximum capacity for brick placement. The first phase which includes lesson 1 and 2 are the Bricks surrounding the window openings all around the pavilion. The second phase comprises only 1 level which what is called the "Lentil Level for window openings". In the latter, the main goal was to place as much bricks as possible to create a good base for the upper bricks of the dome phase.

Epsilon, and learning rate were supposed to be decaying with a constant slope, however the sudden drop in the values is due to a change midst training of the maximum number of steps allocated for training. Reducing the number of steps led to the recalibration of both parameters so that they would decay and reach zero by the end of the allowed maximum steps of 50 million. Previously, it was set at 200 million steps.

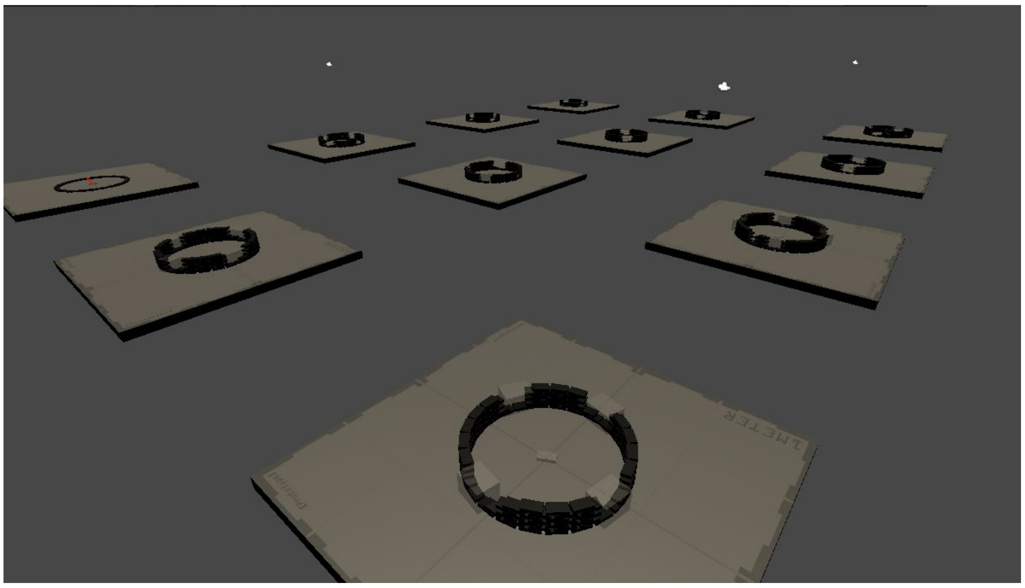


Figure 129: 12 Instances of the "Building Area" with agents being trained at the End of a successful training session for Phase Window Bricks

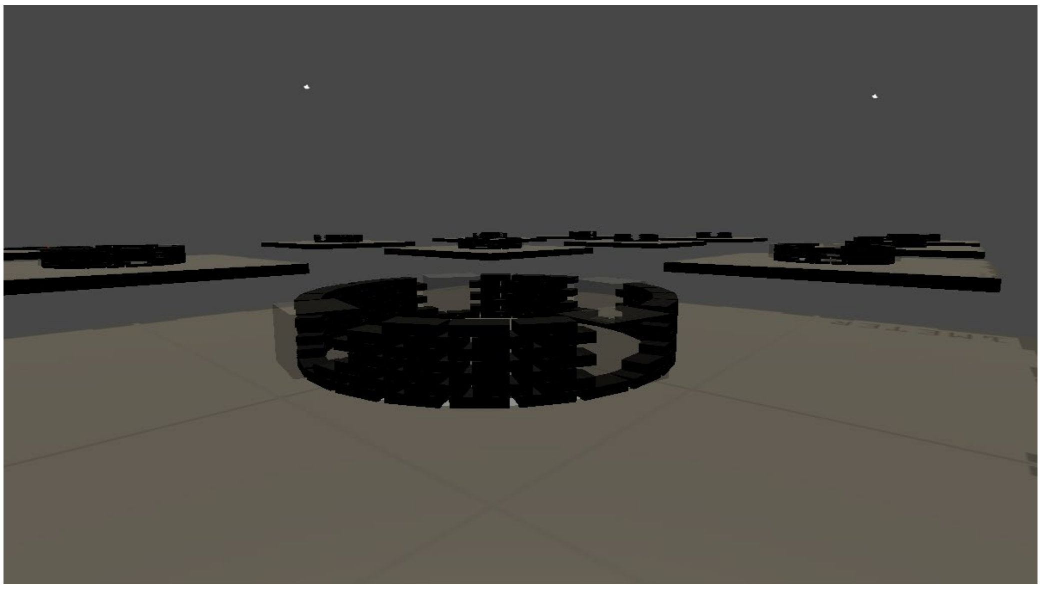


Figure 130: Close caption of an agent during training for Phase Window Bricks

On the other hand, during this episode, a curiosity reward was among the modules experimented. Curiosity works best in environments where the agent receives rare or infrequent rewards (i.e. sparse-reward), an agent may never receive a reward signal on which to bootstrap its training process. This is a scenario where the use of an intrinsic reward signals can be valuable. Curiosity is one such signal which can help the agent explore when extrinsic rewards are sparse.

Accordingly, during the adjustment of the training model, all of the instantaneous rewards were removed from the game and only rewards resulting from a game termination scenario where kept. The curiosity Reward Signal enables the Intrinsic Curiosity Module. This is an implementation of the approach described in Curiosity-driven Exploration by Self-supervised Prediction by Pathak, et al [89]. It trains two networks [78]:

1. an inverse model, which takes the current and next observation of the agent, encodes them, and uses the encoding to predict the action that was taken between the observations
2. a forward model, which takes the encoded current observation and action, and predicts the next encoded observation.

The loss of the forward model (the difference between the predicted and actual encoded observations) is used as the intrinsic reward, so the more surprised the model is, the larger the reward will be.

2.9.5. Phase Dome Bricks

Phase Dome Bricks was trained on 2 consecutive runs. Thus, the second run was initialized from where the first run has stopped. The difference is updates added to the curriculum training as well as some hyperparameters. The first run is indicated in (Cyan) while the second is in (Green).

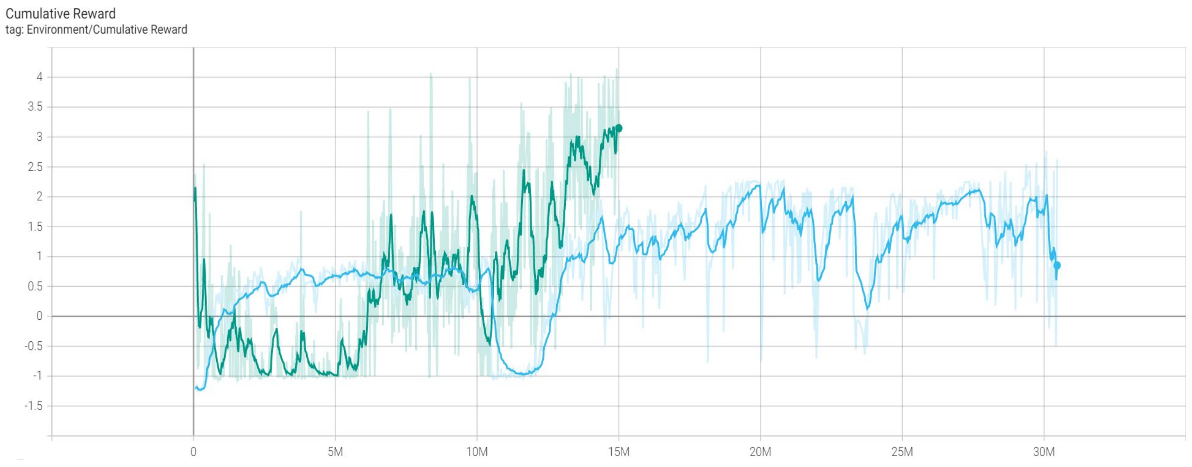


Figure 131: Cumulative Reward for Phase Dome Bricks

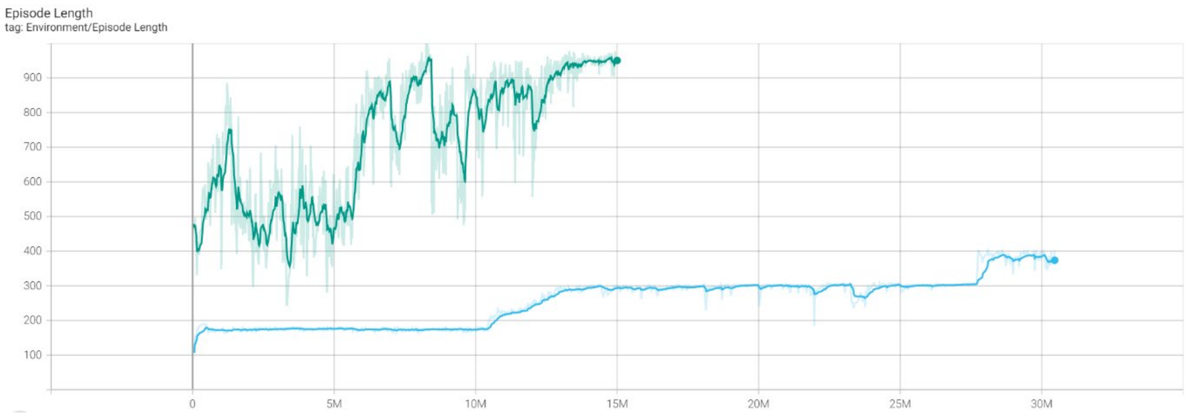


Figure 132: Episode Length per Step for Phase Dome Bricks

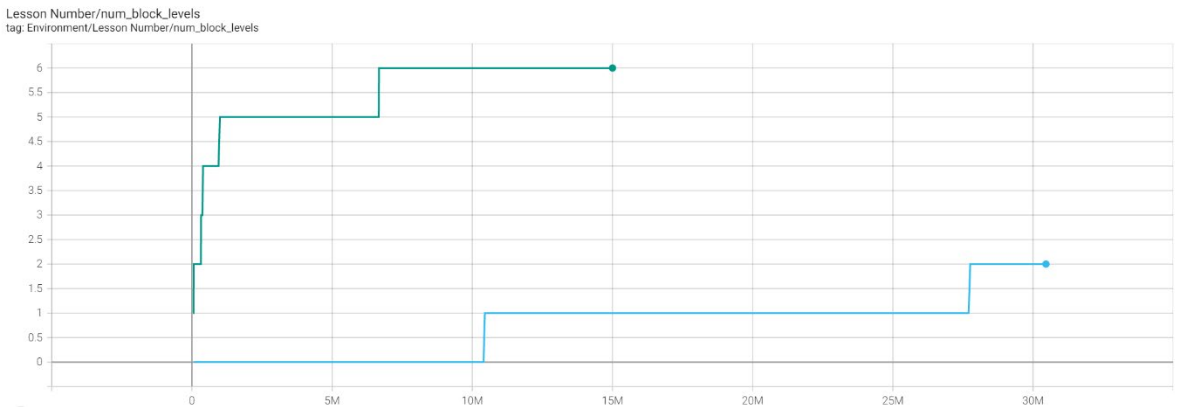


Figure 133: – Lesson number Per step for Number of Brick Levels for Phase Dome Bricks

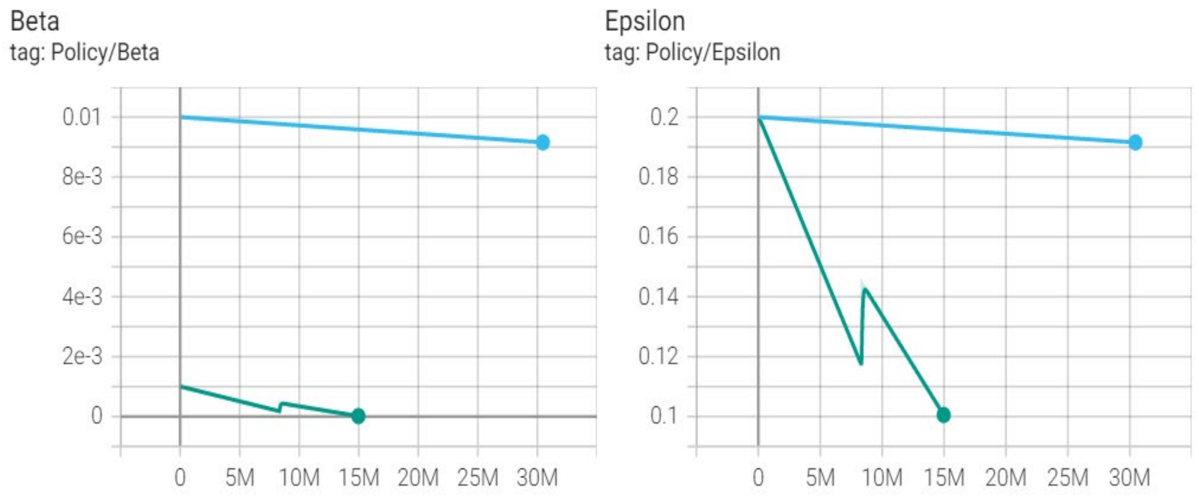


Figure 134: (left) Beta Evolution per Step- (right) Epsilon Evolution per Step Phase Dome Bricks

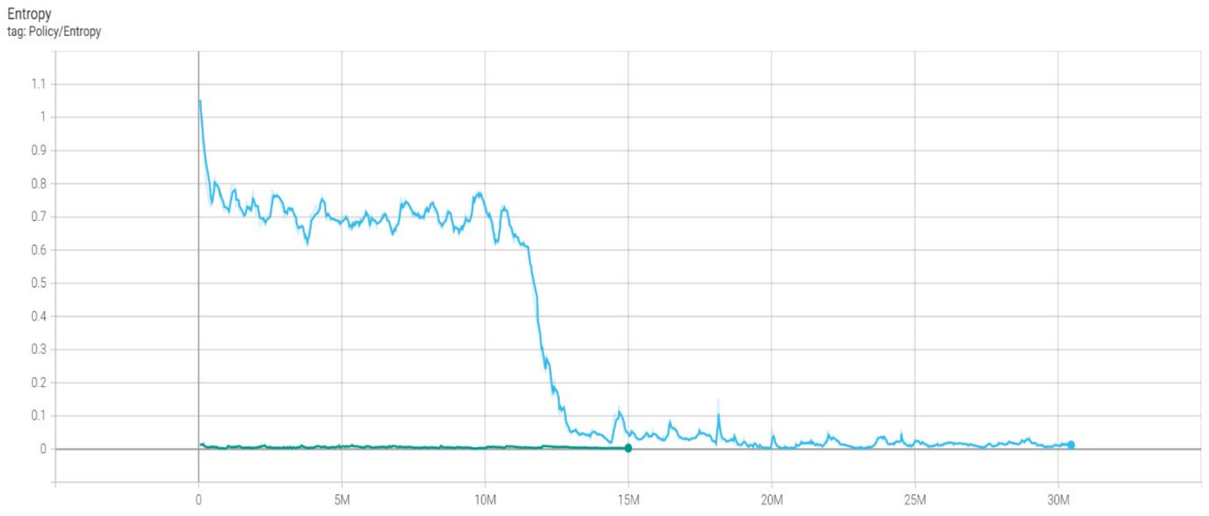


Figure 135: Entropy Evolution per Step Phase Dome Bricks

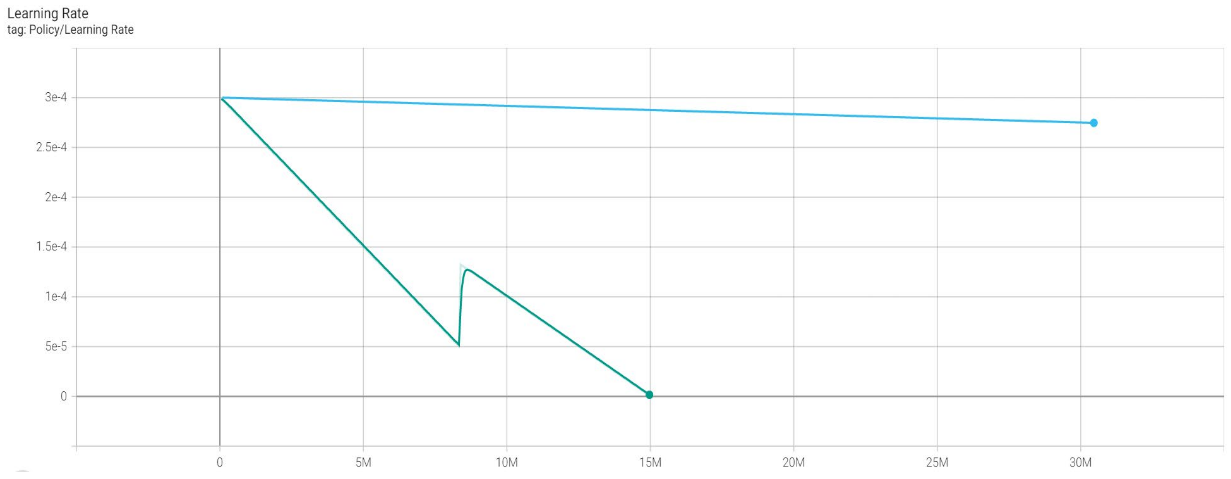


Figure 136: - Learning Rate Evolution per Step for Phase Dome Bricks

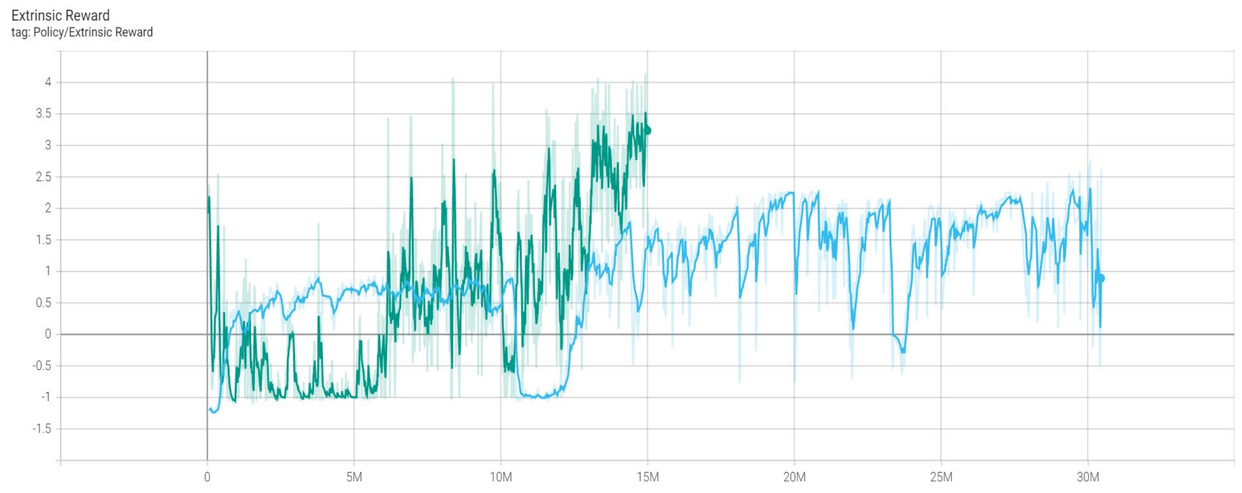


Figure 137: - Extrinsic Reward per Step for Phase Dome Bricks

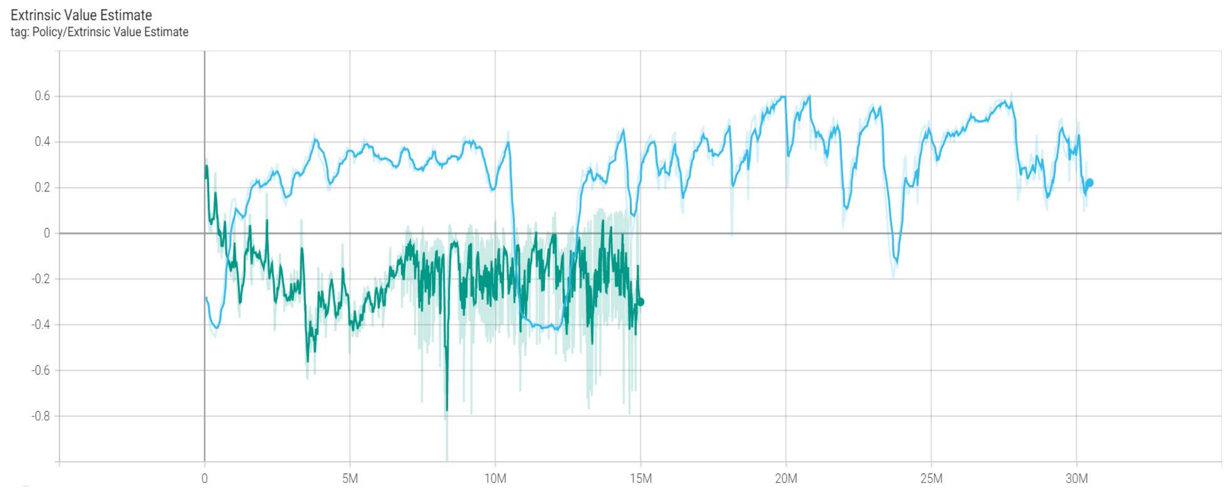


Figure 138: Extrinsic Value Estimate Per Step for Phase Dome Bricks

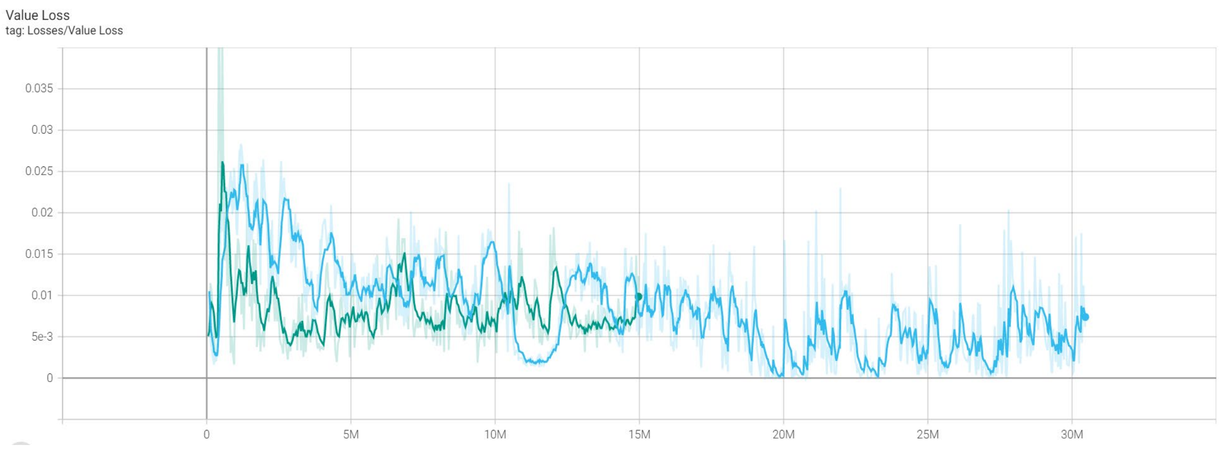


Figure 139: Value Loss per Step for Phase Dome Bricks

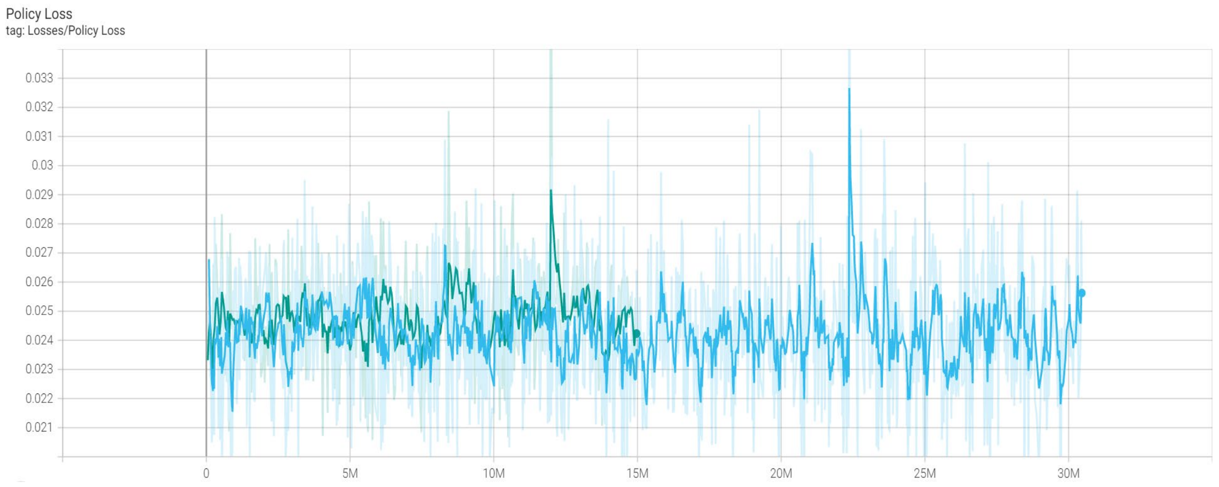


Figure 140: Policy Loss per Step for Phase Dome Bricks

- *Summary Points of Statistics Output*

By observing the statistical results, it can be concluded the following main points:

- The first training session has started till reaching 30.46 million steps reaching an average cumulative reward of 2.63. Afterwards session 2 has been initialized by the weights developed during the training of the neural network of session 1. The training has continued till reaching 3.036 as an average cumulative reward after training for 15 million steps.
- Both sessions have been trained using curriculum learning (see section 3.8.7). Session 1 has succeeded in reaching the third lesson in its designated curriculum, ending with an average episode length of 403.1 steps. Session 2 however has started with what session 1 has ended with (Lesson Three), that has fixed the maximum dome height required at 8 levels. However, the threshold for passing to the next lesson was lowered to be 1.0 as an average cumulative reward instead of 2.7 that was previously set as the threshold in session 1. The agent has successfully surpassed all the lessons assigned to in session 2, reaching the final lesson (9th Lesson) with a maximum dome height of 25 levels. It finishes with an average episode length of 959.3.
- Policy loss (the mean loss of the policy function update) values were oscillating throughout the training session with the absence of a clear downward slope towards the end of training session.
- Entropy was consistently decreasing throughout the training session for session 1, while session 2 entropy was more or less stable around zero.
- Epsilon, and learning rate values were decreasing more rapidly in session 2.

- The chosen option for “Learning rate” hyperparameter was selected to linearly decay on the span of the maximum steps, reaching zero at the end of the steps.
- Hyper parameter “Beta” has been decreased from 0.01 in session 1 to 0.001 in session 2 training.
- Time horizon was raised from window bricks phase to be 1024 instead of 512. This value remained constant for both sessions 1 and 2.

- *Discussion of the Training Summary and hyperparameters*

This session is considered the most challenging one. Given its added complexity with the introduction of the reduced radius on each newer upper level in order to create the dome. This has led to several challenges during the training process.

Regarding the first point in the summary, it is noticeable that the first training session has taken much more time than the second one, however it has only reached level 8 as the maximum dome height. On the other hand, session 2 has reached a final dome height of 25 levels in nearly half of the time of session 1. On the contrary, there is not much improvement regarding the rewards achieved in session 2. Though it has managed to reach 17 extra levels, the average cumulative reward has increased by only 15% compared to what session 1 has ended with.

This is mainly because each session had a diverse goal to achieve. As for the first session, it took more time and the promotion from one lesson to the other required a heigh reward to achieve it. This has an impact on achieving a sturdy dome base in the beginning. This is a crucial part in withstanding all the upper loads. It also focused on the proper placement of bricks in an arranged consecutive way that was explained earlier. This arrangement was taken into consideration in the reward system to mimic an optimized strategy for energy management on site by reducing the transition distance between supply and demand points and reduce risks associated to path planning issues.

On the other hand, after achieving high rewards on the first 8 levels, it was sought to give priority to the crucial aspects of stability in the higher levels and turn a blind eye on the complementary tasks. The motivation behind that was to accelerate the speed of learning and reduce the level of complexity in an already challenging task.

This was evident when the agent was able in session 2 to successfully complete the assigned lessons in a quick pace till reaching the final lesson, however at the expense of some other aspects. One of these major aspects that raised our concern was mainly design related. A source of this design failure is the increased number of gaps between bricks that eventually got wider by increasing heights and led to the split of the brick fabric making it impossible to reconnect bricks on top one more time till the upper oculus (see Figure 141). This would consequently jeopardize the initial design idea.

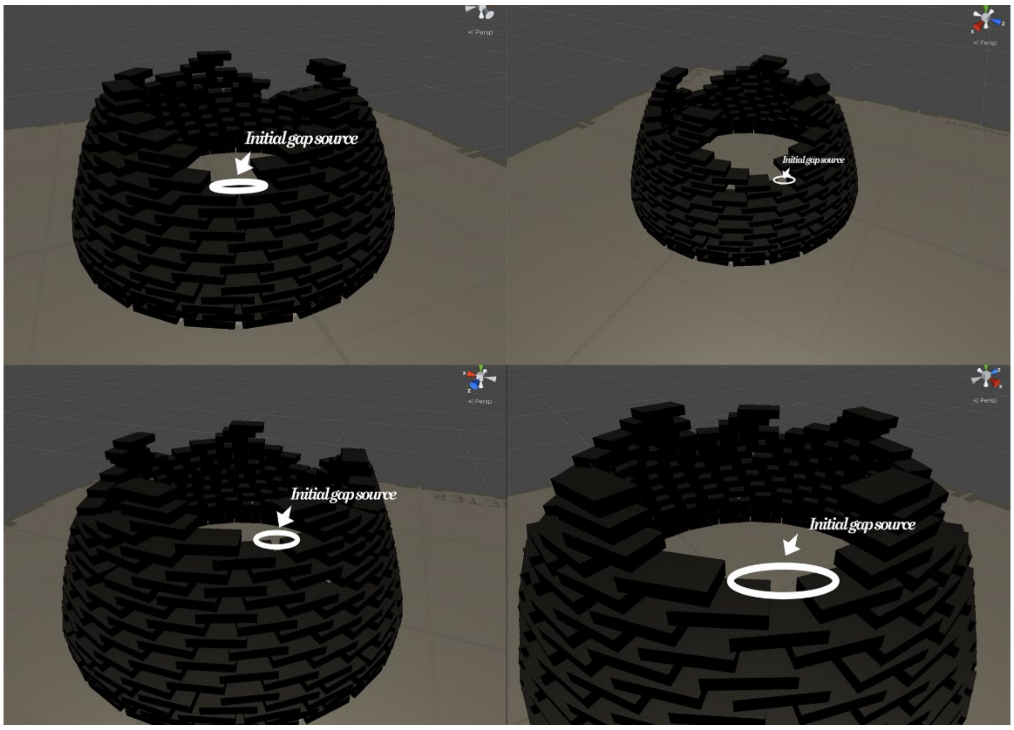


Figure 141: Gap Propagation leading to failure in complying with Design Requirements

The intervention we have proposed was an introduction of the "Gap Check" and setting a maximum number of allowable gaps. This threshold, if exceeded, would directly lead to game termination due to design failure (as explained in rewards sub-section 3.8.6). In addition, the penalty for such an error was quite large with respect to other errors. The training was then restarted with the new addition to the reward system and the final results has shown a great improvement in that manner.

This also explains the limited increase of rewards in session 2. In some cases, the agent would create an empty gap, thus is penalized. However, it has learnt how to recover from this mistake and to prevent gap propagation by adjusting the bricks to be placed on top of such gap.

The additional step that was also introduced was a small incentive rewarded to inserting bricks in "Star Points". This reward was turned on and experimented on higher levels in the dome (for instance from level 5 from dome start and then upwards). This incentive has also indirectly contributed to the reduction of gaps between bricks.

Regarding "Beta Parameter", its definition in [90] states that it ensures that agents properly explore the action space during training. Increasing this will ensure more random actions are taken. Accordingly, this explains why it had was set to a large value in Session 1 and a very low value in Training Session 2. The main goal was to let the agent "explore" as much as possible different scenarios in order to eventually find its way to the optimum strategy to be used. While in session 2, once we initialize it from the “more experienced” agent from session 1, we don’t want it to explore more, but rather keep the “rhythm” acquired from training session 2 and try to maintain the same approach as much as possible in Session 2.

Furthermore, this phase has witnessed the highest value for time horizon. Since time horizon corresponds to how many steps of experience to collect per-agent before adding it to the experience buffer. When this limit is reached before the end of an episode, a value estimate is used to predict the overall expected reward from the agent's current state.

Accordingly, after many trials, the most convenient solution was to increase the time horizon to capture as much events as possible. This case differs greatly from other phases including window bricks and phase base bricks because of the incompatibility of applying the repetitive concept. In other words, at lower levels and since the radius remains constant it is possible to consider that once a brick level is completed in sequence, there is less urge to capture all succeeding actual rewards from the game.

This approach could be convenient if what occurs on one level can be just “repeated” on all upper levels afterwards. This strategy is sought to be inefficient in the changing radius dome. Despite the effort to try generalizing observations, especially when it comes to height calculations, it is still not possible to assume that a brick placed on a radius of 5 meters has the same circumstances as another one placed at a radius of 4 meters or less.

Finally, the following images are some screenshots of the training sessions for Dome phase. These shots were taken at the final steps of both training sessions.

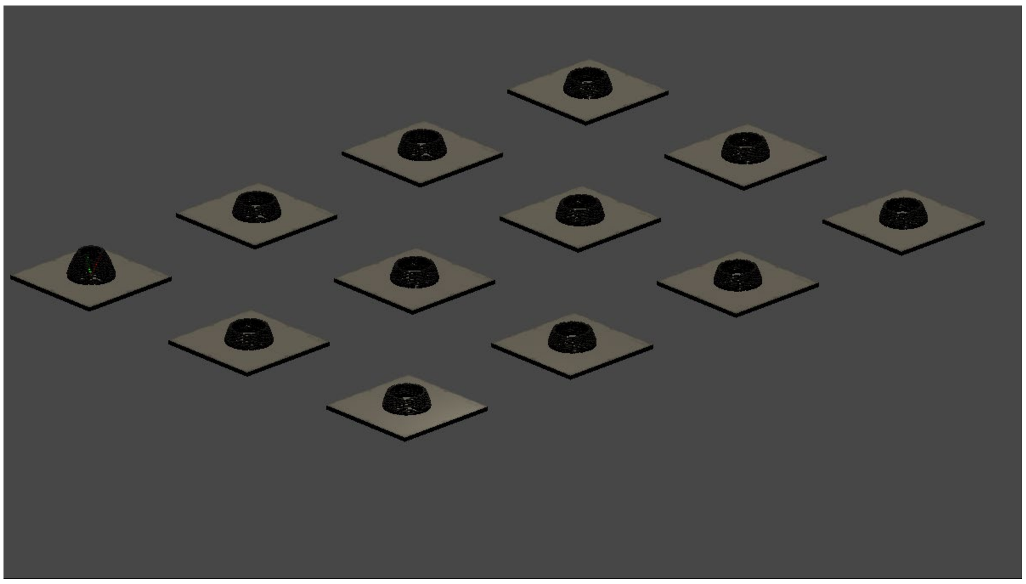


Figure 142: 12 Instances of the “Building Area” with agents being trained at the end of session 2

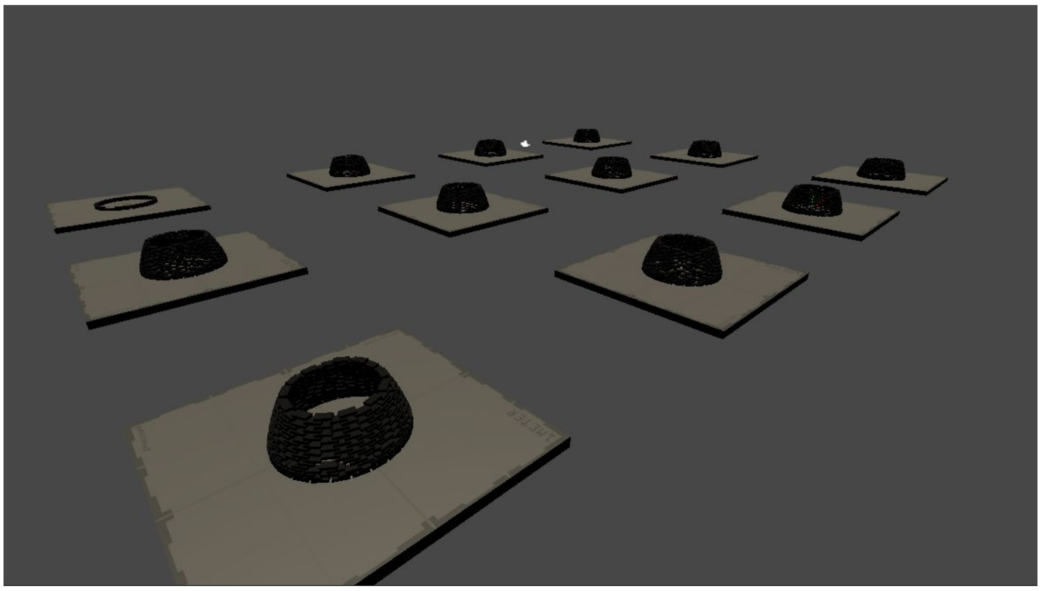


Figure 143: Close Caption of Simultaneous Training of Agents in phase 5 (Dome Creation)

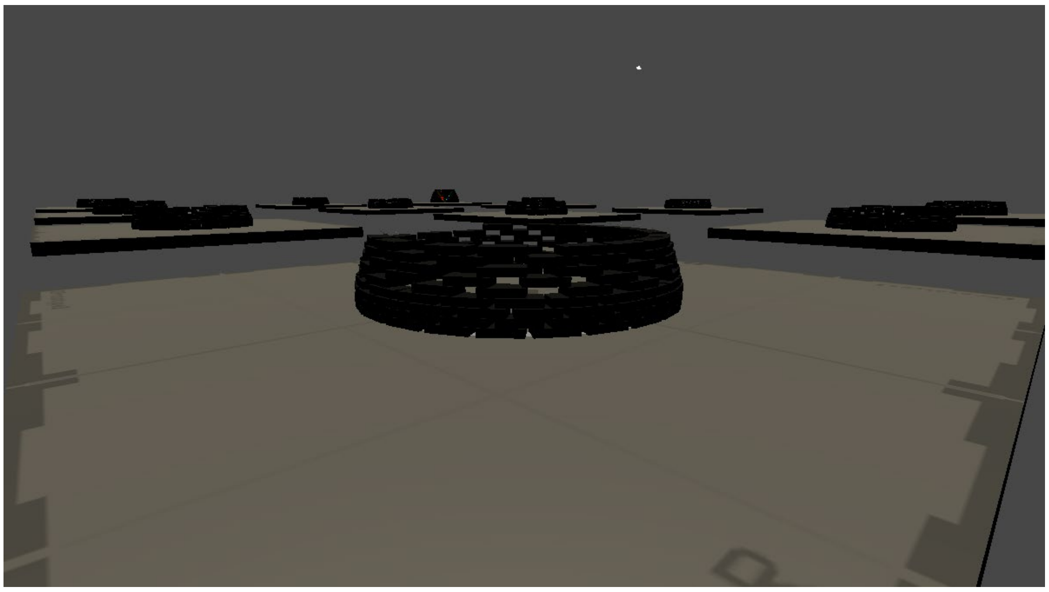


Figure 144: Agents Brick Placement Behavior after the introduction of small incentives for selecting "Star Points" and its effect on the overall Design of the Pavilion

2.9.6. Review of Inference Phase

Once all phases were trained, comes the turn of inference phase. All trained models are then connected to the game object with the Agent component attached to it (Robuilder Agent). Despite the training of the models was executed in simplified configuration regarding the absence of a complete lower structure for phases built on existing bricks, the inference phase was carried out successfully.

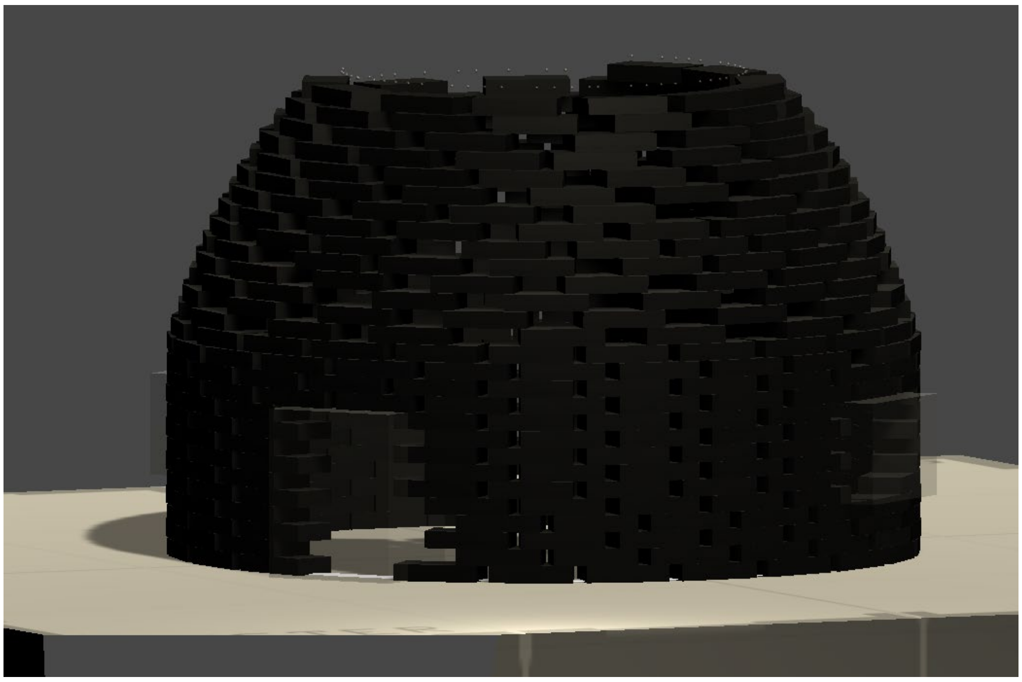


Figure 145: Trained Models Compiled in Inference phase, creating the whole structure (1/3)

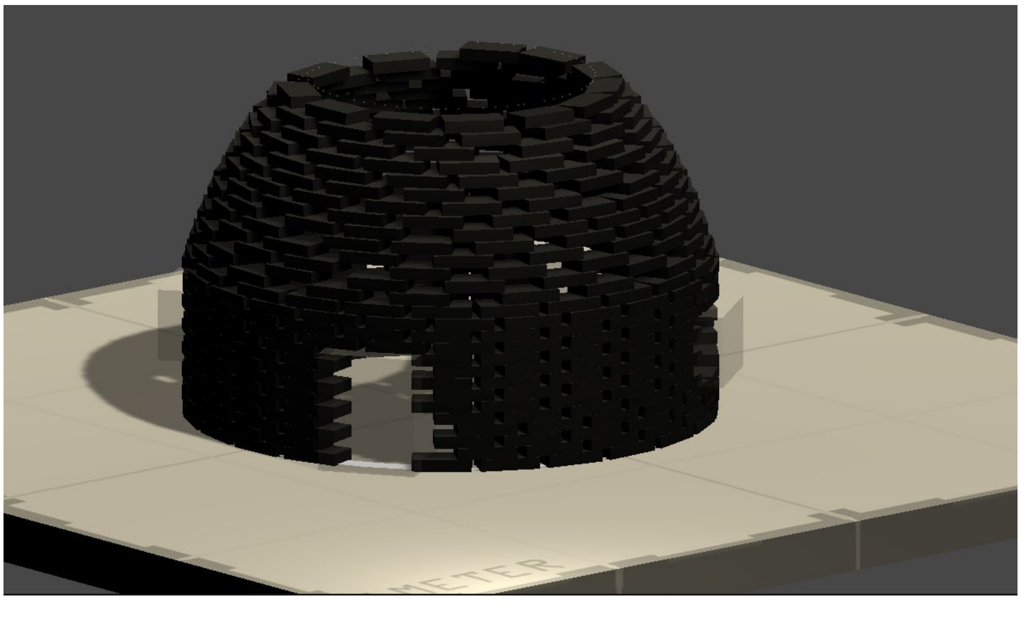


Figure 146: Trained Models Compiled in Inference phase, creating the whole structure (2/3)

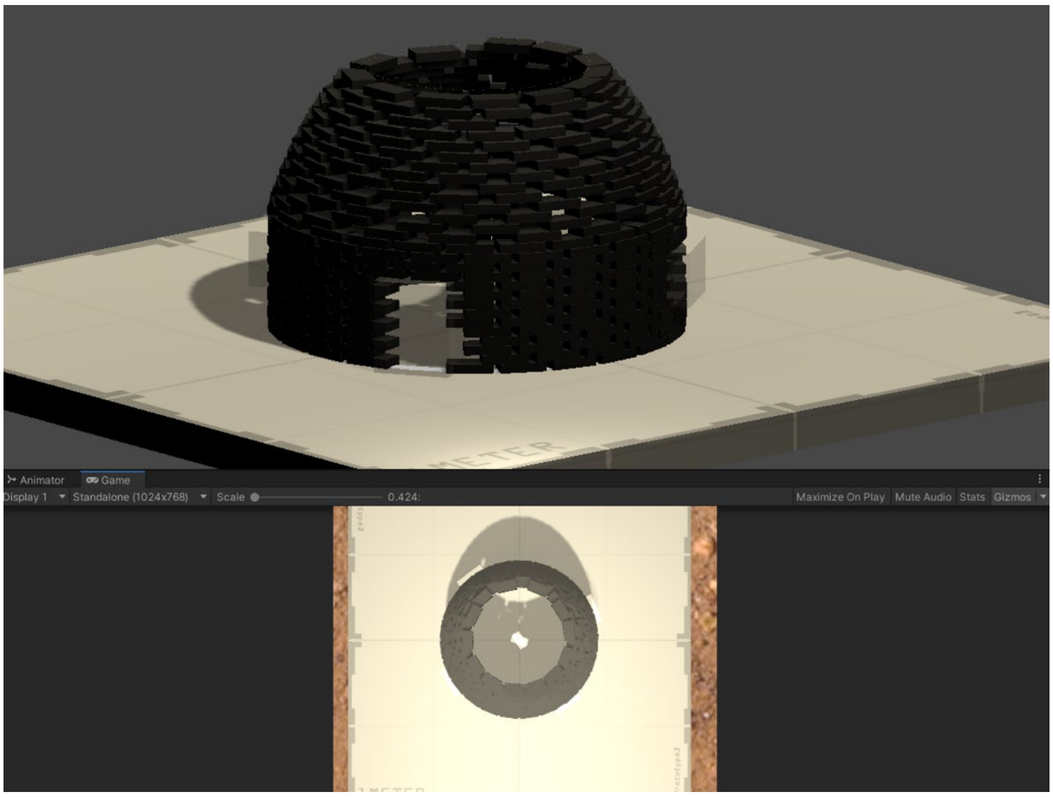


Figure 147: Trained Models Compiled in Inference phase, creating the whole structure (3/3)

One of the main reasons that needed to be optimized in order to guarantee this smooth connection between different phases during inference mode is the generalization of Observations. This issue was creating problems in the beginning of the early trials. From one point most of the upper brick phases were trained with only one level of bricks below, while in the actual scenario these bricks should have been placed above several brick layers.

This issue has created a discrepancy for observations related to height calculations. The solution that we decided to follow was to use relative height definitions instead of absolute ones. For instance, a brick height to be used for observation variable is given as the height calculated from the beginning of the phase and not from the ground. In that case, no matter how many brick levels are present below the current phase, the observation will have the same value in both inference and training modes.

2.9.7. Final Notes

This sub-section is dedicated to some final bullet points regarding RV and the training of the agents.

- *Time Speed:*

During the training of the agents, the speed of the game goes multiple times faster than the actual (real) speed. At one point, this is very crucial method to reduce the amount of time consumed in the learning process. However, it creates a drastic effect on the physics behavior of the game objects.

For example, if a brick was starting to tilt at a normal speed, the brick might reach the maximum threshold and the game could terminate. But, at 10x the average speed of the game, and the tilting is still happening at the normal speed, there is a high chance that a new brick will be placed to stabilize it before even starting to tilt. This creates fake situations and when the game is played in the normal speed after training, the Agent might not be able to react correctly to this behavior.

Therefore, the solution was to link the physics behavior speed with the current training speed used. This would simulate movements at the same high speed of the training process.

- *One hot Observations*

One hot style was used in describing categorical information that are fed to the observation vectors. During the initial trials, float values were also tried in replacement of One hot style. For instance, to represent the selection of the 2nd Segment of the 12 Segments in the observation vector, there are 2 possible ways:

1. By using One-hot Style (This consumes 12 variables in the Observation Vector). The Vector will contain all zeroes except for the value of 1 at position 2.
2. By using 1 float value. This can be done by dividing the value 2 (second segment) by 12 (total number of segments). In this case, no matter which segment is chosen, it will be divided by the total number of segments and will only consume 1 position in the Observation Vector.

Despite that the second option may seem more intuitive to reduce the complexity of the problem, it turned out during training that the neural network can identify in an easier way categorical information when conveyed in one-hot style. Of course, when we only have 3 categories for example instead of 12, there wouldn't be great difference between both representations. Therefore, Categorical information in the range of (10-20) items, is best represented in the observation vector using One-hot style.

- *Time Horizon Hyper parameter*

During the several trials done throughout the training of agents, finetuning the hyperparameters was regarded a crucial factor to success. One of these important hyperparameters was `time_horizon`. According to ML—Agents the definition of `time-horizon` is: How many steps of experience to collect per-agent before adding it to the experience buffer [90].

When this limit is reached before the end of an episode, a value estimate is used to predict the overall expected reward from the agent's current state. As such, this parameter trades off between a less biased, but higher variance estimate (long time horizon) and more biased, but less varied estimate (short time horizon).

It turned out that, it is possible to adjust this parameter to lower its value in repetitive processes. On the contrary, in phases that have too many details that aren’t replicated, it is required to increase its value as much as possible in order to capture all these events during training.

For instance, in lower bricks phases, since the radius remains constant, it is possible to consider each brick level as an event that is repeated along the episode several times. Therefore, value of time horizon could be equal to the number of steps needed to complete only one level of bricks.

On the other hand, in Phase Dome Creation, since every level has its own unique properties, it is required to increase the time horizon to a much larger value.

2.10. Conclusion and Future Research

In this research, a proposed framework for automating design and construction processes has been presented. This framework was inspired by the achievements of RL Algorithms in gaming industry as well as the modern advancements in simulation environments supported by game engines.

Through the proper transformation of real case constraints into the simulation environment, the designer could have a new role in this framework. This role is mainly concerned with supervising the training process of AI agents that learn how to execute this project through exploring the simulated environment.

These agents by the help of RL Algorithms could learn simultaneously the design and construction of structures in the simulated environment. This learning process is directed by a reward system that is set by the designers prior to the training process. Once trained properly, the agents are able to deploy their experience on site by the help of robotic execution. These AI-Agents then become the brain that directs these robots on site on what to do.

Eventually, the team of designers and engineers would form a new modernized “Master Builder”, that supervises his crafts men on site (agents in the simulation environment) in order to later convey the proper instructions to the labor force (the execution robots).

As a proof of concept, RoBuilDeR application was developed in Unity Platform and ML-Agents framework was used for RL training of the agents. The Game objective is 2 design and construct a Brick Pavilion inspired by the pantheon’s dome.

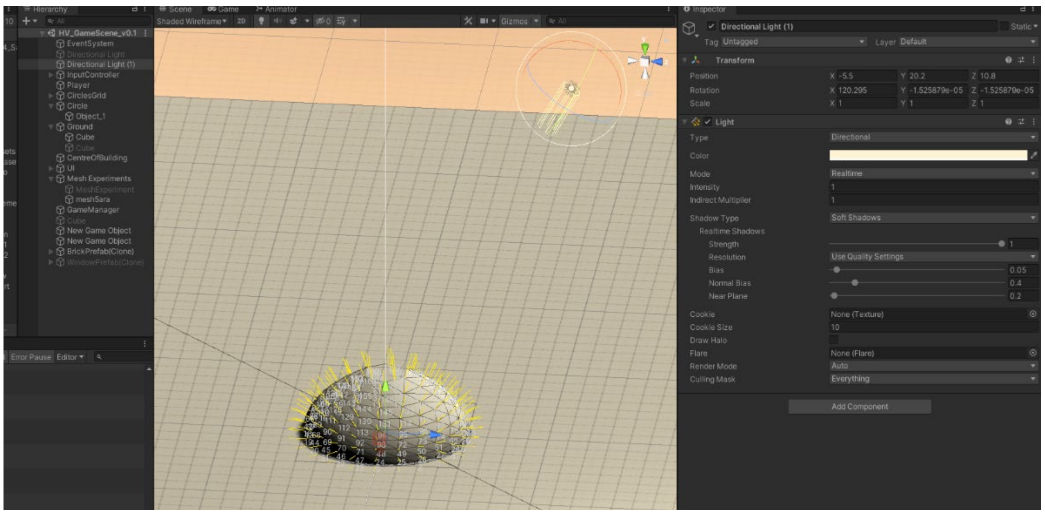
2 Versions of the game were provided. The UV (Human Version) is for the designer to check the simulated environment and adjust its parameters and the RV (Robot Version) that is considered the learning playground for the AI-Agents.

Proximal Policy Algorithm (PPO) was used to train the Agents. The complete project was divided into 5 phases to reduce its complexity for the Agents. During the training process, several methods were adopted to facilitate the learning process of the Agents. Examples of these processes include: curriculum learning, curiosity module, parallel training of several instances, random generation of environment parameters, generalization of observation parameters etc. Finally, the 5 phases were integrated together in the inference phase and the results were discussed.

There are several topics in this research that are still open for development and improvement. For instance, regarding the division of the project into phases, perhaps with bigger scale projects the use of hierarchical reinforcement learning could be a solution instead of the “manual" process that has been done here.

Hierarchical Reinforcement Learning (HRL) enables autonomous decomposition of challenging long-horizon decision-making tasks into simpler subtasks. During the past years, the landscape of HRL research has grown profoundly, resulting in copious approaches. [91].

Furthermore, there are some additional design parameters that were not included in this research but are still under development. An example is sunlight analysis and better checking of “design gaps” through mesh analysis and calculations of normal vectors to mesh faces. This method could transform the bricks into meshes. Then a plenty of analysis can be executed based on the rich data that meshes convey,



Directional Light (1) Static
Tag Untagged Layer Default
Transform
Position X -5.5 Y 20.2 Z 10.8
Rotation X 120.295 Y -1.525879e-05 Z -1.525879e-05
Scale X 1 Y 1 Z 1
Light
Type Directional
Color
Mode Realtime
Intensity 1
Indirect Multiplier 1
Shadow Type Soft Shadows
Realtime Shadows
Strength 1
Resolution Use Quality Settings
Bias 0.05
Normal Bias 0.4
Near Plane 0.2
Cookie None (Texture)
Cookie Size 10
Draw Halo
Flare None (Flare)
Render Mode Auto
Culling Mask Everything

Figure 148: Mesh Creation out of brick Pattern (Under Development)

Accordingly, this research has drawn the attention to some points that would eventually lead in the near future to major disruptions in the AEC industry. The following points summarizes some of these ideas:

- ML-Agents is still in its primary stage, however with increased implementation of RL state-of-the-art algorithms, many applications could be built up on top of it. Eventually, it would become as “grasshopper” is to “Rhino” but for deploying deep RL algorithms.
- The continuous development of game-engines as well as the new direction towards the “Meta Verse” would drastically improve the qualities present in simulated environments and increase the “real” essence of them. Thus, less discrepancy would be found between what the Agent learns in the simulation and what it could face in real world
- 3D GANs is also regarded as a new tool, that once properly developed, it is expected to significantly contribute to the faster development of new design ideas. It could also be linked with Agent-Based Learning methods like Deep RL.

To sum up, this new paradigm shift would essentially require in the first place a change in the mindset of modern designer and architects and rearrangement for their learning priorities. By acquiring the needed skills for these new tools and techniques they could lead the transformation process of our AEC Industry into a more modern version that is built on sustainability and agility.

References

[1] J. H. Fetzer, “What is Artificial Intelligence?,” in *Artificial Intelligence: Its Scope and Limits*, J. H. Fetzer, Ed. Dordrecht: Springer Netherlands, 1990, pp. 3–27. doi: 10.1007/978-94-009-1900-6_1.

[2] “Definition of INTELLIGENCE.” https://www.merriam-webster.com/dictionary/intelligence (accessed Apr. 09, 2022).

[3] “What is the Turing Test?,” *SearchEnterpriseAI*. https://www.techtarget.com/searchenterpriseai/definition/Turing-test (accessed Apr. 09, 2022).

[4] N. Leach, “Do Robots Dream of Digital Sleep?,” 2019. Accessed: Apr. 09, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia19_298

[5] “Blade Runner (1982) - IMDb.” https://www.imdb.com/title/tt0083658/ (accessed Apr. 09, 2022).

[6] “Recordings,” *Intelligent Machinery, Identity and Ethics*, Oct. 04, 2018. https://intelligentmachinerycourse.com/recordings/ (accessed Apr. 09, 2022).

[7] “MAKOTO SEI WATANABE | 渡辺 誠.” https://www.makoto-architect.com/aitect.html (accessed Apr. 10, 2022).

[8] “Deep Dream Generator.” https://deepdreamgenerator.com/#gallery (accessed Apr. 10, 2022).

[9] C. Y, “Creating Art with Generative Adversarial Network,” *Medium*, Mar. 01, 2022. https://medium.com/@ymingcarina/creating-art-with-generative-adversarial-network-refik-anadols-wdch-dreams-159a6eac762d (accessed Apr. 10, 2022).

[10] J. Takeno, *Self-Aware Robots: On the Path to Machine Consciousness*. CRC Press, 2022.

[11] “The Difference Between AI, Machine Learning, and Deep Learning?,” *NVIDIA Blog*, Jul. 29, 2016. https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/ (accessed Apr. 10, 2022).

[12] J. Vrana and R. Singh, *The NDE 4.0: Key Challenges, Use Cases, and Adaption*. 2020.

[13] “Artificial Neural Network,” *NVIDIA Developer*, Apr. 23, 2018. https://developer.nvidia.com/discover/artificial-neural-network (accessed Apr. 10, 2022).

[14] “NVIDIA Blog: Supervised Vs. Unsupervised Learning,” *NVIDIA Blog*, Aug. 02, 2018. https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/ (accessed Apr. 10, 2022).

[15] “The Tree of Machine Learning Algorithms | Teradata Blog.” https://www.teradata.com/Blogs/The-Tree-of-Machine-Learning-Algorithms (accessed Apr. 10, 2022).

[16] T. Hong, Z. Wang, X. Luo, and W. Zhang, “State-of-the-art on research and applications of machine learning in the building life cycle,” *Energy Build.*, vol. 212, p. 109831, Apr. 2020, doi: 10.1016/j.enbuild.2020.109831.

[17] S. Blundell, “Making a foray into AI for applications in the AEC industry,” *Planning, BIM & Construction Today*, May 24, 2021. https://www.pbctoday.co.uk/news/construction-technology-news/ai-in-aec/93782/ (accessed Apr. 10, 2022).

[18] B. and J. G. Andrade Zandavali, “Automated Brick Pattern Generator for Robotic Assembly using Machine Learning and Images,” 2019. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ecaadesigradi2019_605

[19] “Form Follows Function, Function Follows Form : Journal of Craniofacial Surgery.” https://journals.lww.com/jcraniofacialsurgery/Citation/2020/04000/Form_Follows_Function,_Function_Follows_Form.4.aspx (accessed Apr. 10, 2022).

[20] I. S. P. and P. B. As, “Artificial intelligence in architecture: Generating conceptual design via deep learning,” 2018. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ijac201816406

[21] M. del Campo, “Architecture,Language and AI - Language,Attentional Generative Adversarial Networks (AttnGAN) and Architecture Design,” 2021. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/caadria2021_389

[22] W. Z. Huang, “Architectural Drawings Recognition and Generation through Machine Learning,” 2018. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia18_156

[23] A. B. Mohammad, “Hybrid Elevations using GAN Networks,” 2019. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia19_370

[24] L. Brown, “Drawing Recognition - Integrating Machine Learning Systems into Architectural Design Workflows,” 2020. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ecaade2020_047

[25] V. Eisenstadt, “Generation of Floor Plan Variations with Convolutional Neural Networks and Case-based Reasoning - An approach for transformative adaptation of room configurations within a framework for support of early conceptual design phases,” 2019. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ecaadesigradi2019_648

[26] M. C. del Campo, “How Machines Learn to Plan,” 2020. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia20_272

[27] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and Improving the Image Quality of StyleGAN,” *ArXiv191204958 Cs Eess Stat*, Mar. 2020, Accessed: Apr. 10, 2022. [Online]. Available: http://arxiv.org/abs/1912.04958

[28] K.-H. Chang, C.-Y. Cheng, J. Luo, S. Murata, M. Nourbakhsh, and Y. Tsuji, “Building-GAN: Graph-Conditioned Architectural Volumetric Design Generation,” 2021, pp. 11956–11965. Accessed: Apr. 10, 2022. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2021/html/Chang_Building-GAN_Graph-Conditioned_Architectural_Volumetric_Design_Generation_ICCV_2021_paper.html

[29] H. and B. Zhang, “3D Architectural Form Style Transfer through Machine Learning,” 2020. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/caadria2020_234

[30] G. and N. Rossi, “Haptic Learning - Towards Neural-Network-based adaptive Cobot Path-Planning for unstructured spaces,” 2019. Accessed: Apr. 11, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ecaadesigradi2019_280

[31] J. Bard, A. Bidgoli, and W. W. Chi, “Image Classification for Robotic Plastering with Convolutional Neural Network,” in *Robotic Fabrication in Architecture, Art and Design 2018*, Cham, 2019, pp. 3–15. doi: 10.1007/978-3-319-92294-2_1.

[32] H. Perez, J. H. M. Tah, and A. Mosavi, “Deep Learning for Detecting Building Defects Using Convolutional Neural Networks,” *Sensors*, vol. 19, no. 16, p. 3556, Aug. 2019, doi: 10.3390/s19163556.

[33] C.-L. and H. Cheng, “Biomimetic Robotic Construction Process - An approach for adapting mass irregular-shaped natural materials,” 2016. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ecaade2016_079

[34] N. M. A. K. A. Larsen, “Exploring Natural Wood,” 2019. Accessed: Apr. 10, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia19_500

[35] G. Mirra, A. Holland, S. Roudavski, J. S. Wijnands, and A. Pugnale, “An Artificial Intelligence Agent That Synthesises Visual Abstractions of Natural Forms to Support the Design of Human-Made Habitat Structures,” *Front. Ecol. Evol.*, vol. 10, 2022, Accessed: Mar. 21, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fevo.2022.806453

[36] D. Wang and R. Snooks, “INTUITIVE BEHAVIOR The Operation of Reinforcement Learning in Generative Design Processes,” 2021.

[37] D. Wang and R. Snooks, “Artificial Intuitions of Generative Design: An Approach Based on Reinforcement Learning,” in *Proceedings of the 2020 DigitalFUTURES*, Singapore, 2021, pp. 189–198. doi: 10.1007/978-981-33-4400-6_18.

[38] Z. Han, W. Yan, and G. Liu, “A Performance-Based Urban Block Generative Design Using Deep Reinforcement Learning and Computer Vision,” in *Proceedings of the 2020 DigitalFUTURES*, Singapore, 2021, pp. 134–143. doi: 10.1007/978-981-33-4400-6_13.

[39] P. Veloso and R. Krishnamurti, “An Academy of Spatial Agents Generating spatial configurations with deep reinforcement learning.”

[40] T. Xu, D. Wang, M. Yang, X. You, and W. Huang, “AN EVOLVING BUILT ENVIRONMENT PROTOTYPE A Prototype of Adaptive Built Environment Interacting with Electroencephalogram Supported by Reinforcement Learning,” 2018.

[41] Z. Fang, “Towards multi-drone autonomous construction via deep reinforcement learning,” thesis, Carnegie Mellon University, 2021. doi: 10.1184/R1/14138024.v1.

[42] A. A. Apolinarska *et al.*, “Robotic assembly of timber joints using reinforcement learning,” *Autom. Constr.*, vol. 125, May 2021, doi: 10.1016/J.AUTCON.2021.103569.

[43] T. T. Hosmer, “Deep Reinforcement Learning for Autonomous Robotic Tensegrity (ART),” 2019. Accessed: Apr. 11, 2022. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/acadia19_16

[44] C.-H. Huang, “REINFORCEMENT LEARNING FOR ARCHITECTURAL DESIGN-BUILD Opportunity of Machine Learning in a Material-informed Circular Design Strategy,” 2021.

[45] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft,” *IEEE Trans. Comput. IN^{TEL}LIGENCE AI GAMES*, vol. 5, no. 4, p. 293, 2013, doi: 10.1109/TCIAIG.2013.2286295.

[46] “What is Real-Time Strategy (RTS)? - Definition from Techopedia,” *Techopedia.com*. http://www.techopedia.com/definition/1923/real-time-strategy-rts (accessed Mar. 10, 2022).

[47] “Current AI in games : a review | QUT ePrints.” https://eprints.qut.edu.au/45741/ (accessed Apr. 11, 2022).

[48] “AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Deepmind*. https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning (accessed Mar. 10, 2022).

[49] M. Buro, “Real-Time Strategy Gaines: A New AI Research Challenge”.

[50] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, “On Reinforcement Learning for Full-length Game of StarCraft,” *ArXiv180909095 Cs Stat*, Feb. 2019, Accessed: Mar. 10, 2022. [Online]. Available: http://arxiv.org/abs/1809.09095

[51] B. Weber and M. Mateas, *A data mining approach to strategy prediction*. 2009, p. 147. doi: 10.1109/CIG.2009.5286483.

[52] G. Synnaeve and P. Bessiere, “A Dataset for StarCraft AI & an Example of Armies Clustering,” in *Artificial Intelligence in Adversarial Real-Time Games 2012*, Palo Alto, United States, Oct. 2012, p. pp 25-30. Accessed: Mar. 11, 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00752893

[53] “Towards digital architecture, engineering, and construction (AEC) industry through virtual design and construction (VDC) and digital twin | Elsevier Enhanced Reader.” https://reader.elsevier.com/reader/sd/pii/S266612332100060X?token=F466FCE668CCD24EF1C2ABE90120E83FCEF96FBB515DAFB9DEE2AD9FAD61840683EA2E372B54900B31F4225EB6FA40CD&originRegion=eu-west-1&originCreation=20220311032230 (accessed Mar. 11, 2022).

[54] Y. An, H. Li, T. Su, and Y. Wang, “Determining Uncertainties in AI Applications in AEC Sector and their Corresponding Mitigation Strategies,”

Autom. Constr., vol. 131, p. 103883, Nov. 2021, doi: 10.1016/j.autcon.2021.103883.

[55] A. Darko, A. P. C. Chan, M. A. Adabre, D. J. Edwards, M. R. Hosseini, and E. E. Ameyaw, “Artificial intelligence in the AEC industry: Scientometric analysis and visualization of research activities,” *Autom. Constr.*, vol. 112, p. 103081, Apr. 2020, doi: 10.1016/j.autcon.2020.103081.

[56] A. Zabin, V. A. González, Y. Zou, and R. Amor, “Applications of machine learning to BIM: A systematic literature review,” *Adv. Eng. Inform.*, vol. 51, p. 101474, Jan. 2022, doi: 10.1016/j.aei.2021.101474.

[57] “College of Engineering & Applied Science,” *College of Engineering & Applied Science*. https://www.colorado.edu/engineering/ (accessed Mar. 15, 2022).

[58] PlanGrid, “The History of Blueprints,” *PlanGrid Construction Productivity Blog*, Apr. 12, 2016. https://blog.plangrid.com/2016/04/the-history-of-blueprints/ (accessed Mar. 15, 2022).

[59] E. L. Flavell, “Master Builder: Historical Icon?,” *Leadersh. Manag. Eng.*, vol. 11, no. 2, pp. 78–79, Apr. 2011, doi: 10.1061/(ASCE)LM.1943-5630.0000105.

[60] V. (Vincenzo M. Emprin-Gilardini, “Master builder of the Middle Ages and design build of today : an analysis and comparison,” Thesis, Massachusetts Institute of Technology, 2000. Accessed: Mar. 15, 2022. [Online]. Available: https://dspace.mit.edu/handle/1721.1/8976

[61] “(200) Pinterest,” *Pinterest*. https://www.pinterest.it/pin/329536897704842012/ (accessed Mar. 15, 2022).

[62] “Prevailing winds - Energy Education.” https://energyeducation.ca/encyclopedia/Prevailing_winds#cite_note-env-1 (accessed Mar. 16, 2022).

[63] “Reinforcement Learning via Markov Decision Process,” *Analytics Vidhya*, Nov. 28, 2020. https://www.analyticsvidhya.com/blog/2020/11/reinforcement-learning-markov-decision-process/ (accessed Mar. 17, 2022).

[64] H. com Editors, “Pantheon,” *HISTORY*. https://www.history.com/topics/ancient-greece/pantheon (accessed Mar. 18, 2022).

[65] “Pantheon,” *Pantheon Rome*. https://www.pantheonroma.com/en/pantheon-history/ (accessed Mar. 18, 2022).

[66] “Pantheon di Roma: 3 curiosità che non tutti conoscono!,” *Ristorante Grano*, Jun. 29, 2016. https://www.ristorantegrano.it/blog/pantheon-roma/ (accessed Mar. 18, 2022).

[67] G. R. H. Wright, *Chapter Seven: Roman Concrete Construction*. Brill, 2009, pp. 269–283. doi: 10.1163/9789047430896_008.

[68] “The Pantheon by David Moore.” http://www.romanconcrete.com/docs/chapt01/chapt01.htm (accessed Mar. 18, 2022).

[69] “Il Pantheon, storia e segreti,” *ItalyGuides.it*. https://www.italyguides.it/it/lazio/roma/antica-roma/pantheon (accessed Mar. 18, 2022).

[70] “rotunda | architecture | Britannica.” https://www.britannica.com/technology/rotunda-architecture (accessed Mar. 18, 2022).

[71] R. Mark and P. Hutchinson, “On the Structure of the Roman Pantheon,” *Art Bull.*, Aug. 2014, Accessed: Mar. 18, 2022. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00043079.1986.10788309

[72] “Principles of Roman Architecture: Wilson Jones, Mark: 9780300102024: Amazon.com: Books.” https://www.amazon.com/Principles-Roman-Architecture-Wilson-Jones/dp/030010202X (accessed Mar. 18, 2022).

[73] T. A. Marder and M. W. Jones, *The Pantheon: From Antiquity to the Present*. Cambridge University Press, 2015.

[74] “Gramazio Kohler Research.” https://gramaziokohler.arch.ethz.ch/web/e/forschung/216.html (accessed Mar. 18, 2022).

[75] “Unity (game engine),” *Wikipedia*. Mar. 11, 2022. Accessed: Mar. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1076485156

[76] U. Technologies, “Unity - Manual: Physics.” https://docs.unity3d.com/Manual/PhysicsSection.html (accessed Mar. 21, 2022).

[77] “Anyone can use NVIDIA’s physics simulation engine,” *Engadget*. https://www.engadget.com/2018-12-03-nvidia-physx-open-source.html (accessed Mar. 21, 2022).

[78] *Unity ML-Agents Toolkit*. Unity Technologies, 2022. Accessed: Mar. 21, 2022. [Online]. Available: https://github.com/Unity-Technologies/ml-agents

[79] M. Saroufim, "Building your own game simulations for Reinforcement Learning with Unity ML agents — A code deep…," *Medium*, Oct. 30, 2019. https://marksaroufim.medium.com/building-your-own-game-simulations-for-reinforcement-learning-with-unity-ml-agents-a-code-deep-e69a7bbc601e (accessed Mar. 21, 2022).

[80] U. Technologies, "Unity - Manual: Supported platforms." https://docs.unity3d.com/2018.2/Documentation/Manual/UnityCloudBuildSupportedPlatforms.html (accessed Mar. 21, 2022).

[81] N. Vishwakarma, "How Unity Supports Cross Platform Feature," *Medium*, Apr. 20, 2018. https://niraj-vishwakarma.medium.com/how-unity-supports-cross-platform-feature-ae722321cfa (accessed Mar. 25, 2022).

[82] "KMR QUANTEC," *KUKA AG*. https://www.kuka.com/en-de/products/mobility/mobile-robots/kmr-quantec (accessed Mar. 25, 2022).

[83] blackburn, "Introduction to Reinforcement Learning : Markov-Decision Process," *Medium*, Jul. 25, 2021. https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da (accessed Mar. 31, 2022).

[84] EBatlleP, *Català: Diagrama d'un procés de decisió de Markov en aprenentatge per reforç basat en una figura del llibre "Reinforcement Learning An Introduction" segona edició de Sutton and Barto.* 2020. Accessed: Mar. 31, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Markov_diagram_v2.svg

[85] "Proximal Policy Optimization — Spinning Up documentation." https://spinningup.openai.com/en/latest/algorithms/ppo.html#quick-facts (accessed Mar. 31, 2022).

[86] R. M. Kretchmar, "Parallel Reinforcement Learning," 2002.

[87] "Class Agent | ML Agents | 1.0.8." https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Agent.html (accessed Mar. 31, 2022).

[88] "Class DecisionRequester | ML Agents | 1.0.8." https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.DecisionRequester.html (accessed Apr. 02, 2022).

[89] "Curiosity-driven Exploration by Self-supervised Prediction." https://pathak22.github.io/noreward-rl/ (accessed Apr. 12, 2022).

[90] AlexZou, *Unity ML-Agents Toolkit (Beta)*. 2020. Accessed: Apr. 08, 2022. [Online]. Available: https://github.com/gzrjzcx/ML-agents/blob/476504b547b39e0bd6974d4b2951dd0e97c95f79/docs/Training-PPO.md

[91] B. Hengst, “Hierarchical Reinforcement Learning,” in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 495–502. doi: 10.1007/978-0-387-30164-8_363.

Appendices

Appendix A

This Appendix includes video links for the RoBuilDeR:

1. Human Version Explanation:

   https://youtu.be/banTFV1uat8

2. Robot Version Training Steps

   https://youtu.be/CaVdg1k1oTQ