BARC0141: Architectural Computation Dissertation

"Deep Reinforcement Learning for Training Intelligent Agents in Cooperative Housing Environments Using Decentralised Agents and a Central-Critic"

by

David de Miguel Vidal

19165952 12/09/202

Word count: 9986

Final Project submitted in part fulfilment of the Degree of MSc Architectural Computation

Bartlett School Architecture
University College London

MSc/MRes Adaptive Architecture and Computation Dissertation Submission Form

A signed and dated copy of the following MUST be inserted after the title page of your dissertation. If you fail to submit this statement duly signed and dated, your submission will not be accepted for marking.

1. DECLARATION OF AUTHORSHIP

I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.

I certify that the work submitted is my own and that it has been also submitted electronically and that this can be checked detection service, Turnitin®.

I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

2. COPYRIGHT

The copyright of this report remains with me as its author. However, I understand that a copy may be given to my funding body (alongside limited feedback on my academic performance). A copy may also be given to any organisation which has given me access to data and maps (if requested and if appropriate).

I also understand that a digital copy may be deposited in the UCL public access repository and copies may be available on the UCL library bookshelves.

Please write your initials in the box if you DO NOT want this report to be made available publicly either electronically or in hard copy.

Name: David de Miguel Vidal

Signed: 

Date: 12 September 2021

Deep Reinforcement Learning for Training Intelligent Agents in Cooperative Housing Environments Using Decentralised Agents and a Central-Critic

David de Miguel Vidal

MSc Architectural Computation

Bartlett School of Architecture, UCL

September 2021

ABSTRACT

Multi-Agent Reinforcement Learning (MARL) has enjoyed great success in decision-making domains such as robotics, games, or autonomous driving. Simultaneously, housing prices have continued to rise, contributing to social inequality. This thesis seeks to study the use of Deep Reinforcement Learning (DRL) methods for designing collaborative housing aggregations. It examines whether intelligent agents can cooperate and learn behaviours that create complex systems.

The research explores the training and evaluation phases, using game engines to compute interactive environments. Distilling housing problems into sets of simple rules, it prioritises urban parameters such as height, sun-light, protected green areas or floor-to-area ratio (FAR). Computationally, the training phase comprises several configurations, based on the novel MultiAgent POsthumous Credit Assignment (MA-POCA) that uses decentralised agents and a central critic. The resulting behaviours are evaluated using the same conditions and comparing to understanding the rules that yield more efficient housing aggregations. After training, key concerns are maximising occupancy ratios, FAR and time to compute aggregations.

Tools that enable decentralised mass housing facilitate access to high-quality, affordable homes for those excluded from homeownership. This study suggests an innovative use of MARL systems for collaborative housing where agents continuously interact to meet their briefs. The proposed computation allows for the implementation of additional rules and can be applied to other architectural problems.

Keywords: deep reinforcement learning, multi-agent, MARL, DRL, MA-POCA, collaborative, housing, decentralised, agents, central-critic, housing.

ACKOWLEDGMENTS

I would like to thank Tommaso Casucci and Sherif Tarabishy, the thesis and technical supervisors, for their continuous support, clear guidance, and generosity with their time. Without them, none of this would have been possible. I would also like to thank other ZHA members, specially Shajay and Vishu Bhooshan, for their input and facilitating discussions with Mubbasir Kapadia, researcher and professor at Rutgers University.

Special thanks to the team at Atkins for all the personal and technical support during the past two years. First, to Declan O'Carroll, National Practice Director, and Paul Medhurst, National Operations Director, for their understanding and facilitating time and availability. Thanks to Caroline Paradise, Head of Design Research, and Babak Tizkar, Head of BIM and research counsellor, for providing guidance and discussing the practical and philosophical implications of this work. Also, thanks to Stephen Workman, Head of Residential and long-time mentor.

Finally, thanks to the other professors and students of AC who helped me throughout the last two years. I have been lucky to share with you this experience at The Bartlett. I would like to also mention the continuous support of family and friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. Background

Artificial intelligence (AI) and the use of deep learning (DL) methods are relatively new domains in architectural research. They create opportunities for significant change in the discipline (Chaillou and Gsd 2019). However, planning and design problems are wicked, lacking definite answers (Rittel and Webber 1973). DL entails accurate datasets whilst the design process is complex and subjective, compromising the gathering of reliable data (Harputlugil et al. 2014). Unlike other DL, deep reinforcement learning (DRL) does not need existing data, training from its interaction with the environment (Goodfellow, Bengio, and Courville 2016). Thus, making architectural problems highly suitable for DRL methods.

Since the 1970s, housing is a primary amplifier of social inequality, affected by hyperinflation. Housebuilders operate on a build-to-sell basis with social providers following similar principles, thus providing packaged products (Parvin et al. 2011). Some argue that building new homes should be a participatory bottom-up process with future occupants involved from inception (Turner and Fichter 1972). In addition, the idea of modularity has been part of architectural housing efforts for almost a century. In the last decade, resources are driven towards the digital implementation of discrete assemblies (Retsin 2016). This is particularly relevant to automating the construction of new homes, good illustrations are Urbansplash and Katerra ("Urban Splash" 2021; "Home | Katerra" 2021).

Previous computational research for mass- housing include Shape Grammars (Shekhawat et al. 2019; Garcia and Menezes Leitão 2018; Correia et al. 2012), Cellular Automata solutions (Dounas et al. 2017), graph-based algorithms (Heckmann et al. 2020) or Generative Neural Networks (Chaillou 2019). This thesis specifically draws inspiration from machine learning approaches, using multi-agent decision-making systems. Recent examples are the exploration of assemblages through wave function collapse and reinforcement learning (Mintrone and Erioli 2021), optimisation of the orientation of the urban blocks using sunlight-driven criteria (Cocho-Bermejo and Navarro-Mateu 2019), or intelligent generation of layouts (C. Cai, Tang, and Li 2019).

B. Motivation

Academic studies in the field of video games have their origin in the late 1990s and early 2000s. In addition, the earlier Game Theory is heavily influenced by the work of mathematicians such as John von Neumann, Alan Turing, or John Nash and refers to mathematical models that study decision-making processes (Wolf 2021). Both disciplines laid the foundations of recent achievements in the field of DRL. For instance, AlphaZero

achieves super-human performance at general games, according to its authors, beating world-champions and previous programs at chess, shogi and Go (Silver et al. 2017).

Game development tools provide access to computational frameworks and often provide 3D representation, physics simulation or DRL toolkits (Juliani et al. 2018; Unity Technologies 2021). In architecture, this is essential for computational simulations in virtual environments such as the present thesis.

On the other hand, there is a longstanding shortage of affordable quality housing (Heffernan and Wilde 2020). Three-quarters of all new homes are built by private initiatives, seeking their own agenda and short-term gains. In this context, self-provided housing can deliver higher-quality homes that are affordable, sustainable and more resilient (Parvin et al. 2011).

C. Structure of the Thesis

The thesis is organised as follows. Chapter II provides research precedents and the theoretical aspects regarding Game Theory and Reinforcement Learning, introducing the concepts that are most relevant to understand the logic applied in this research. Chapter III covers the initial testing and the implementation of the housing agents and a detailed description of the training configuration. This is followed by Chapter IV that provides an evaluation of the training and results obtained. Finally, chapter V discusses the results and the implications of this exercise, explaining its limitations and potential future work.

D. Research Question and Objectives

This thesis seeks to discuss the advantages of adopting novel methods based on multi-agent reinforcement learning systems for cooperative self-build housing. A research question underpins its methodology: can the process of designing these housing aggregations be distilled into simple rules that allow intelligent agents to learn behaviours?

To answer the above, this thesis seeks to assess the application of such multi-agent systems in the domain of collaborative housing. The agents continuously interact with each other to meet their housing brief that results in assemblies of discrete elements that arise from examples of real layouts.

This thesis proposes the exploration of housing design solutions where non-human intelligent agents are embedded in the decision-making process using game engines and DRL. However, the approach here discussed is easily applicable to other architectural problems.

It further tests methods for planning and design considerations such as height policies, high-quality communal spaces, and green areas. According to Mercer, cost and amount of quality public space are key factors that make their Quality of Living Index (Mercer 2021). The previous research and computational tools studied place a strong emphasis on optimising the built parts of the housing aggregations. This research seeks to fill this gap, focusing on protecting the quality of communal green areas and public spaces.

In addition, the testing of game development tools such as the Unity engine is also part of the research objectives. It pursues to inspire further research in the field of architecture using their full potential engines for testing design solutions.

The image below shows the holistic workflow that underpins the proposed computation. A key aspect of collaborative self-build housing is that future dwellers participate and influence the design process since its inception. They can choose the housing program, sizes, views, amenities, and other requirements. This is then translated into layouts of discrete modular elements. These form the agents' brief, they are required to interact with the environment, receiving rewards according to their actions. Their goal is to assemble the parts that make up the brief whilst respecting the urban policies (sunlight, FAR, etc.) and maximising their cumulative reward until they reach a final state in equilibrium. This thesis focuses on the last two parts.

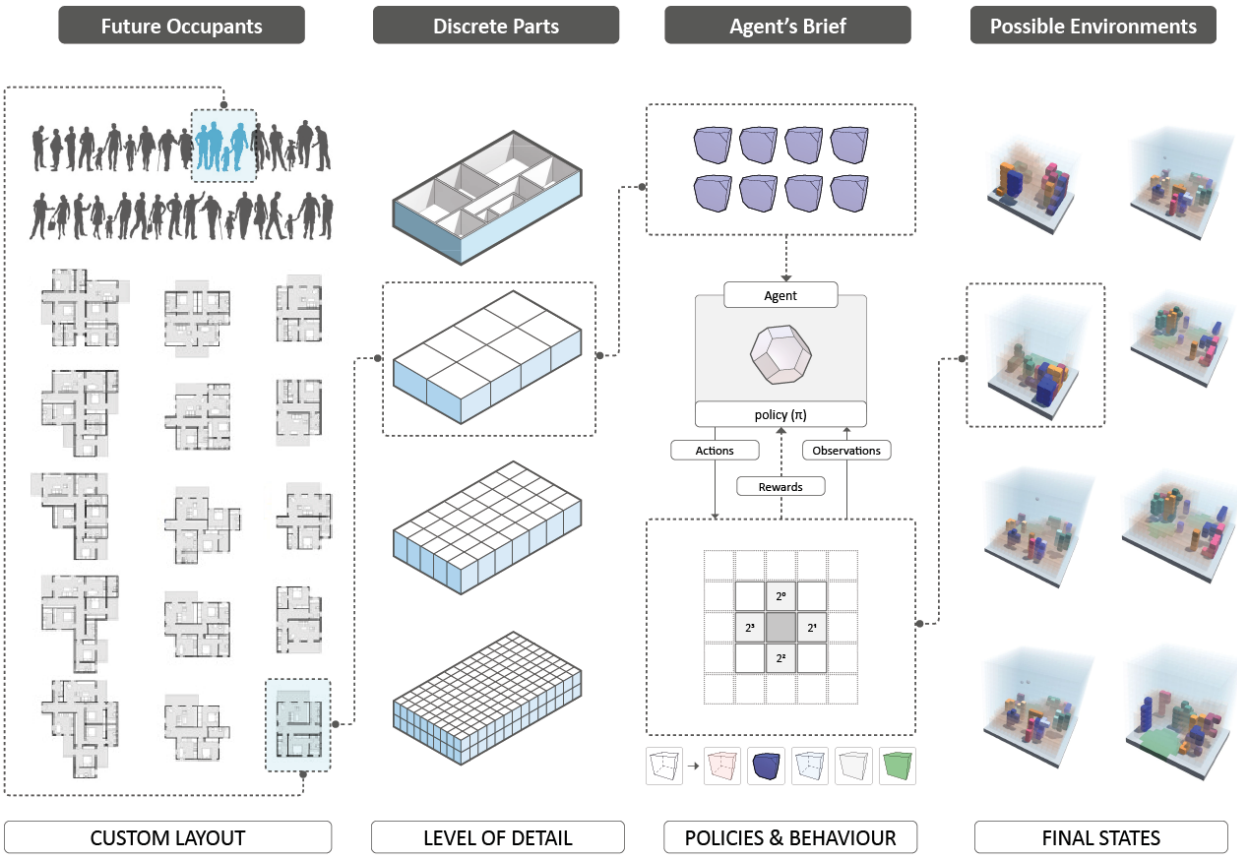
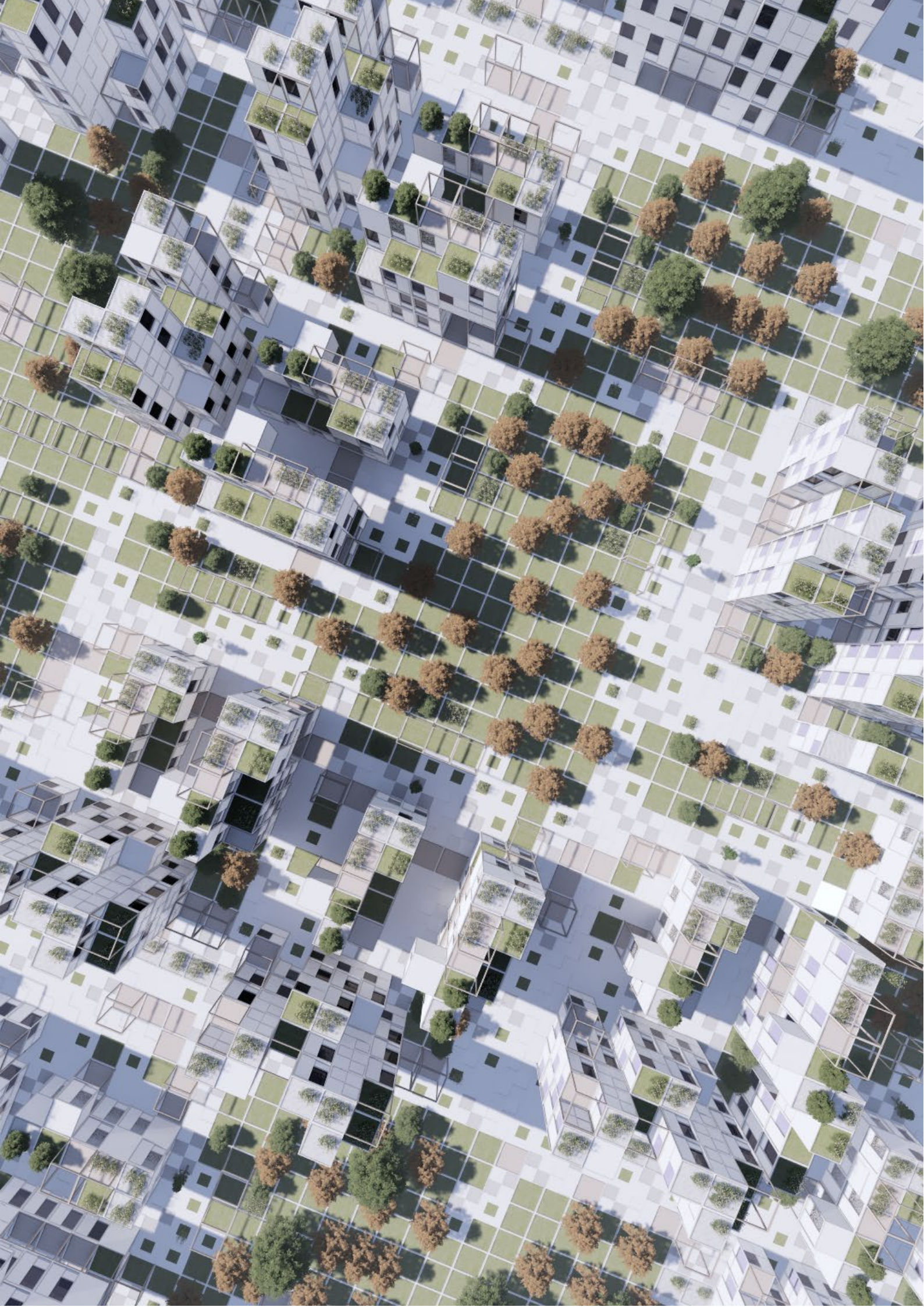


Figure 1: Overview of the holistic workflow.



II. LITERATURE REVIEW

A. General Concepts

1. Housing and Space Planning Tools

This project-thesis builds on previous efforts to automate the design of housing solutions. They generally follow the principle of co-creation, acting as digital tools that help designers create rapid testing and optioneering. They usually tackle the initial phases of the design where speed of testing is key.

Examples of this are TestFit, Archistar or Spacemaker. Likely, the most advanced is Prism, a tool commissioned by the Mayor of London and developed by Bryden Wood.

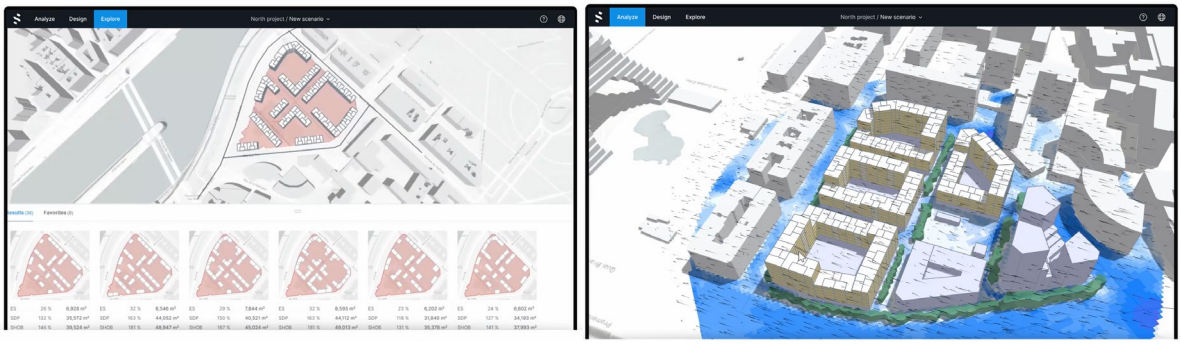


Figure 2: Spacemaker interface.

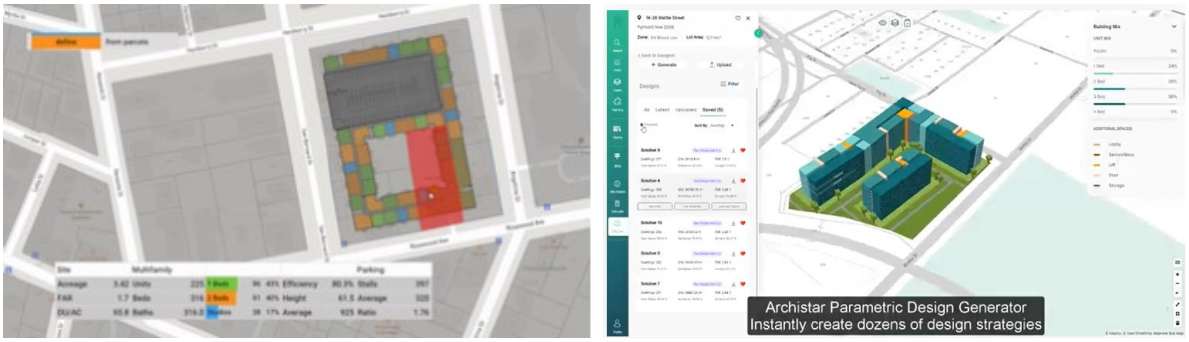


Figure 3: Interfaces for TestFit (left) and Archistar (right).

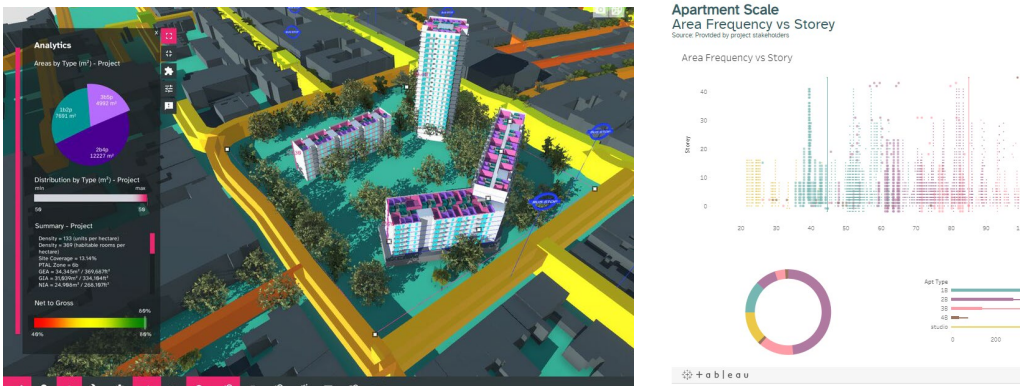


Figure 4: Interface and data analytics in Prism.

A key consideration in digital tools for space planning is the addition of modular fabrication. An example of this is the Roatán Próspera Residences (*Dezeen Magazine* 2020), a housing complex in Honduras by Zaha Hadid Architects. It provides a user interface where future occupants use to design custom housing configurations based on modular elements automatically translated into discrete parts ready for construction.

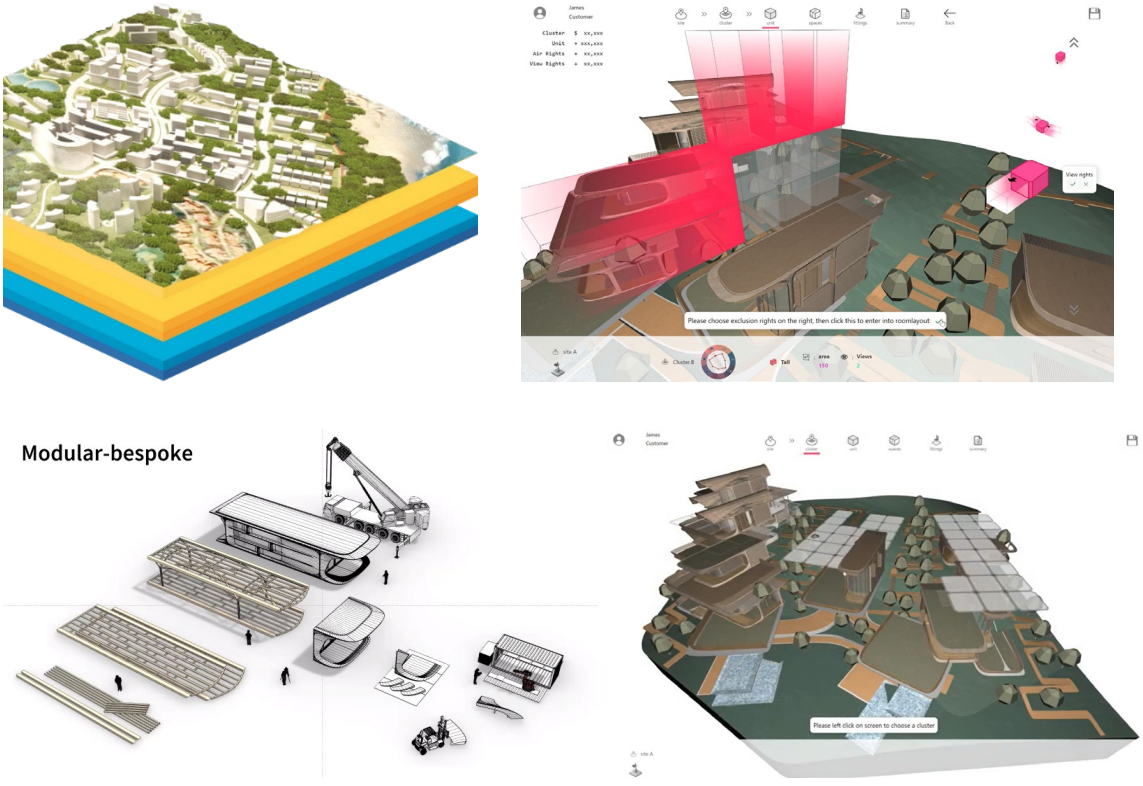


Figure 5: Several steps of the interface for the Roatán Próspera Residences.

2. Discrete and Modular Assemblies

Academic and real-life studies have explored bottom-up approaches for housing aggregations based on modular and discrete assemblies. Examples are Block Party (Bhooshan and Nahmad 2017) and Retsin's research on digital materials (Retsin 2016). Industry efforts are led by Katerra, a company that specialises in fabrication of discrete elements that at later factory-assembled and delivered on-site. These approaches enable construction automation, fast and efficient delivery, improved quality, sustainability gains, reduced embodied carbon and better energy-performance

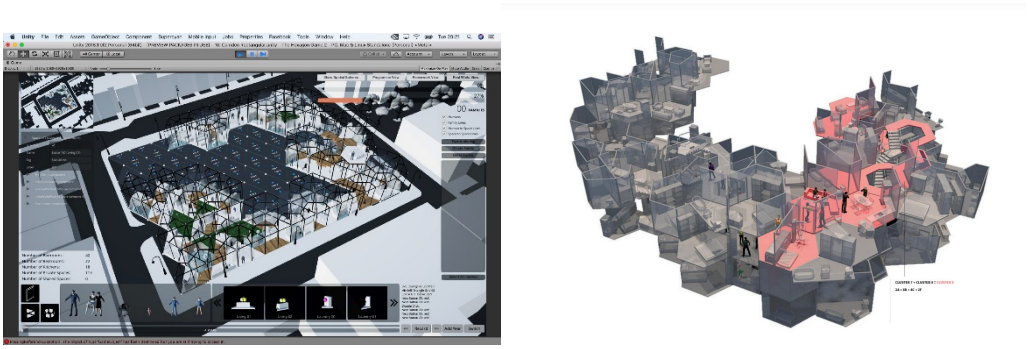


Figure 6: Block Party interface.

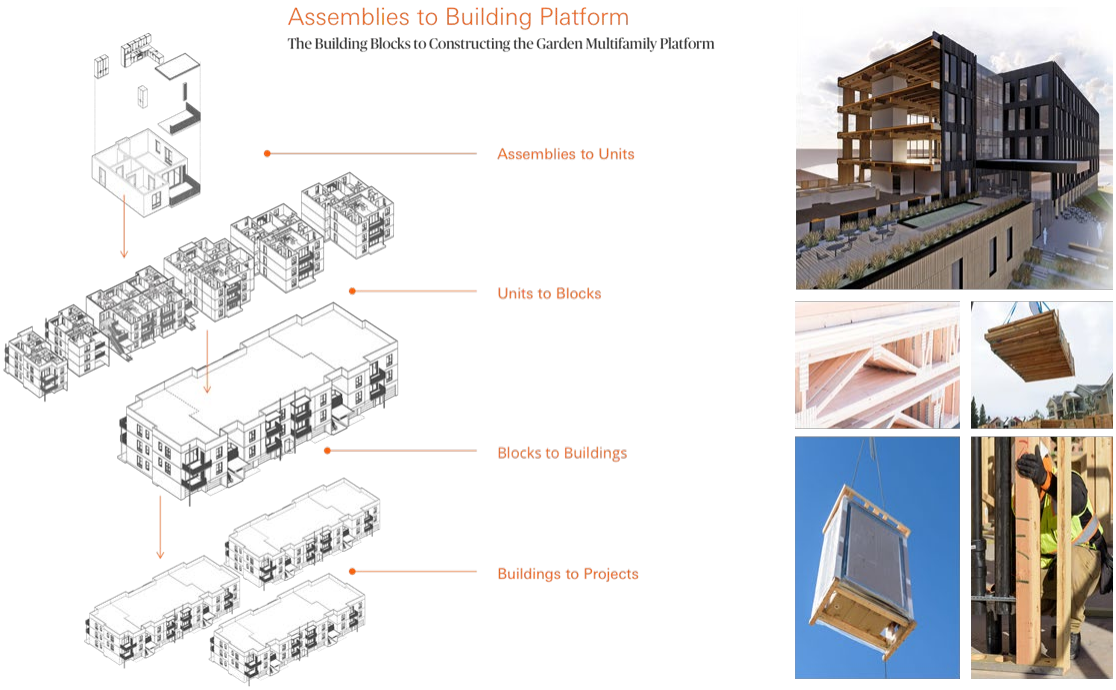


Figure 7: Extracts from Katerra's website.

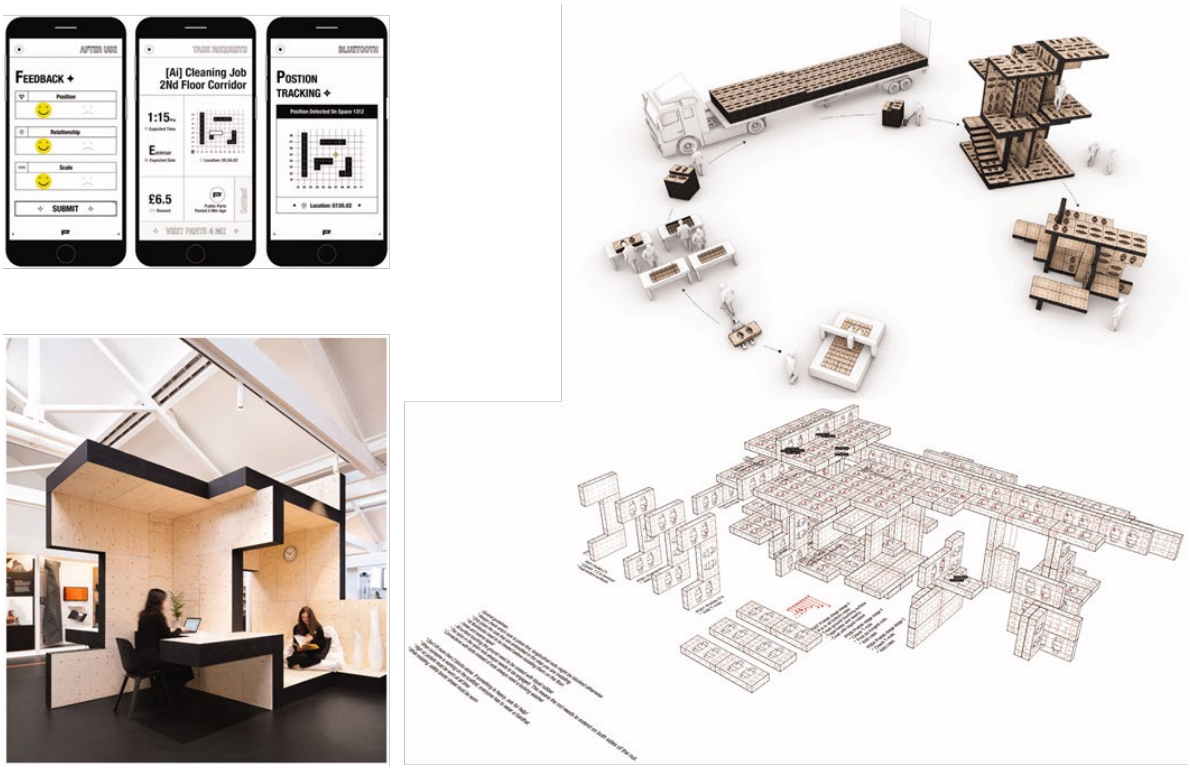


Figure 8: Extract from the Automation and the Discrete (Retsin 2016).

3. Simulated Ecologies Through Gaming

Game-based approaches and gamification provide clear opportunities to design and test solutions for the built environment. Many games, albeit mostly inadvertently, provide precedents of this approach. Similarly, architecture and the built environment have served as inspiration in many games. Perhaps, one of the clearest examples is Sim City, a well-known planning game to build a functional city with interdepend parts.

Figure 9: Snapshots of the Sim City gameplay interface.

Traditionally, computer science specialists have dominated game development. However, the simplification of game engines and their mainstream distribution means that other professionals gained access to such tools. An example of this is José Sánchez, architect, researcher and video game developer responsible for indie games such as the community-oriented Block' hood and resources driven Common'hood.

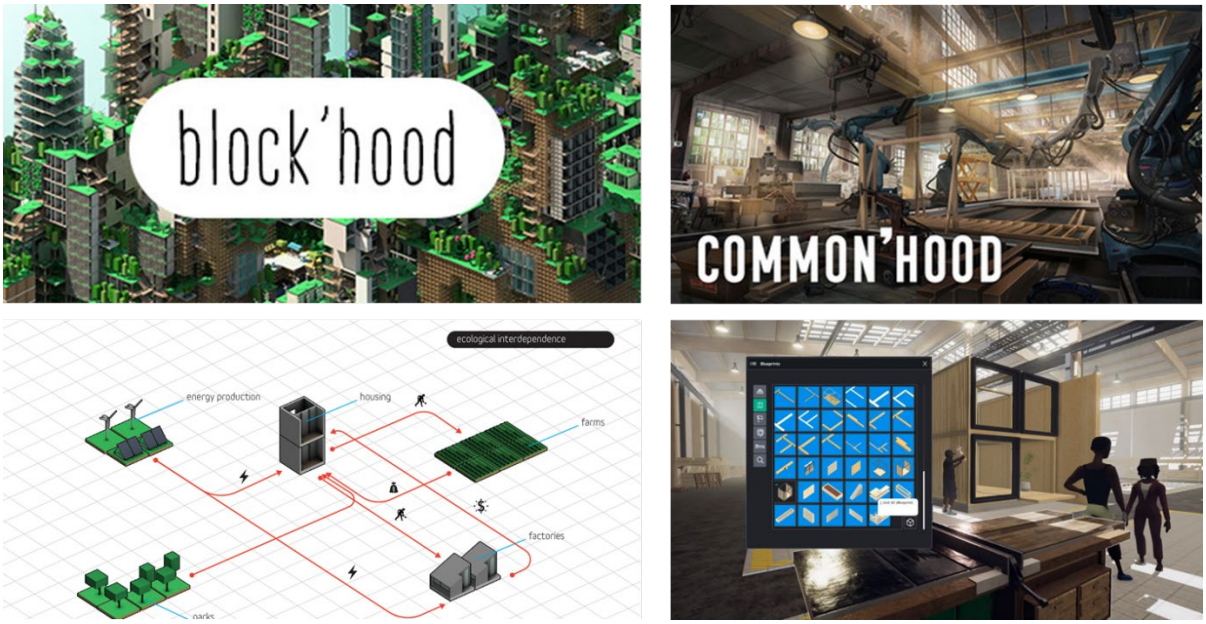


Figure 10: Snapshots of Block' hood and Common'hood.

Somewhere between simulated ecologies and real urban regeneration lies the charitable initiative Block-by-Block. Based on Minecraft and backed by the UN, it seeks to engage marginalised communities, especially children, with tools for visualising, collaborating and ultimately develop community-driven projects.

Figure 11: Several stages of the Block-by-Block initiative.

B. Game Theory Research

This chapter provides an overview of the decision planning and game theory algorithms assessed throughout the development of the proposed study.

1. Decision Planning Algorithms

Planning algorithms were used for motion planning, sequencing logical actions and for stability, feedback, and optimality. Also called Problem Solvers (PS), they comprise planning methods that obtain a sequence of actions leading to a goal state (LaValle 2006).

In most cases, this is a space of finite states. An example is the moves on a 2D grid, including other discrete problems such as labyrinths, puzzles, or Rubik's cube.

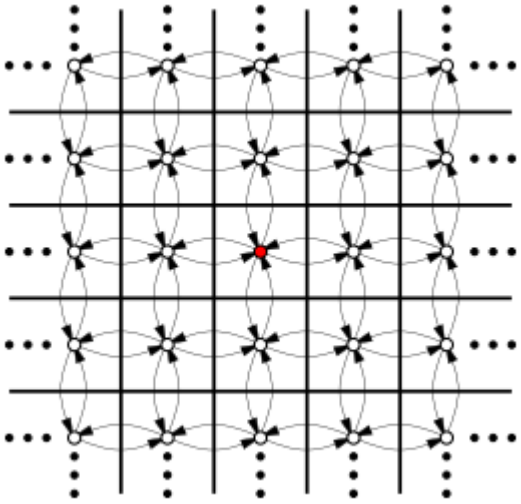


Figure 12: The state transition graph (LaValle 2006).

2. Game Theory and Game Trees

Game Theory models interactive decision-making situations. The decision-makers are players seeking to fulfil specific goals. This interactivity separates Game Theory from Decision Planning (Maschler, Solan, and Zamir 2020).

Game trees are the most common representation of games. They are directed graph where each vertex corresponds to a state of the game. The initial states are called roots and, the game terminations are leaves. Every vertex that is not a leaf needs to specify the player acting next.

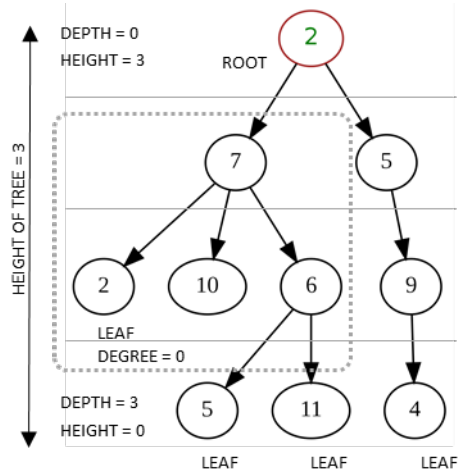


Figure 13: Main parts of a tree.

3. Competitive Games

Games with opponents – chess, checkers Othello or, Go – are generally deterministic and observable, meaning every player knows the entire state (Ertel 2011). Every two-player game finite and with perfect information can only have three outcomes: first player wins, second player wins or both draw (von Neumann and Morgenstern 1944). They are called zero-sum games because the gain of a player is the loss of the other.

An example is a Tic-Tac-Toe game where two players on a three-by-three board attempt to place three positions in a straight line. Identifying each player with an X or an O, they choose in turns from the squares not yet selected by any player. The game ends when a player meets the target, or all positions are selected.

Despite its simplicity, its tree is significantly large with 255,168 possible leaf nodes. More complex games exhibit exponentially complex problems and are not fully computable. Relevant to the topics explored, this thesis will study Tic-Tac-Toe and Minesweeper games for testing planning and spatial awareness.

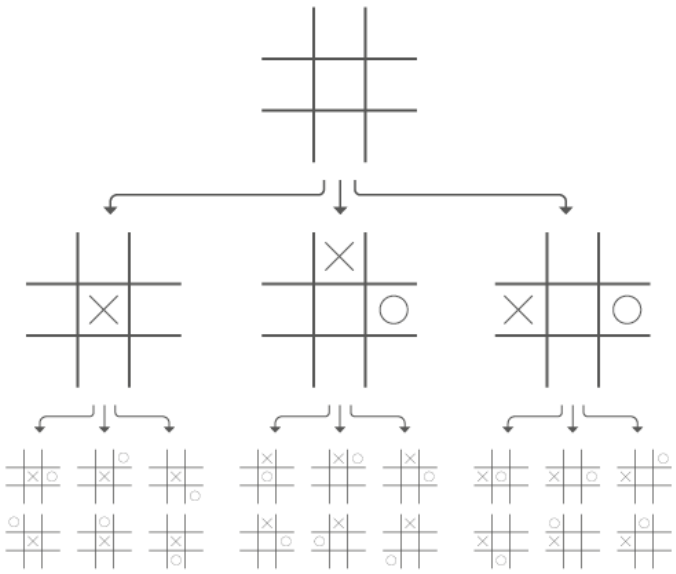


Figure 14: First three plies for Tic-Tac-Toe.

4. Minimax Search

Minimax search finds the best possible move, recursively from the end-states to the current one to determine the best move. It minimises a player's losses for a worst-case scenario, assuming the opponent always maximises its gains. It evaluates each state of the game, using a brute force approach. As an example, a Tic-Tac-Toe tree that exhibits depth 9 and 9! nodes has 255,168 leaves (Allis 1994).

The image below shows a minimax game with depth limited to four, MAX seeks the maximum reward and MIN is the lowest outcome for MAX.

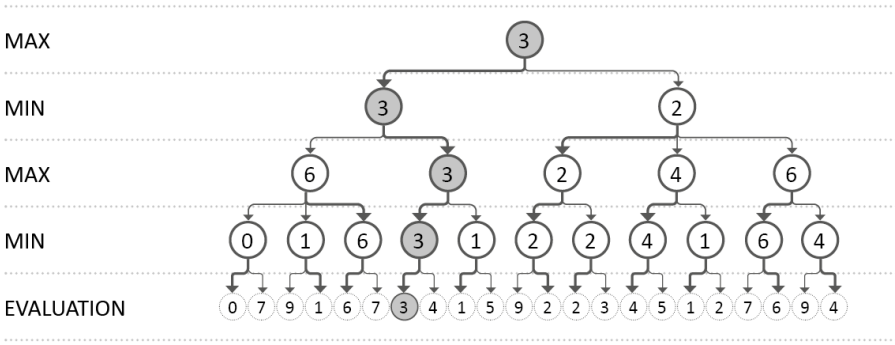


Figure 15: A minimax game tree with depth limited. (Ertel 2011).

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Figure 16: Pseudocode for a depth limited minimax.

5. Alpha-Beta-Pruning

Alpha-beta pruning is an improvement of minimax. It does not perform a search for branches that are not relevant to the result. Its worst possible performance is equal to minimax. As Ertel explains, the same computing power at chess achieves eight half-moves instead of six (Ertel 2011).

The image below shows an alpha-beta pruning tree with depth limited to four. The blue nodes are not traversed as are not relevant.

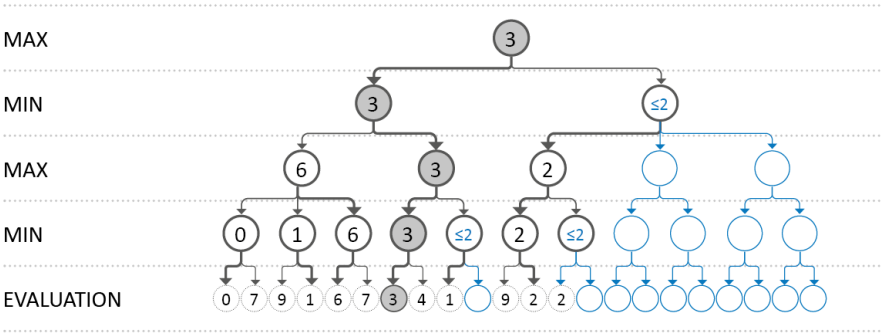


Figure 17: Alpha-beta game tree (Ertel 2011).

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            if value ≥ β then
                break (* β cutoff *)
            α := max(α, value)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            if value ≤ α then
                break (* α cutoff *)
            β := min(β, value)
        return value
```

Figure 18: Pseudocode for depth limited alpha-beta pruning.

6. Non-deterministic

Non-deterministic systems present an unreal player to model uncertainty. Called nature, it acts randomly whilst other players don't know what actions will choose. If players gather sufficient information, the approach is called probabilistic and, prior to choosing actions, they perform observations (LaValle 2006). The previous algorithms are applicable introducing randomness. For instance, minimax for backgammon looks like this: MAX, dice, MIN, dice, …

7. Cooperation, Stability and Nash Equilibrium

Non-competitive situations exhibit the principle of stability. The Nash equilibrium is the strategy at which no player has a profitable deviation, meaning that achieves the highest payoff for all (Maschler, Solan, and Zamir 2020). A simple illustration is the Prisoner's Dilemma, shown in the image below. It depicts the prison years for each player with D standing for Defection – player confesses the crime – and C for Cooperation - cooperates with the fellow criminal. Below, the Nash equilibrium is the strategy C-C.

		Player II	
		D	*C*
Player I	*D*	6, 6	0, 10
	C	10, 0	1, 1

Figure 19: Representation of the Prisoner's Dilemma.

Recent research suggests that combining competition and coordination is also possible. Through repetition, reaching stability is reasonable and expected in this type of game. Cai and Daskalakis' research proposes that strictly competitive games can also converge to Nash equilibrium, thus mimicking coordination games. (Y. Cai and Daskalakis 2011). This is relevant for collaborative housing aggregations.

8. Conclusions on Game Theory

Minimax and alpha-beta pruning assume zero-sum games where gains and losses are in an equilibrium that results from strict competition. Many argue that these algorithms can be limiting. The use of worst-case scenarios might be too conservative. Also, expecting players to behave rationally is not realistic (LaValle 2006). An irrational move at the start might cause subsequent volatility or sensitivity to initial conditions.

Furthermore, in non-competitive games, the Nash equilibrium does not always provide good predictions. This is the case of the repeated Prisoner's Dilemma or the Centipede games (Maschler, Solan, and Zamir 2020). a drawback for further applying this method to this research.

Another disadvantage to the above methods is that they are only viable for limited depths. This holds for current computers. In most cases, this factor forces the evaluations to depend on handcrafted functions, hard to fine-tune. An example is the chess computers used in the decades before AlphaZero (Silver et al. 2017).

This chapter has provided some background concepts to understand issues surrounding the modelling of mass-housing aggregations. However, they cannot deal with more complex modelling such as the increasing complexity derived from a large number of players. Furthermore, the evaluation of most spatial games so far explored is limited to 2D environments. Designing successful housing will require higher dimensionality in 3D or 4D environments. This will increase the complexity even further.

In conclusion, decision planning algorithms and game theory provide systems specific to their domains. Generalization is often not possible unless significant human effort is spent. For the above reasons, they remain comparatively weak as general problem-solving systems (Silver et al. 2017) and are considered not the preferred option for this research.

Nevertheless, their methods and the terminology they used is essential to understanding the state-of-the-art in the field of games and decision-making. The next chapter will explore that some of these in combination with deep convolutional neural networks provided the basis for the algorithms for this research, including Deep Reinforcement Learning such as PPO and MA-POCA.

C. State of the Art in Game Theory

1. An overview of Reinforcement Learning

RL has been part of the Artificial Intelligence toolkit from its early days in the 1950s. For decades, researchers favoured supervised and unsupervised learning methods. In 2016, when DeepMind won the world champion Lee Sedol at Go, this prompted a new wave of research in RL (Mitchell 2019).

Placed between supervised and unsupervised (Otterlo and Wiering 2012), RL is employed to solve decision planning and similar problems with limited information from the environment. It uses a framework where the decision-maker is called agent, tasked with observing the environment before taking actions. It then obtains cumulative rewards (Hu et al. 2020). Agent and environment interact at discrete time steps t=0,1,2,3,4... with the agent receiving partial information at each step. Mathematically, the environment is a Markov decision process (MDP) (Sutton and Barto 2018).

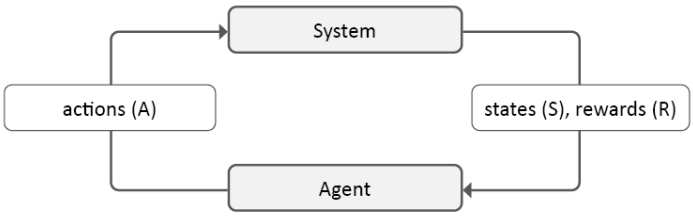


Figure 20: Agent to environment interaction in Markov Decision Process.

Similar to MDPs the key ingredients for RL frameworks are the environment and the environment states called states space or **S**, the agent, referred to as **A**, the probability **Pa** of transitioning from one state to another at a given time under an action **a** and, the potential reward after each transition known as **Ra**.

Probability of transitioning from one state to another:

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$

During the learning process, the RL agent needs to make modifications to its policy function π to ensure it learns the actions that maximise the cumulative reward. At each time t, the agent observes the present environment's state **st** and reward **rt,** then chooses the next action at, moving to a new state **st+1** and obtaining a reward **rt+1**.

The balance between exploration and exploitation is critical (Sutton and Barto 2018). Exploration implies random actions from the agent to learns new policies. Exploitation is reward maximisation following policies already learned. The agent's configuration must consider trade-offs between them (Russell and Norvig 2016).

2. Single-Agent and Multi-Agent Reinforcement Learning

Initially intended as single-agent MDPs, the same algorithms that apply to single-agent – value and policy methods – are relevant to multi-agent systems (Zhang Zhuoran Yang and Bas 2021) and this research.

Multi-agent reinforcement learning (MARL) addresses the sequence of decisions from several agents. Rewards the overall state of the environment depend on their joint actions. Often, the solution achieves a Nash equilibrium regardless of the setting being cooperative, competitive or, a mix of both. These are applicable to robotics, telecommunications, distributed control, etc. Multi-agent systems help to resolve complex computations and to represent decentralised systems (Nowé, Vrancx, and Hauwere 2012).

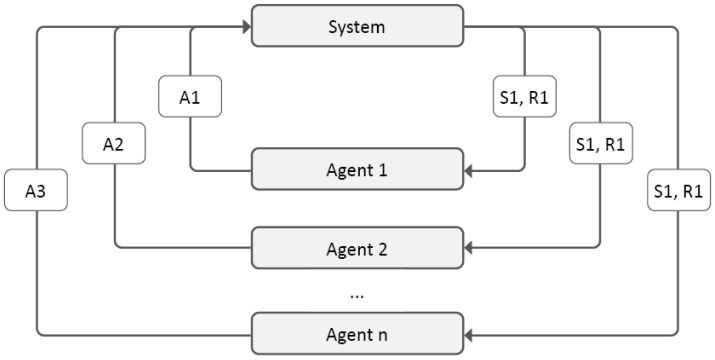


Figure 21: Multi-agents setting in a Markov Game.

3. Markov Decision Processes and Extensive-Form Games

The state representation of the environment has the Markov property if retains relevant information from past time steps to inform future ones. Then, it has the Markov property, formally defined as:

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\},$$

Where p is the probability of a reward and S represents states of the environment and A those actions chosen by the agent. We call this a Markov Decision Process (MDP). There are three main methods to resolve these: Dynamic Programming (DP), Monte Carlo Tree Search (MCTS) and Temporal Difference (TD). Researchers

consider TD methods more complex but allowing for partial information and sequential learning (Sutton and Barto 2018). This makes the latter better suited to the computation of housing agents that observe incomplete information about the environment and iteratively respond to conditions changing throughout time (Sutton and Barto 2018).

Games explained in the previous section are considered Markov games that require agents to observe the full game state. Real-life applications often involve partial observations, rendering Markov games as not practical (Zhang Zhuoran Yang and Bas 2021). In contrast, extensive-form games allow compute incomplete information and act as non-deterministic. They are relevant to this research. Agents exhibit perfect recall, remembering all actions leading to a state. In this setting, agents make decisions and perform actions (A), observe the system state (S) and receive individual rewards (R) at the terminal history. These depend on the actions of the other agents as per the below image.

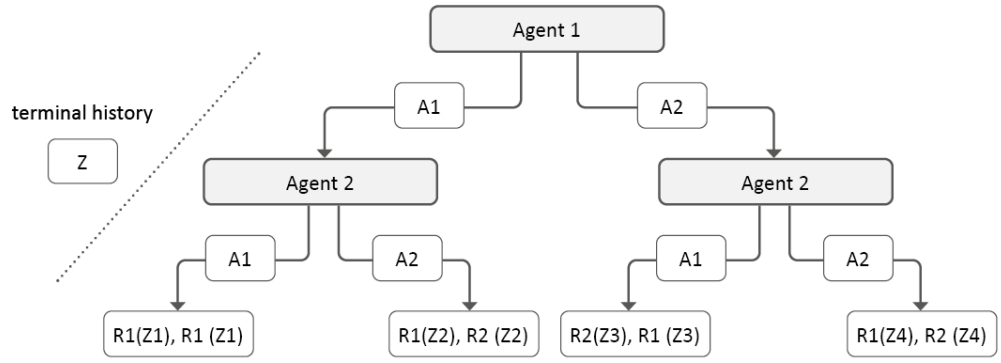


Figure 22: Extensive-form game diagram.

4. Action-Value, Policy Approximation and Actor Critic Methods

The majority of RL methods exhibit three key steps. They try to estimate value functions, then store information from previous decisions to create a model that serves decision-making and achieve new possible states. Finally, they interactively update value and policy functions, or state-action pairs, to continually align their response to the environment.

Value or action-value RL methods propose that the agent learns to assign values to the environment states. They represent the cost of choosing an action, also called the payoff. These values control the actions directly. Optimality is usually defined as those policies that exploit returns from initial states (Sutton and Barto 2018).

Alternatively, policy approximation directly searches the policy space. The best performing policy search method seems the actor-critic, dominating at Atari games and 3D navigation (Mnih et al. 2016). Actor-critic methods are TD methods, separating policy and value functions. The term critic arises from the fact that the

value function criticises the policy function, the actor since it selects actions. After the agent carries an action, the critic determines if it has led to a better or worse outcome calculating a TD error using the simplified formula:

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V(S_t),$$

Where t are the time steps, R the reward and V the value function implemented.

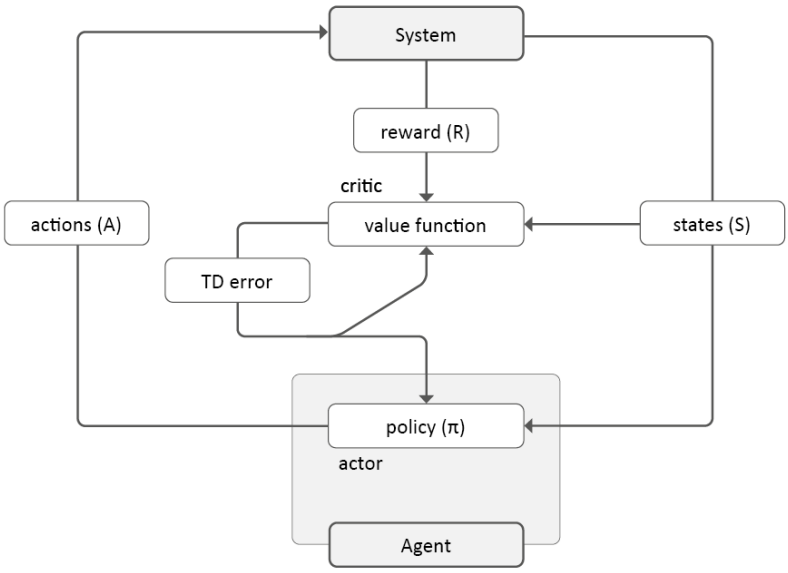


Figure 23: Actor-critic architecture.

5. Unity ML Agents

Since 2017, the Unity ML-Agents is freely available on Github. It is an open-source toolkit that allows researchers and developers the generation of simulated environments using the game engine Unity. The agents can be trained using mostly Deep RL built using Pytorch, an open-source machine learning library. The agents are generally non-playable characters (NPC). After the learning phase, their pre-trained neural network models can be used inside the Unity environment as the agents' brains. In addition, several training configurations are available from single to multi-agent. The later accepts cooperative, competitive and hybrid models (Unity Technologies 2021).

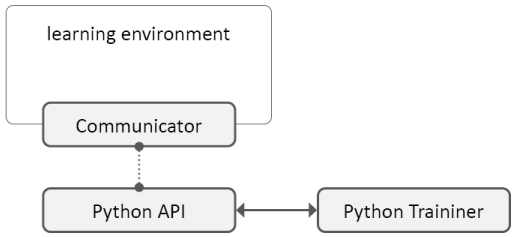


Figure 24: Simplified block diagram of ML-Agents.

The image below shows the implementation of a basic training scenario carried out as part of the initial testing for this research. The agent performs continuous moves towards its target (ball) without touching the platform edges. The platform colour varies from yellow for positive outcomes to red for negative ones. In addition, the image shows several environments in parallel to speed training.

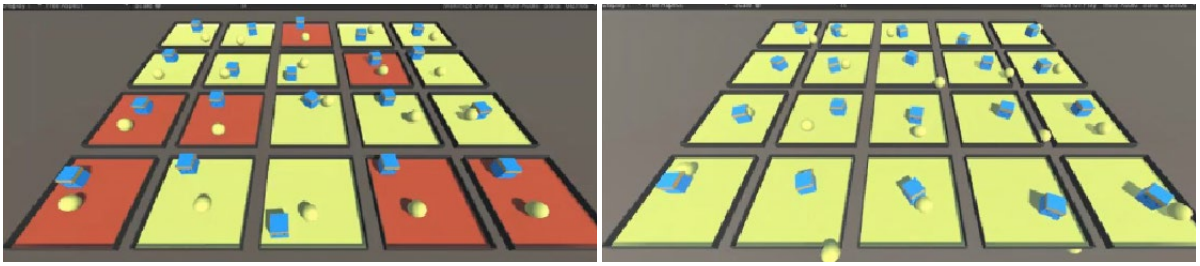


Figure 25: Basic testing of the ML-Agents Toolkit.

6. PPO and MA-POCA Algorithms

From the ML-Agents Toolkit two algorithms were explored. The Proximal Policy Optimisation (PPO) and the MultiAgent POsthumous Credit Assignment (MA-POCA), added in 2021 to the toolkit.

Generally used in single-agent models, the PPO is a policy gradient method that uses multiple epochs of stochastic gradient ascent at each policy update (Schulman et al., n.d.). This algorithm was successfully implemented during the first stages of testing the virtual environment.

On the other hand, exploration of the multi-agent domain was based on MA-POCA. A deep RL algorithm based on policy approximation and developed from two previous pieces of research proposed in 2017. One of them is the multi-agent model called Multi-Agent Deep Deterministic Policy Gradient Algorithm (MADDPG) that allows cooperation, competition, or a mix of both. It functions as a policy gradient procedure where individual agents are tasked with observing the environment and deciding the actions. The updates to their policies occur as a centralized critic that manages the ensemble. The authors argue that that policy ensembles perform better than single policies (Lowe et al. 2017).

The diagram below shows the relations between decentralised agents and a centralised critic.

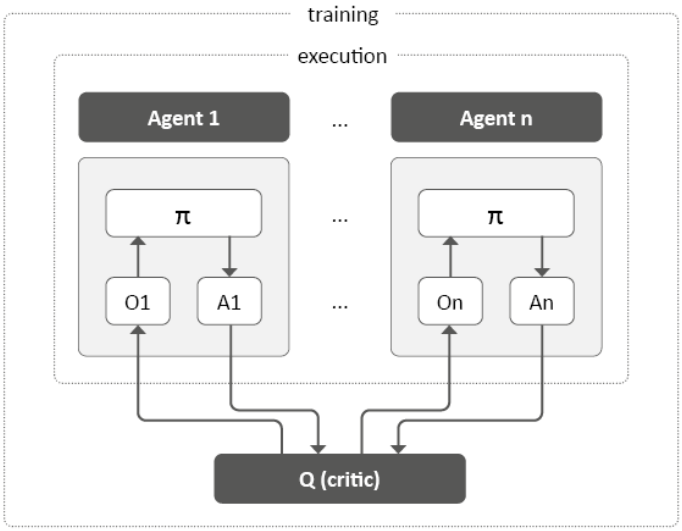


Figure 26: Decentralised agents and centralised critic.

The other influence for MA-POCA is counterfactual multi-agent (COMA) policy gradients, a model for cooperation that includes a central critic and Q-learning functions. For policy updates, the agents use a "counterfactual baseline that marginalises out a single agent's action, while keeping the other agents' actions fixed" (Foerster et al. 2017).

The image below is published by Foerster et al. Diagram (a) displays the interactions between decentralised agents, their environment, and centralised critic; all the red elements are useful only during training. Diagrams (b) and (c) explain the relationships between the agent and the critic

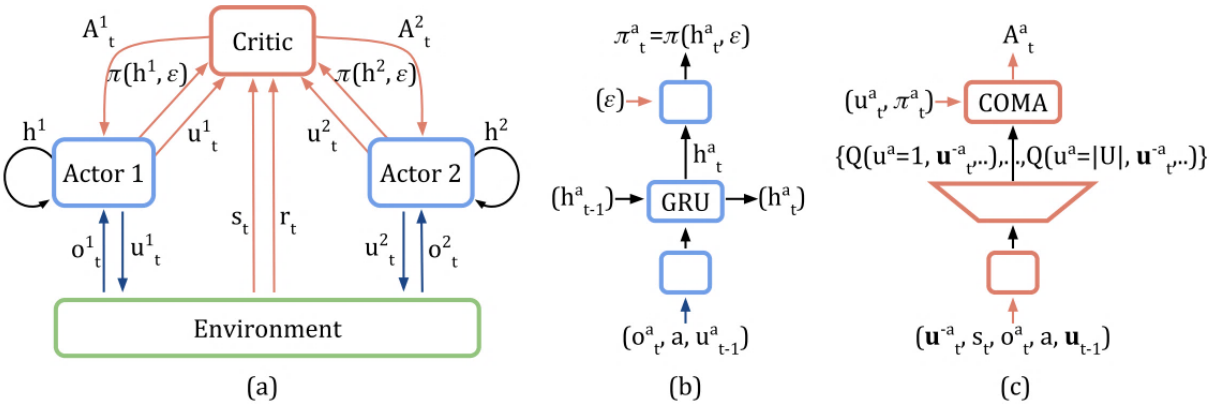


Figure 27: Diagram of the COMA algorithm (Foerster et al. 2017).

III. Methodology

A. Initial Testing: Spatial Games

Rhinoceros 3D and Grasshopper programming environment (McNeel and others 2010) served the computation in this section. The only exception is computation number 4 as Unity3D was better suited for gameplay and user-generated virtual environments.

1. Minimax Algorithm for Tic-Tac-Toe

As part of the initial experimentation, this research implemented the minimax algorithm for the game Tic-Tac-Toe. In this context, the solution provided by the minimax or alpha-beta pruning algorithm corresponds to Nash equilibrium. The image below shows different moves carried out by a human playing against an agent. The A represents the moves carried out by the human player and B those of the agent. The algorithm is allowed to explore all possibilities with no depth limiting applied. If the human plays efficiently, the best possible outcome is a draw against the agent.

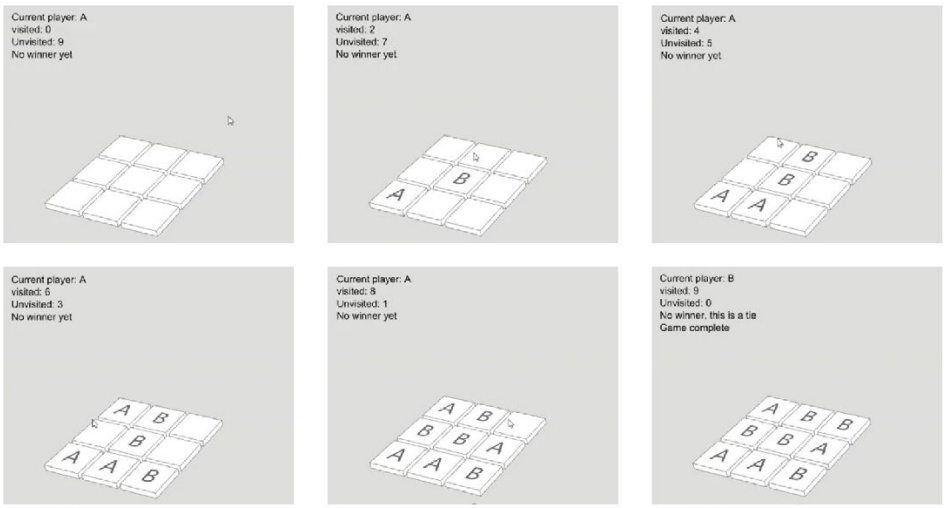


Figure 28: Tic-Tac-Toe game using the minimax algorithm.

2. Non-Deterministic Algorithm for Minesweeper

In this research, the game minesweeper has been implemented. The combination of uncertainty, incomplete observations that are limited to neighbouring positions and limited depth search made this game a good

choice to test solutions for non-deterministic problems. The implementation of the minesweeper game requires that each cell is aware and can display a numerical representation of its immediate neighbourhood.

The image below shows sequences of the implementation of a minesweeper game from initial to a losing state where the human player has hit a mine. The objective is to clear all the cells in the board without selecting any mines.

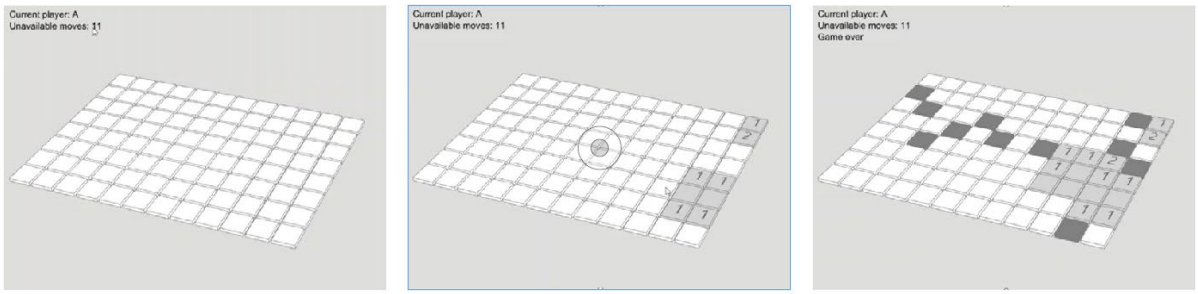


Figure 29: Minesweeper game.

3. Enabling Human-Player Spatial Competition

Another step prior to the computation of housing agents is the creation of a 2D board, capable of storing the moves of several players. The implementation monitors the number of cells occupied by each player and provides randomly distributed protected cells (in grey). These cannot be occupied by players, testing mechanisms of controlling open space within architectural aggregations. The computation also allows for players to occupy cells already occupied by others – except the initial positions shown in darker colours. In theory, this fluidity would allow future agents to reach stable states resembling Nash equilibrium. However, during the testing of large housing aggregations, it was clear that this approach is not compatible with large numbers of players and the need to preserve open space, often resulting in overcrowded scenarios and parts of housing units isolated from others. The implementation of this 2D environment served as the basis for more complex testing in 3D environments.

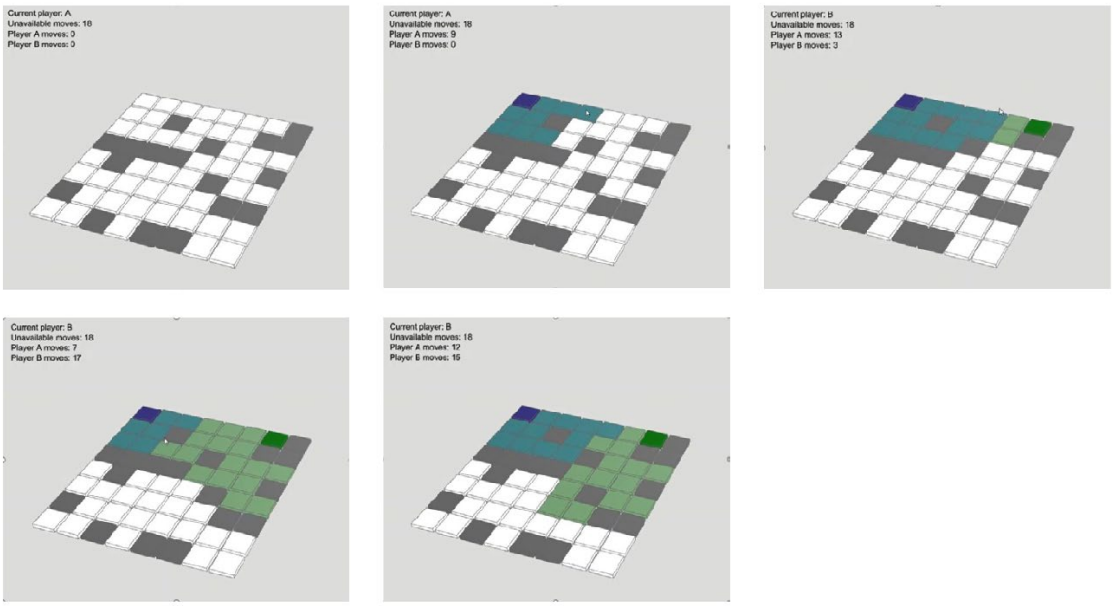


Figure 30: Spatial competition game.

4. First Person Shooter Aggregations

Inspired by the Gamescapes and Polynomio, two case studies from the design research initiative 'Plethora Project', carried out by José Sánchez at The Bartlett School of Architecture ("Plethora Project" n.d.). The below testing, carried out in the Unity environment, proposes the creation of landscapes combining architecture and game studies. It consists of a First-Person Shooter (FPS) that creates aggregations by shooting (left-clicking) or removing (right-clicking) units.

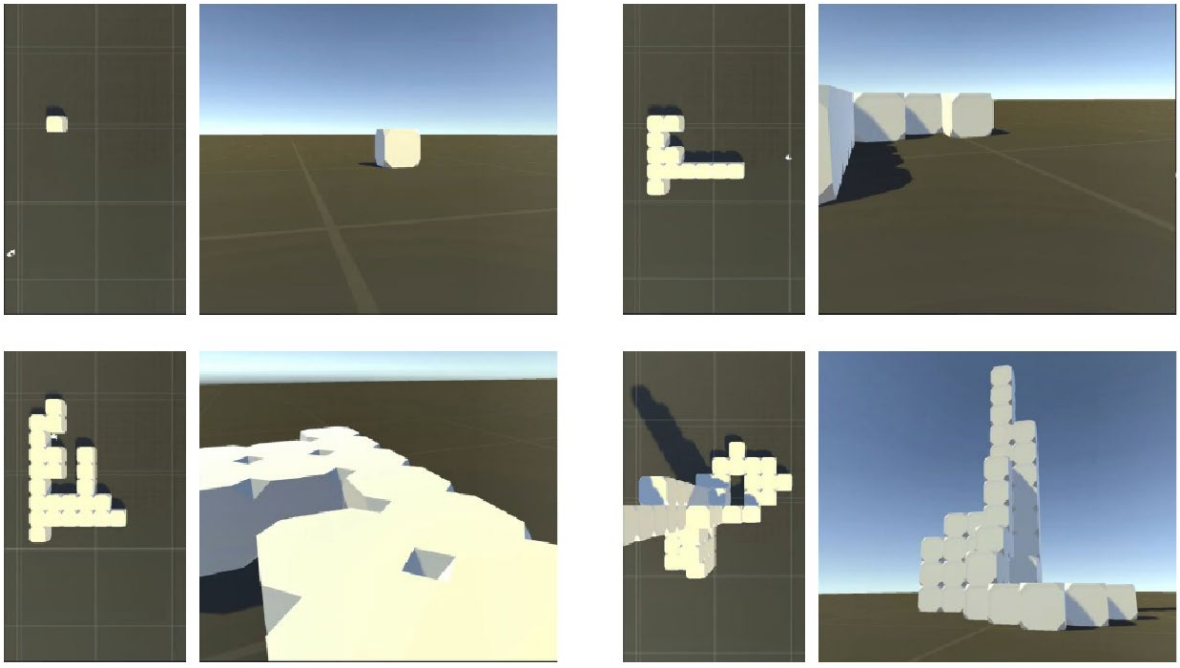


Figure 31: Testing an FPS aggregations.

B. Training Intelligent Housing Agents

1. Overview

The MA-POCA algorithm, provided by the Unity engine as part of its ML-Agents Toolkit. It added functionality for the objectives set by this research proposal. Other decision-making and planning algorithms have been considered and are included in the literature review.

The proposed computation is based on a three-dimensional grid environment where housing agents continuously interact with their environment and its policies. The housing agents must move and occupy the cells forming the plot to meet their objectives. Each one of them represents a housing unit with a specific target size, a simplification applied to this exercise. Potentially, more complex housing briefs could be applied, adding requirements from the future occupants.

The environment comprises the plot and policies such as the heightmap or areas designated for protected green open space. The objective of the environment is to maximise the housing density whilst protecting the environmental policies. It must preclude housing agents from overshadowing the green open spaces, ensure they respect the height policies and allow breathing space between different housing units. New agents are sequentially added while the percentage occupation remains below the limit.

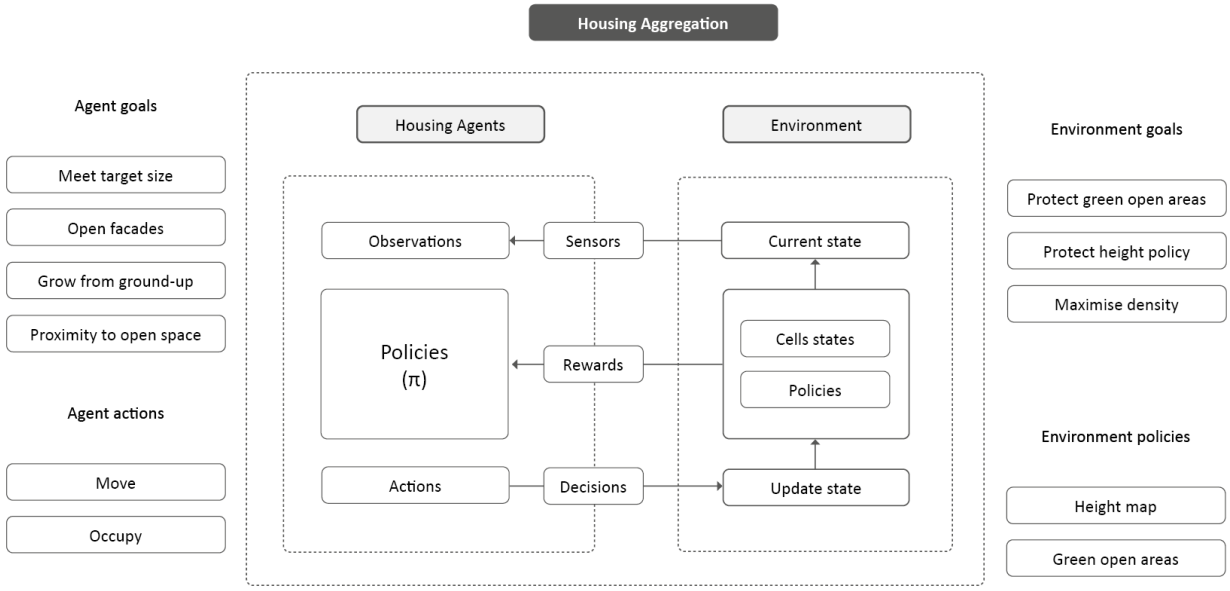


Figure 32: Training phase of the intelligent agents.

Software Considered

The simulation of MARL systems for collaborative housing has been implemented in the Unity game engine. Training of the agents used Unity's ML toolkit called, ML-Agents. It uses PyTorch, an open-source deep-learning library written in Python. The scripting of behaviours in Unity software is done via C# scripting. The connection to the Pytorch packages uses Anaconda's virtual environments but could be performed directly in python using the Command Prompt in the case of Windows machines.

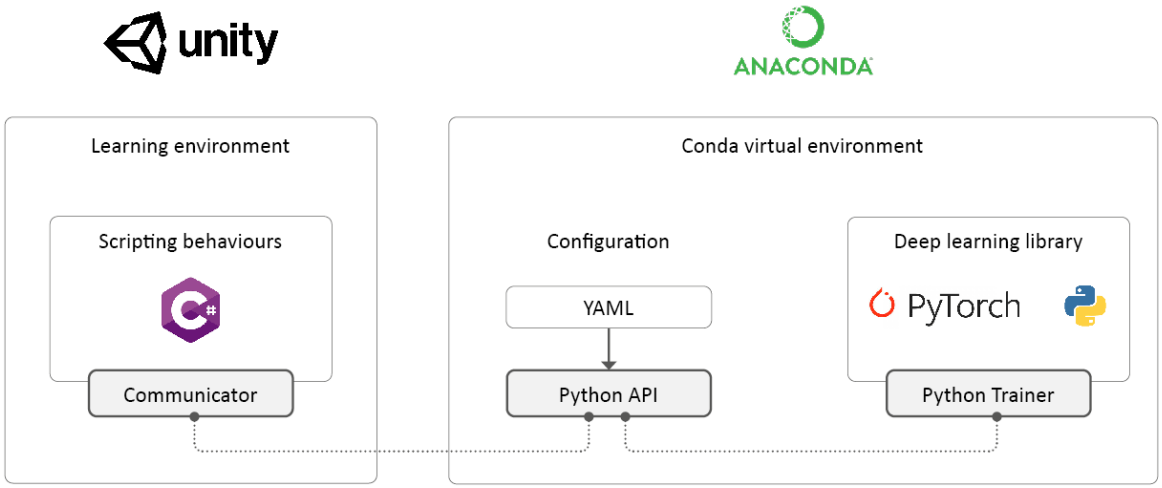


Figure 33: Interface between the software used.

2. Proposed Workflow and Dynamics

The training of the housing agents follows these steps. At the beginning of each episode, the computation sets up the environment. This step includes policies for protected open space and heights. It also adds the minimum number of agents from the start. These can move and occupy the available cells, first at random and throughout training, making intelligent decisions towards the most beneficial arrangements. The computation sequentially feeds agents to the environment until it reaches the occupation percentage. When it achieves this limit or carries out the maximum number of steps defined, generally set as 3,600, the training episode terminates and, a new one starts.

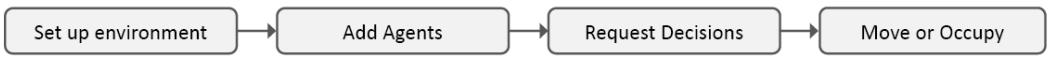


Figure 34: Basic steps carried out in a training episode.

The behaviour of the housing agents has three possible modes. Before training, the heuristic mode serves to test the environments and possible actions. During training, the MA-POCA algorithm makes decisions for each agent individually and adjusts its policies based on the environment rewards are given to the group containing all agents. After the training phase, the computation generates an Open Neural Network Exchange (ONNX) that is linked to each agent to perform inference.

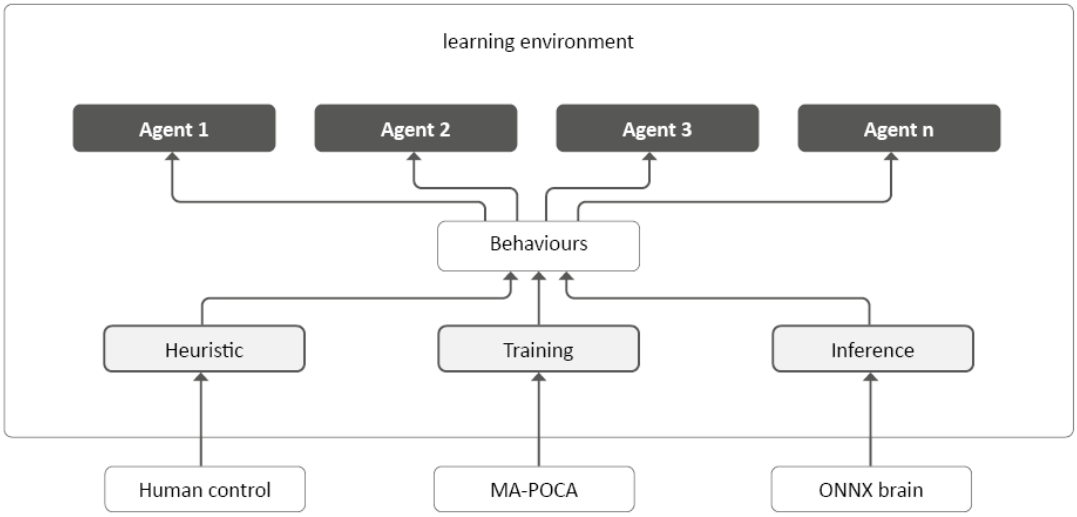


Figure 35: Three modes to control the behaviour of agents.

The users can prevent the episode from restarting by selecting the 'Reach Stable State On' parameter. This allows observation of the stable state. Users can add housing units manually by inputting '1' on the keyboard. If another result is needed, they can also restart the episode by pressing the space bar.

The dynamics between the agents and the environment are explained in the graphic below. These refer to one training episode and are reset at the beginning of each episode.

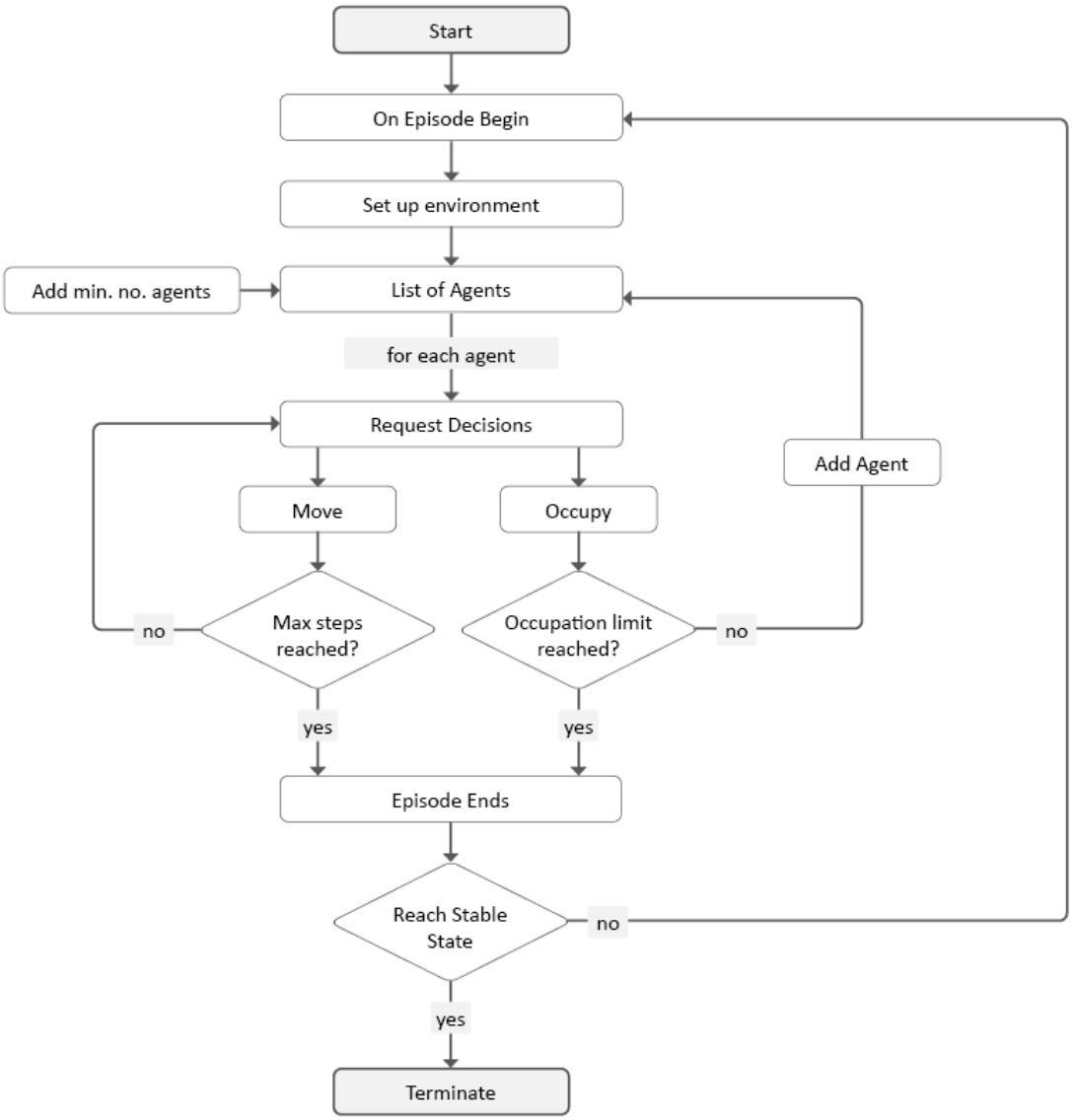


Figure 36: Main steps carried out during the training phase.

3. Learning Environment

The learning environment is implemented in the Unity Engine using mostly C# scripting. As well as the housing agents, the environment comprises a three-dimensional plot and set of policies, heights and protected green open areas. A list of agents ensures that each agent is part of a collaborating group. The parameters set up below register the initial configuration and monitor the states after each update.

The learning environment class performs the workflow shown in the image below. The interface between custom elements and Unity's methods is clearly stated.

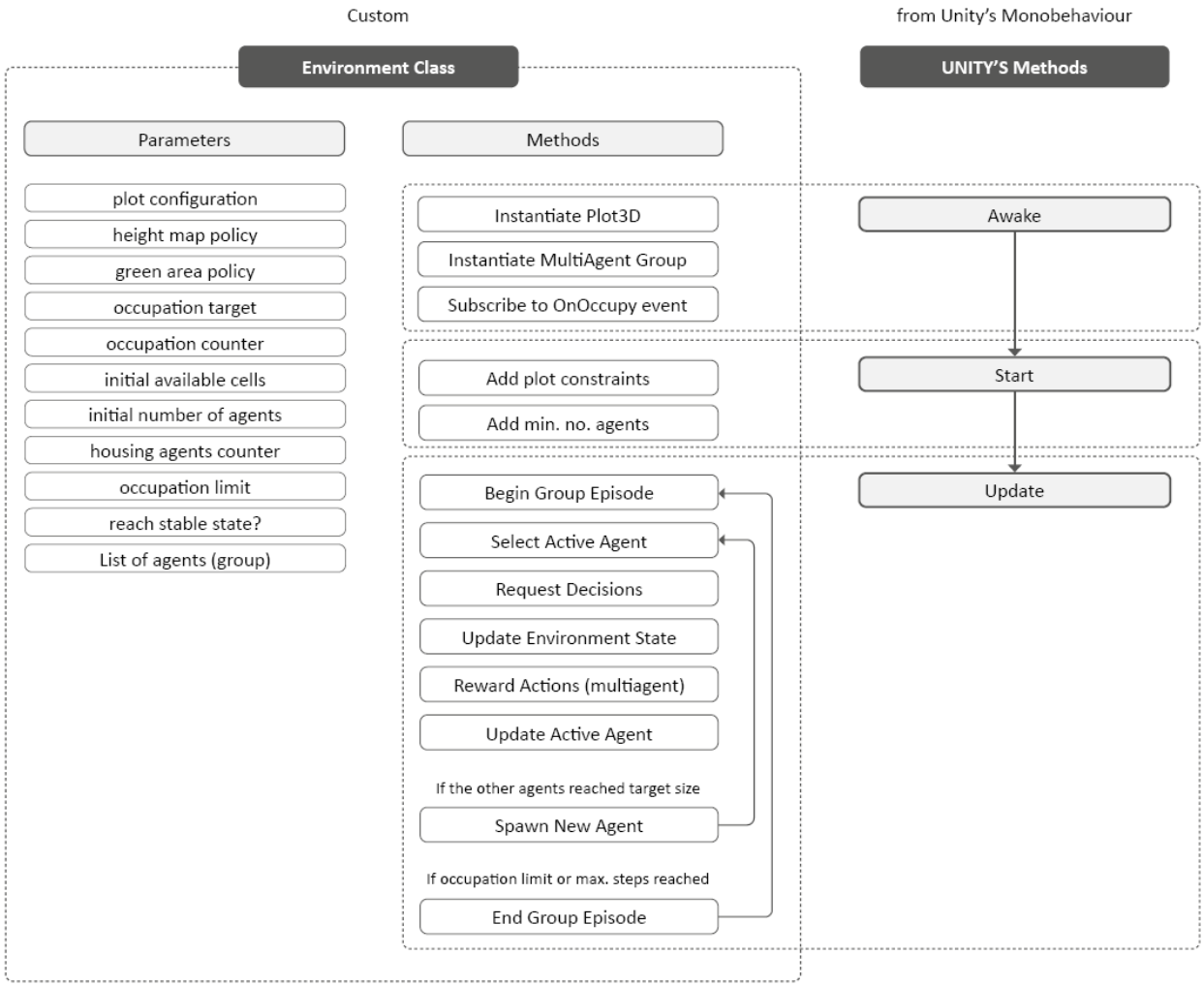


Figure 37: Learning environment class description.

The image below shows the environment initial state: (1) initial agent at a random position, (2) open-air cells are already protected, (3) ready-to-occupy empty cells within the height policy are tinted in red, (4) protected green areas, (5) developable ground, (6) environments parameters.

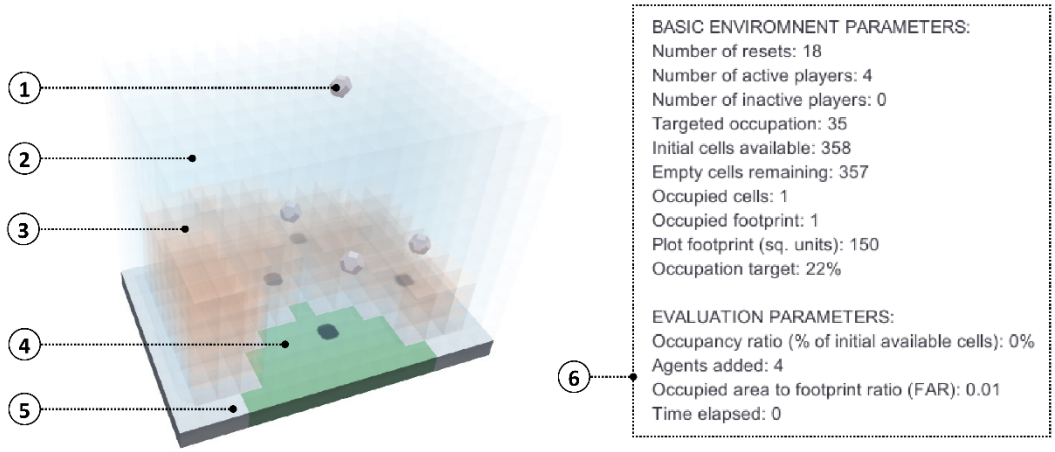


Figure 38: Initial learning environment after the episode set up.

The Plot 3D

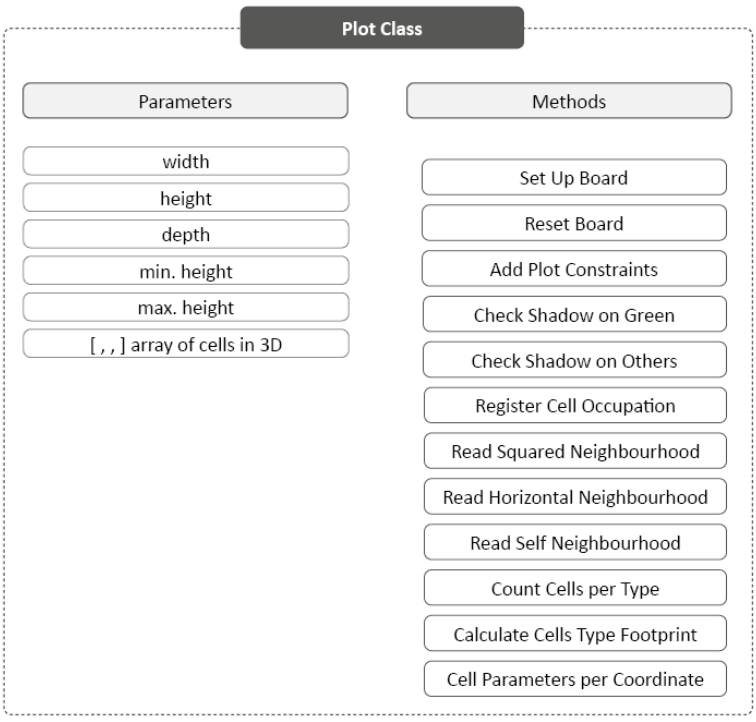


Figure 39: Parts of the plot class.

The behaviours associated with the environment that are not part of the housing-agents are managed by the plot. This class contains critical parameters and methods that provide functionality, registering the environment's configuration, policies, and states It stores a three-dimensional array of cells. They are defined next. Also, it provides the generation of environment configurations, including randomised sizes and policies. It further manages the event 'OnOccupyCell', which agents can subscribe to get rewards when occupying cells.

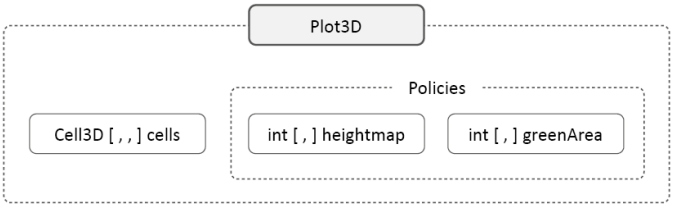


Figure 40: Main components of the three-dimensional plot.

As shown below, the plot class contains the following critical parts.

Cells

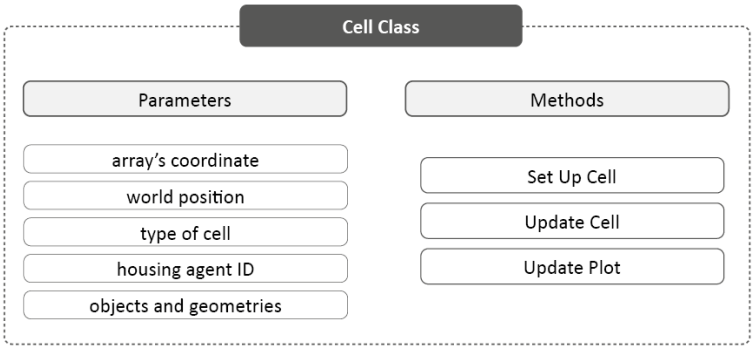


Figure 41: Parts of the cell class.

The cells are the building units of the plot. They store information regarding their relative coordinate within the plot and world position, type of cell and player ID if occupied by one of the agents. The cell types can be empty (developable), occupied, open-air, ground and green open. The environment protects with negative rewards the open air and green positions. The cells implement several methods that facilitate the interactions with the plot such as setting up, updating the cell type, colours and shapes displayed.

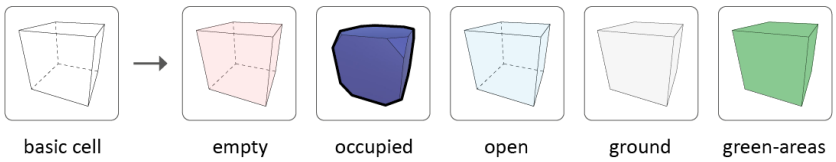


Figure 42: Cell Types diagram.

Height

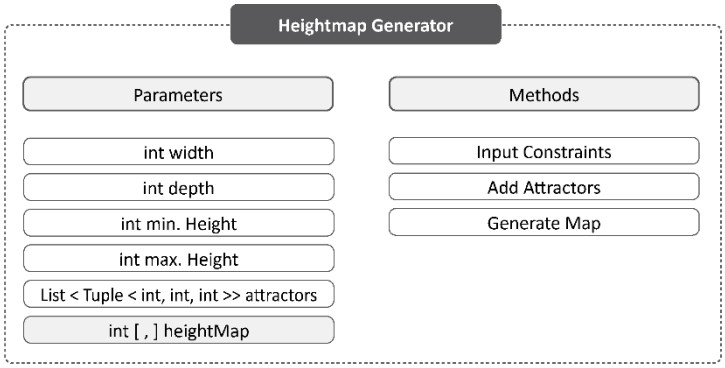


Figure 43: Parts of the heightmap generator class.

Height is one of the key parameters related to urban design. In this computation, a heightmap function determines height policies generated procedurally. It inputs one or more attractors with a smoothing function providing transition between peaks and valleys. Height values are stored in a double array called heightmap.

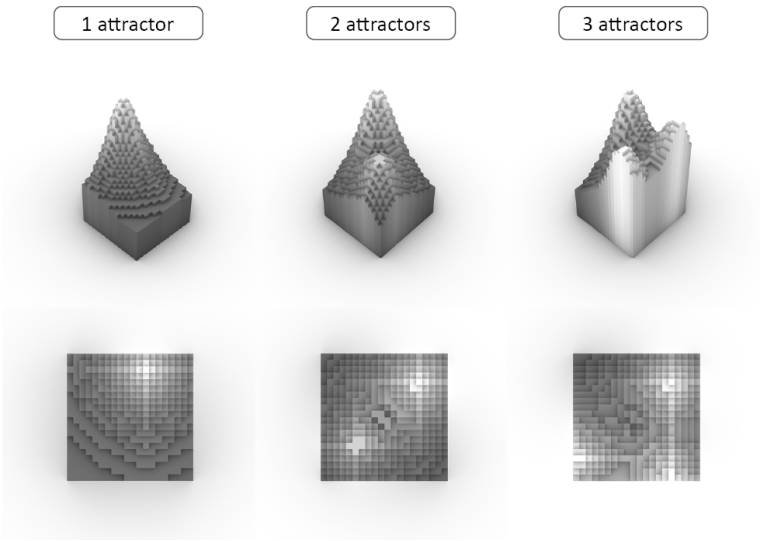


Figure 44: Examples of heightmaps in plan and axonometric view.

Generate Map

```
double [,] distances = new double[width, depth];
double minDist = double.MaxValue;
double maxDist = double.MinValue;

for (int x = 0; x < width; x++) {
 for (int z = 0; z < depth; z++) {

   double value = 0;
   for (int i = 0; i < attractors.Count; i++) {

     double valueM = 1 / (0.5 + (depth + width) + Pow(ManhattanDistance(x, attractors[i].Item1, z, attractors[i].Item2), 2));
     value += attractors[i].Item3 * (valueM + valueE);
   }

   if (value < minDist) minDist = value;
   if (value > maxDist) maxDist = value;
   distances[x, z] = value;
 }
}

for (int x = 0; x < width; x++) {
 for (int z = 0; z < depth; z++) {
   heightMap[x, z] = (int) RemapDoubles(distances[x, z], (minDist), (maxDist), minHeight, maxHeight);
 }
}
```

Figure 45: Pseudocode for generating heightmaps.

Green Open Space

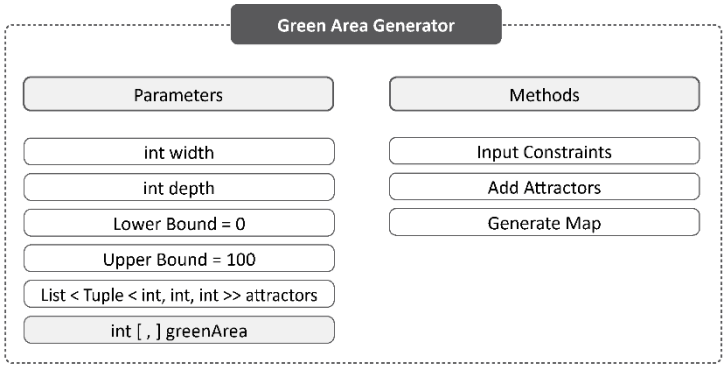


Figure 46:Parts of the green area generator class

The green areas map follows the same principle. Instead of storing height values, it stores binary numbers representing whether the values are above or below a certain threshold.

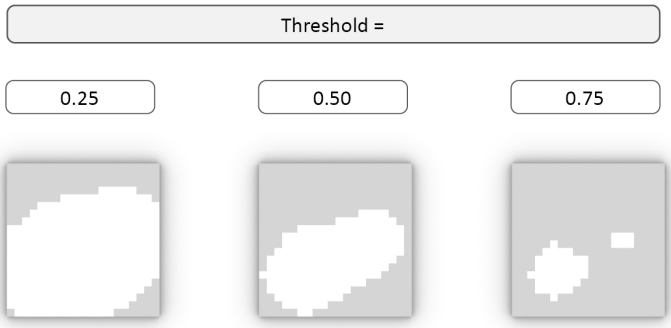


Figure 47: Same green map at different thresholds.

Generate Map

```
double [,] distances = new double[width, depth];
double minDist = double.MaxValue;
double maxDist = double.MinValue;

for (int x = 0; x < width; x++) {
 for (int z = 0; z < depth; z++) {

   double value = 0;
   for (int i = 0; i < attractors.Count; i++) {

     double valueM = 1 / (0.5 + (depth + width) + Pow(ManhattanDistance(x, attractors[i].Item1, z, attractors[i].Item2), 2));
     value += attractors[i].Item3 * (valueM + valueE);
   }

   if (value < minDist) minDist = value;
   if (value > maxDist) maxDist = value;
   distances[x, z] = value;
 }
}

for (int x = 0; x < width; x++) {
 for (int z = 0; z < depth; z++) {
   int newValue = (int) RemapDoubles(distances[x, z], minDist, maxDist, minHeight, maxHeight);
    if (newValue >= threshold) openGFMap[x, z] = 1;
    else openGFMap[x, z] = 0;
 }
```

Figure 48:Pseudocode for generating green area maps.

Shadows on Green Open Space

Below, a method for checking whether a position that the agent tries to occupy will overshadow green open space. The computation is based on casting a sunray from such position to the ground floor, returning true if it hits a green area.

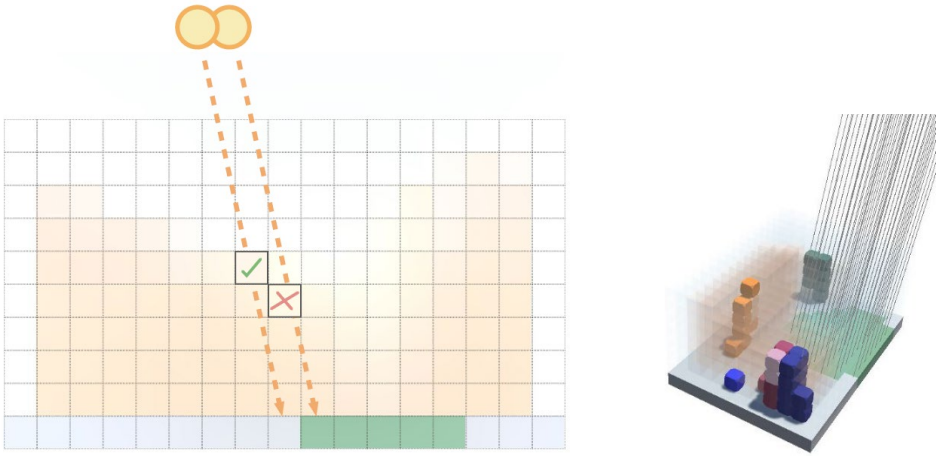


Figure 49: Checking occupation shadows on green open areas.

Neighbourhoods

The notion of neighbourhood is the basis for several of the above methods. From the central position, the adjacent cells are encoded by assigning binary numbers to them. It results in a unique decimal number for each possible configuration. The same principle applied to two and three dimensions. This is relevant to numerically encode the observations performed by the agents.

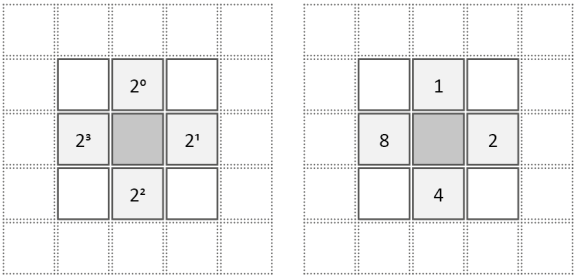


Figure 50: Binary encoding of a cell's neighbourhood.

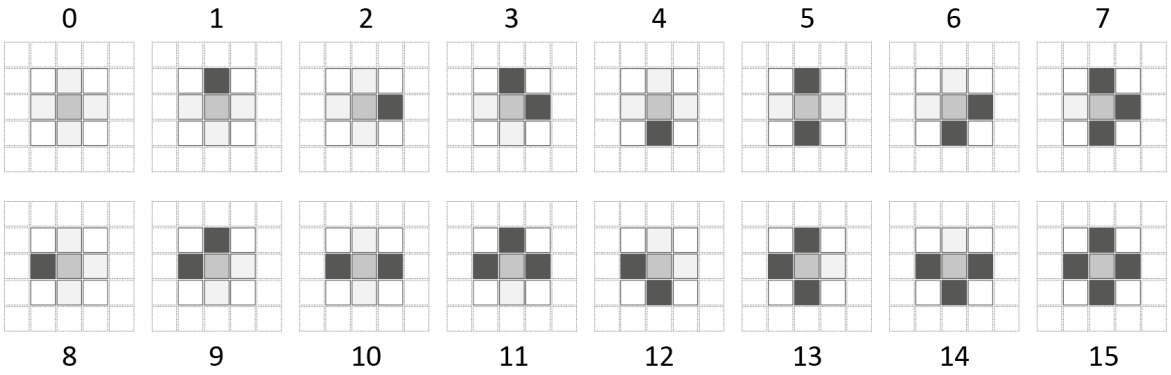


Figure 51: Combinations within a two-dimensional neighbourhood

Neighbourhood Significance

The neighbourhood methods comprising the plot class implement the above encoding. They provide unique numbers according to the cell type input, for instance, representing the adjacent empty, occupied, and open-air cells.

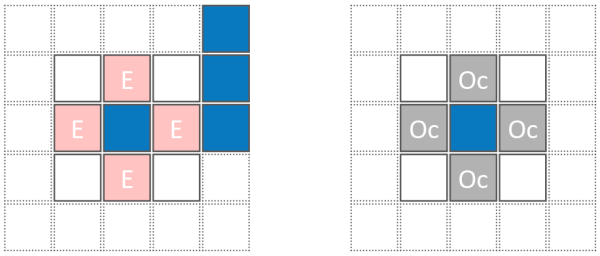


Figure 52: Examples of disconnected (left) and packed (right) occupations.

Two examples of this are shown in the above image. The disconnected occupation will return 15 as a 2D neighbourhood value for empty cells – or 63 in 3D, and zero for occupied, meaning the agent has disconnected positions and is not a realistic housing arrangement. Similarly, a packed neighbourhood implies that the occupied cell at the centre is not able to breathe.

4. Housing Agents

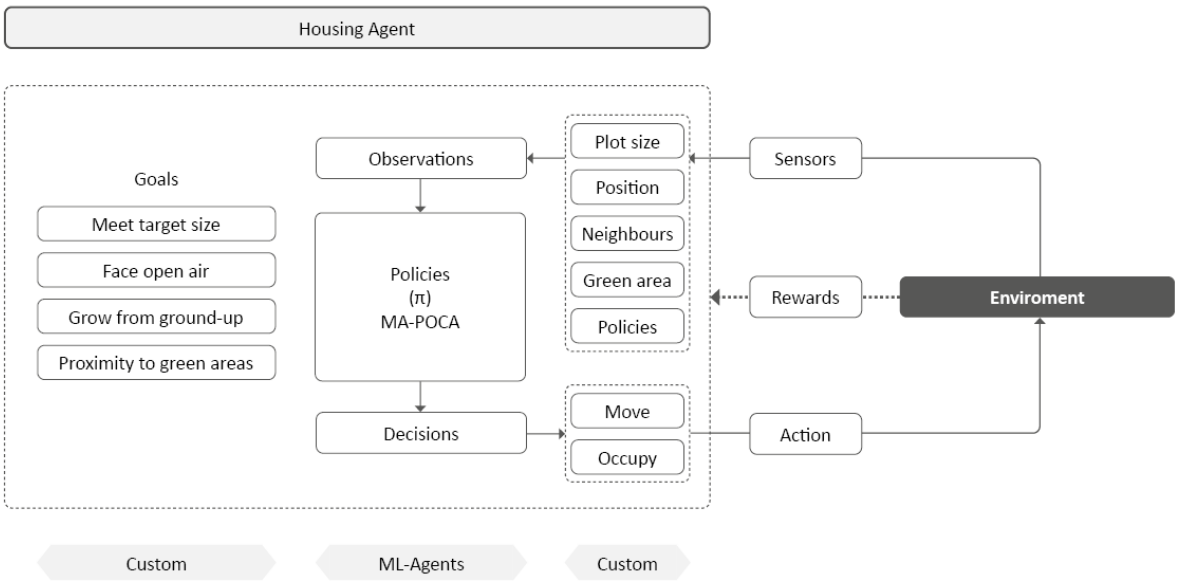


Figure 53: Architecture of the housing agent implementation.

The housing agent class inherits the Agent class from the Unity ML-Agents Toolkit. As described above, it allows for the agents' behaviour to be controlled by three modes learning, inference and heuristic. This section focuses only on the learning part, using the MA-POCA algorithm already described.

Implementation of Custom Elements and ML-Agents

The housing class comprises the parameters shown in the image below, such as the individual target size, house ID, current coordinate and if the agent remains active or has completed its objectives. It also monitors the current size and provides support for the actions of moving through the plot and occupying cells. The basic functionality applies to the learning, heuristic, and inference modes.

The housing agent inherits the Agent class from Unity's ML-Agents. This provides functionality to observe and make decisions based on the environment state and communicates the custom elements with the ML-Agents API that performs the policy updates using PyTorch. The most common methods implemented are episode begin and end, collect the vector observations and request decisions. Common to any RL method, the agent class operates in discrete temporal states. "At each step, an agent collects observations, passes them to its decision-making policy, and receives an action vector in response" (Unity Technologies 2021).

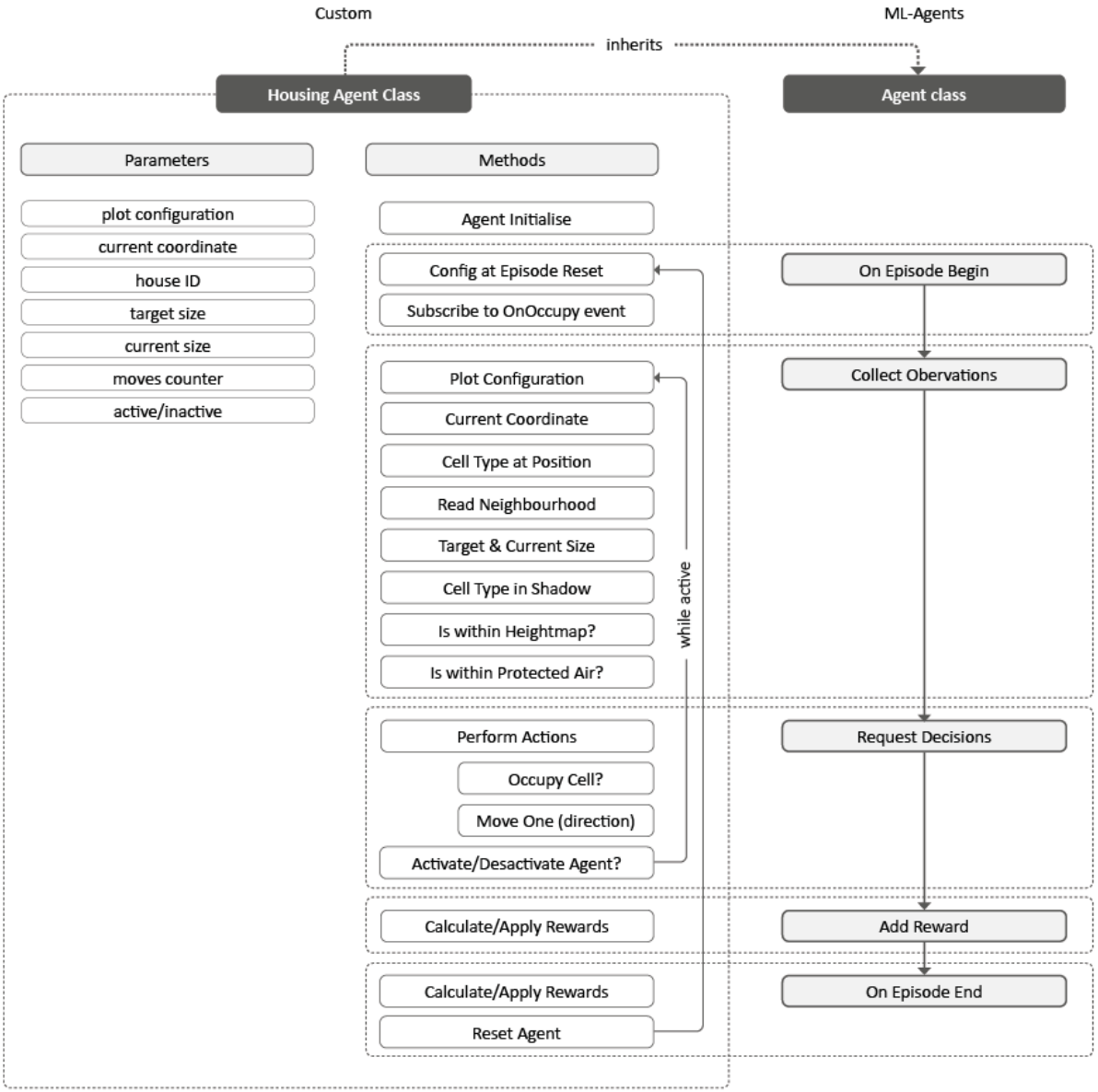


Figure 54: Housing agent class description

Agent and Episode Initialisation

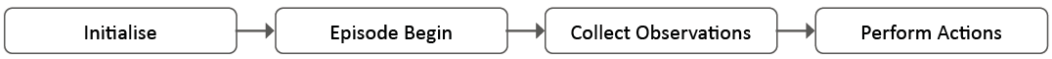


Figure 55: Simplified steps for the implementation of the housing agent.

The first method sets up the plot constraints and initiates the agents. The constraints are randomly generated on each episode, allowing the agent to learn in different conditions determined by the external policies. Such policies loosely resemble planning policies determining height and a designer's decisions to allocate open space areas. The agent initialisation includes a parameter that assigns its unique ID value and target size.

Any cells that the agent occupies inherit the ID parameter to determine the housing aggregation at any given state. After certain steps, one of the objectives is for the current size to match the target size.

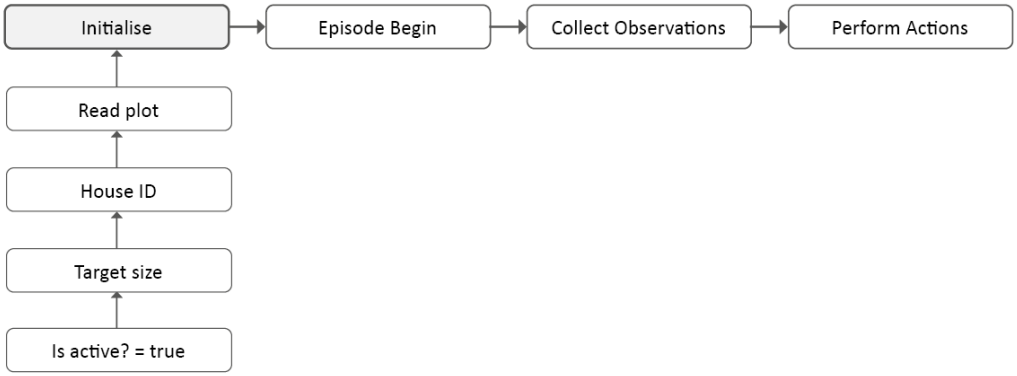


Figure 56: Agent Initialisation

A minimum number of agents is initialised only once at the start of the computation. These are always retained between episodes. Other agents added during the episode's duration to test the plot capacity are destroy when the episode ends. The agents retained, implement a method to reset their parameters when each episode begins.

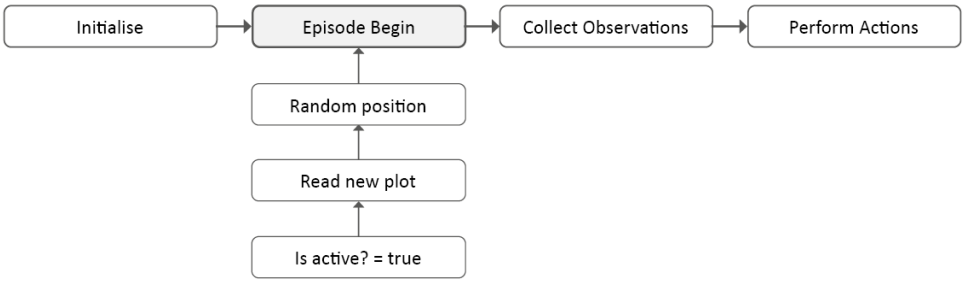


Figure 57: Workflow for retained agents at the begin of an episode.

5. Observations

The observations the housing agent performs are imperfect. They contain a small subset of the environment's state only covering what is around the agent "observing". The toolkit facilitates numerical and/ or visual observations for the agent. For this research, all observations are based on numerical (discrete) inputs. This can facilitate future implementation outside Unity.

The proposed agents perceive a total of eighteen observations from their environment. The list below explains them in more detail:

- Size of the plot 3D: width, height, and depth.

- Agent coordinate, corresponding to its x, y, z positions within the array of cells.
- Cell state at the current coordinates within the plot
- Type of cell at the current coordinate, this can be empty, occupied, or protected.
- Whether the current position is already occupied by this agent.
- The orthogonal neighbourhood values at the current coordinate generally refer to the empty, occupied, and protected values, each encoded as a binary number.
- Neighbourhood value for the cells already occupied by the agent, meaning having its ID.
- Target and current sizes of the housing agent.
- The type of cell receiving shadows from the agent's position.
- Whether the agent is within the height dictated by the heightmap policy.
- Access to open-air from its immediate horizontal neighbourhood, at last one must be air.

The ML-Agents Toolkit provides an implementation for fully connected neural networks. These are used by the agent to learn. The observations form multi-dimensional arrays, also known as vector observations, encode the information that the agent uses to make decisions. These provide the basis for training its behaviour.

6. Actions

The housing agents carry out one action at a time. For these to be executed, the computation must request a decision. Each action generally determines a new state of the environment that the agents can then observe.

Despite a relatively high number of observations, the proposed housing agents can only perform two types of actions: they move, or they occupy - or not - the cell they are in. A small number of possibilities is appropriate for this computation. Complex behaviours can emerge from these simple actions.

The move action provides the agent with a choice of seven possible options. It can move one position to any of the six cells perpendicularly adjacent to it or stay in the same position. The action to occupy or not provides the agent with the opportunity of building a position in its current cell position.

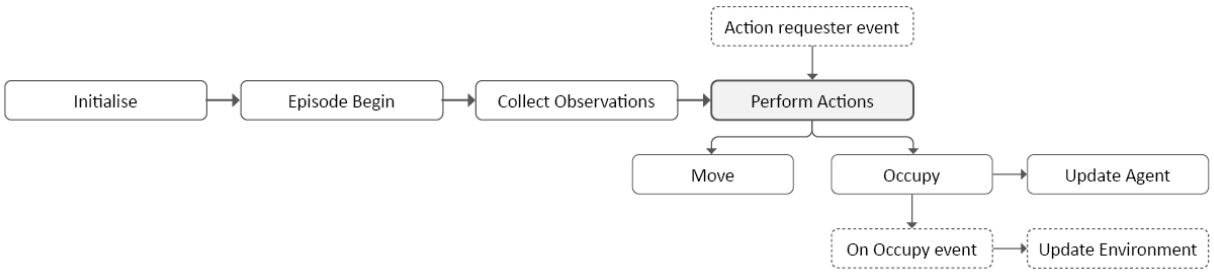


Figure 58: Actions performed by the housing agent.

7. Rewards

Each agent has individual goals. Housing agents seek to meet their programme, coded as several cells to occupy. After each interaction with the environment, the agent obtains rewards. Its value depends on the current state and executed actions although, the agent might process the feedback over several steps. The agent tries to improve its score even if it does not gain rewards for individual moves. Frequently, it is the sequence of actions that leads to a better outcome. Assigning adequate reward – or discount – values is a complex task that helps the agent to learn. As described by Ertel, a policy is the mapping from states to actions. The success of a policy is determined by its efficiency in the long term, leading to optimum or near optimum results. (Ertel 2011).

In this proposal, each agent aims to occupy several voxels, until it meets its target size. First, they observe and make simple decisions. Then, the reward functions are applied. The move action does not carry immediate rewards other than a small penalisation for excessive moves. This is to allow the agent free movement. However, due to the cumulative effect of rewards, the occupy action also influences the choice of moves. For instance, when applying small rewards to the occupation of the low levels or proximity to open space and after a significant amount of training steps, the agents tend to travel into those positions. As shown below, positive rewards are accompanied by negative ones. Both play a significant role in the overall result.

To simulate real-case housing scenarios. some key parameters are utilised in line with those that future occupants usually require. Explained below are the positive rewards given to the agent:

- Occupies an empty cell – this promotes the choice of available ones.
- Grows towards the target size – this avoids the agent's stagnation.
- Complies with the height policy – this promotes respect to urban constraints.
- Occupies a cell within a neighbourhood of empty cells – this promotes some dispersion.
- Occupies a cell that is close to open protected space – promotes some clustering around them.

Likewise, the agent receives negative rewards for the following reasons:

- Casts shadows on green open areas - this promotes quality communal spaces.
- Occupies protected cells such as open-air cells – this reinforces choice towards empty cells.
- Creates disconnected houses – this punishes the occupation of scattered houses.
- Tries to occupy cells already occupied – this punishes disturbance to already settled positions.
- Builds higher units – this promotes occupation from the ground up.
- Occupies positions with no horizontal access to open-air – this promotes that at least one face per position faces open-air cells.
- Occupies positions that preclude others from horizontal access to open-air – same as above.

During the experimentation phase of this research, it was observed that excessive negative rewards for the occupation of non-empty cells stopped the agents from occupying cells at all. This was addressed by lowering the negative rewards values. On the contrary, testing the reward for the agent's speed to fulfil its programme by penalising time spent triggered the premature occupation of cells. This limited future exploration towards more beneficial positions. Despite these drawbacks, negative rewards are necessary to direct the agent's behaviour towards housing aggregations that meet somehow realistic criteria.

The reward values displayed below are split into two categories. Basic ones refer to those providing essential functionality and are included in all the testing scenarios. Enhanced refers to the ones whose selection provides the basis for the configurations tested.

SINGLE-AGENT REWARDS			
Basic		**Enhanced**	
Occupy an empty cell	1.0	Occupy already occupied cells	-1.0
Growth towards target size	1.0	Cost of building higher (factor of height coordinate)	(1)
Comply with heightmap policies	1.0	Occupy neighbourhood of empty cells	1.0
Shadows green open areas	-2.0	Occupy position with no horizontal access to open air	-2.0
Occupy protected space	-2.0	Precludes other cells from access to open air	-2.0
Creates disconnected units	-1.0	Neighbouring green open areas	1.0
		Speed meeting the brief	(2)

Note:
(1) reward = - 0.1 * Y-value
(2) reward = 10 * house target size / moves to complete target

Table 1: Single-agent reward values.

8. Multi-Agent Rewards

In line with other MARL frameworks, this computation addresses serialised observation-decision-reward processes with more than one agent. Their joint and individual actions, and subsequent rewards, influence the evolution of the system's state.

Like real mass-housing experiences, each agent is self-interested, and its rewards may conflict with other agents. Competition is not a pre-established condition but may occur. Furthermore, the agents can cooperate to increase the group's reward. This type of setting is considered a general-sum game or mixed setting. It is aligned with other equilibrium ideas such as the Nash equilibrium which, has influenced this framework (Zhang Zhuoran Yang and Bas 2021).

The rewards given to the group of agents are shared across all the agents added to it. The group of housing agents receives positive rewards for the following:

- Growth towards global occupation target – this avoids stagnation.
- Addition of agents to the environment – this promotes higher densities although, prior to the addition of agents the computation requires the existing ones to have met their targets.
- Higher floor-to-area ratios (FAR) – promotes reduced occupation footprints.

The group of agents does not receive negative rewards. These are only applied to individual agents. In addition, the reward values displayed below are split into the same two categories, basic and enhanced, as the single-agent rewards. Refer to the above for clarity on the criteria used for the split.

MULTI-AGENT GROUP REWARDS	
Basic	**Enhanced**
Growth towards global occupation target (1)	Agent added to the environment 10.0
	Higher floor-area ratio (FAR) (2)

Note:
(1) reward = occupation count / occupation target
(2) FAR reward = occupied floor area / plot area

Table 2: Group reward values.

9. Ending the training episode

During training the training phase multiple episodes are needed. The episode can last a maximum of 3,600 steps, the equivalent of 555 episodes. When it reaches this limit, the episode terminates and a new one starts. An alternative reason for an episode to terminate is when the process reaches the target occupation density, calculated as occupied units between the number of empty cells at the beginning of the episode. The computation allows between 15 and 45 per cent of the initial empty cells to be occupied by agents. A limit within this range is randomly selected per training episode. Finally, a training episode can also terminate if one of the agents performs a third of the total episode moves without occupying any cells.

When the episode terminates the FAR of the housing is calculated by dividing the floor area occupied between the plot footprint area. This is directly added as a reward to the group of agents to encourage more intensive plot occupation, thus, releasing more are for open spaces.

10. Training Configuration

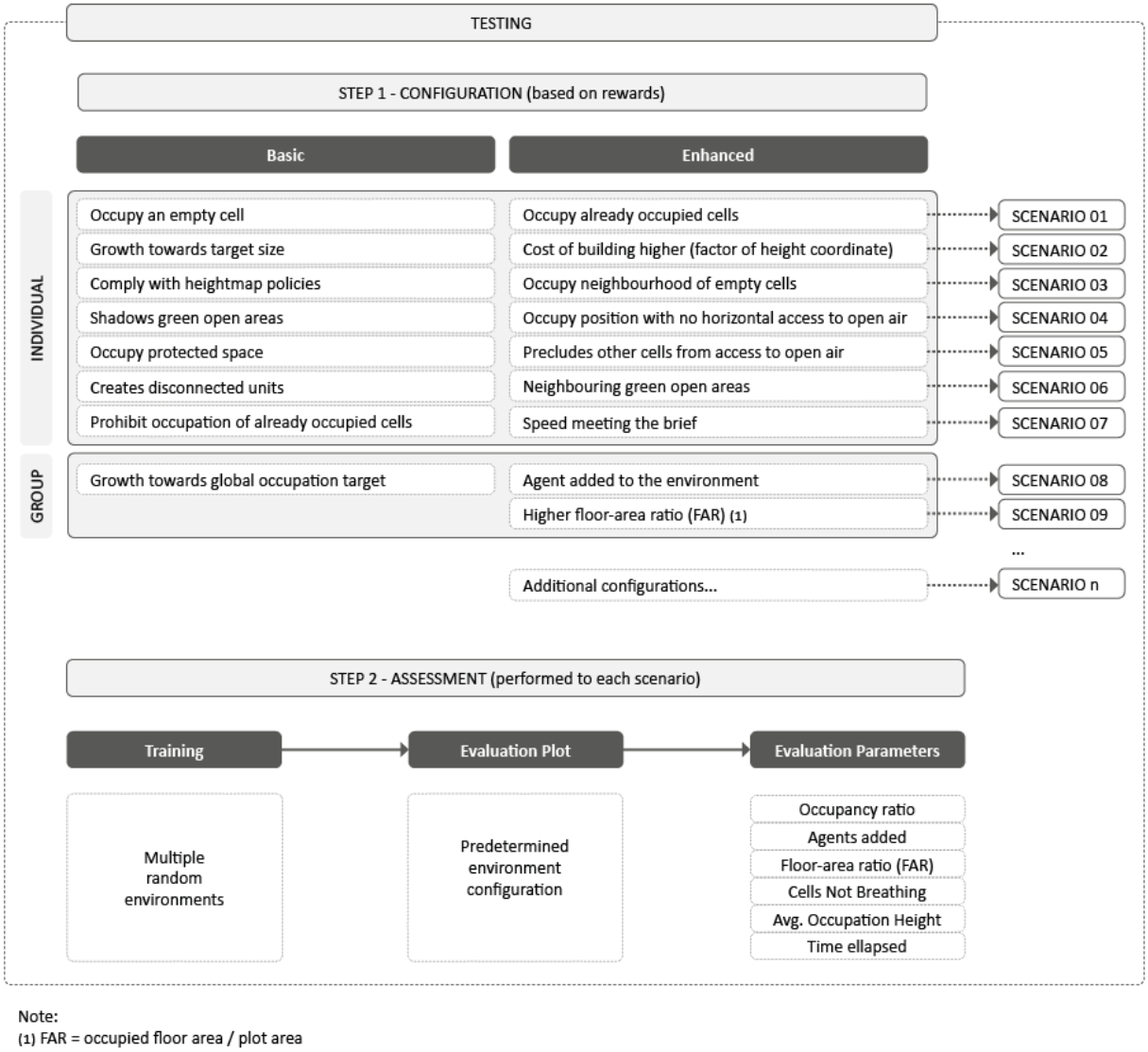
A data serialisation or YAML file is used to feed training configuration and hyperparameters to the deep learning model. There are different types, depending on the type of agents, environment, and type of learning. For multi-agent environments in cooperative settings, the most appropriate is the MA-POCA. It provides support for groups of decentralised agents with a centralised critic. Shown below is the configuration used.

```
trainer_type: poca
hyperparameters:
  batch_size: 512
  buffer_size: 10240
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: constant
network_settings:
  normalize: false
  hidden_units: 128
  num_layers: 2
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
keep_checkpoints: 10
max_steps: 2000000
time_horizon: 64
summary_freq: 50000
```

Figure 59: YAML file used for configuring the training.

IV. RESULTS

This section reports on the results obtained using the MA-POCA algorithm. Prior experimentation with single-agent systems using PPO allowed for testing but is not covered here. In the diagram below, Step 01 explains how different configurations of rewards for testing are set. Basic rewards are always present in each scenario, whereas enhanced are used to define the different tests. Step 02 shows that first the agents train on random plot configurations and then an evaluation is performed to nine plots configurations. These result from the combination of green area thresholds and occupation targets.



Note:
(1) FAR = occupied floor area / plot area

Figure 60: Testing steps and configurations.

A. Evaluating the Training

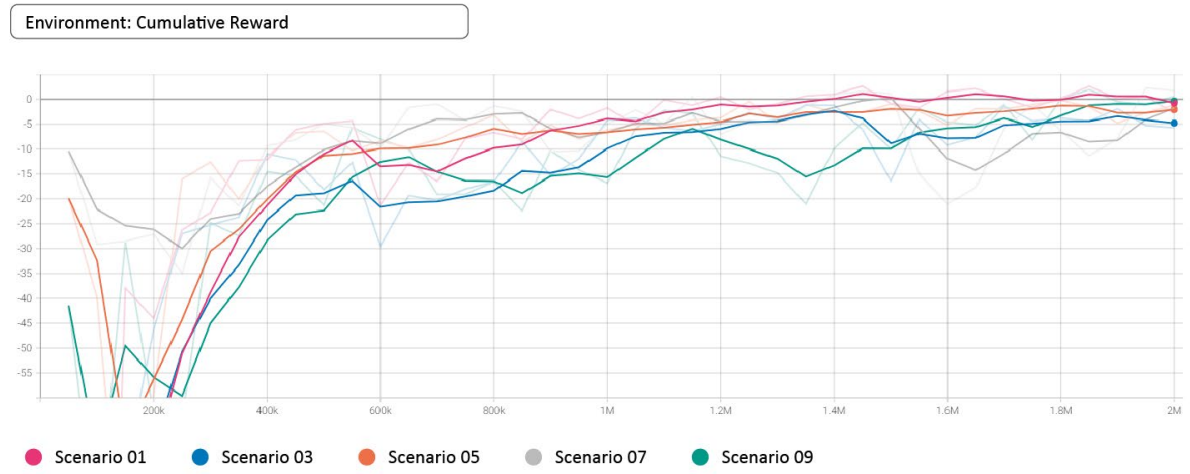
The configuration of the training scenarios is proposed as additional rules to the overall computation. For instance, scenario 02 proposes adding a negative reward for building higher, penalising but not precluding higher positions. This should is added to the configuration of the preceding step that penalises the occupation of already occupied cells. These two considerations together with the basic elements will define the configuration for the referred scenario 02. Due to the time required for the training of each scenario, this thesis will only explore scenarios 01, 03, 05, 07 and 09. It should be understood that the last one will comprise all the previous basic and enhanced configurations. The evaluation of the results obtained from the scenarios studied will determine if sequential improvements are observed or if certain rewards do not provide significant gains. The evaluating parameters will be provided in the next section.

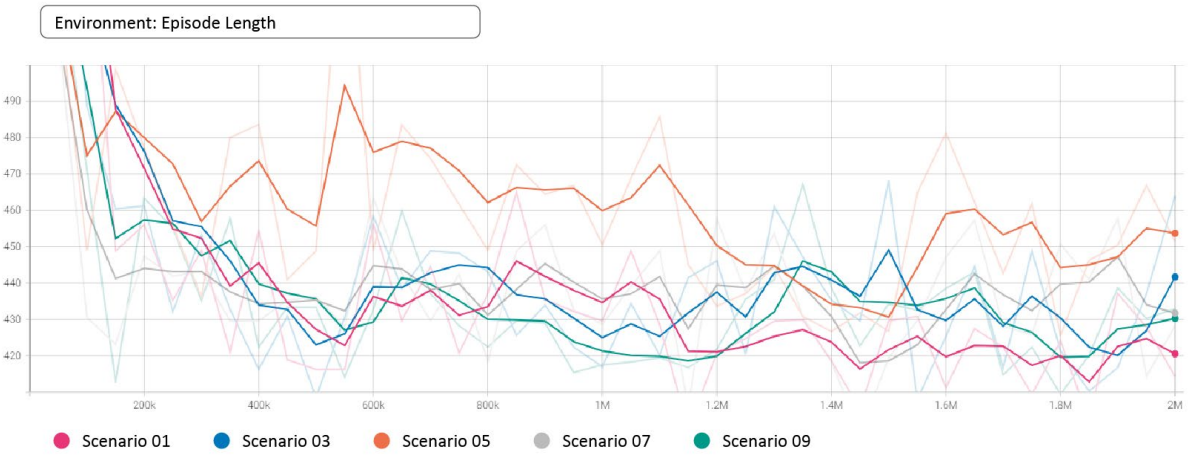
	STEP 1 - CONFIGURATION (based on rewards)							
	Basic	Enhanced						
	ALL	SCENARIO 01	SCENARIO 03	SCENARIO 05	SCENARIO 07	SCENARIO 09		
INDIVIDUAL	Occupy an empty cell	✓	✓	✓	✓	✓	✓	Occupy already occupied cells
	Growth towards target size	✓		✓	✓	✓	✓	Cost of building higher (factor of height coordinate)
	Comply with heightmap policies	✓		✓	✓	✓	✓	Occupy neighbourhood of empty cells
	Shadows green open areas	✓			✓	✓	✓	Occupy position with no horizontal access to open air
	Occupy protected space	✓			✓	✓	✓	Precludes other cells from access to open air
	Creates disconnected units	✓				✓	✓	Neighbouring green open areas
	Prohibit occupation of already occupied cells	✓				✓	✓	Speed meeting the brief
GROUP	Growth towards global occupation target	✓					✓	Agent added to the environment
		✓					✓	Higher floor-area ratio (FAR) (1)

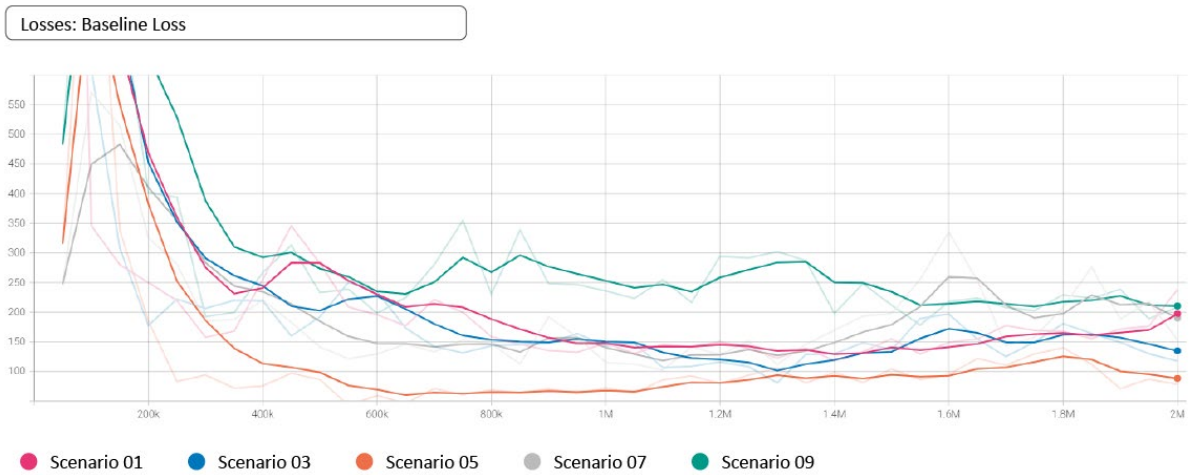
Table 3: Configuration of each scenario

Step 01 or the training phase uses PyTorch for the training of the agents but the visualisation is made using TensorBoard. This platform allows the production of model graphs and statistics. For instance, the graph cumulative reward graph shows a similar learning pattern for all the scenarios. Other graphs are also available on TensorBoard such as episode lengths, policy and value losses, etc. and all display similar outcomes for the training performed. However, it is unlikely that all configurations will achieve equally satisfactory results.

The conclusion from the above is that the training graphs do not provide means to identify successful configurations. As the observation of training graphs is not practical, this study will propose alternative tools to evaluate the configuration of each scenario. These should be relevant to the context of housing environments. The following section will explain what parameters are used for this assessment.


Environment: Cumulative Reward
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
-55
200k
400k
600k
800k
1M
1.2M
1.4M
1.6M
1.8M
2M
Scenario 01
Scenario 03
Scenario 05
Scenario 07
Scenario 09


Environment: Episode Length
490
480
470
460
450
440
430
420
200k
400k
600k
800k
1M
1.2M
1.4M
1.6M
1.8M
2M
Scenario 01
Scenario 03
Scenario 05
Scenario 07
Scenario 09


Losses: Baseline Loss
550
500
450
400
350
300
250
200
150
100
200k
400k
600k
800k
1M
1.2M
1.4M
1.6M
1.8M
2M
Scenario 01
Scenario 03
Scenario 05
Scenario 07
Scenario 09

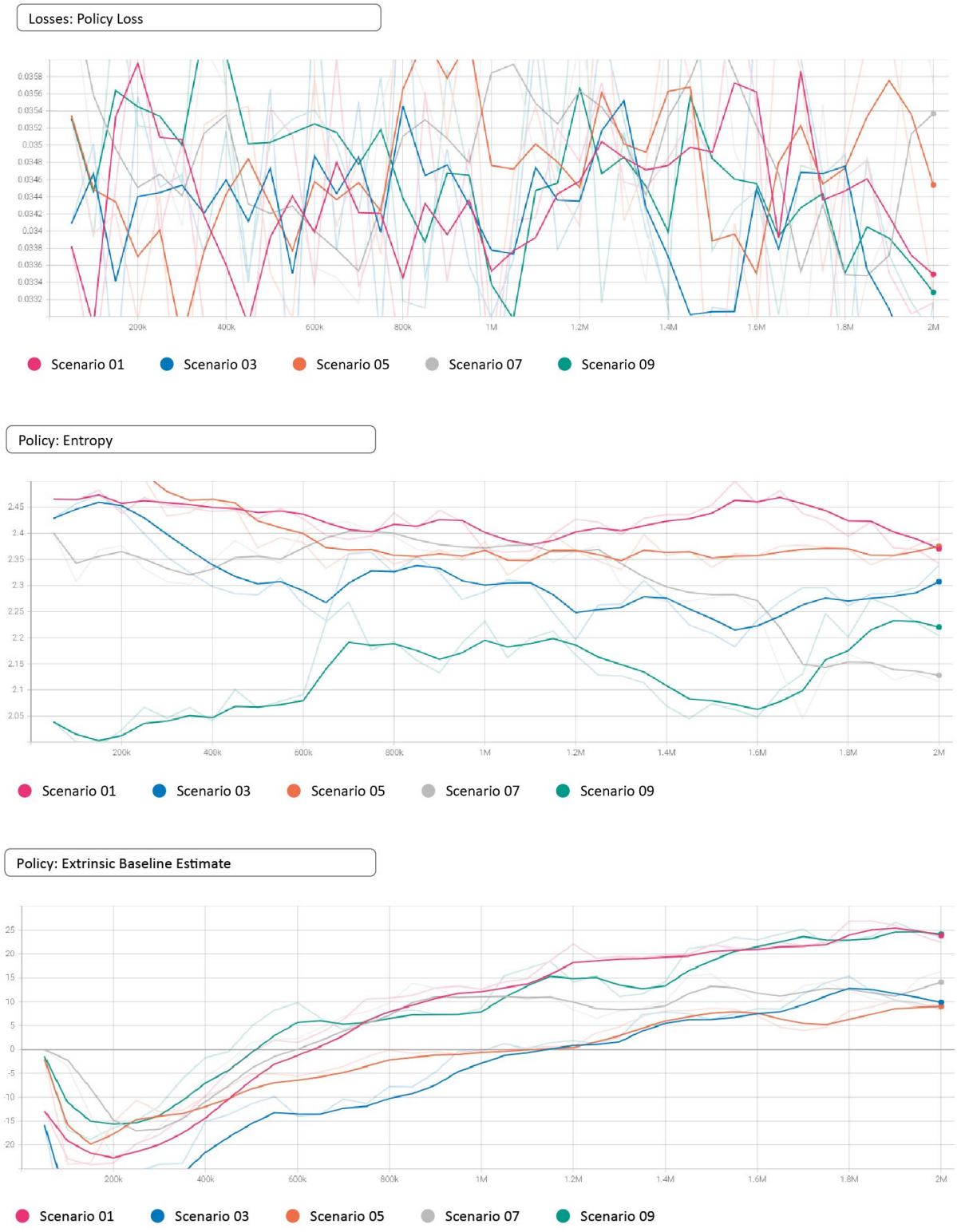


Figure 61: Model graphs for the training scenarios performed.

B. Evaluating the Results

The methodology section covered the scripting of behaviours in Unity, the use of the ML-Agents Toolkit and the Python API to train the agents using the TensorFlow library. The results shown below constitute examples of inference control performed by the ONNX brain after training.

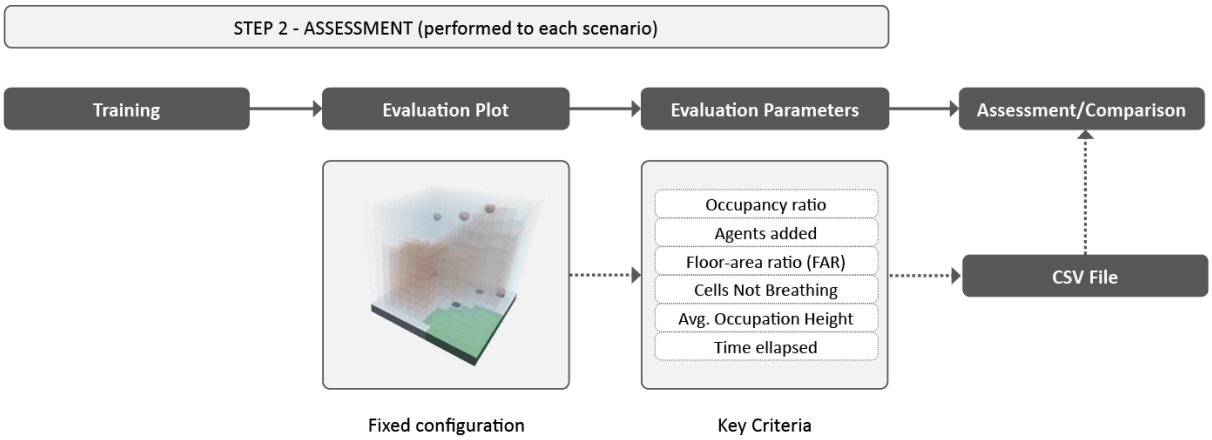


Figure 62: Step 2 - Evaluation of the agents' behaviours.

During training, the creation of identical plots is almost impossible due to the stochastic nature of the system. This thesis proposes the testing of the agents under similar conditions to facilitate the comparison of results.

As stated, the time required for the training has limited the number of scenarios that can be explored in this thesis. Only scenarios 01, 03, 05, 07 and 09 will be evaluated. The procedure used could be applied to the other scenarios. Using the Unity game engine, this section provides a detailed evaluation of several scenarios. The evaluation is based on tailored parameters, namely occupancy ratio, total agents added, FAR, cells not facing any open air (not breathing), the average height of occupied cells and time elapsed. The last one seeks to check if the computation can achieve the occupancy limit before reaching the time.

In addition, this evaluation also seeks to address the research questions set out in the introduction section. First, if it is possible to use simple rules to train MARL systems for housing. Second, if some rules are beneficial, which ones they are. A key concern is to evaluate if they maximise occupancy ratios, FAR and form aggregations from the ground up. Also, whether the agents exhibit collaborative behaviour and in what form.

Finally, the previous research studied will be compared to the present one.

Scenario 01

Reference: Test_Scenario_01						
Iteration	**Occupancy_Ratio**	**Agents**	**FAR**	**Packed**	**Avg_Height**	**Time**
1	9.87	8	0.3611	0	6.17	3600
2	9.68	8	0.3542	0	5.02	3600
3	9.87	8	0.3611	0	6.29	3600
4	7.59	6	0.2778	0	6	3600
5	9.11	7	0.3333	1	6.17	3600
6	9.11	7	0.3333	0	6.15	3600
7	8.92	7	0.3264	0	4.79	3600
8	9.87	8	0.3611	0	5.29	3600
9	7.78	6	0.2847	0	5.07	3600
10	9.11	7	0.3333	0	4.56	3600
11	8.73	7	0.3194	1	6.28	3600
12	7.97	7	0.2917	0	5.52	3600
13	8.92	7	0.3264	0	6.26	3600
14	7.21	6	0.2639	0	5.34	3600
15	9.87	8	0.3611	0	5.54	3600
Average	**8.91**	**7.13**	**0.33**	**0.13**	**5.63**	**3600.00**

Table 4: Scenario 01 results.

Scenario 03

Reference: Test_Scenario_03						
Iteration	**Occupancy_Ratio**	**Agents**	**FAR**	**Packed**	**Avg_Height**	**Time**
1	11.57	9	0.4236	0	2.59	3600
2	11.2	9	0.4097	0	3.31	3600
3	12.14	10	0.4444	0	2.78	3600
4	11.76	9	0.4306	0	2.68	3600
5	11.95	10	0.4375	0	2.79	3600
6	11.95	10	0.4375	2	3.86	3600
7	11.95	10	0.4375	0	2.76	3600
8	12.14	10	0.4444	0	2.69	3600
9	11.76	9	0.4306	0	3.26	3600
10	12.52	10	0.4583	0	3.15	3600
11	12.33	10	0.4514	0	3.02	3600
12	11.57	9	0.4236	0	3.69	3600
13	11.39	9	0.4167	0	2.6	3600
14	12.14	10	0.4444	0	2.89	3600
15	12.14	10	0.4444	0	2.78	3600
Average	**11.90**	**9.60**	**0.44**	**0.13**	**2.99**	**3600.00**

Table 5: Scenario 03 results.

Scenario 05

Reference: Test_Scenario_05						
Iteration	Occupancy_Ratio	Agents	FAR	Packed	Avg_Height	Time
1	11.57	9	0.4236	0	4.18	3600
2	11.95	10	0.4375	0	3.35	3600
3	12.33	10	0.4514	0	2.89	3600
4	11.57	9	0.4236	0	2.87	3600
5	11.76	9	0.4306	1	2.95	3600
6	12.14	10	0.4444	0	2.55	3600
7	11.76	9	0.4306	0	3.05	3600
8	11.95	10	0.4375	0	2.71	3600
9	11.95	10	0.4375	0	2.71	3600
10	12.14	10	0.4444	0	3.09	3600
11	12.14	10	0.4444	1	3.03	3600
12	10.82	9	0.3958	0	2.74	3600
13	10.25	8	0.375	0	2.98	3600
14	11.95	10	0.4375	1	3.27	3600
15	12.33	10	0.4514	1	2.98	3600
Average	**11.77**	**9.53**	**0.43**	**0.27**	**3.02**	**3600.00**

Table 6: Scenario 05 results.

Scenario 07

Reference: Test_Scenario_07						
Iteration	Occupancy_Ratio	Agents	FAR	Packed	Avg_Height	Time
1	9.3	8	0.3403	0	3.2	3600
2	11.39	9	0.4167	0	3.12	3600
3	10.44	8	0.3819	0	3.8	3600
4	9.68	8	0.3542	0	3.65	3600
5	11.01	9	0.4028	0	3.31	3600
6	10.63	9	0.3889	0	3.55	3600
7	11.95	10	0.4375	0	3.11	3600
8	11.39	9	0.4167	0	4.35	3600
9	9.87	8	0.3611	0	3.27	3600
10	11.95	10	0.4375	0	4.24	3600
11	11.01	9	0.4028	0	2.76	3600
12	11.95	10	0.4375	1	2.78	3600
13	8.16	7	0.2986	0	2.53	3600
14	11.01	9	0.4028	0	3.62	3600
15	11.95	10	0.4375	1	2.38	3600
Average	**10.78**	**8.87**	**0.39**	**0.13**	**3.31**	**3600.00**

Table 7: Scenario 07 results.

Scenario 09

Reference: Test_Scenario_09						
Iteration	**Occupancy_Ratio**	**Agents**	**FAR**	**Packed**	**Avg_Height**	**Time**
1	11.76	9	0.4306	0	4.55	3600
2	11.95	10	0.4375	0	3.87	3600
3	12.14	10	0.4444	0	4.58	3600
4	11.57	9	0.4236	1	2.98	3600
5	12.14	10	0.4444	0	3.17	3600
6	11.95	10	0.4375	0	3.27	3600
7	11.76	9	0.4306	0	3.29	3600
8	12.14	10	0.4444	0	3.39	3600
9	12.14	10	0.4444	0	3.25	3600
10	11.95	10	0.4375	0	2.9	3600
11	12.33	10	0.4514	0	3.58	3600
12	10.82	9	0.3958	0	3.28	3600
13	11.95	10	0.4375	0	3.14	3600
14	11.95	10	0.4375	0	3.94	3600
15	12.14	10	0.4444	0	2.88	3600
Average	**11.91**	**9.73**	**0.44**	**0.07**	**3.47**	**3600.00**

Table 8: Scenario 09 results.

Evolution of one Testing Episode

Figure 63: Sequence of states during one iteration of scenario 09.

Example of Final States during the Testing

Figure 64: One final state for each scenario during testing.

V. DISCUSSION

A. Interpretation of the Results

All the scenarios studied have shown that the agents can learn basic tasks. They leaned to occupying primarily empty cells, growing to reach the target size and maximise the overall occupation of the plot, respecting the height and green area policies and not occupy cells already occupied. However, the agents still tend to create houses that are disconnected from time to time. In addition, they seem to follow random moves that sometimes result in not advantageous configurations. Resolving this will require more testing to adjust the rewards.

Comparison: Average Values						
Scenario	Occupancy_Ratio	Agents	FAR	Packed	Avg_Height	Time
01	8.91	7.13	0.33	0.13	5.63	3600.00
03	11.90	9.60	0.44	0.13	2.99	3600.00
05	11.77	9.53	0.43	0.27	3.02	3600.00
07	10.78	8.87	0.39	0.13	3.31	3600.00
09	11.91	9.73	0.44	0.07	3.47	3600.00

Table 9: Comparative of average results from each scenario during testing.

Concerning the parameters specific to each scenario, the agents exhibit trade-offs to maximise their rewards. For instance, scenario 01 achieves a higher number of agents added. However, it is not penalised for building higher, creating floating elements. From scenario 03 onwards, the configurations include the cost of building in higher cells. Surprisingly, this has not affected the occupancy ratio but has the negative effect of increasing the FAR. Also, from scenario 03 the agents seem to create unrealistic aggregations with the occupied cells too scattered. This is likely the effect of promoting the agents the occupation of empty neighbourhoods. This is not how efficient housing aggregations work in real cases and it should be further refined in the future.

Aggregations 05 and 07 do not seem to create visually different designs. However, 05 resulted in more packed cells and higher occupancy. It also exhibits marginal lower heights when compared to 07. The higher ratio of packed cells for scenario 07 is likely a result of rewarding faster performances. It might favour the premature occupation of positions as early as possible at the cost of other parameters such as access to open air. Differently adjusting the rewards should resolve this issue.

Prior studies, resulted in more satisfactory patterns exhibiting growth from the grown-up whilst minimising their footprint and occupy positions in proximity to open space. However, in the final studies, this seems less

obvious, potentially due to the level of unpredictability linked to the several reward values. Unfortunately, the time limitation for the studies does not allow for re-testing to revert to those conditions.

While additional research time might be necessary, scenario 09 is the one closer to exhibiting such patterns. It can be seen in the above comparison. Scenario 09 provides the most realistic results visually. The values obtained do not provide the best result for any of the parameters studied. However, it seems to achieve good or near-optimal values for most of them, an effect that resembles results from multi-objective optimisation methods.

For the above reasons, the study does not endorse the use of the proposed computation in real housing scenarios. The methodology shows promising results and, further work is required before it is ready for real-life applications. Given the empirical success in other disciplines (Zhang Zhuoran Yang and Bas 2021) and the precedent set in this study, it is safe to consider that MARL systems will play a pivotal role in the context of decentralised housing.

Figure 65:Interpretation of the results, view from a housing unit.

B. Significance of the Study

The generation of diverse and high-quality housing environments remains a top priority within architects. Most of the areas and sectors in the field of architecture required successful spatial arrangements. At the most basic level, this research proposes the use of MARL to generate these arrangements. This research proves that starting from discrete elements and a set of clear rules, it is possible to successfully train a multi-

agent system that complies with sets of external conditions. This research has tested the use of heights and green areas policies to influence the agents' behaviour, but others should be equally valid. Therefore, the workflow proposed in this thesis applies to other sectors and domains.

Previous computational research for mass- housing include Shape Grammars (Shekhawat et al. 2019; Garcia and Menezes Leitão 2018; Correia et al. 2012), Cellular Automata solutions (Dounas et al. 2017), graph-based algorithms (Heckmann et al. 2020) or Generative Neural Networks (Chaillou 2019). They provide solutions that are hard to readapt if the given constraints change. The methodology of this thesis allows easy-to-implement rules and can adapt to a great variety of environments.

Closer examples, like the integration of Wave Function Collapse algorithm and DRL (Mintrone and Erioli 2021), distribute tileset patterns but do not support the addition of planning and urban policies such as height or green space. They mimic learned patterns but, unlike MARL systems, are not capable of novel solutions.

The methodology proposed demonstrates that intelligent agents can create housing aggregations by learning to comply with simple rules and policies. Furthermore, previous research in the field addressed the creation of housing blocks but lacked emphasis on communal spaces, urban parameters, and planning policies. This study proposes a way to facilitate decentralised self-build mass housing. Despite its limitations, it proves that MARL systems can learn collaborative behaviours that meet individual goals whilst complying with urban and communal policies.

C. Limitations

Despite the number of parameters already considered, the designs obtained do not yet provide a clear hierarchy of spaces other than respecting the green areas and the need for housing units to have at least one side facing open-air (Mintrone and Erioli 2021). The tasks that the human brain performs in the context of housing design are very complex. Researchers should not underestimate this complexity when training MARL systems. Training intelligent agents require lengthy processes, requiring large amounts of training, many parameters, and complex observations.

The eighteen observations performed by the housing agents are collected as mathematical models partially representing their environment. Further exploration with alternative sensors such as ray perception or machine vision could prove equally successful. They can also provide increased flexibility. Another limitation is the number of states stacked by the agent before feeding information to the neural network. This research tested 1, 3 and 5, with the last one being more successful. Perhaps with more time available, other configurations could have exhibit improved results.

Other limiting aspects in this thesis are the randomness applied to the generation of plots and the relatively low number of cells. Coarser resolutions allow for faster computations, but finer-grain definitions should result in richer and more complex environments. The time available for experimentation and testing of the training configurations has also greatly limited this research.

D. Further Research

Promising research in the field of economics has greatly influenced this work. A recent paper by Zheng et al. studied the improvement of government taxation. They have proposed a DRL model that uses two loops. The inner one comprises four categories of greedy agents, simulating parts of the social fabric. The outer mimicked the role of a social planner, balancing the policies for the common good (Zheng et al. 2020). The implementation of a similar methodology would be an immediate next step for the present research. In the context of maximising the common good, clear advantages favour the use of agents that learn to control position and quantum of height and green area policies.

Further work is also necessary to determine improved methods to evaluate the state of the environment and housing units. An improvement that could allow the application of more effective rewards lead to more efficient behaviours. Another proposed improvement is an implementation that ensures the agents remain active even after meeting their brief. Constantly seeking more efficient aggregations as agents are added to the environment to ensure the evolves with the environment states.

This research has considered urban density, FAR, size of the plot, urban policies, etc. The generation of the agents' brief has considered only their size, namely the number of cells. The brief given to the housing agents could implement additional constraints to achieve more realistic briefs. For instance, future occupants might have a diverse appetite for views, amenity space, layout arrangements – compartmented or open-plan – number of bathrooms, etc. Topological relations between the housing cells can further improve this study, linking it to efforts in automation based on the research of discrete assemblies and digital materials (Retsin 2016; Rossi and Tessmann 2017). The implementation of graph-based topological relationships, therefore, seems a natural step. Higher topological relationships, like those between housing units and cores or communal areas, are also possible.

Experiments with human participants have not been tested in this study. However, during the implementation testing was required to test the computation. Videogame like human control was used and should still be compatible with the current code provided. Real-time human interaction could provide for participatory environments. Such a tool could have learning and development possibilities in line with the Common'hood experiment (Sanchez 2018).

Advice to Future Researchers

Someone seeking to continue the line of research set out in this study might consider the following. First, additional reward parameters to promote horizontal growth of individual housing units. In combination to lower FAR, this might exhibit new behaviours and trade-offs. Second, the training on random plots might not be the most effective way for agents to learn behaviours. Instead, it is suggested that future research implements ways to repeat the same scenarios for several intervals before their configuration changes. This could allow the agents to test behaviours several times under the same conditions before adapting to new ones. A final note is also due to reemphasize the need to allow for long training times when considering research in the field of MARL, this issue can be improved with parallel computing or cloud-based solutions.

The code implemented and other parts of this thesis are available on Github in the link below:

https://github.com/dforman33/Housing-Agents

Figure 66: Interpretation of the results, view of a green open space.

VI. REFERENCES

Allis, Louis Victor. 1994. *Searching for Solutions in Games and Artiicial Intelligence*. CIP- Gegevens Koninklijke Bibliotheek, den Haag. http://fragrieu.free.fr/SearchingForSolutions.pdf.

Bhooshan, Shajay, and Alicia Nahmad. 2017. "Block Party Phase I Booklet by Bhavatarini Kumaravel - Issuu." *AA School of Architecture*, 2017. https://issuu.com/bhavatarinikumaravel2/docs/phase_1_booklet___b5___final.

Cai, Chenyi, Peng Tang, and Biao Li. 2019. "Intelligent Generation of Architectural Layout Inheriting Spatial Features of Chinese Garden Based on Prototype and Multi-Agent System. A Case Study on Lotus Teahouse in Yixing." *CAADRIA* 1: 291–300.

Cai, Yang, and Constantinos Daskalakis. 2011. "On Minimax Theorems for Multiplayer Games." In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 18. https://doi.org/10.1137/1.9781611973082.20.

Chaillou, Stanislas. 2019. "Space Layouts & GANs | GAN-Enabled Floor Plan Generation."

Chaillou, Stanislas, and Harvard Gsd. 2019. "AI + Architecture Towards a New Approach."

Cocho-Bermejo, Ana, and Diego Navarro-Mateu. 2019. "User-Centered Responsive Sunlight Reorientation System Based on Multiagent Decision-Making, UDaMaS." *ECAADe 37* 2: 695–704.

Correia, Rodrigo, José Duarte, António Leitão, Ist -Utl Portugal, and Fa -Utl Portugal. 2012. "GRAMATICA A General 3D Shape Grammar Interpreter Targeting the Mass Customization of Housing." *Shape Studies* 1: 30.

Dezeen Magazine. 2020. "Zaha Hadid Architects Unveils Roatán Próspera Housing Complex for Honduras," 2020. https://www.dezeen.com/2020/07/29/zaha-hadid-architects-roatan-prospera-housing-complex-honduras/.

Dounas, Theodoros, Benjamin Spaeth, Hao Wu, Chenke Zhang, and Xi ' An Jiaotong. 2017. "SPECULATIVE URBAN TYPES A Cellular Automata Evolutionary Approach."

Ertel, Wolfgang. 2011. *Introduction to Artificial Intelligence Series Editor*.

Foerster, Jakob N, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. "Counterfactual Multi-Agent Policy Gradients." *AAAI 2018*.

Garcia, Sara, and António Menezes Leitão. 2018. “Shape Grammars as Design Tools: An Implementation of a Multipurpose Chair Grammar.” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM* 32 (2): 240–55. https://doi.org/10.1017/S0890060417000610.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press. https://mitpress.mit.edu/books/deep-learning.

Harputlugil, Timuçin, A. Tanju Gültekin, Matthijs Prins, and Y. Ilker Topçu. 2014. “Architectural Design Quality Assessment Based on Analytic Hierarchy Process: A Case Study (1).” *Metu Journal of the Faculty of Architecture* 31 (2): 139–61. https://doi.org/10.4305/METU.JFA.2014.2.8.

Heckmann, Oliver, Michael Budig, Zack Xuereb Conti, Ray Chern, X I Cheng, Sky Lo, and Tian Tian. 2020. “User-Driven Parcellation of High-Rise Units for Future Urban Habitation.” *Participatory Computational Design Tools for Future Urban Habitation*.

Heffernan, Emma, and Pieter de Wilde. 2020. “Group Self-Build Housing: A Bottom-up Approach to Environmentally and Socially Sustainable Housing.” *Journal of Cleaner Production* 243 (January): 118657. https://doi.org/10.1016/j.jclepro.2019.118657.

“Home | Katerra.” 2021. 2021. https://katerra.com/.

Hu, Junyan, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. 2020. “Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning.” *IEEE Transactions on Vehicular Technology* 69 (12): 14413–23. https://doi.org/10.1109/TVT.2020.3034800.

Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, et al. 2018. “Unity: A General Platform for Intelligent Agents.” *CoRR* abs/1809.0. https://github.com/Unity-Technologies/ml-agents.

LaValle, Steven M. 2006. *Planning Algorithms / Motion Planning*. Cambridge University Press. http://planning.cs.uiuc.edu/.

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel Uc, Berkeley Openai, and Igor Mordatch Openai. 2017. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.” *ArXiv E-Prints*. https://ui.adsabs.harvard.edu/abs/2017arXiv170602275L.

Maschler, Michael, Eilon Solan, and Shmuel Zamir. 2020. *Game Theory*. Second edi. Cambridge University Press.

McNeel, Robert, and others. 2010. “Rhinoceros 3D, Version 7.0.” *Robert McNeel & Associates, Seattle, WA*.

Mercer. 2021. “Quality of Living Reports | Mercer.” 2021. https://www.imercer.com/products/quality-of-living.

Mintrone, Alessandro, and Alessio Erioli. 2021. “Training Spaces Fostering Machine Sensibility for Spatial Assemblages through Wave Function Collapse and Reinforcement Learning.” In *ECAADe*, 17–76.

Mitchell, Melanie. 2019. *Artificial Intelligence : A Guide for Thinking Humans*. Farrar, Straus and Giroux. https://us.macmillan.com/books/9780374257835.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, and Koray Kavukcuoglu. 2016. “Asynchronous Methods for Deep Reinforcement Learning.”

Neumann, John von, and Oskar Morgenstern. 1944. *Theory of Games and Economic Behavior*. Princeton University Press. https://press.princeton.edu/books/paperback/9780691130613/theory-of-games-and-economic-behavior.

Nowé, Ann, Peter Vrancx, and Yann-Michaël De Hauwere. 2012. “Game Theory and Multi-Agent Reinforcement Learning.” *Adaptation, Learning, and Optimization* 12: 441–70. https://doi.org/10.1007/978-3-642-27645-3_14.

Otterlo, Martijn van, and Marco Wiering. 2012. “Reinforcement Learning and Markov Decision Processes.” *Adaptation, Learning, and Optimization* 12: 3–42. https://doi.org/10.1007/978-3-642-27645-3_1.

Parvin, Alastair, David Saxby, Cristina Cerulli, and Tatjana Schneider. 2011. “A Right To Build. The next Mass-Housebuilding Industry.” https://static1.squarespace.com/static/5a9837aa8f513076715a4606/t/5eac3dafcdbffc0abd29ce95/1588346311289/RIGHT+TO+BUILD_LR.pdf.

“Plethora Project.” n.d. Accessed August 30, 2021. https://www.plethora-project.com/.

Retsin, Gilles. 2016. “Discrete Assembly and Digital Materials in Architecture.”

Rittel, Horst W. J., and Melvin M. Webber. 1973. “Dilemmas in a General Theory of Planning.” *Policy Sciences* 4 (2): 155.

Rossi, Andrea, and Oliver Tessmann. 2017. “DESIGNING WITH DIGITAL MATERIALS A Computational Framework for Discrete Assembly Design.” *Protocols, Flows and Glitches; Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA 2017*.

Russell, Stuart J. (Stuart Jonathan), and Peter Norvig. 2016. *Artificial Intelligence : A Modern Approach*. Third

Edit. Pearson.

Sanchez, Jose. 2018. “Platforms for Architecture: Imperatives and Opportunities of Designing Online Networks for Design.” In *ACADIA 2018: Recalibration. On Imprecision and Infidelity. [Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture]*, 108–17. Mexico City, Mexico. http://papers.cumincad.org/data/works/att/acadia18_108.pdf.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai. n.d. “Proximal Policy Optimization Algorithms.”

Shekhawat, Krishnendra, Yd Pinki, and José P. Duarte. 2019. “A Graph Theoretical Approach for Creating Building Floor Plans.” In *Communications in Computer and Information Science*, 1028:3–14. Springer Verlag. https://doi.org/10.1007/978-981-13-8410-3_1.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, et al. 2017. “A General Reinforcement Learning Algorithm That Masters Chess, Shogi and Go through Self-Play.” *CoRR* abs/1712.0. http://arxiv.org/abs/1712.01815.

Sutton, Richard S, and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction Second Edition, in Progress*. London, England: The MIT Press.

Turner, John F. C., and Robert Fichter. 1972. *Freedom to Build, Dweller Control of the Housing Process*. New York: Collier macmillan. http://www.communityplanning.net/JohnTurnerArchive/pdfs/FreedomtoBuildCh7.pdf.

Unity Technologies. 2021. “Unity ML-Agents Toolkit Documentation.” April 2021. https://github.com/Unity-Technologies/ml-agents/tree/main/docs.

“Urban Splash.” 2021. 2021. https://www.urbansplash.co.uk/.

Wolf, Mark. 2021. “Refining and Redefining ‘Game Studies.’” *Academia Letters*, January. https://doi.org/10.20935/AL156.

Zhang Zhuoran Yang, Kaiqing, and Tamer Bas. 2021. “Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms.” https://arxiv.org/abs/1911.10635.

Zheng, Stephan, Alexander Trott, Sunil Srinivasa, Nikhil Naik, Melvin Gruesbeck, David C Parkes, Richard Socher, et al. 2020. “The AI Economist: Improving Equality and Productivity with AI-Driven Tax Policies.”