MCAST

Investigating Emergent Behaviours in Deep Reinforcement Learning Agents Through Curriculum-Based Object Interaction

Luca Alfino

Supervisor: Owen Sacco

June - 2025

A dissertation submitted to the Institute of Information and Communication Technology in partial fulfilment of the requirements for the degree of BSc (Hons) in Software Development

Authorship Statement

This dissertation is based on the results of research carried out by myself, is my own composition, and has not been previously presented for any other certified or uncertified qualification.

The research was carried out under the supervision of Owen Sacco

....

Date Signature

Copyright Statement

In submitting this dissertation to the MCAST Institute of Information and Communication Technology, I understand that I am giving permission for it to be made available for use in accordance with the regulations of MCAST and the Library and Learning Resource Centre. I accept that my dissertation may be made publicly available at MCAST's discretion.

....

Date Signature

Acknowledgements

I would like to express my sincere gratitude to my mentor, Owen Sacco, for his guidance throughout this research. Moreover, I would like to thank Jael Vella for her help in proofreading the dissertation prior to publishing.

Abstract

Gameplay mechanisms have been increasing in complexity for years, leading to more resources needed to be spent on general NPCs and game testing for different playstyles [1]. This paper delves into the use of artificial intelligence, specifically reinforcement learning and deep learning techniques, to investigate emergent behaviours of deep reinforcement learning agents inside a hide-and-seek environment. The emergent behaviours are akin to NPCs and game testing, which shows the potential utilization of artificial intelligence for a variety of different areas where emergent behaviours are of high value. The agents are first trained to a satisfactory level where the gameplay is fluid and balanced between the seeker agent and the hider agent. For training, the Unity ML-Package has been used in the Unity game engine. To analyse this, three metrics will be observed: Cumulative Reward, Policy Loss, and Value Loss. For analysis, TensorBoard has been used for the friendly user interface and clear graphs. Afterwards, curriculum learning is introduced to further provoke emergent behaviours by introducing new objects inside the environment, which can be used by the agents to exploit the game engine. The results of this paper are compared with the base paper [2], which also contained a focus on emergent behaviours inside a physics engine.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ML	Neural Network
DL	Deep Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DRLA	Deep Reinforcement Learning Agents
MARL	Multi-Agent Reinforcement Learning
PPO	Proximal Policy Optimization
SGA	Stochastic Gradient Ascent
GAE	Generalized Advantage Estimation
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
RND	Random Network Distillation
TD	Temporal Difference
SARSA	State Action Reward State Action
UI	User Interface
YAML	Yet Another Markup Language
CL	Curriculum Learning
NPC	Non-Playable Character

Chapter 1: Introduction

1.1 Research Background

This study centres on machine learning, a subfield of artificial intelligence that employs algorithms and data that replicates knowledge processes and enhances overall precision [8]. Machine learning is categorized into three categories: Supervised, Unsupervised and Reinforcement Learning. Reinforcement Learning is the most common choice for this particular problem such as papers [2] and [5]. Similarly, this paper utilizes deep reinforcement learning agents to investigate emergent behaviours. C# scripts will be created for a hider agent and seeker agent with their model results to be analysed quantitatively for the chosen metrics via tensorboard.

1.2 Research Purpose

The study's main goal is to illustrate the potential utilization of machine learning for NPCs, automated game testing and other similar areas, by investigating the emergent behaviours of the deep reinforcement learning agents. For analysis, tensorboard is used for the agents training. Whereas a real-time snapshot is taken of the agent's model mid-observation of an emergent behaviour, all emergent behaviours are then compared with the base paper [2]. This study serves as a valuable reference for researchers interested in the utilization of machine learning for deep reinforcement learning agents. Furthermore, it is useful for NPCs,

automated game testing and areas of similar nature.

1.3 Hypothesis and Research Questions

The Hypothesis of this research is: "Deep Reinforcement Learning Agents (DRLA) will exhibit emergent behaviours throughout their training process, mainly during curriculum learning with the introduction of external objects." This study will support or refute the deep reinforcement learning agents by observing emergent behaviours. The results will be evaluated against earlier research. The research questions of the study are as follows:

- **RQ1:** Do trained Deep Reinforcement Learning Agents (DRLA) exhibit fluid and balanced gameplay before curriculum learning is introduced?
- **RQ2:** Do Deep Reinforcement Learning Agents (DRLA) detect exploitable mechanics within the game engine to gain an advantage?
- **RQ3:** Do Deep Reinforcement Learning Agents (DRLA) exhibit emergent behaviours, particularly during curriculum learning with the introduction of external objects?

1.4 Significance of the research

The significance of this study lies in its empirical exploration of the application of Deep Reinforcement Learning Agents. By investigating the efficacy of these agents in the context of game development, this study contributes to the growing body of knowledge in both the reinforcement learning and gaming domains.

Firstly, the findings aim to contribute and advance the understanding of how reinforcement learning techniques can be tailored and optimized for specific gaming applications.

Secondly, the findings aim to contribute to advancing the understanding of how reinforcement learning agents can be utilized for thorough NPC optimization, game testing,, and other similar areas. This empirical insight may lead to the development of more sophisticated and adaptive AI systems within games, enhancing player experiences and immersion.

1.5 Research Boundaries

This study focuses specifically on the use of machine learning to investigate emergent behaviors inside a gaming domain. Hence, the use case is gaming systems, mainly NPCs and game testing. However, it does not delve into other potential areas such as robotics or cybersecurity where emergent behaviours are crucial.

1.6 Dissertation Overview

The dissertation contains the introduction, which focuses on the research background, the purpose, significance of the research, and the hypothesis and research questions. Furthermore, it explores the research boundaries for this dissertation.

The literature review delves into past work, mainly three research papers which highly relate to my work. This consists of environments, configurations, curriculum learning and emergent behaviours akin to game-testing and gaming domains.

The research methodology contains my implementation for the dissertation. It consists of the research pipeline, the environmental setup/configurations, the training done on the agents, and curriculum learning.

The analysis of results and discussion contains the results of the research methodology, questionnaire results, and research comparisons to the papers in the literature review. It is mapped directly in correspondence to the methodology environments, containing the environment results. Furthermore, it contains the curriculum learning results and the exploitable mechanics / emergent behaviour findings. Finally, it ends with the questionnaire results and research comparisons with previous papers.

Finally, the conclusion was made with recommendations for future work.

Chapter 2: Literature Review

This chapter will contain a thorough literature review of the selected research topic. It will cover three research papers which are closely related to my own research.

2.1 Introduction

The world of Artificial Intelligence (AI) has been trending for some time, with multiple categories of Machine Learning (ML) techniques being created and further optimized for different use cases. One of such techniques is Deep reinforcement learning (DRL), which combines Deep Learning and Reinforcement Learning. Deep Learning (DL) consists of multiple processing layers, which together make up a deep neural network which rely on function:

$$y = f(x;\theta) \quad [3]$$

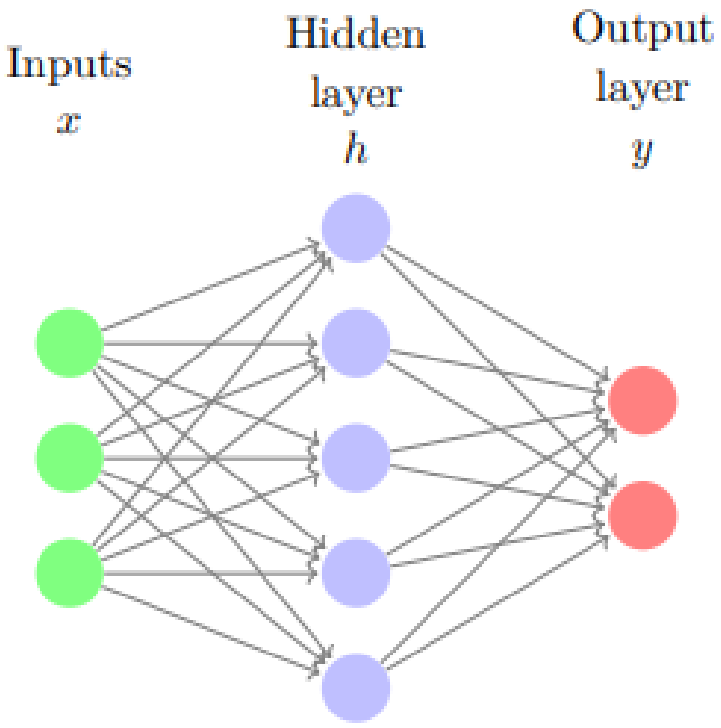


Figure 2.1: *Example of a neural network with one hidden layer [3]*

Reinforcement Learning (RL) consists of sequential decision-making. A key component of RL is that an agent learns good behaviour, this means that it can incrementally acquire new skills and behaviours [3]. Such examples include dexterous in-hand manipulation [9]

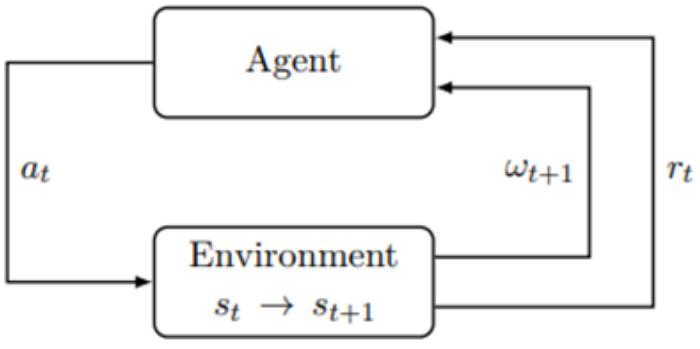


Figure 2.2: *Basic architecture of Reinforcement Learning Model [2]*

Understanding how Deep Learning and Reinforcement Learning work together

to create the DRL methodology is crucial for the upcoming research papers. Although different experiments are performed, all three use DRL extensively throughout.

2.2 Related Work

2.2.1 Emergent Tool Use From Multi-Agent Autocorricula

Environment and Policy Optimization

To create intelligent artificial intelligence which can solve a wide range of complex human-like tasks has been a long-standing challenge in the AI Community, particularly agents which can interact with objects in a physical environment. One approach to creating these agents is to thoroughly specify the desired tasks and incrementally train a RL agent to solve them [2].

Bowen Baker et al. [2] created a hide and seek environment based within the popular MuJoCo physics simulation engine, which is suitable for RL simulations [10]. The environment consists of agents divided into two teams consisting of one to three hiders and one to three seekers. The hider attempts to avoid the seeker's line of sight, while the seeker attempts to maintain vision of the hider. The environment consists of movable objects such as boxes and ramps, which are interactable via locking. The hiders get a preparation phase which takes up to 40% of the entire episode, where seekers are immobile. Neither the hiders nor the seekers can get rewards during this time. After the 40% timer, the hiders become active. Hiders earn a reward of +1 if hidden, -1 if seen. Likewise, the seekers get +1 if hiders are maintaining vision, -1 if they are not. Agents can

move via force alone x/y axes, rotate and grab whilst having a 135-degree cone of vision and 30-range sensor, similar to a lidar widely used in autonomous driving and accurate human motion recognition [10]. The game is simulated within a randomized environment to encourage emergent strategies [2]. In this paper, multi-agent reinforcement learning (MARL) was used to train the agents. Unlike traditional RL which is applicable to single-agent systems, MARL contains two or more agents co-operating or competing in a shared environment [11].

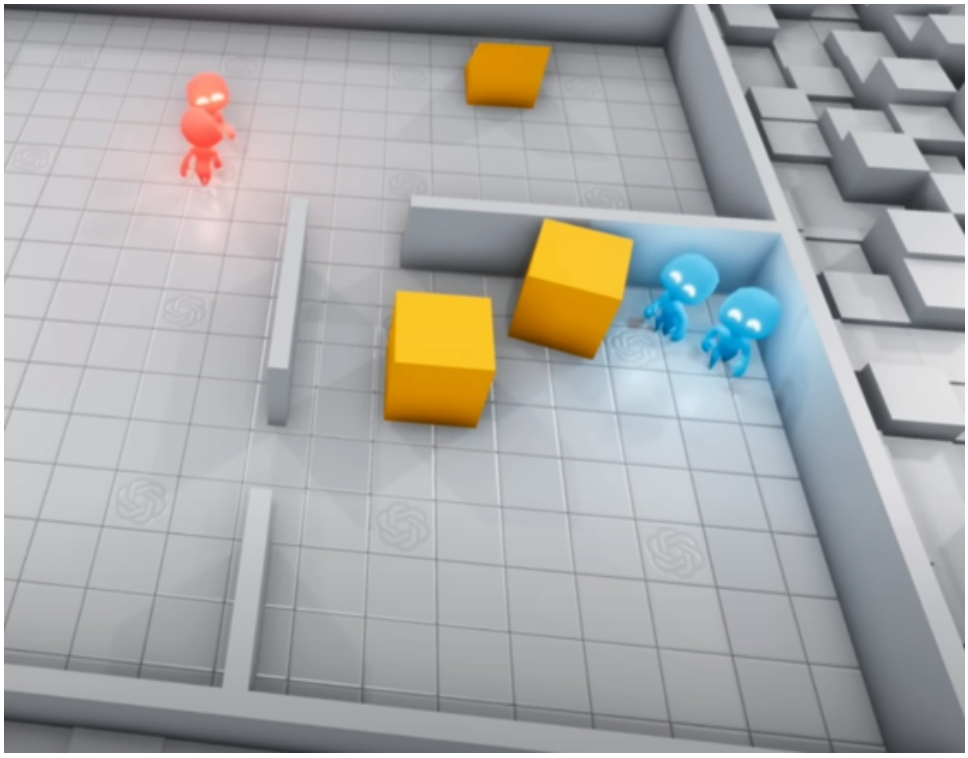


Figure 2.3: *Agents trained in a simple environment for the first few million rounds [2]*

Proximal Policy Optimization (PPO) is a policy optimization method in RL that uses multiple epochs of stochastic gradient ascent (SGA) to perform each policy update. SGA:

$$\hat{g} = \mathbb{E}_t \left[\hat{\nabla}_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

Figure 2.4: *Stochastic Gradient Ascent function [2]*

is an algorithm that finds the maximum (highest point in a curve) of a function that computes the gradient using a single randomly selected data point or a small random subset. Generalized Advantage Estimation (GAE) is also used in PPO to reduce the variance of policy gradient estimates while introducing some bias [12].

GAE uses function:

$$\hat{A}_t = \sum_{l=0}^{H} (\gamma\lambda)^l \delta_{t+l}$$

Figure 2.5: *Generalized Advantage Function [2]*

PPO modifies traditional policy gradient methods to prioritize stability and sample efficiency. It achieves this by restricting how much the policy can change during each update using a clipping mechanism.

It does this by using the key objective function:

$$L = \mathbb{E}_t \left[\min \left(l_t(\theta)\hat{A}_t, \text{clip}(l_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

Figure 2.6: *Proximal Policy Objective Function [2]*

PPO achieves a balance between samples efficiency, stability and simple implementation.

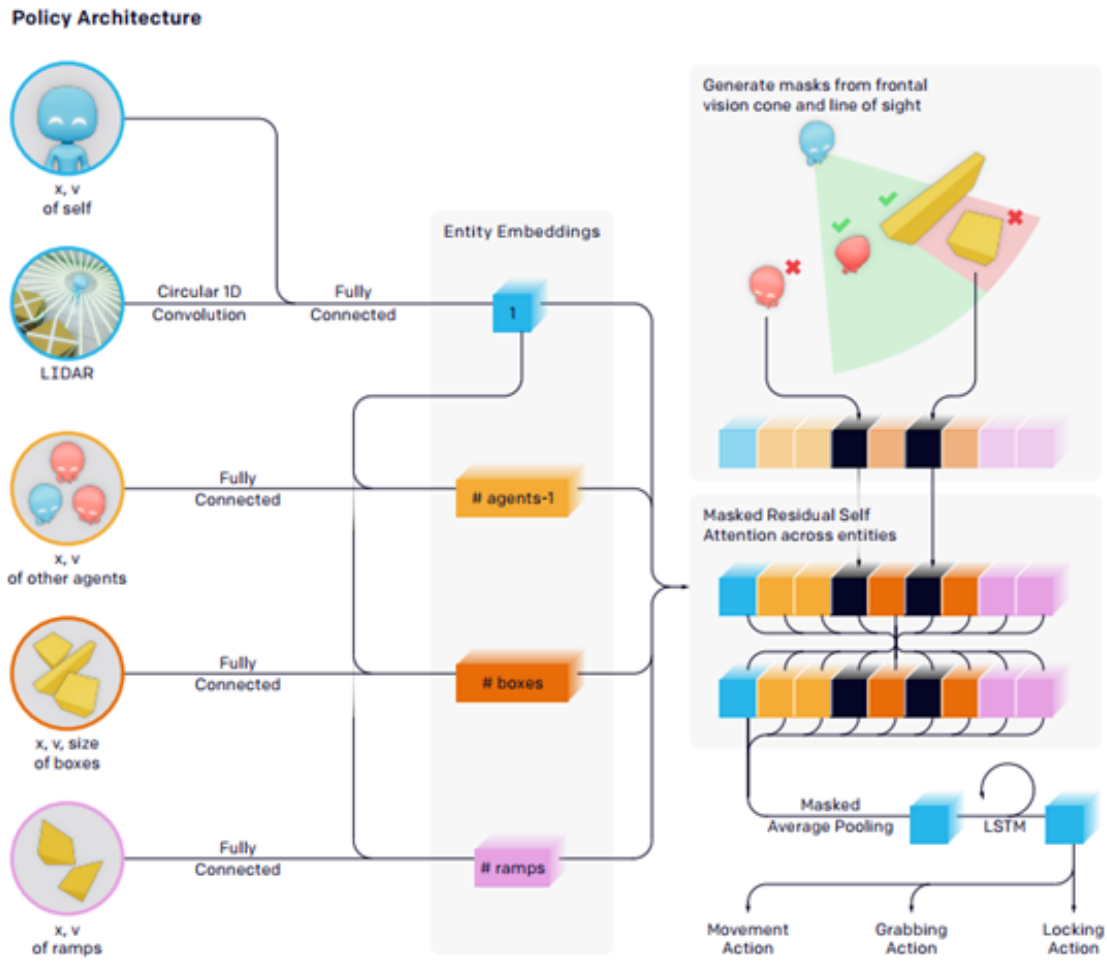


Figure 2.7: *Agent Policy architecture, which depicts the previously discussed environment and policies in one structured framework [2]. A Long short-term memory (LSTM) network is a type of Recurrent Neural Network (RNN) which can learn long-term dependencies in sequential data [4]*

Curriculum Learning

Throughout the years, both classic discrete games such as Backgammon [13] and Go [14] as well as continuous real-time domains such as Dota [15] and StarCraft [16] have showed success in leveraging multi-agent autocorricula to solve multi-player games. However, despite the impressive emergent complexities in these environments, the learned behaviours are abstract and disembodied from the physical world [2]. Whereas in other work such as international conference of Learning representations [14], the studies showcased emerging complexities in physically grounded environments. The success of these configurations exhibits a show of confidence that inducing autocorricula in physically grounded environments could enable agents to achieve a number of human-relevant skills [2]. During the study, a mix of competitive and cooperative physics-based environments were introduced in which agents compete in a simple game of hide-and seek. Agents manage to learn multiple emergent skills and strategies where they intentionally change the environment to suit their needs, by only using a visibility-based reward function and competition. For example, hiders learn to block the seekers by barricading doors or constructing multi-object forts whereas to counter this, the hiders learn to use ramps to jump into the hider's shelter [2]. Moreover, it is observed that through multi-agent competition and standard reinforcement learning algorithms, signs of dynamic and growing complexities became apparent. As many as six distinct adaptations of strategy and counterstrategy emerged, which further proves that multi-agent co-adaptation may scale better with environment complexity and centres more around human-interpretable be-

haviour than intrinsically motivated agents [2].

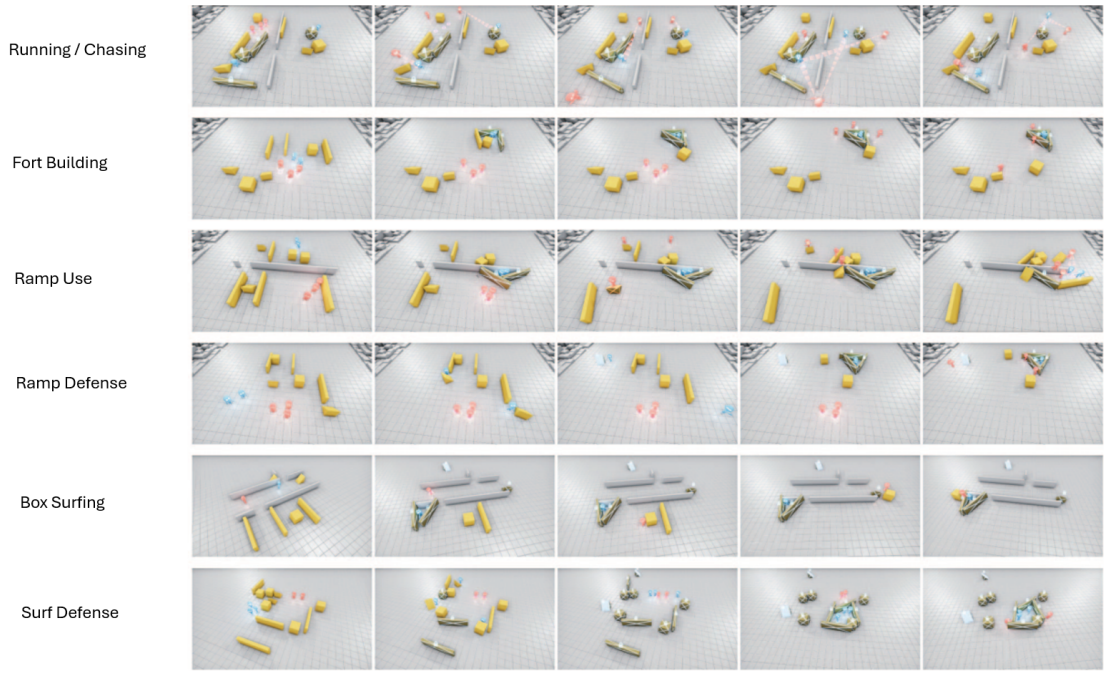


Figure 2.8: *Each image, from left to right, illustrates the progression of each emergent strategy [2]*

There are no direct incentives for agents to explore or interact with objects, strategies are a result solely of auto-curriculum induced by multi-agent competition [2]. This emphasises the focus of auto-curriculum methodology over intrinsic motivation (more on this below), with the breakdown of how agents adapted over time being as follows:

Initial Learning Phase (0 Episodes): Hiders and seekers learn basic behaviours, such as running away (hiders) and chasing (seekers).

After 25 million episodes: Hiders learn to use tools (e.g., boxes) to intentionally modify the environment. They begin constructing secure shelters by pushing boxes together or against walls and locking them in place to hide more effectively.

After 100 million episodes: Seekers acquire rudimentary tool use. They figure out how to move and use ramps to jump over obstacles, enabling them to breach the hiders' shelters.

After 110 million episodes: Hiders counter the seekers' new strategy by bringing ramps to the edge of the play area and locking them in place, removing the seekers' primary tool.

After 380 million episodes: Seekers innovate by using a ramp and a box together. They push a box to the edge, jump on it using a ramp, and surf the box into the hiders' shelter by applying force while airborne.

Hiders' response (at 380M episodes): Hiders adapt again by locking all boxes in place before constructing shelters, preventing seekers from using the boxes for movement.

Findings and Results

The default model uses a batch size of 64,000, 1.6 million parameters and requires 132.3 million episodes (31.7 billion frames) over 34 hours of training to reach stage four of the skill progression (i.e. ramp defence). It was observed that larger batch sizes lead to much quicker training time which reduces the number of required optimization steps [2].

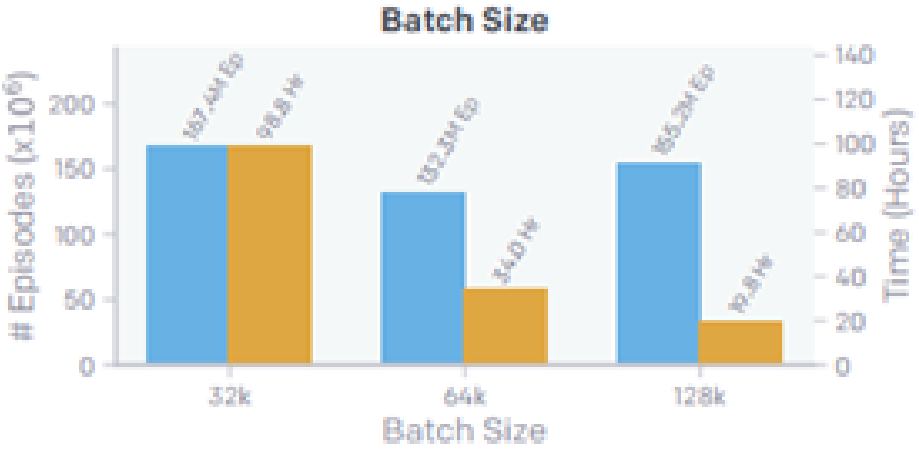


Figure 2.9: *Number of episodes and wall clock required to reach stage 4 [2]*

It is also found that emergent auto-curriculum is robust as long as the environment during training was robust. If randomization is reduced, fewer stages of skill progression emerge, with less sophisticated strategies instead emerging. Intrinsic motivation is a popular paradigm for encouraging unsupervised exploration and discovery, such as Random Network Distillation (RND) [2].

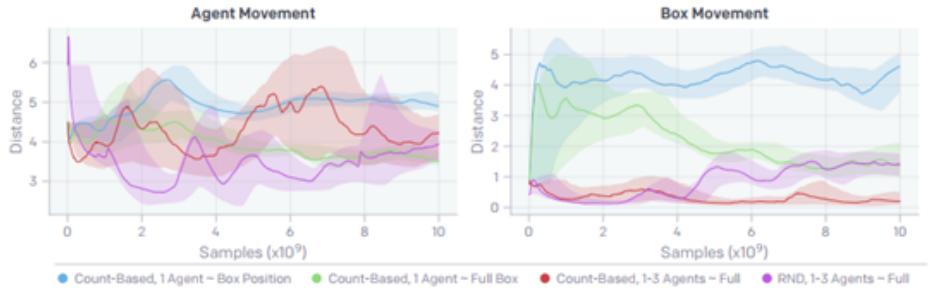


Figure 2.10: *Comparison of behavioural statistics (net box movement and maximum agent movement) for count-based exploration variants and Random Network Distillation (RND) across different state representations and agent configurations [2].*

It is found that count-based exploration leads to larger agent and box movement if state-representation is set at 2-D Location of the boxes. Whereas intrinsic motivation, which is the training of agents via exploration and learning

without relying on external rewards, performs slightly better in complex environments consisting of higher dimensions [2]. As for the rewards, the hider started with an initial decline which turned to a heavy dominant incline after a certain number of episodes until the end. Oppositely, the seekers started with a healthy incline for rewards but were at a heavy decline until the end of training.

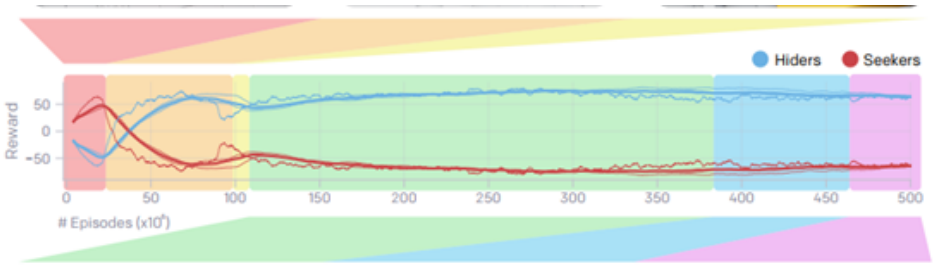


Figure 2.11: *Hider and Seeker Reward Values [2].*

This work demonstrates that simple game rules, multi-agent competition, and standard reinforcement learning algorithms can lead to the emergence of complex strategies and skills. Through the hide-and-seek environment, up to six distinct rounds of strategy and counterstrategy were observed, highlighting the potential for open-ended growth in complexity via multi-agent self-play in complex environments. The results serve as a proof of concept that multi-agent auto-curricula can induce physically grounded, human-relevant behaviours. However, the agents required vast amounts of experience to progress due to misalignment between reward functions and behaviour. Reducing sample complexity will be an important future direction, with potential improvements through better policy learning algorithms and architectures to enhance sample efficiency and performance on transfer tasks [2].

2.2.2 Multi-Agent Reinforcement Learning - Implementation of Hide and Seek

Introduction

This paper [5] focuses on a similar concept of the previously discussed paper [2] with different policies, environments, implementations and results. It experiments with different reinforcement theories and algorithms to which the best performing one was chosen to implement the game of hide and seek. On-Policy methods consist of decisions which are made using the value of the policy, the results are determined by the mapping of the action to the policy which updates the value function [17]. They are "soft" and non-deterministic which means that the policy will always contain an element of exploration, three such common examples include Soft, Greedy and Soft-Max [18]. Off-Policy can also be used for learning behaviour and estimation, which also contains the "soft" element. However, Off-Policy can aid in the updates of the estimated value function, opposite to On-Policy which is based purely on experience [19] [20]. In brief, Off-Policy algorithms possess the ability to distinguish exploration from control [18].

Reinforcement of Theories and Algorithms

Temporal difference (TD) learning algorithms can be both On-Policy and Off-Policy, this study explores an On-Policy and Off-Policy algorithm using the TD reinforcement theory.

TD algorithms are used to evaluate value functions [21], in contrast to methods that require waiting until the final reward is received before updating the value function, TD learning updates the value function incrementally during the course of the episode [2]. Once the final reward is received, the state-action pair values are refined and the course taken to reach the goal state is traced to update each value accordingly [5]. Represented mathematically as:

$$V(\text{state}_t) \leftarrow V(\text{state}_t) + \alpha \left(\text{Reward}_t - V(\text{state}_t)\right)$$

Figure 2.12: *basic Temporal Difference (TD) learning update rule [5].*

Where alpha is a constant parameter.

Each state-action value is updated at each state by estimating the final reward represented mathematically as:

$$V(\text{state}_t) \leftarrow V(\text{state}_t) + \alpha \left[\text{Reward}_{t+1} + \gamma V(\text{state}_{t+1}) - V(\text{state}_t)\right]$$

Figure 2.13: *standard Temporal Difference (TD) update rule [5].*

Where Reward t+1 is the reward observer at time t+1 and y (gamma) is the estimation factor. Q-Learning is an Off-Policy learning algorithm with the ability to learn the optimal policy, even in cases of exploratory methods or random policy selection [22]. The Q-Learning algorithm consists of this methodology:

1. Set the Q-values table, $Q(s_t, a_{ct})$.
2. Observe the current state s_t.
3. For that state, select an action a_{ct} using a policy (e.g., Greedy, ε-Greedy, or SoftMax).

4. Observe the reward r for the selected action and the resulting new state.

5. Update the Q-table using the observed reward and transition.

6. Set the current state to the new state and repeat the cycle until the goal is met [5].

Mathematical Representation of Q-Learning:

$$Q(\text{state}, \text{action}) = (1 - \alpha) \cdot Q(\text{state}, \text{action}) + \alpha \cdot \left(\text{reward} + \gamma \cdot \max_a Q(\text{next state}, a)\right)$$

Figure 2.14: *Q-Learning Action-Value Update Equation [6].*

State Action Reward State Action (SARSA) is an On-Policy TD learning algorithm which opposite to Q-Learning, does not update the Q-Values using the maximum reward. The next action as well as the reward is determined using the same set of policies which were used for the same previous action [5].

The Sarsa algorithm consist of this methodology:

1. Initialize the Q-table values: $Q(s_t, a_{ct}, r, s_{t+1}, a_{t+1})$.

2. Observe the current state s_t.

3. Select an action a_{ct} for that state using a policy (e.g., Greedy, ε-Greedy, or SoftMax).

4. Execute the action and observe the reward r, the new state s_{t+1}, and the new action a_{t+1}.

5. Update the Q-table using the observed reward and estimated future reward

6. Set the current state to the new state and repeat the cycle until the goal is met [5].

Mathematical representation of Sarsa:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

Figure 2.15: *Sarsa Action-Value Update Equation [5].*

For this paper, it was concluded that Q-Learning was the best algorithm. Since it is a model-free, fast-iterating reinforcement learning approach that efficiently learns optimal policies in dynamic environments without requiring prior knowledge of the environment's transitions or rewards. Its adaptability and simplicity make it ideal for solving the given problem statement [5].

Environment and Policies

The environment consists of walls that acts as structures, made up of predefined classes of square and rectangles from pygame's library. The walls which were the environment, were made up of single/multiple characters of 'W' in the string structure which in turn made the environment dynamic. This makes the environment easy to change since the agent's training is independent of the environment. Moreover, the user can change the structure of the game without the need of understanding the code [5].

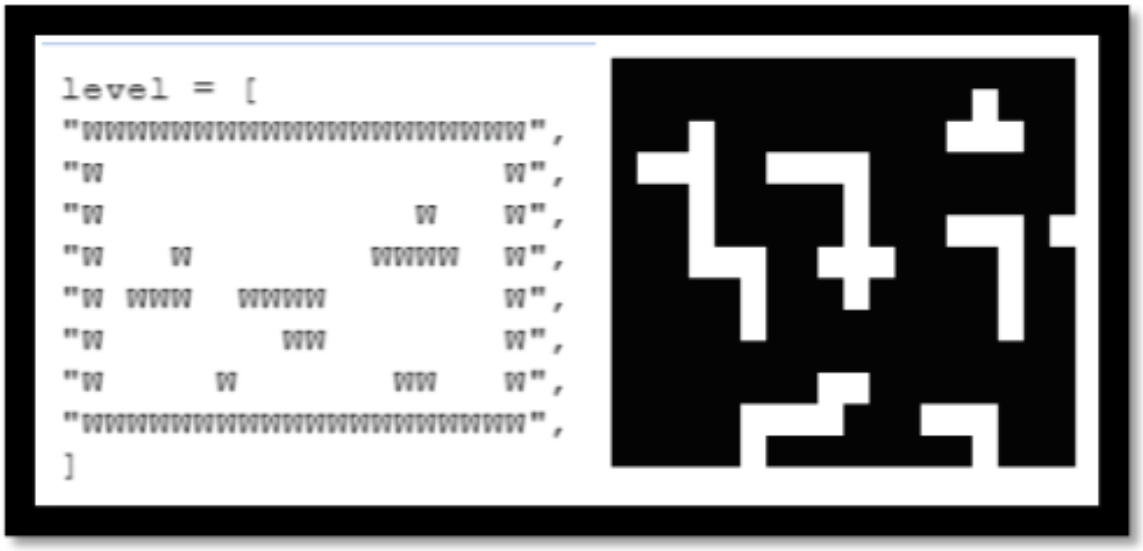


Figure 2.16: *Environment Creation [5].*

Agents move randomly by selecting random X,Y coordinates within a fixed range and combining linear movements (LEFT/RIGHT/UP/DOWN) with angular movements (CLOCKWISE/ANTI-CLOCKWISE). Linear movement allows free navigation, while rotation is handled using the formula angle=(angle+1) %360, enabling agents to rotate incrementally. This ensures dynamic, unrestricted movement. The agent also needs to be able to see and react to the environment, thus the Vision arc was introduced. The vision arc consists of a number of lines originating from the center of the agent. These help the agent detect and understand the environment with the given policies [5].

Figure 2.17: *Three-dimensional presentation of Agent Ray cast [5].*

The agents contain similar policies to the previously discussed paper [2], it goes as follows:

General Policies

- **Exploration:** Agents receive a reward for covering more area and a penalty for limited movement.
- **Wall Avoidance:** Penalty is applied for collisions with walls to encourage safer navigation.

Hider Policies

- **Distance Maintenance:** Hiders are rewarded for staying far from the seeker and penalized for being too close.
- **Escape Vision:** A large penalty is applied if a hider is caught within the seeker's field of vision.

Seeker Policies

- **Chasing Hider:** Seekers are rewarded for reducing the distance to hiders and penalized if the distance increases.
- **Catch Hider:** A large reward is given when a hider enters the seeker's vision.

Q-Learning: Balances exploration (random actions) and exploitation (best-known actions) [5].

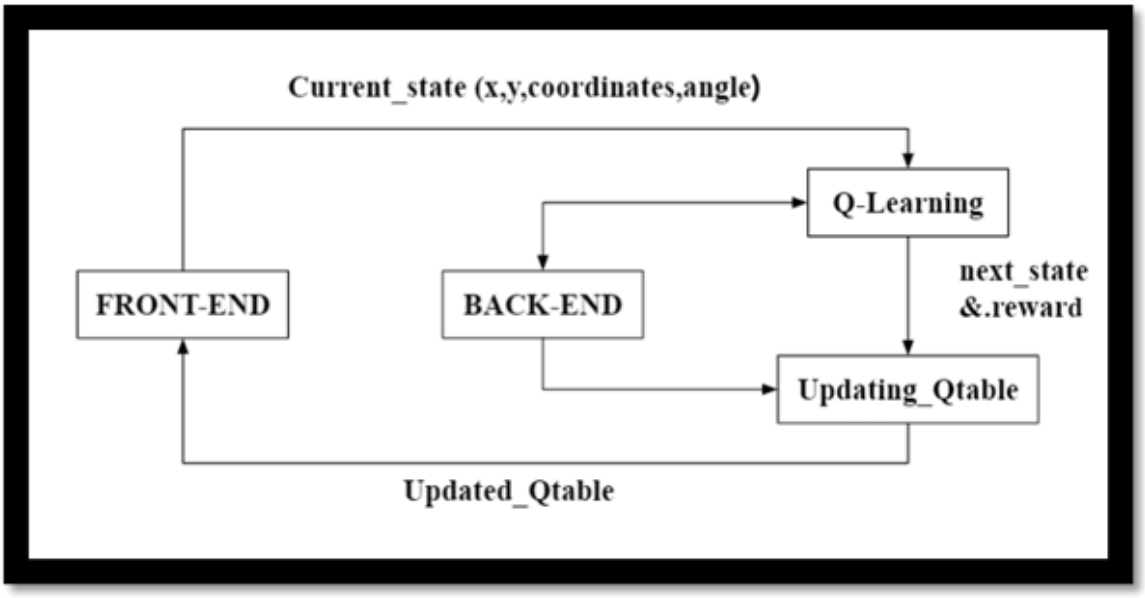


Figure 2.18: *Overview of the system [5].*

Results

The model was trained for a long duration, in the initial stage the agents were understanding the environment and their end goal, similarly to the previously discussed paper [2]. However, some strategies emerge after a certain number of episodes with the hiders being able to hide from the seekers whereas the seekers had learned to catch the hiders [5].

A noteworthy outcome to the results from this study, shows how it differs from the results from the previously discussed paper [2]. We can see that from the initial start, the seeker's rewards start to decline, however it gains a steady and dominating incline.

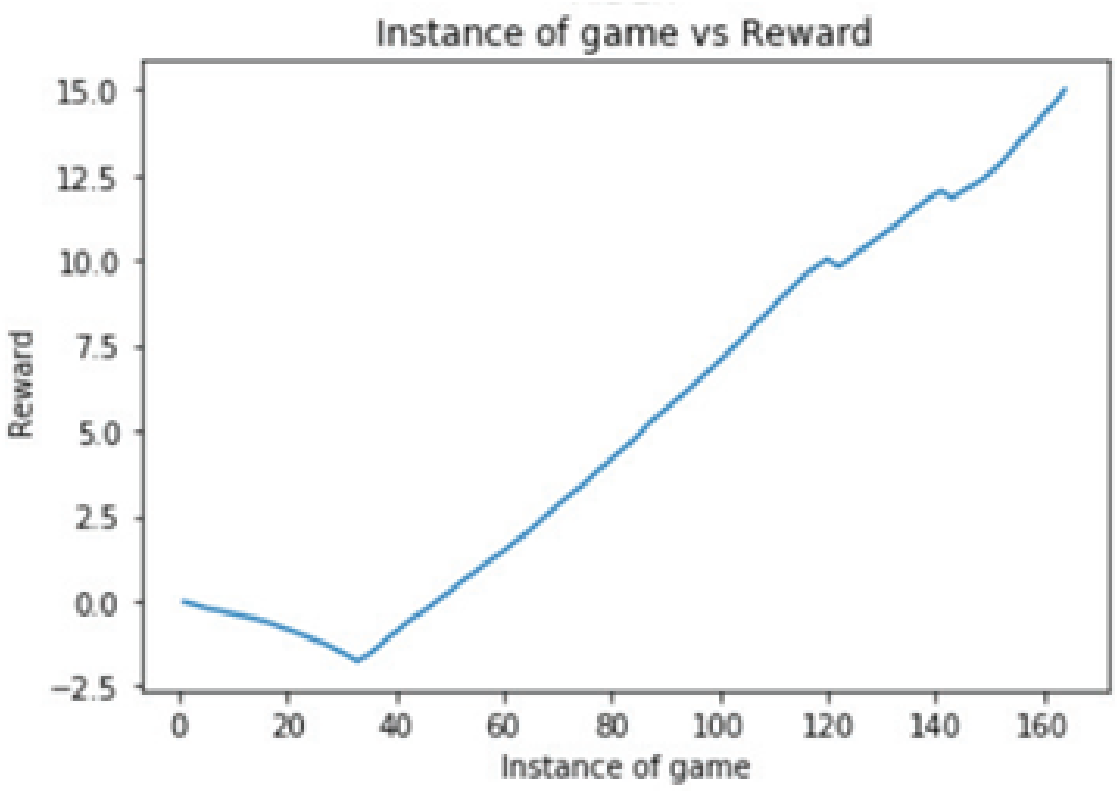


Figure 2.19: *Seeker vs reward plot [5].*

Whereas for the hider, it starts with a healthy incline for the rewards, however it ends up declining heavily towards the later episodes.

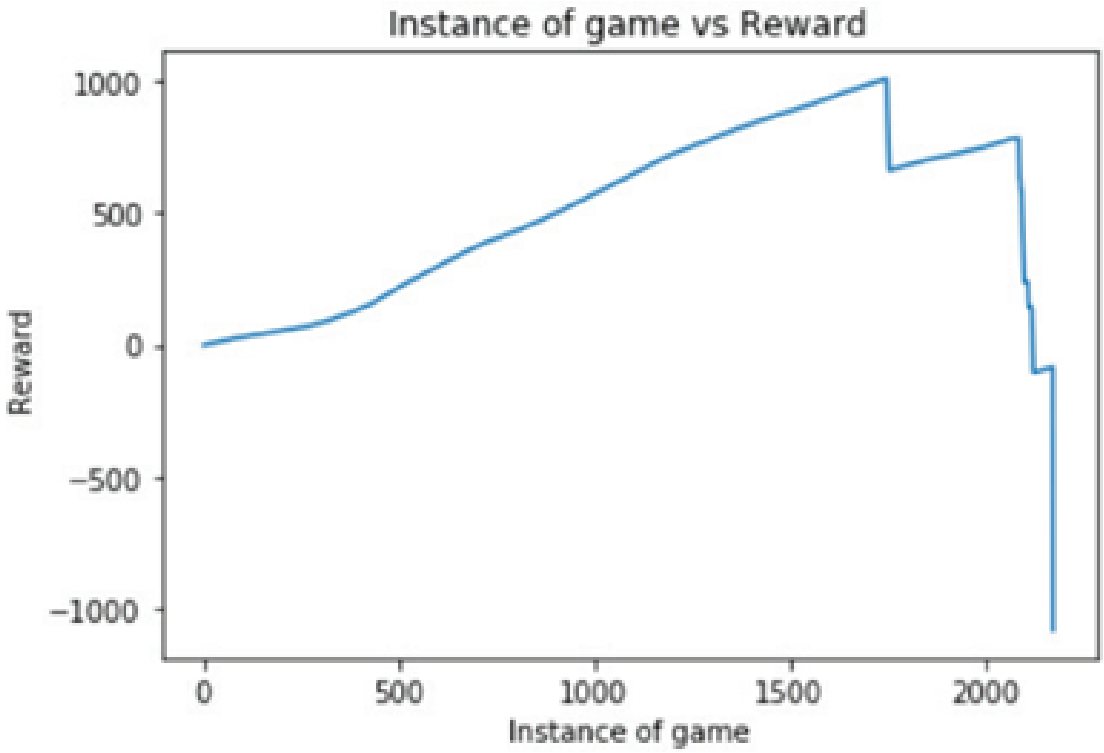


Figure 2.20: *Hider vs reward plot [5].*

When the rewards are plotted against each other, it outputs table:

Instance No	Hider Reward	Seeker Reward
1	0.989	-0.065
2	630.999	-15.677
3	586.288	-0.119
4	191.428	34.775
5	45.385	69.099
6	-103.586	131.803
7	-206.365	376.431
8	-91.584	499.468
9	-82.292	514.991
10	**-1082.291**	**+1515.916**

Figure 2.21: *Hider and Seeker Reward Values [5].*

Which when compared to the previous study [2]:

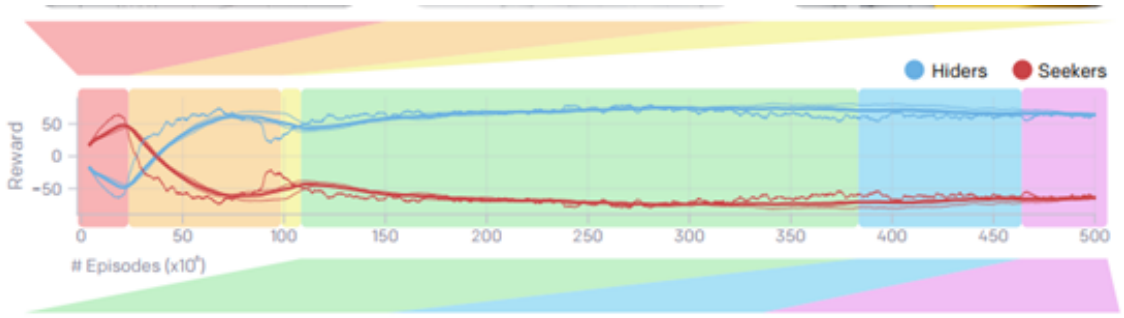


Figure 2.22: *Hider and Seeker Reward Values [2].*

Shows results that contradict each other, possibly due to differences in environments, agent policies, and implementation details. The variations in how the hide-and-seek problem was modelled, such as the reward structures, agent behaviours, or training setups, significantly influenced the outcomes between both studies.

2.2.3 Augmenting Automated Game Testing with Deep Reinforcement Learning

Introduction

When creating games, thousands of developers and designers are involved. This includes map sizes which are measured in square kilometres with an abundant number of characters and abilities. Thus, the need for NPC optimization and automated testing increases. Scripting behaviours is a common way of doing it, however this method presents drawbacks when it comes to adaptability and learnability [7]. RL models present the option to implement current scripts and automated solutions by learning directly by playing the game without the need for human intervention. This is due to modern RL algorithms being able to explore

complex environments [23] and exploit in-game mechanics [2] [7].

Environment, Policies and Training

The hypothesis is tested on a set of environments where multiple test bug classes were tested, such as navigation bugs, exploits etc. For this research, the following environment will be focused on:

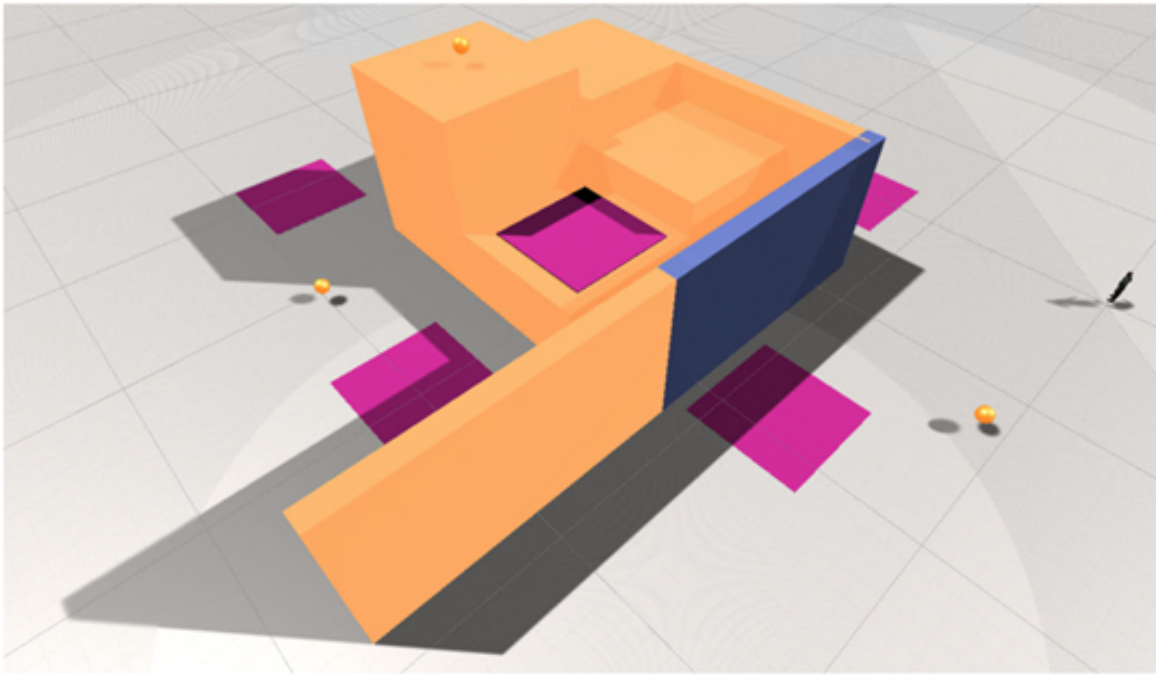


Figure 2.23: *Blue wall – missing collision mesh [7].*

The figure includes the following elements:

- **Yellow spheres:** Represent navigation goals.
- **Blue wall:** Indicates a missing collision mesh.
- **Pink squares:** Areas where the agent will get stuck upon entering [7].

The observation state includes an aggregated vector of normalized features: agent's position (R^3), velocity (R^3), rotation (R^4), goal distance (R), climbing status (B), ground contact (B), jump cooldown (R), reset timer (R), and a 12-ray vision array. Agents are rewarded (+R) for moving towards the goal and penalized (-R) for moving away [7].

Different algorithms were compared:

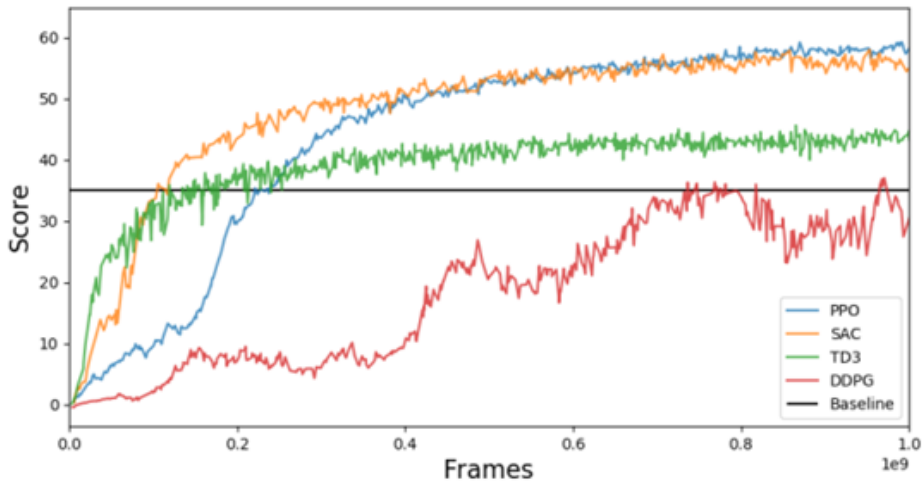


Figure 2.24: *Comparison of algorithms [7].*

PPO was found to perform the most efficiently, similarly to paper [2]. Hence, the algorithm was used in a server hosted on a single machine.

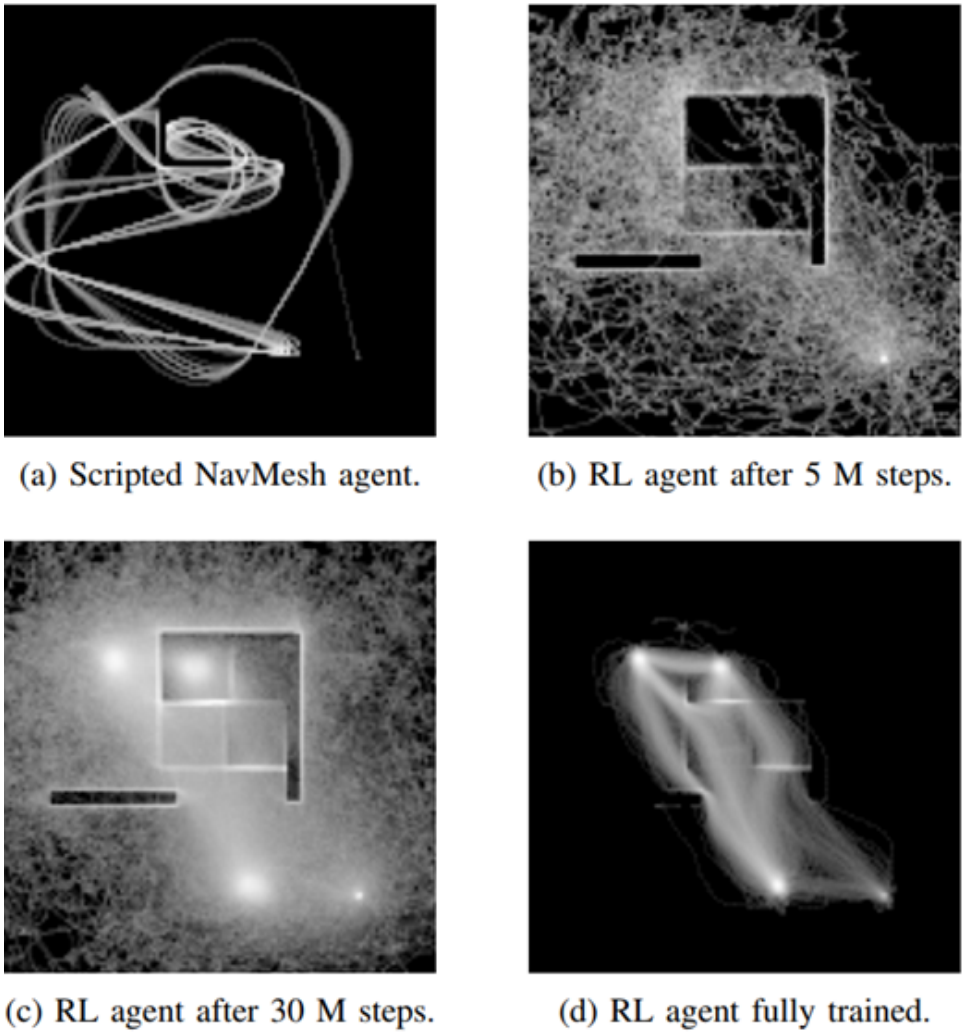
The machine used for training included:

- **Processor:** AMD Ryzen Threadripper 1950X @ 3.4 GHz
- **GPU:** NVIDIA GTX 1080 Ti

The agent took between 50-1000M environmental frames depending on the complexity of the environment [7].

Results

For the chosen environment above:



(a) Scripted NavMesh agent. (b) RL agent after 5 M steps.

(c) RL agent after 30 M steps. (d) RL agent fully trained.

Figure 2.25: *Generated Heat-maps [7].*

Figure 2.26 illustrates agent behaviour during training in the Exploit sandbox. Fig 2.26(a) shows the scripted agent following its predefined navigation system. Figs. 2.26(b), 2.26(c), and 2.26(d) depicts how an RL agent movement evolves over time. Early in training, visited states are evenly distributed across the map. As training progresses, agents refine their behaviour which by the end identify near-optimal paths to the goals [7].

This paper shows that RL can be used to augment traditional scripting methods to test video games especially when focusing on single isolated tasks, while noting that not all problems are solved more effectively with RL [7].

2.3 Literature Conclusion

The first two papers [2] [5] will serve as the primary references for implementing the hide-and-seek environment, as well as defining the policies and configurations used throughout this research. Both provide comprehensive frameworks and methodologies specifically tailored to agent-based environments and multi-agent systems, aligning closely with the requirements of this study. Their detailed focus on environmental dynamics, training policies, and agent behaviours ensures that they are relevant for addressing my Hypothesis and research questions. The third paper [7] will mainly be utilized for its game-testing methodologies, particularly in resolving practical challenges such as issues with missing wall-collision meshes and other implementation-related concerns. This paper emphasizes rigorous testing strategies, which are essential for ensuring the robustness and reliability of the hide-and-seek environment. However, as mentioned the main focus of this research is not based solely on game testing. It is based on emergent behaviours which in this case are within a gaming domain. These three research papers have been selected because they collectively offer a targeted and high-quality foundation for this work. Their focus on relevant methodologies and problem areas directly supports the objectives of my study. While other papers were reviewed, they were either too general in scope, lacked practical implemen-

tation details, or did not sufficiently address the specific technical challenges and hypotheses explored for my dissertation. Therefore, these three references were chosen for their depth, relevance, and applicability to the core research questions.

Chapter 3: Research Methodology

3.1 Research Methodology Pipeline

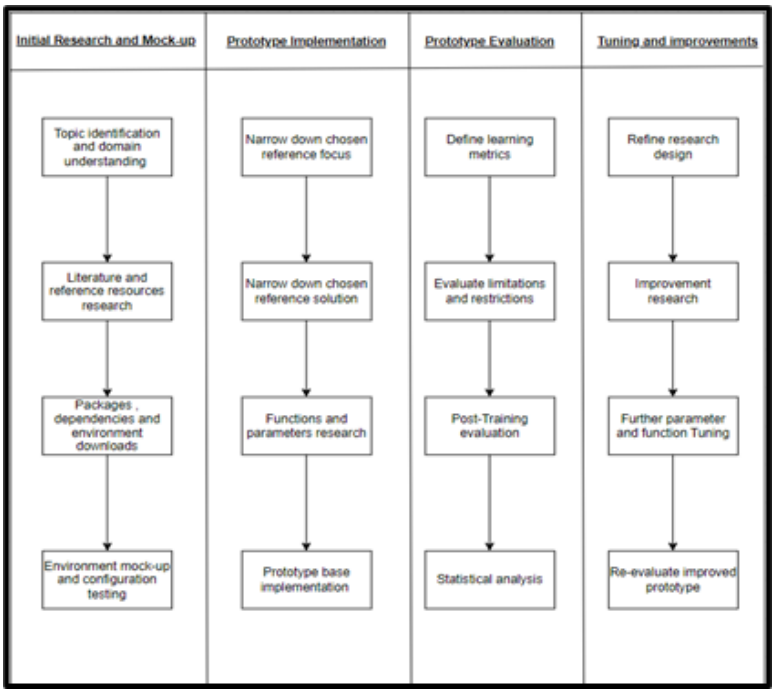


Figure 3.1: *Research Methodology Pipeline*

The research methodology is outlined in the pipeline above, which consists of the following stages:

Initial Research and Mock-up

- **Topic Identification and Domain Understanding**
  - Define the research problem and understand the domain to be explored.
- **Literature and Reference Resources Research**
  - Gather relevant academic references and existing resources.

- **Packages, Dependencies, and Environment Download**
  - Download necessary packages and dependencies and configure the environment.
- **Environment Mock-up and Configuration Testing**
  - Create and test a mock-up to ensure the environment functions correctly.

Prototype Implementation

- **Narrow Down Chosen Reference Focus**
  - Refine the focus by selecting the most relevant references.
- **Narrow Down Chosen Reference Solution**
  - Select efficient solutions from within the chosen references.
- **Functions and Parameters Research**
  - Identify critical functions and tunable parameters.
- **Prototype Base Implementation**
  - Develop the initial working prototype.

Prototype Evaluation

- **Define Learning Metrics**
  - Establish the criteria for evaluating learning performance.

- **Evaluate Limitations and Restrictions**
  - Identify challenges and constraints in the setup, including hardware limitations.
- **Post-training Evaluation**
  - Evaluate the results of the trained prototype.
- **Statistical Analysis**
  - Analyse results using TensorBoard.

Tuning and Improvements

- **Refine Research Design**
  - Adjust the research design based on initial findings.
- **Improvement Research**
  - Explore further areas for potential prototype improvements.
- **Further Parameter and Function Tuning**
  - Optimize parameters and functions based on the findings.
- **Re-evaluate Improved Prototype**
  - Test and analyse the results of the improved prototype.

The process of prototype evaluation and tuning follows an iterative loop. After each evaluation phase, the prototype is analysed based on the defined performance

metrics to identify areas for improvement. Subsequently, tuning is performed by adjusting parameters or changing the environment to address identified shortcomings. This refined prototype is then subjected to further evaluation.

This is a mono-method approach where the prototype produces quantitative results. Moreover, a questionnaire will be given to people over 18 to understand others' perceptions and interpretations based on this subject.

3.2 Environmental Setup / Configuration

The environment is built on the Unity game engine since modern game engines are powerful for visual simulations with sophisticated physics and agent interactions. Moreover, these game engines are designed with intuitive user interfaces (UI) and easy to use [24].

The Unity Machine-Learning Package is an essential component which will be installed to facilitate the extensive development and implementation of the environment. Moreover TensorBoard, a tool created for efficient and versatile programming frameworks due to AI and DL rapid growth [25], will be used as a tool to visualize training progress and results. For this situation, Cumulative Reward, Policy loss and value loss will be analysed for these reasons:

- **Cumulative Reward:**

  Cumulative reward is the most direct measure of an agent's success in achieving its objective within the environment. For the hider agent, this reflects its ability to evade the seeker agent, while for the seeker agent, it represents its efficiency in locating the hider agent. By tracking cumulative

reward, it is possible to assess the effectiveness of the agents' strategies over time and determine whether meaningful learning is occurring.

- **Policy Loss:**

  Policy loss captures how well the agent is optimizing its decision-making process to achieve its objectives. In reinforcement learning, this metric reflects the updates to the agent's policy based on its interactions with the environment. Fluctuations or spikes in policy loss indicate the challenges associated with learning in a dynamic, multi-agent environment, where optimal strategies are constantly evolving. Tracking this metric provides insight into the stability and progress of the agent's learning process, particularly in a competitive setting like hide and seek.

- **Value Loss:**

  Value loss measures the discrepancy between the predicted value of a state and the actual reward obtained, indicating how accurately the agent is learning to predict long-term outcomes. This metric is crucial in environments where agents must plan strategically, as in hide and seek. While an increase in value loss can suggest challenges in predicting long-term rewards—often due to the complexity of the environment or evolving agent policies—the steady improvement in cumulative reward provides reassurance that the agents are learning effectively. Thus, value loss serves as a diagnostic tool to understand the complexities of reward prediction in competitive and adaptive scenarios.

3.3 Standard Settings / Configurations

Throughout the dissertation, there will be multiple environments/modifications to train the agents. However, there will be common configurations / policies amongst the agents and environments. This includes:

- A single hider agent and a single seeker agent are included.
- Both agents' positions are randomized at the start of each episode to prevent overfitting, which can hinder model generalization and reduce prediction reliability [26].
- The seeker agent remains inactive for a few seconds at the beginning of each episode. During this period, all rewards are disabled.
- The hider agent receives rewards when it remains out of the seeker's vision and avoids physical contact. It loses rewards if seen or touched. Conversely, the seeker earns rewards for seeing or making contact with the hider and is penalized otherwise.
- Both agents can see and label environmental objects through ray casting.
- Both agents use Proximal Policy Optimization (PPO) as the reinforcement learning algorithm. PPO was chosen for its superior performance in the third paper [7], as shown in Figure 2.25, and because it was also the algorithm used in the base paper [2]. Additionally, PPO scales well in large and continuous state spaces, unlike Q-learning.

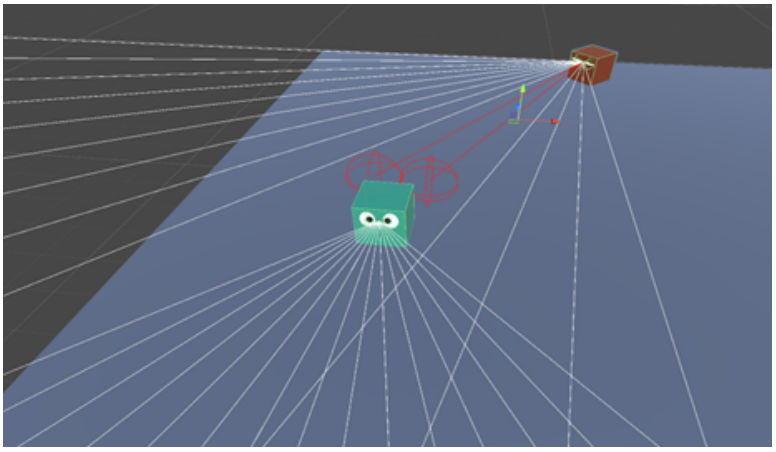


Figure 3.2: *Agent's Ray casts*

In the directory, there will also be a Yet Another Markup Language(yaml) file which contains important configurations for the agents (refer to Appendix C). Each environment/modification will correspond to the results section in order.

3.4 Training and refinement process

3.4.1 First Environment

For the first environment, the agents were left to train on a large single 40 * 40 platform

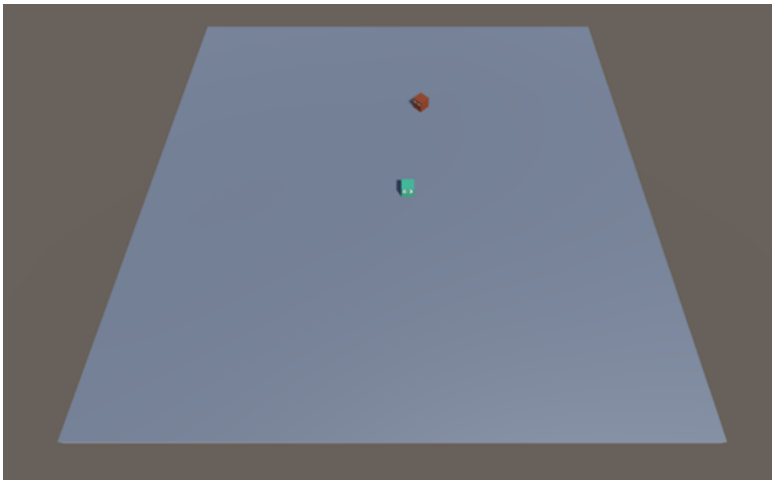


Figure 3.3: *First environment consisting of a large single 40 * 40 platform*

The configurations were as stated above in section 3.3

First Environment, First Modification

At attempting to improve the performance of both agents, intrinsic rewards which originate from the agent itself such as exploration and extrinsic rewards which are externally provided by the environment as feedback [27] , were modified. The curiosity parameter of both agents, which is an intrinsic reward, was increased to encourage exploration, likewise the gamma parameter of both agents, an extrinsic reward, was also increased to encourage the agent to plan for long-term outcomes

First Environment, Second Modification

In an attempt to optimize the hider agent. It has been given a massive advantage over the seeker.

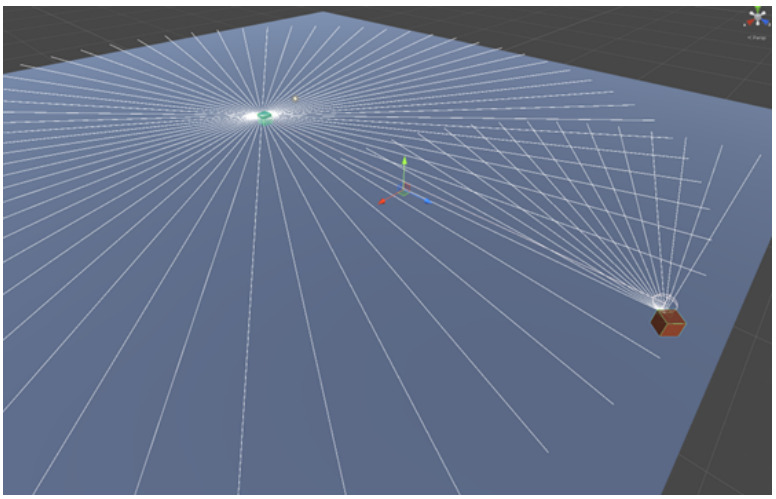


Figure 3.4: *Seeker vs Hider agent raycast modification*

The raycast has been given a substantial number of rays per direction, 360° view and the length of the rays also cover the whole area. No negative rewards

will be given for this run should the seeker be in the hider's vision, since its raycast covers the whole area.

First Environment, Third Modification

Since no difference was observed with the modified raycast for the hider agent, other than minor improvements in fleeing. The hider agent will now receive a negative reward of -0.5 should the seeker agent be in the hider's raycast while maintaining a consistent reward accumulation when the seeker agent is not within range. The positive reward has been increased from +0.5 to +0.8. The raycast will remain a revolution to work as intended. In theory, this will encourage the hider agent to move away once the seeker agent is spotted.

To make it more balanced, the distance of the raycast has been decreased and the seeker agent has also been given a revolution raycast:

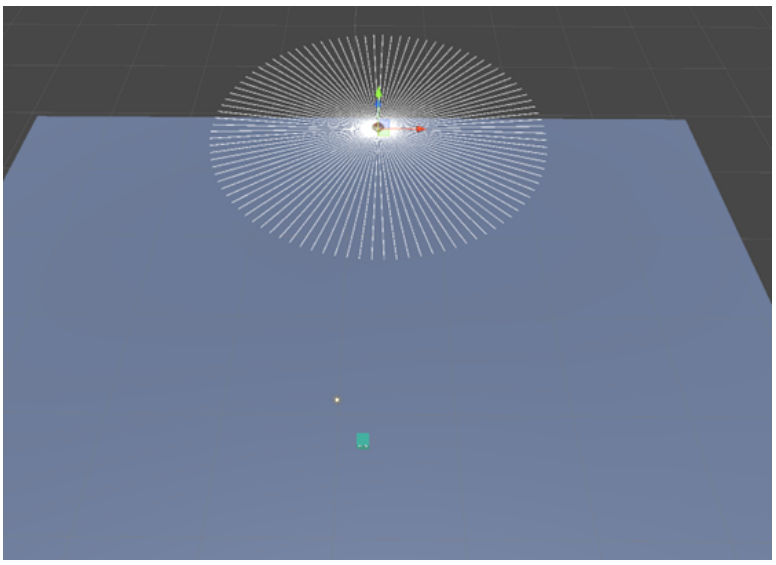


Figure 3.5: *Seeker vs Hider agent second raycast modification*

3.4.2 Second Environment

Due to the hider agent exploiting the game in unforeseen ways(refer to 4.2.3) , the environment needed to be updated. Hence, a restricted zone acting as a wall has been added to the platform to force both agents to stay within the play area.

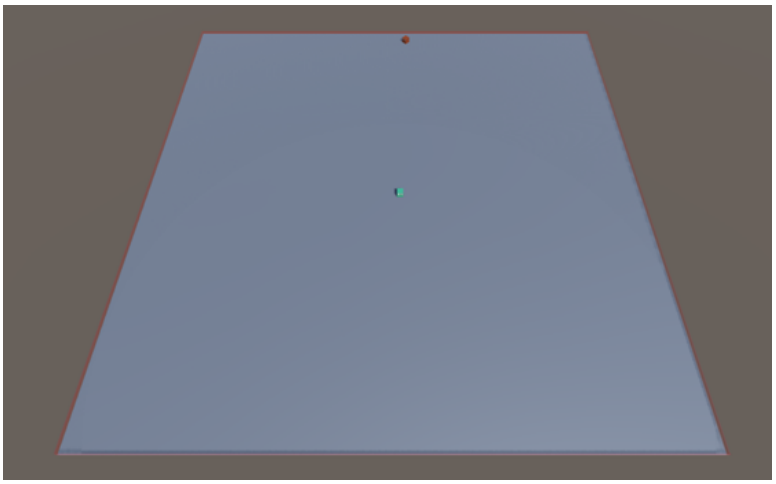


Figure 3.6: *Restricted Zone*

This restricted zone will cause the agents to lose a reward of -1.0 should they collide with it, randomizing their position back on the platform to continue the episode.

3.4.3 Third Environment

Due to the hider agent finding another exploitation in my environment (refer to 4.3), the restricted zone will be replaced with a physical wall which cannot be used to its advantage.

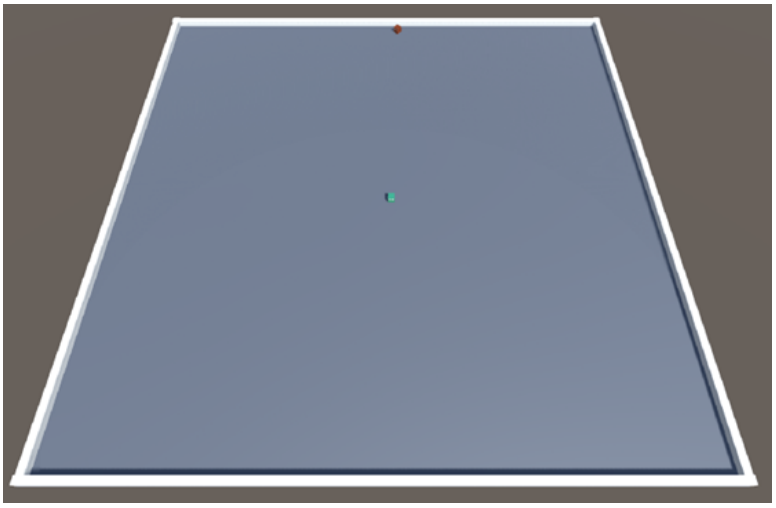


Figure 3.7: *Physical Walls*

3.4.4 Fourth Environment

The third environment consists of the seeker agent being entirely favored (Refer to 4.4). Hence, the fourth environment will contain important changes to help the hider agent perform better while maintaining balance. These are:

- Decreased area size by 25%.
- Increased curiosity strength for both agents from 0.1 to 0.25.
- Decreased raycast length from 25 to 17.5 for both agents.
- Added internal walls to the environment for the hider agent to use to its advantage.

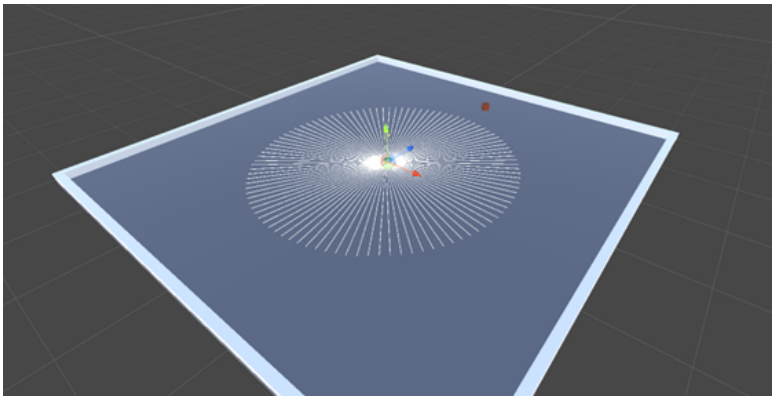


Figure 3.8: *Fourth environment size decrease / RayCast range decrease*

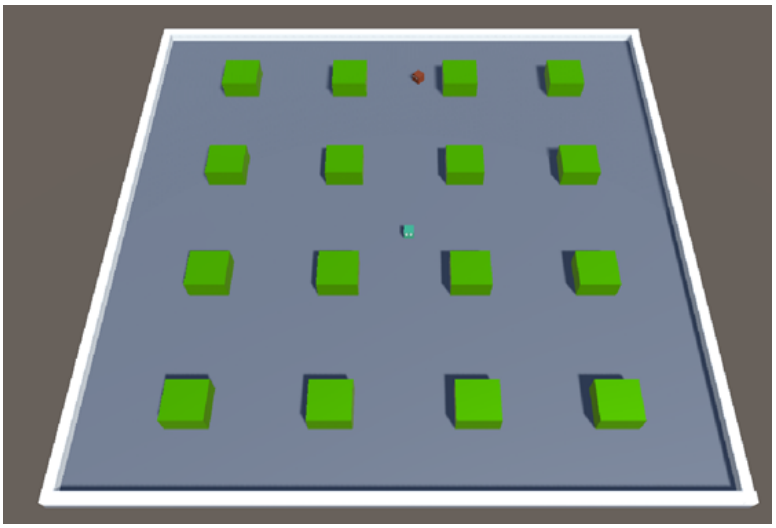


Figure 3.9: *Fourth environment, Internal walls*

Due to the internal walls, both agents will have their positions randomized around the middle of the platform so that they will not spawn inside the walls

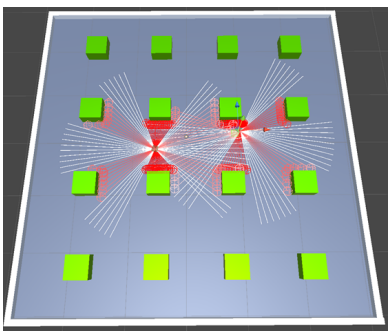


Figure 3.10: *Fourth environment, random positioning*

Fourth Environment, First Modification

To cater for the raycast passing through internal walls(refer to 4.5), the scale of the agent's size has been increased:

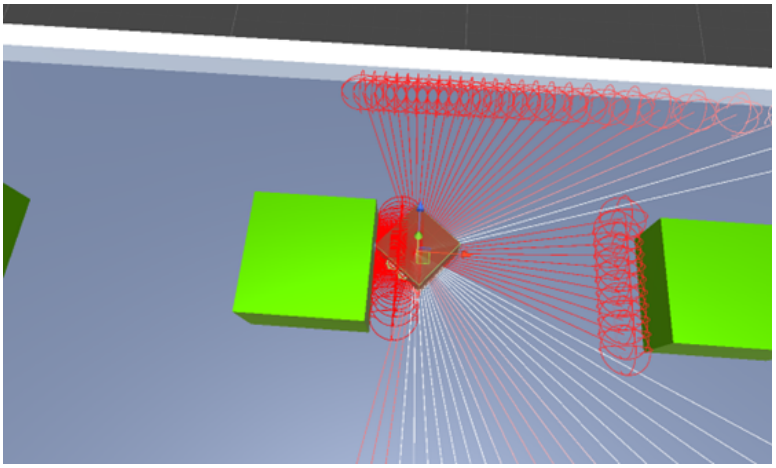


Figure 3.11: *Fourth environment, Agent size scaling*

To help the hider agent further improve its performance, incentives were introduced at the corner of the maps to act as hiding zones. They give a reward of +0.4 each second the hider agent is within the zone. Moreover, to help with the hider agent finding these incentives, as well as encourage the hider agent to look around the area for the seeker – the curiosity factor has been increased further to a strength of 0.3 via the yaml parameters

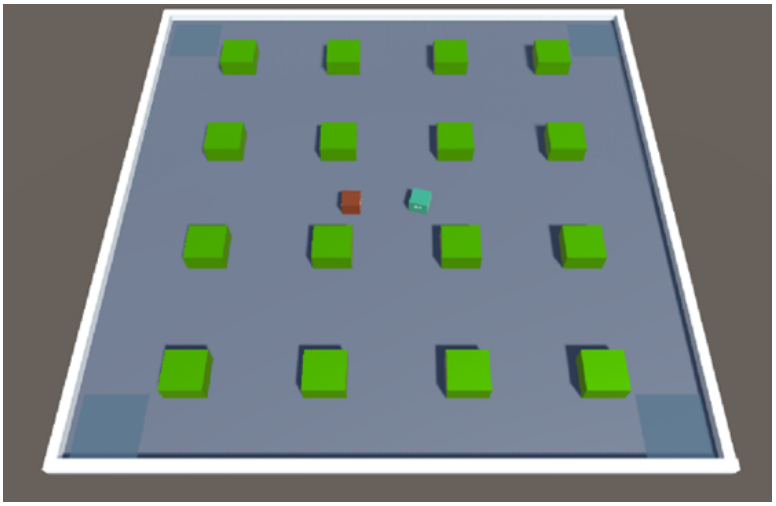


Figure 3.12: *Fourth environment, added hider agent Incentives*

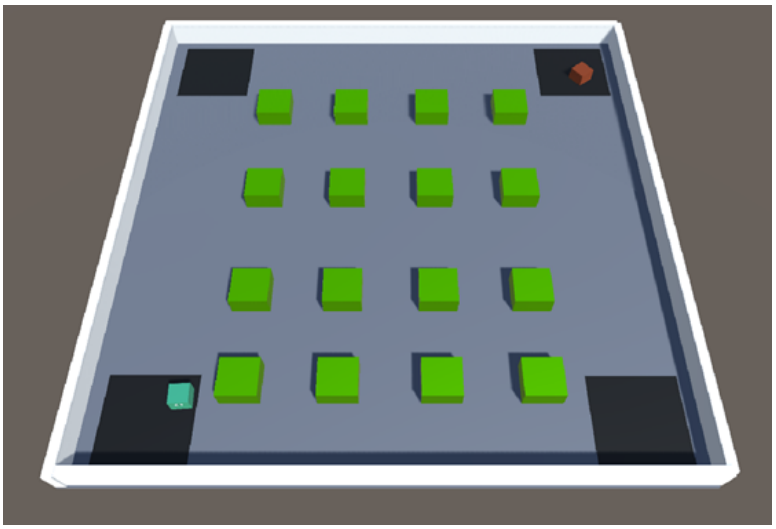


Figure 3.13: *Hider hiding from seeker, Seeker searching for hider*

Fourth Environment, Second Modification

Based on the observed results from the policy and value loss, as well as the identified issue with the raycast being obstructed by the incentive platform (refer to 4.5.1), further adjustments will be made to optimize the training environment. The area will be reduced in size to accelerate the agents' learning process by focusing their interactions within a more constrained space. Additionally, the ray-

cast will be repositioned slightly above the agent's original point to ensure it can detect entities located on top of the incentive platform, thereby mitigating the impact of the previously identified bug

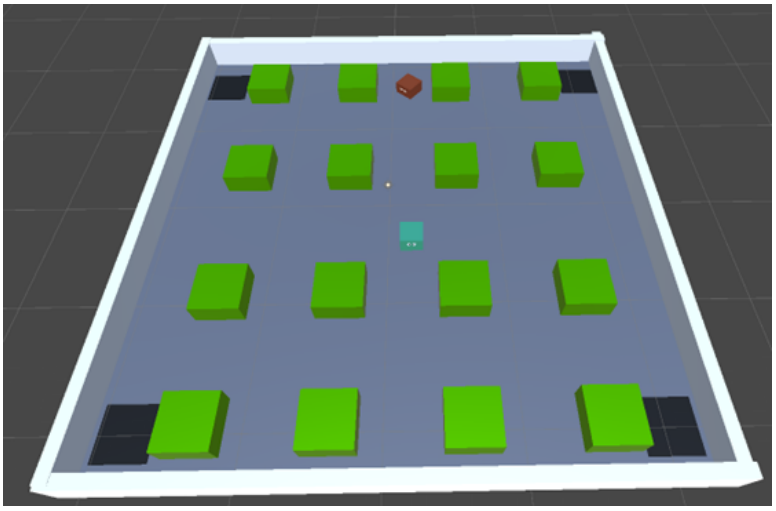


Figure 3.14: *Reduced environment size*

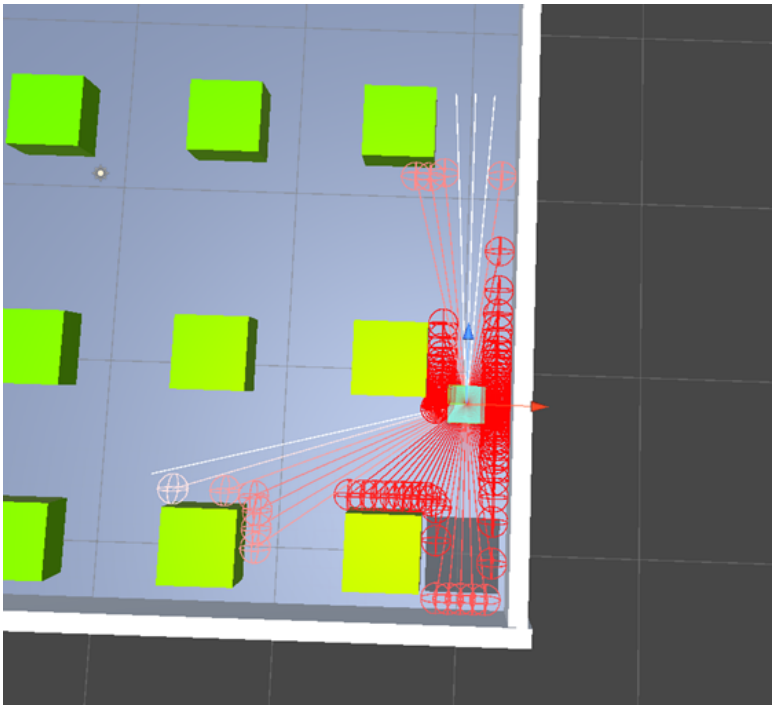


Figure 3.15: *Agent raycast passing over the incentive platform*

Fourth Environment, Third Modification

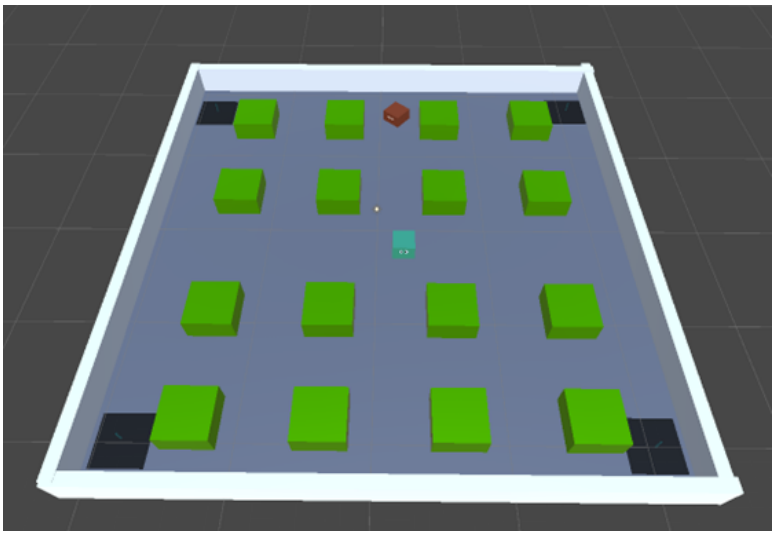


Figure 3.16: *Added incentive Poles*

Incentive poles were added in the middle of the incentives to help the agents better detect where the incentive platforms are located. Additionally, the hider agent had its curiosity strength increased from 0.1 to 0.2, while also having the reward for the incentive platform increased from +0.4 to +0.8. This is done so that the hider agent will be able to utilize the incentive platforms to a more optimal effect for better results.

3.5 Curriculum Learning

The objective of this phase is to refine the agents' learning process progressively by introducing increasingly complex tasks and challenges, facilitated by curriculum learning. This approach is expected to foster the emergence of sophisticated and adaptive behaviours, potentially mirroring patterns observed in NPC behaviour, game testing scenarios and other similar events in gaming domains. By iteratively increasing the difficulty and scope of the tasks, it is anticipated that the agents will exhibit emergent behaviours aligned with the overarching goals of this research. Each curriculum attempt will be trained using the fourth environment, third modification of brain models, as these configurations have produced the best results. Furthermore, curriculum learning will not be used in the traditional sense to further train the agents. Since satisfactory hide-and-seek outcomes have already been achieved, the primary focus is now to encourage and provoke additional emergent behaviours. Consequently, Policy Loss and Value Loss will no longer be considered, as the primary objective is no longer to optimize agent training but to provoke novel behaviours in a manner similar to game testing as mentioned above The first research question, "Do trained DRLAs exhibit fluid and balanced gameplay before curriculum learning is introduced?", has been addressed up to this point. The results indicate that the DRLAs successfully demonstrated fluid and balanced gameplay, providing a solid foundation for introducing curriculum learning. This, in turn, enables further investigation of the remaining research questions and hypotheses. The significance of this research question lies in its role as a prerequisite for progressing with the subsequent phases of the

study.

3.5.1 Curriculum Learning, First Environment

The first environment consists of speed boosts which only the seeker agent can use. Once the seeker agent interacts with a single speed boost, it will double its speed for a duration of 4 seconds to give it an advantage at finding and chasing the hider agent. The speed boost itself will not give a reward to the seeker agent, as it is expected that the agent will misinterpret the rewards which will affect training by provoking it to go after the speed boosts inefficiently.

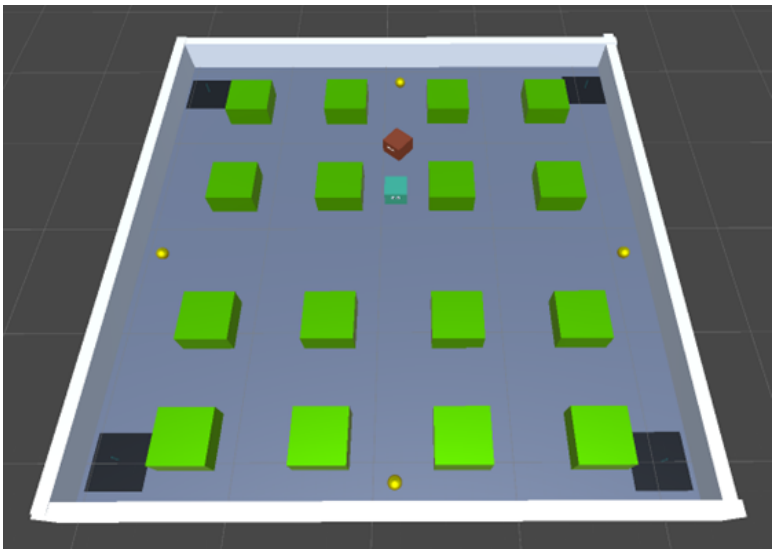


Figure 3.17: *Added speed boosts*

3.5.2 Curriculum Learning, Second Environment

The second environment consists of a purple lever placed at the corner of the map, only the hider agent can activate this lever. Once the lever is activated, walls will spawn to block the seeker agent from being able to enter around the incentive area. It is not expected for the hider agent to jump off as it has a

much higher reward potential by staying on the incentive.

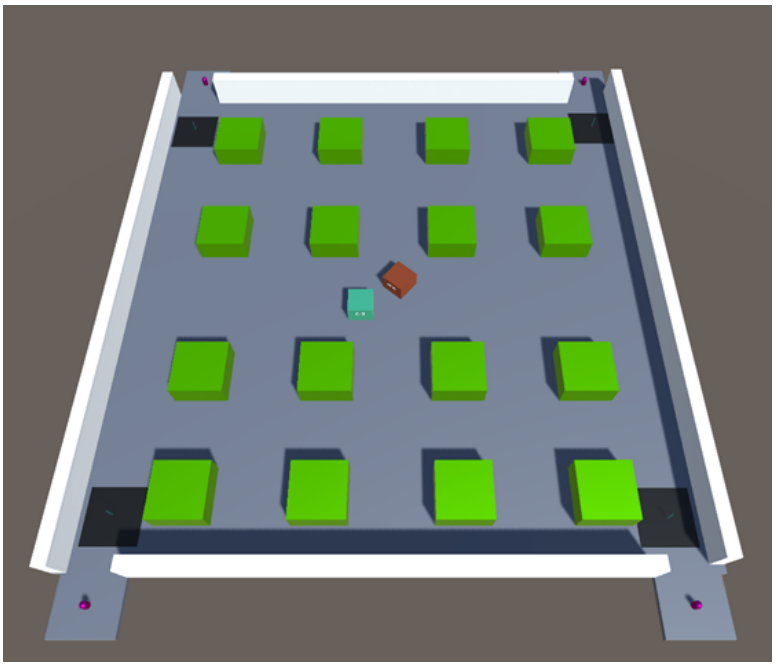


Figure 3.18: *Curriculum Learning, second environment*

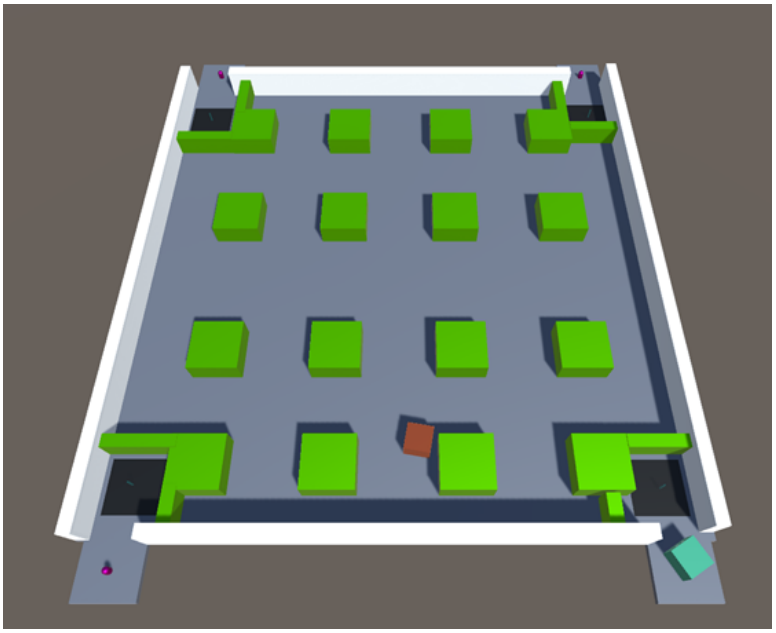


Figure 3.19: *Walls triggered by lever*

3.5.3 Curriculum Learning, Third Environment

The third environment will keep the levers for the hider agent; however, both agents will have the ability to pass through the four walls that connect the spawned walls. While Figure 3.20 depicts these walls as slightly transparent, this is purely a visual aid to help users understand the environment's implementation. For both agents, these walls are perceived as standard, with no visual distinction from regular walls.

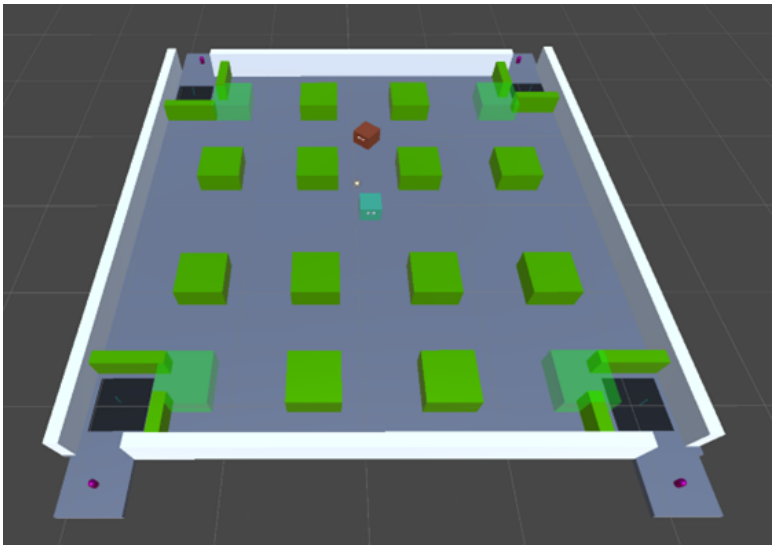


Figure 3.20: *Curriculum Learning, Third Environment*

This environment is inspired by paper [7] which consisted of a similar walk-through wall which the agents had also learned to exploit (refer to figure 2.24).

3.6 Methodology Conclusion

Once the DRLAs were trained to a sufficient level, curriculum learning had been introduced to provoke emergent behaviours by the DRLAs. Although not each environment provided the expected results, it proved to be effective in finding game engine exploits even with the given hardware limitations as per appendix D. It is also worth noting that game exploits by the DRLAs were seen during the initial training and refinement process (3.4) – although these were minor.

Chapter 4: Analysis of Results and Discussion

4.1 Methodology – Results Mapping

In this chapter, the results of the experiments are presented in alignment with the structure outlined in Chapter 3 (Methodology). Each environment/modification will be analysed individually, with results discussed in the same order and context as the corresponding methodology sections. This ensures a clear and consistent comparison between the experimental setup and the outcomes observed, providing a direct link between the proposed methodology and its corresponding results

4.2 First Environment Results

Using Tensorboard visualizations for analysis, training was inefficient. With both agents performing poorly gaining around -8 to +6 points for the entirety of the training.

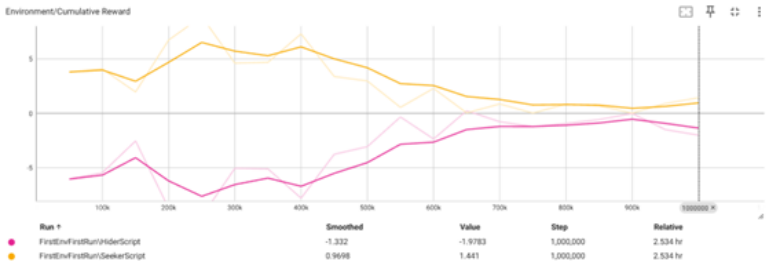


Figure 4.1: *First environment cumulative reward*

The policy loss and value loss consist of a slow decline, although this is a good sign which shows improvement, stabilization and accuracy - the values are trivial.

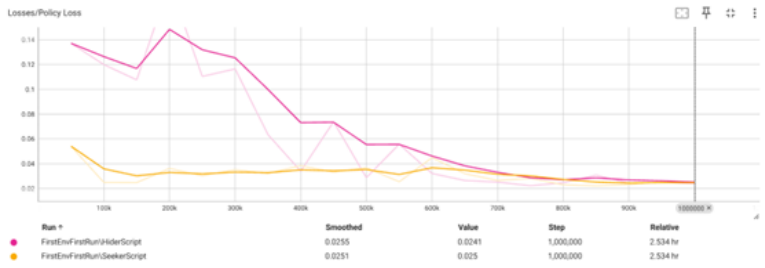


Figure 4.2: *First environment policy loss*

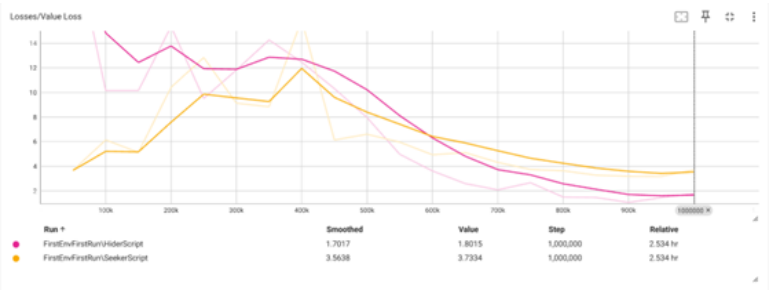


Figure 4.3: *First environment value loss*

There is no clear advantage for either agent, with both performing poorly.

First Environment, First Modification Results

The first modification done had the desired outcomes:

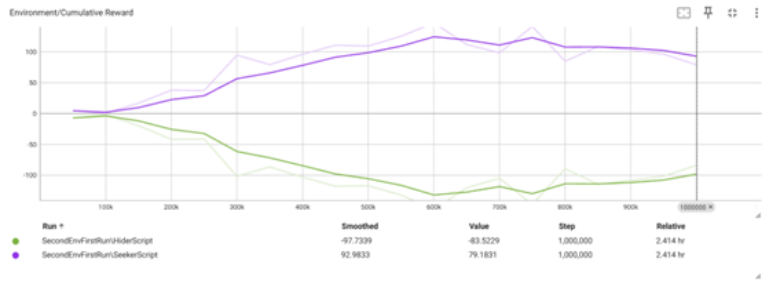


Figure 4.4: *First environment first modification, cumulative reward*

The seeker agent performed well achieving a cumulative reward of 127 which is a large step over the previous standard configurations. The hider agent on the other hand achieved a cumulative reward of -129.2 which is representative of the seeker's reward.

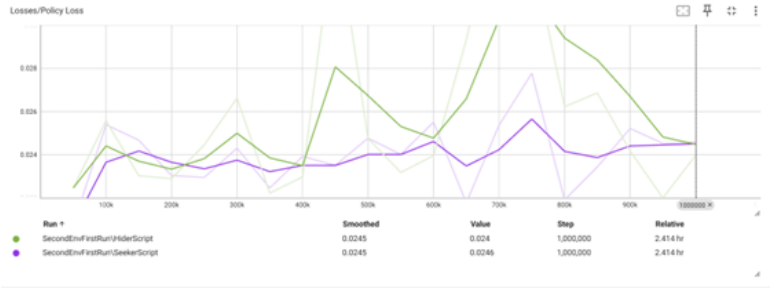


Figure 4.5: *First environment first modification, Policy Loss*

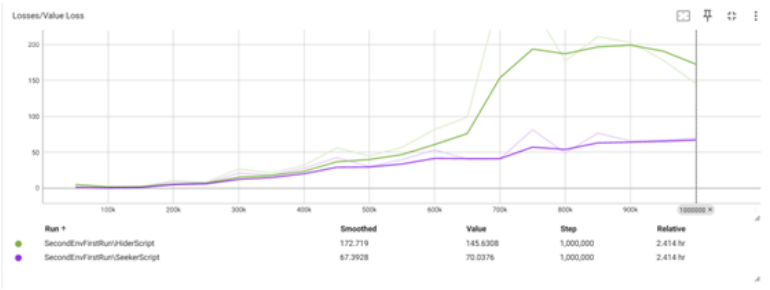


Figure 4.6: *First environment first modification, Value Loss*

As seen by Policy and Value loss, the seeker agent also manages to perform significantly better compared to the hider agent.

First Environment, Second Modification Results

As seen below, although the hider agent once again performed better than the seeker agent. The modified raycast had a positive effect on the hider agent with an increased performance when the chasing commenced. However, the hider agent still performed poorly in terms of being spotted by the seeker agent.

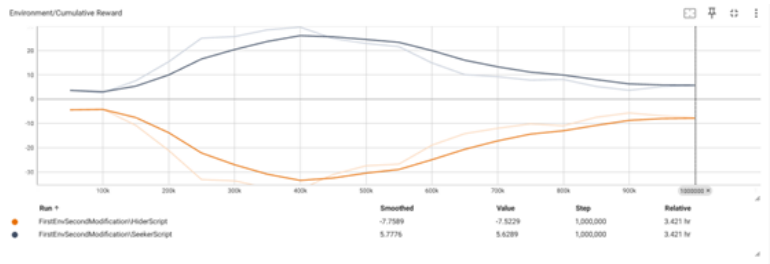


Figure 4.7: *First environment second modification, cumulative reward*

Moreover, the hider agent also outperformed the seeker agent regarding Policy and Value loss even though this contradicts the cumulative rewards.

Figure 4.8: *First environment second modification, Policy Loss*

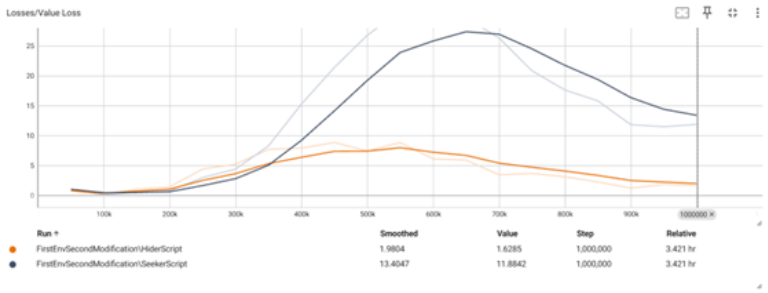


Figure 4.9: *First environment second modification, Value Loss*

First Environment, Third Modification Results

Although the third modification has helped the hider agent to improve. The seeker agent has managed to improve. Locking on instantly to the hider and chasing almost instantly which can be seen below with the reward cumulative rewards, Policy Loss and Value Loss respectively:

Figure 4.10: *First environment third modification, Cumulative Reward*

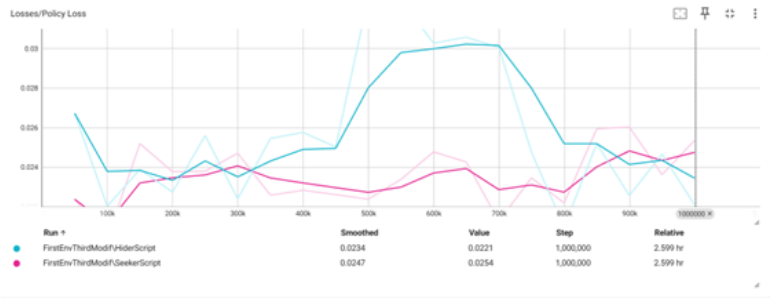


Figure 4.11: *First environment third modification, Policy Loss*

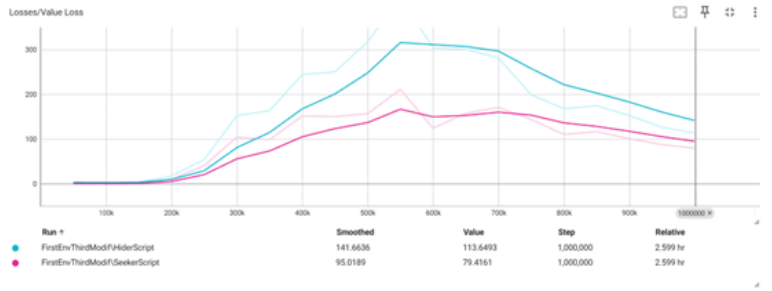


Figure 4.12: *First environment third modification, Value Loss*

However, an important observation is that once the seeker chases the hider, the hider runs away and jumps off the area.

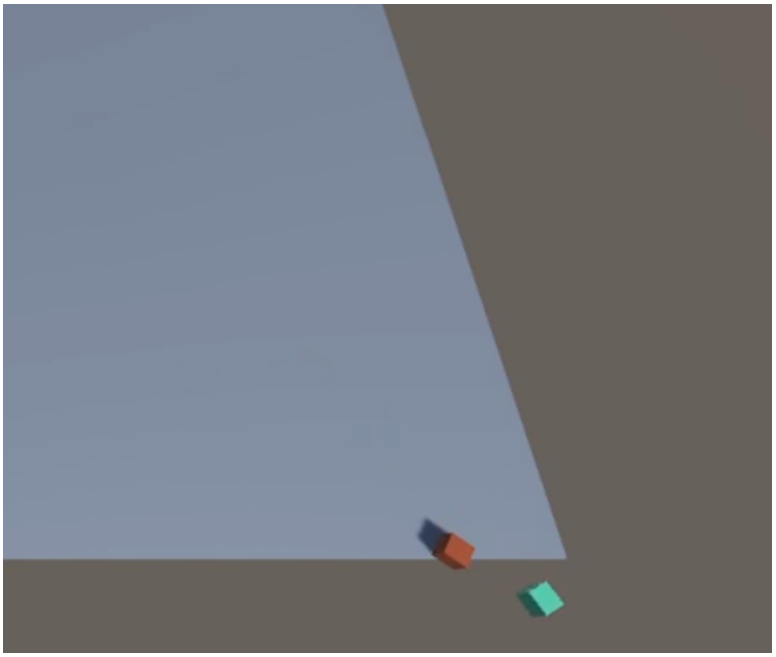


Figure 4.13: *Hider agent running off the platform mid chase*

This was unforeseen, as it was originally believed that the hider would simply run away around the edges of the area, as it had done previously. This indicates that the hider agent was trained well enough not only to utilize the environment to its advantage but also to exhibit behaviour akin to 'game-testing,' exploring and pushing the limits of the given constraints.

4.3 Second Environment Results

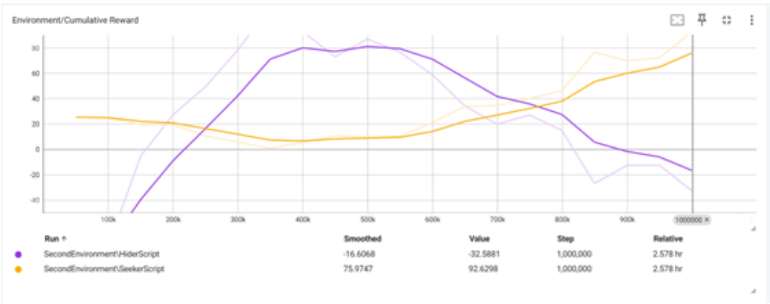


Figure 4.14: *Second environment, Cumulative Reward*

The second environment, which consists of a newly added restricted zone, showed a significant difference in the cumulative rewards compared to the previous environment and modifications. Although the seeker agent still outperformed the hider agent, which was expected due to the agent not having any clear advantage the moment he is seen and chased:

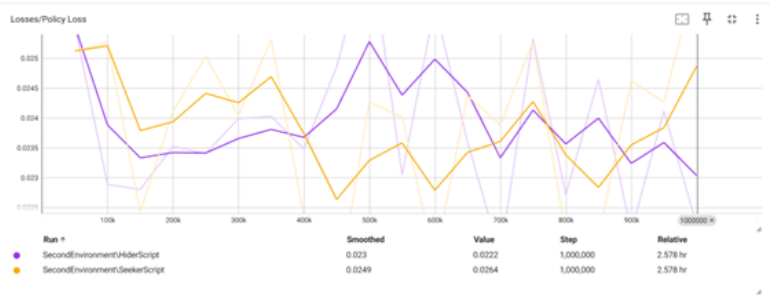


Figure 4.15: *Second environment, Policy Loss*

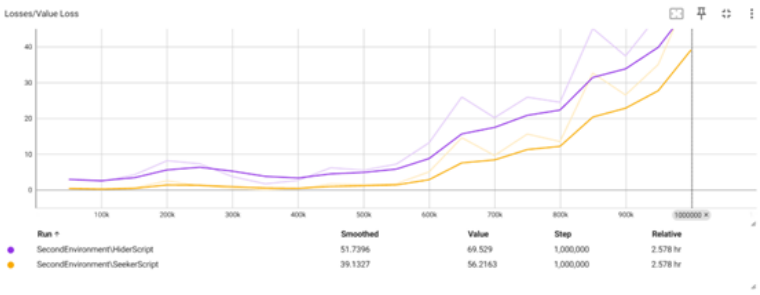


Figure 4.16: *Second environment, Value Loss*

The hider agent was once again using the restricted zone to its advantage. It has been hypothesized that the hider agent will learn to not run into the restricted zone due to a penalty of -1.0 reward. However, the hider agent learned that it was more cost-effective to run into the restricted zone once it is being chased to escape the hider agent by having its position randomized once again. This ends the chase and allows the hider to keep earning its constant reward by not being spotted and/or chased.

4.4 Third Environment Results

In the third environment, the seeker agent had its best performance yet. Achieving a peak reward of +716.8. Alternatively, the hider agent had its poorest performance yet, achieving a negative reward of -757.3. This was expected due to the physical walls blocking the hider agent, allowing the seeker agent to spot and chase the hider agent until the end of the episode. The environment inherently favours the seeker agent up until this point.

Figure 4.17: *Third environment, Cumulative Reward*

It is observed that the policy loss for the seeker agent consists of exploring more aggressively or learning from more rewarding feedback signals, leading to better performance. Whereas the Value loss likely reflects that the seeker agent has also learned an accurate and stable estimate of its expected rewards when compared to the hider agent.

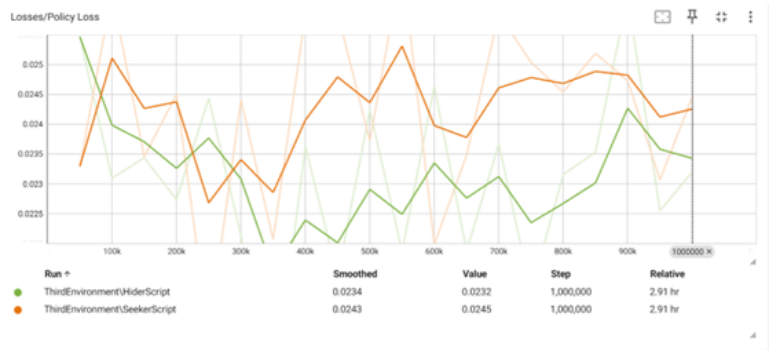


Figure 4.18: *Third environment, Policy Loss*

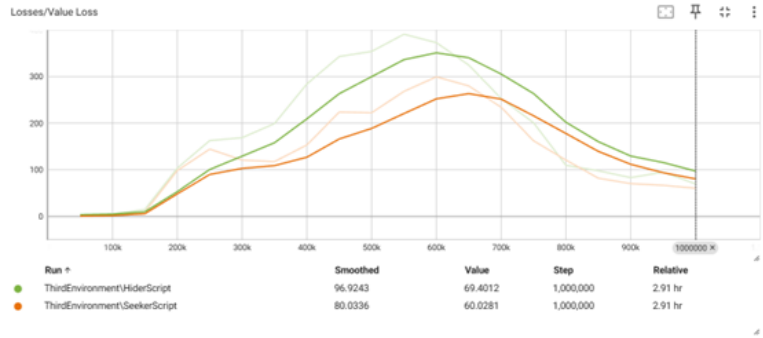


Figure 4.19: *Third environment, Value Loss*

4.5 Fourth Environment Results

For the fourth environment, the seeker agent once again outperformed the hider agent:

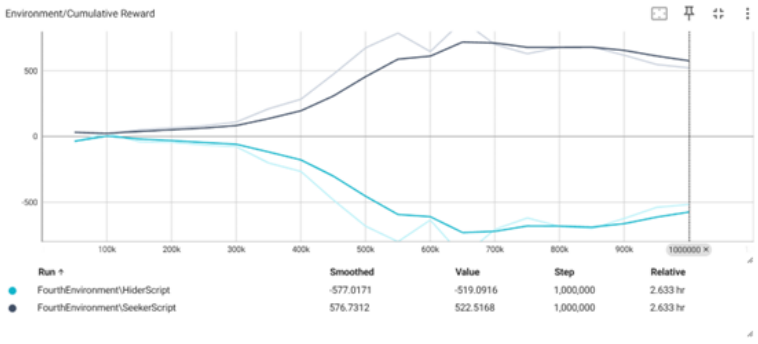


Figure 4.20: *Fourth environment, Cumulative Reward*

The hider agent had not been using the internal walls as cover effectively. However, this could also be due to a crucial bug spotted during training:

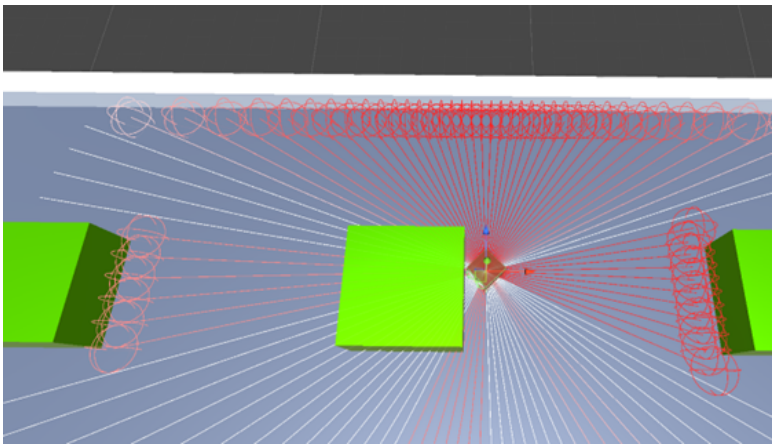


Figure 4.21: *Raycast passing through internal walls*

The agent's raycast was able to pass through internal walls after making contact with them. This behaviour occurred because the raycast originated from the center of the agent, which upon contact with the wall, allowed it to extend through to the other side.

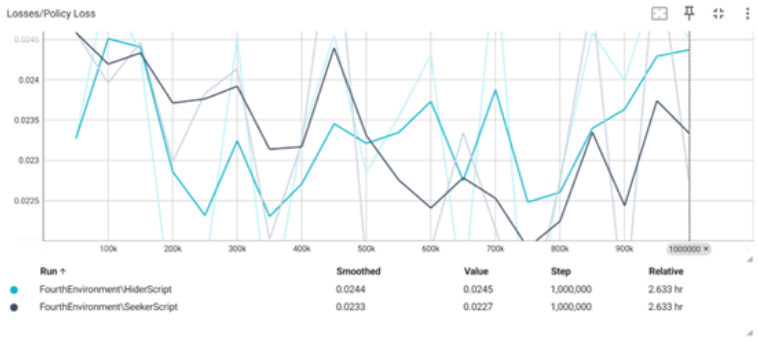


Figure 4.22: *Fourth environment, Policy Loss*

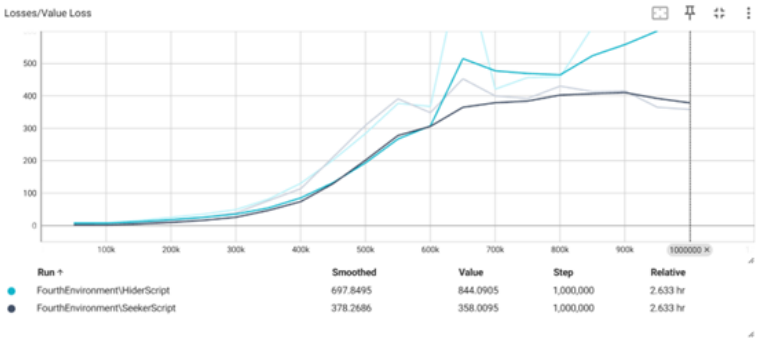


Figure 4.23: *Fourth environment, Value Loss*

As expected, the seeker agent also performed better than the hider agent, given the new-found bug and lack of understanding by the hider agent.

4.5.1 Fourth Environment , First Modification Results

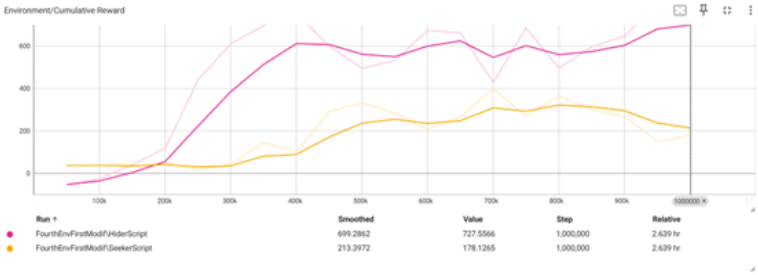


Figure 4.24: *Fourth environment first modification, Cumulative Reward*

The hider agent achieved its highest recorded cumulative reward of +699.3, surpassing the seeker agent by a large difference. This demonstrates significant

progress in the hider agent's ability to optimize its strategy while competing against a constantly well-trained opponent. Notably, the seeker agent also maintained a strong performance, achieving a positive cumulative reward of +321.8, comparable to results in previous runs. These results highlight the effectiveness of training both agents in a balanced and competitive environment, where each agent learns and adapts to the strategies of the other which outputs a smoother looking game of hide and seek.

Figure 4.25: *Fourth environment first modification, Policy Loss*

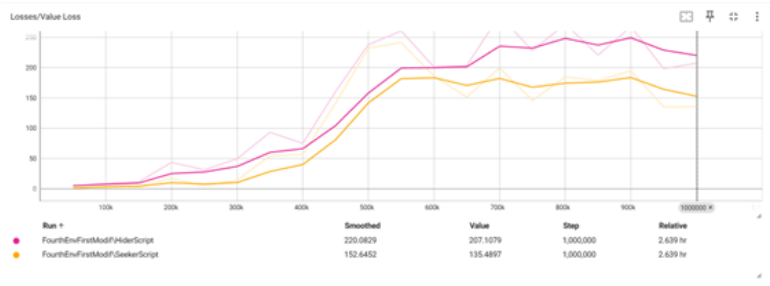


Figure 4.26: *Fourth environment first modification, Value Loss*

This dynamic learning process is further evidenced by the observed fluctuations in policy loss, which reflect the continuous challenge of adapting to an evolving competitive environment. The value loss, on the other hand, exhibits an increasing trend, likely due to the value network's difficulty in accurately predicting long-term rewards as the agents' policies evolve. This phenomenon is particularly pronounced as agents develop new and more sophisticated hiding and

seeking strategies. However, the steady improvement in cumulative rewards indicates that both agents are learning and employing more effective strategies over time, achieving the goal of the training process. In this context, the rising value loss is not necessarily problematic but rather shows the inherent complexity of reward prediction in a competitive and adaptive environment.

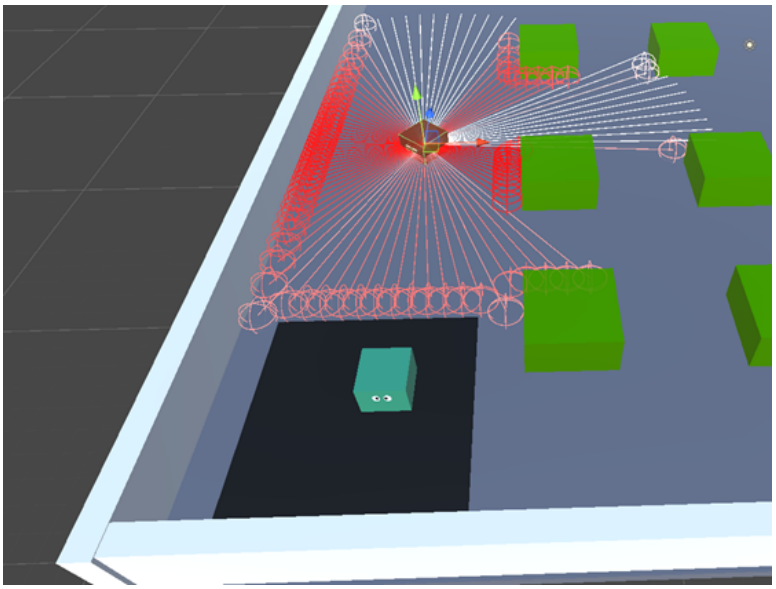


Figure 4.27: *Raycast blocked by incentive*

Despite the incentive platforms being designed to be minimal in size, the seeker agent experienced a disadvantage due to an in-game issue. Specifically, a bug in the system caused the platform to inadvertently block the seeker agent's raycast, preventing it from detecting the hider agent positioned on the incentive platform. This issue, while not immediately visible to the user, significantly impacted the seeker's ability to perform effectively in such scenarios.

4.5.2 Fourth Environment , Second Modification Results

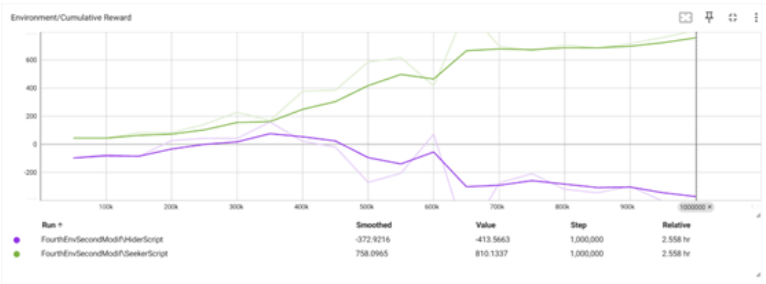


Figure 4.28: *Fourth environment Second modification, Cumulative Reward*

The adjustments made to the environment, particularly the resolution of the ray-cast obstruction caused by the incentive platform bug, have had a significant impact on the training outcomes. Following this change, the seeker agent has regained an advantage, consistently outperforming the hider agent. This highlights the importance of addressing such environmental issues to ensure balanced competition and accurate evaluation of agent performance for NPC optimization.

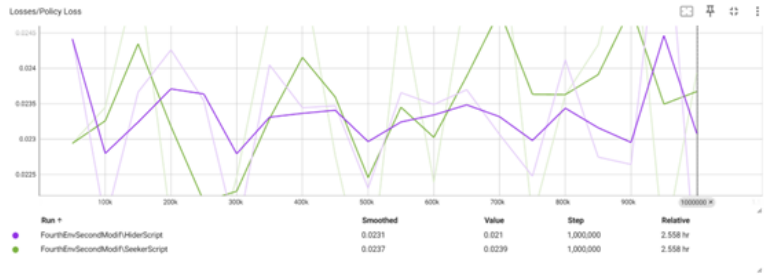


Figure 4.29: *Fourth environment Second modification, Policy Loss*

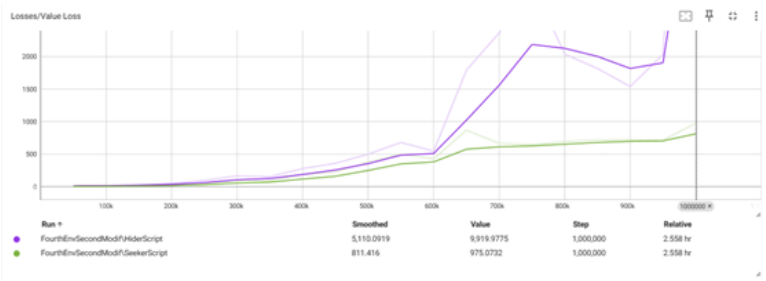


Figure 4.30: *Fourth environment Second modification, Value Loss*

The trends observed in the policy and value loss metrics remain consistent with those seen in the previous modification. This is likely due to the increasing complexity of the environment and the inclusion of additional objects, which continue to challenge the agents' learning processes.

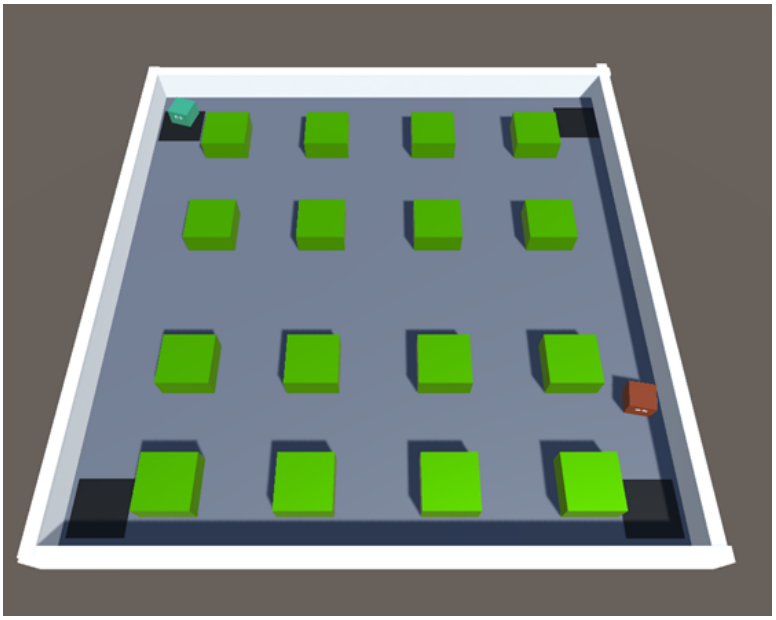


Figure 4.31: *Hider agent on incentive, seeker agent searching for hider agent*

4.5.3 Fourth Environment , Third Modification Results

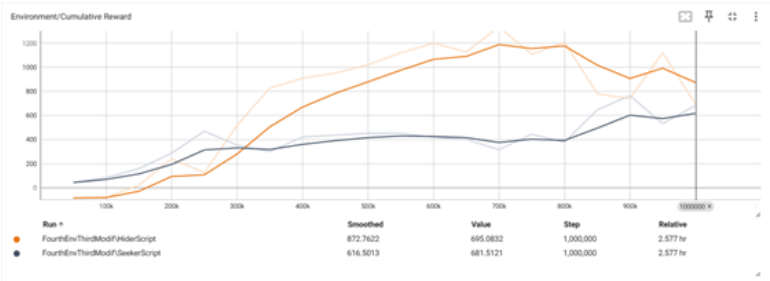


Figure 4.32: *Fourth environment Third modification, Cumulative Reward*

This environment, with its third modification created the best results. The seeker agent was outperforming the hider agent up to around the step 300k. At which point the hider agent started outperforming the seeker agent up to around step

800k. Finally, the cumulative reward for both agents started to converge symmetrically up to the final step of 1M.

Figure 4.33: *Fourth environment Third modification, Policy Loss*

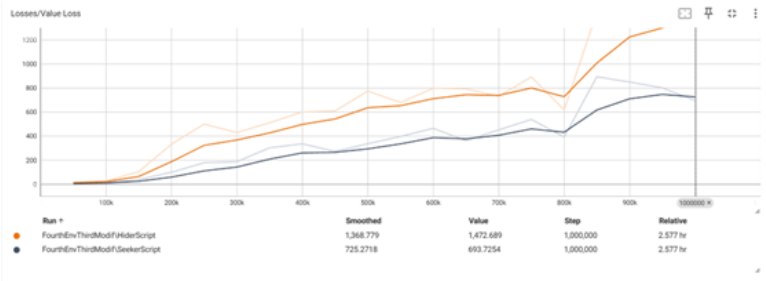


Figure 4.34: *Fourth environment Third modification, Value Loss*

This can be also seen corresponding mainly with the Policy Loss, where the highest level of spikes had been taking place around the 300k step. Value Loss had been steady throughout similarly to other environments/modifications.

Since the final environment has yielded satisfactory results, it had been deemed optimal to proceed to the next phase of the experiment, which involves the implementation of Curriculum Learning (CL). This stage will build upon the pre-trained hider and seeker models, enabling them to further develop their ability to interact with and utilize objects within the environment to provoke new emergent behaviours.

4.6 Curriculum Learning, First Environment Results

Around the 500k step, the seeker agent started to use the speed boosts more effectively. The main advantage is seen when the seeker agent runs quickly around the map searching for the incentives at the corners. This allowed the seeker agent to quickly find the hider agent and chase. Towards the last 100k steps, the seeker agent started to outperform the hider agent.

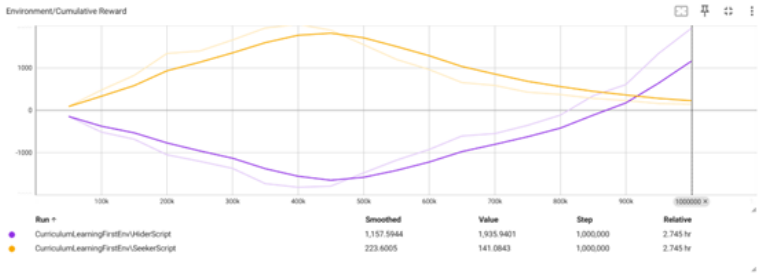


Figure 4.35: *Curriculum Learning, First Environment Cumulative Reward*

No exploitable mechanics or emergent behaviour were identified during training.

4.6.1 Curriculum Learning, Second Environment Results

Around the 200,000-step mark, the hider agent demonstrated proficiency in utilizing the newly introduced levers effectively. As a result, it significantly outperformed the seeker agent, achieving a maximum positive reward of 3,612, whereas the seeker agent attained a maximum positive reward of only 465.4. The hider agent also rarely jumped off the map as expected.

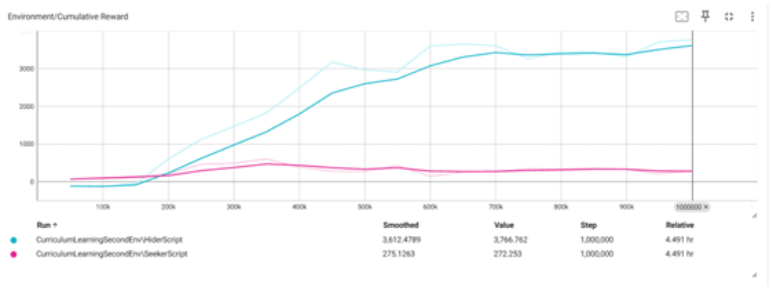


Figure 4.36: *Curriculum Learning, Second Environment Cumulative Reward*

Although the hider agent outperformed the seeker agent, the seeker agent managed to discover an exploitable mechanic within the game engine. Specifically, in certain instances, it was able to pass through randomly spawned walls after the hider activated the lever. This unintended behaviour allowed the seeker agent to bypass a fundamental constraint of the environment, completely undermining the advantage for the hider agent.

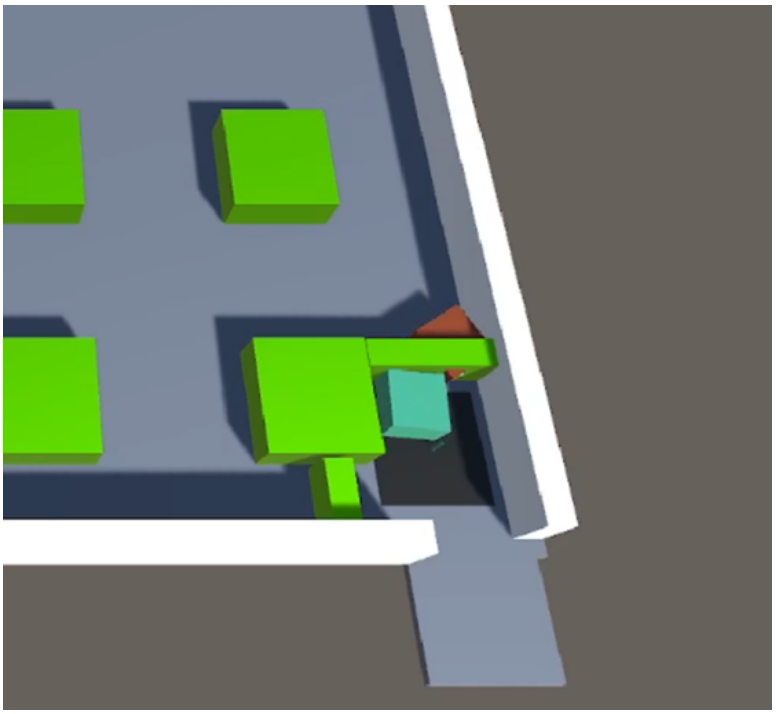


Figure 4.37: *Hider agent exploiting the physics engine*

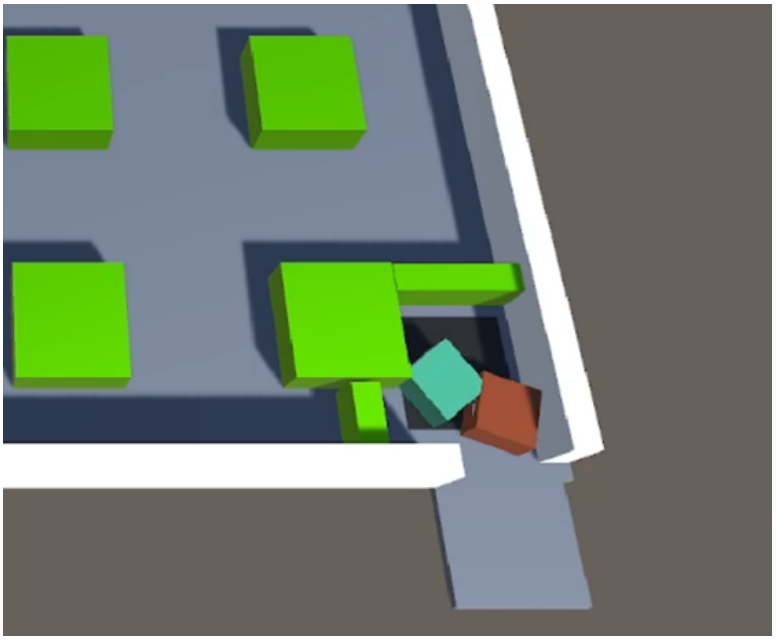


Figure 4.38: *Hider agent passing the spawned walls and trapping the hider agent*

This could occur for a multitude of reasons, in my case it was due to floating-point precision errors which can cause small misalignments in object positioning, leading to unintended gaps between walls that reinforcement learning agents can exploit. These errors arise because game engines represent object positions using floating-point numbers, which have limited precision and can introduce tiny discrepancies in calculations.

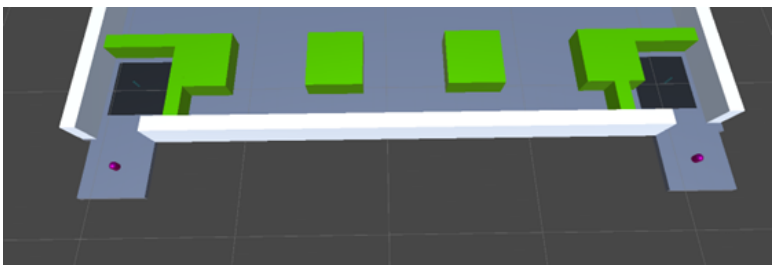


Figure 4.39: *Spawned walls updated to have perfect floating-point precision*

Issues are tested by manually controlling the agents heuristically, which are fixed and updated as per figure 4.39

4.6.2 Curriculum Learning, Third Environment Results

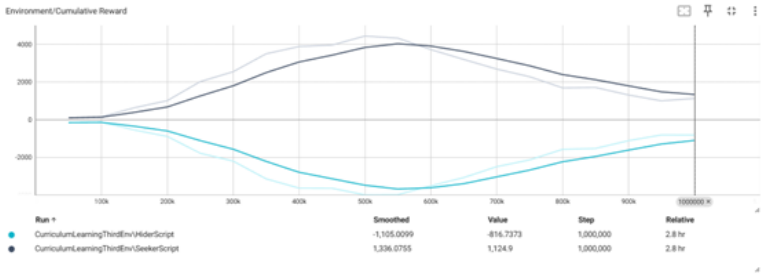


Figure 4.40: *Curriculum Learning, Third Environment Cumulative Reward*

The seeker agent has outperformed the hider agent as expected, utilizing the passable walls effectively which is depicted by graph above (Figure 4.40). Although the hider agent had started to counter the seeker agent by utilizing the passable wall to run away once the seeker agent makes contact, it was not effective enough to outperform the seeker agent. Moreover, no exploitable mechanics were observed.

4.7 Exploitable Mechanics/Emergent Behaviours – Findings

This section will focus on the findings of exploitable mechanics and emergent behaviours during the methodology for both the initial training phase as well as the curriculum learning phase.

4.7.1 First Exploitable Mechanic

The first exploitable mechanic was in the fourth environment.

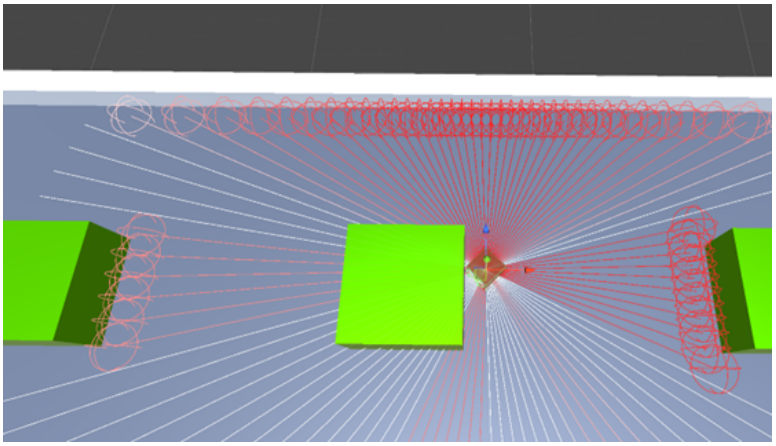


Figure 4.41: *Raycast passing through internal walls (Relates to 4.21)*

Both agent raycast were able to penetrate internal walls upon contact. This behaviour resulted from the raycast being emitted from the centre of the agent, which, upon collision with the wall, allowed it to pass through to the opposite side. The seeker agent used this to its advantage to spot the hider agent through walls which also resulted in an unbalanced game favouring the hider agent.

4.7.2 Second Exploitable Mechanic

The second exploitable mechanic was in the first environment, first modification.

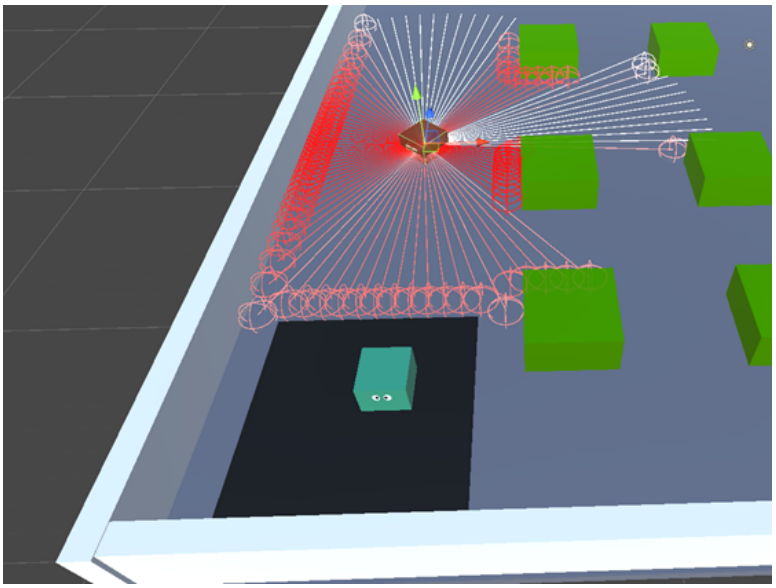


Figure 4.42: *Raycast blocked by incentive (Relates to 4.27)*

Although the incentive platforms were intentionally designed to be minimal in size, the seeker agent faced a significant disadvantage due to an in-game issue. A system bug caused the platform to obstruct the seeker agent's raycast, preventing it from detecting the hider agent positioned on the incentive platform. While this issue was not immediately apparent to the user, it substantially impaired the seeker's performance in such scenarios. This issue presented a significant advantage for the hider agent, as it could remain undetected once within an incentive zone. Although this represents a bug within the machine learning implementation rather than the actual game mechanics, it underscores the potential impact of game-related bugs on overall gameplay. In this instance, the bug specifically affected the line of sight between the seeker and the hider.

4.7.3 Third Exploitable Mechanic

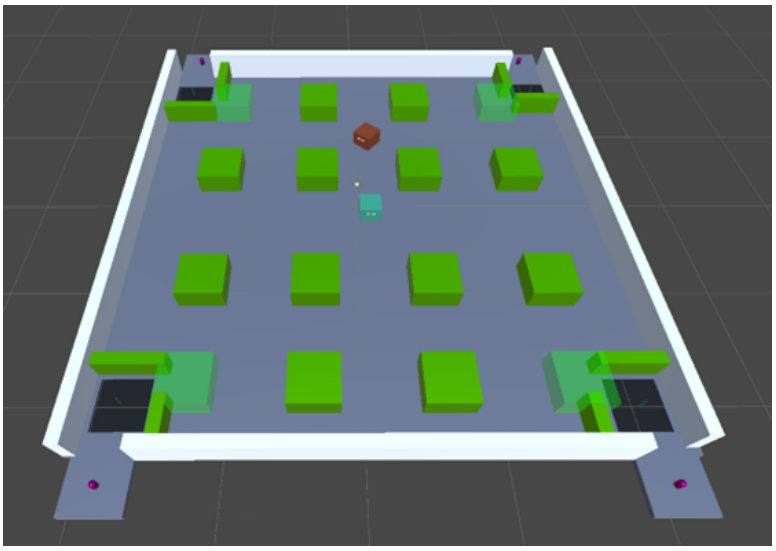


Figure 4.43: *Curriculum Learning, Third Environment (Refer to 3.20)*

This is labeled as an exploitable mechanic instead of an emergent behaviour since it was anticipated due to the intentional design of passable walls, It reinforces the tendency of agents to exploit environmental weaknesses to their advantage. This highlights the potential of DRLAs in identifying unintended system flaws that may otherwise go unnoticed. Even a minor oversight, such as an unchecked configuration setting affecting an object's behaviour within a complex environment, can go undetected by manual game testers. This point of the study answers research question: 'Do Deep Reinforcement Learning Agents (DRLA) detect exploitable mechanics within the game engine to gain an advantage?' in which the DRLAs managed to find three instance of exploitable mechanics throughout their training, which they used to their advantage.

4.7.4 First Emergent Behaviour

The first emergent behaviour was in the first environment, third modification.

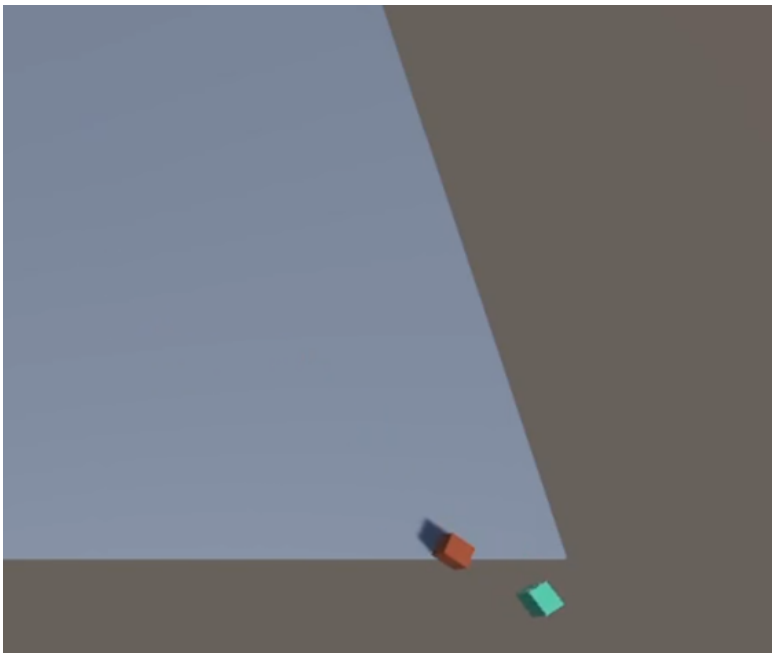


Figure 4.44: *Hider agent running off the platform mid chase (Relates to 4.13)*

The hider agent exploited the environment by running off the platform upon contact with the seeker agent, resulting in an unbalanced game heavily favouring the hider.

4.7.5 Second Emergent Behaviour

The second emergent behaviour was in the second curriculum learning environment

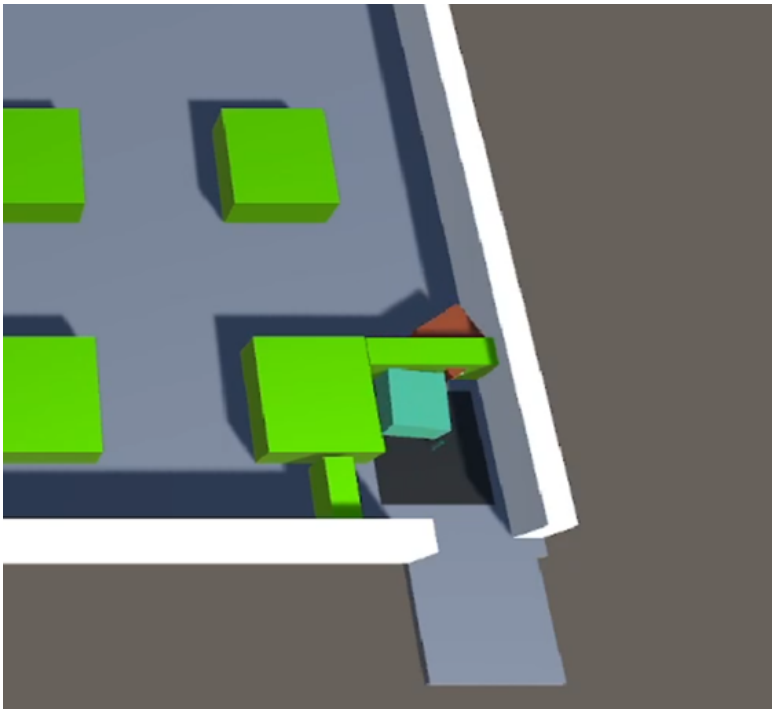


Figure 4.45: *Hider agent exploiting the physics engine (Relates to 4.37)*

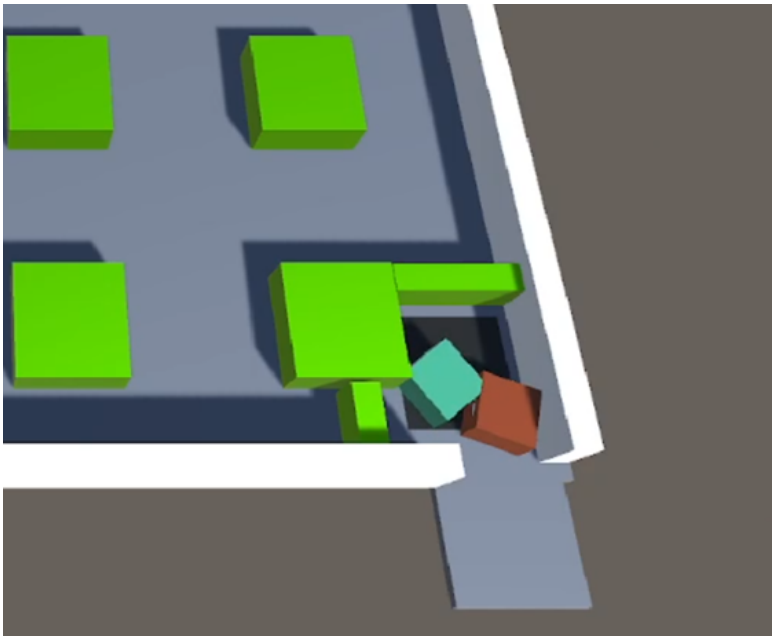


Figure 4.46: *Hider agent passing the spawned walls and trapping the hider agent (Relates to 4.38)*

Both agents could pass through the randomly spawned walls after the hider activated the lever. This allowed the seeker agent to bypass a constraint within the game engine, completely undermining the advantage for the hider agent. In my case, this happened due to the floating-point precision errors which caused a small misalignment between my object's position. This led to unintended gaps between walls that the DRLAs managed to exploit. This point of the study answers research question: 'Do Deep Reinforcement Learning Agents (DRLA) exhibit emergent behaviours, particularly during curriculum learning with the introduction of external objects?' in which DRLAs showed emergent behaviours throughout their training, in both the initial training phase as well as the curriculum learning phase.

4.8 Questionnaire Results

Likert scales have dominated research fields with their fixed-point ordinal system, consisting of a series of statements or items. Respondents are asked to indicate their level of agreement or disagreement with each statement. A traditional 5-point Likert scale has five response options, ranging from" Strongly Disagree" to" Strongly Agree", with a neutral midpoint such as" Neutral". The participant selects the option that best represents their point of view on the given statement, and then the responses are numerically coded for analysis [28]. The conventional use of Likert scales generally involves a relatively small number of response options (2-7 points) [28], the response options for this research will consist of 5 points:

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

This scale allows participants to express varying degrees of agreement or disagreement, striking a balance between simplicity and depth. Moreover, the inclusion of the neutral midpoint accommodates participants that feel indecisive [29]. The Likert scale itself will be used for measures related to the hide-and-seek and not to general demographic questions.

The questionnaire focuses on people's perceptions/interpretations on the utilization of artificial intelligence to provoke emergent behaviours, to help in the development of NPCs, automated game testing and other similar areas.

The questionnaire received a total of 89 responses (refer to Appendix E – Questionnaire Section). Regarding demographics, the largest age group was 18-25, making up 33.7% of respondents. The 25-35 and 35-45 age groups accounted for 23.6% and 24.7%, respectively. Meanwhile, the 45-60 and 60+ age groups had the lowest representation, at 14.6% and 3.4%, respectively. In terms of gender, 59.6% of respondents were male, while 40.4% were female.

Education levels varied, with the majority holding an undergraduate certificate (37.1%) or a bachelor's degree (30.3%), totalling 67.4%. Additionally, 13.5% had completed post-secondary education, and 14.6% held a master's degree, collectively accounting for 28.1%. A small percentage had a secondary education certificate (1.1%) or a doctorate (2.2%), while 1.1% preferred not to disclose their education level.

Regarding employment status, 44.9% of respondents were employed full-time, while 28.1% were students. The remaining participants were employed part-time (10.1%), self-employed or contractors (9%), and unemployed (7.9%).

The majority of respondents were not familiar with artificial intelligence (50.6%), while 39.3% indicated familiarity, and 10.1% remained neutral. However, most respondents (56.2%) were familiar with games and gaming terminology, whereas 39.3% were not, and 4.5% remained neutral.

A significant portion of respondents (89.9%) were familiar with the game of hide-and-seek, while only 2.2% were not, and 7.9% remained neutral. However, when asked about NPCs and manual game testing, the majority (64.1%) reported unfamiliarity, while 29.2% were familiar, and 6.7% remained neutral.

Most respondents (71.9%) agreed that AI can be used to optimize NPCs and detect bugs or exploits, while 3.4% disagreed, and 24.7% remained neutral. Similarly, 60.6% believed AI could detect bugs/exploits more effectively than a human game tester, while 6.7% disagreed, and 32.6% remained neutral.

The respondents were provided with a section (see Appendix D) and asked to share their interpretations. The majority of respondents (89.9%) agreed that the observed behaviour was strategic and intentional rather than random, while 10.1% remained neutral. No respondents believed the behaviour was random. Additionally, 88.7% agreed that the seeker was exploiting unintended game mechanics, while 11.2% remained neutral, with no disagreement.

When asked whether the displayed behaviour indicated intelligence, 85.4% agreed, 12.4% remained neutral, and 2.2% disagreed. However, opinions were more divided on whether the agents would behave differently in a more complex environment: 50.6% remained neutral, 45% agreed, and 4.5% disagreed.

Analysis from the acquired results:

1. **Limited AI Familiarity but Strong Gaming Knowledge**
    - While 50.6% of respondents were unfamiliar with AI, a majority (56.2%) were familiar with gaming terminology.
    - This indicates that while many respondents may not have a deep technical understanding of AI, they can still assess its application in gaming.
    - The gap in AI knowledge may influence perceptions.
2. **Strong Agreement on AI's Role in NPC Optimization & Bug Detection**
    - 71.9% of respondents agreed that AI can be used to optimize NPCs and detect bugs/exploits, with only 3.4% disagreeing.
    - 60.6% believed AI is more effective than human testers at identifying bugs, though a significant 32.6% remained neutral, suggesting some scepticism about AI replacing human testers entirely.
    - This supports the idea that AI-driven automation is gaining acceptance, but human oversight remains important.
3. **Emergent Behaviour is Perceived as Strategic & Intelligent**

- 89.9% of respondents viewed the AI's behaviour as intentional rather than random, reinforcing the idea that AI can develop strategic decision-making rather than just reacting unpredictably.
- 88.7% recognized the AI exploiting unintended mechanics, indicating that AI-driven agents may uncover game-breaking exploits that human testers might overlook.
- 85.4% agreed that the AI displayed intelligence, suggesting that AI-generated behaviours are perceived as meaningful rather than arbitrary.

4. **Uncertainty About AI's Adaptability to More Complex Environments**

- While 45% of respondents agreed that the AI would behave differently in a more complex environment, a significant 50.6% remained neutral.
- This suggests uncertainty about AI's ability to generalize beyond its trained environment, emphasizing the need for further research into AI adaptability in dynamic game scenarios.

Further Observations:

1. **AI-Driven NPCs & Game Testing**

- The findings support AI's viability in optimizing NPC behaviour and detecting exploits, but neutral responses indicate some scepticism.
- This suggests that a hybrid approach, where AI assists rather than replaces human testers, may be the most effective strategy.

2. **AI's Role in Exploit Detection**

- The high agreement that AI exploits unintended mechanics suggests it could be a valuable tool for game balancing and quality assurance.

3. **Emergent Behaviour as an Indicator of Intelligence**

   - The perception that AI behaviour is strategic and intentional reinforces that AI can mimic intelligent decision-making.

4. **Challenges in AI Generalization**

   - The mixed responses about AI's behaviour in more complex environments indicate a need for improved AI training methodologies.
   - Future research could examine how environment complexity impacts emergent behaviours, especially in reinforcement learning models like PPO.

4.9 Research Comparisons

In this section, the initial training phase will be compared via cumulative rewards as well as emergent behaviours. Exploitable mechanics will also be discussed since the scope of this is to show the level of competency that the agents have reached, which once finding exploitable mechanics, then translate into the emergent behaviours.

4.9.1 Research Comparisons – Initial Training Phase

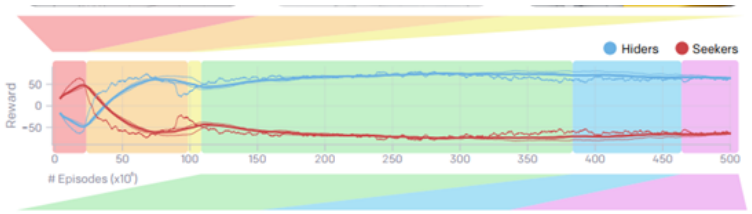


Figure 4.47: *Hider and Seeker Reward Values [2] (Relates to 2.12)*

Instance No	Hider Reward	Seeker Reward
1	0.989	-0.065
2	630.999	-15.677
3	586.288	-0.119
4	191.428	34.775
5	45.385	69.099
6	-103.586	131.803
7	-206.365	376.431
8	-91.584	499.468
9	-82.292	514.991
10	**-1082.291**	**+1515.916**

Figure 4.48: *Hider and Seeker Reward Values [5] (Relates to 2.22)*

The base paper [2] and second paper [5] were already compared in section 2.2.2.4, which shows contradicting results. This was possibly due to the difference in environments, agent policies and implementation details. Moreover, the variations in how the hide-and-seek problem was configured such as their reward

policies, agent behaviours or general setups, influence the outcomes of both studies. This is depicted in figures 4.47 & 4.48.

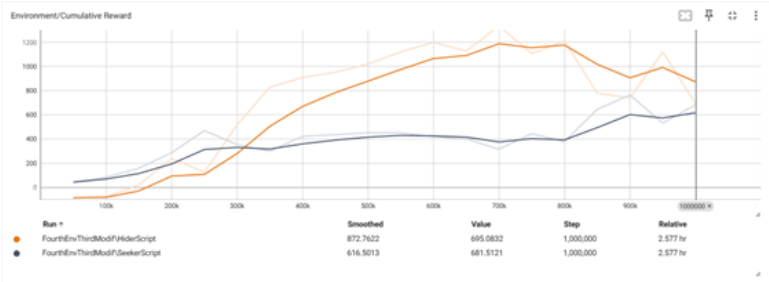


Figure 4.49: *Fourth environment Third modification, Cumulative Reward (Relates to 4.32)*

Comparing figures 4.47 & 4.48 to results of my final initial training phase as per figure 4.49, it is shown that they closely resemble the results of the base paper (refer to figure 4.47) with the hider outperforming the seeker by a significant margin between steps +300k to +800k. However, possibly due to my effort at creating a balanced environment for both agents, the final models consist of a converging graph that represents a balanced playground for both agents. With both agents containing a steady reward between +600 to +1000. This point of the study answered RQ1: 'Do trained Deep Reinforcement Learning Agents (DRLA) exhibit fluid and balanced gameplay before curriculum learning is introduced?" The agents manage to learn to play the game effectively in a balanced manner with fluid gameplay throughout the episodes.

4.9.2 Exploitable Mechanics

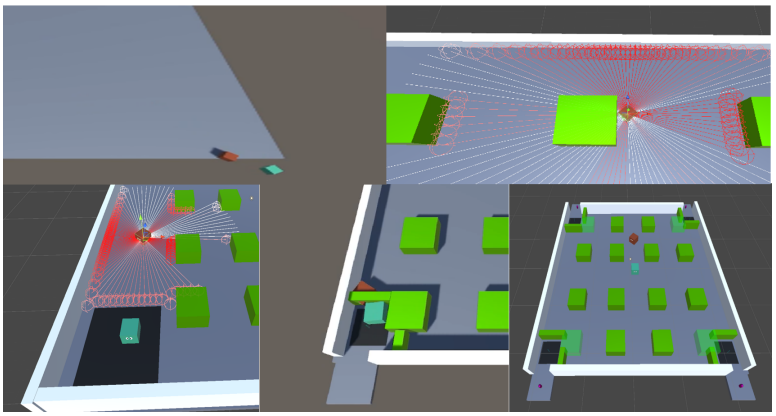


Figure 4.50: *The five exploitable mechanics from top left to bottom right*

The five exploitable mechanics observed are seen in figure 4.50, which are as follows from top left to bottom right:

- No enclosed walls, making it possible for the agents to run off the given area.
- Raycast is able to pass through the wall due to the placement of its axis.
- Raycast is unable to see over the incentive platform, causing the seeker agent to fail to spot the hider agent when on the platform.
- Agents are able to pass through walls, making it possible for the seeker agent to nullify the hider's positional advantage.
- Similarly, the seeker agent can pass through walls connected to the spawnable walls, although this is deliberate.

This point of the study answered RQ2: 'Do Deep Reinforcement Learning Agents (DRLA) detect exploitable mechanics within the game engine to gain an

advantage?' The agents managed to detect the exploitable mechanics in the environments effectively and gain an advantage over each other. The way this is done is discussed in the next section (4.11)

4.9.3 Research Comparisons – Emergent Behaviours

For the comparisons of emergent behaviours, only the base paper [2] will be compared as it is the only study dedicated to the focus of this research

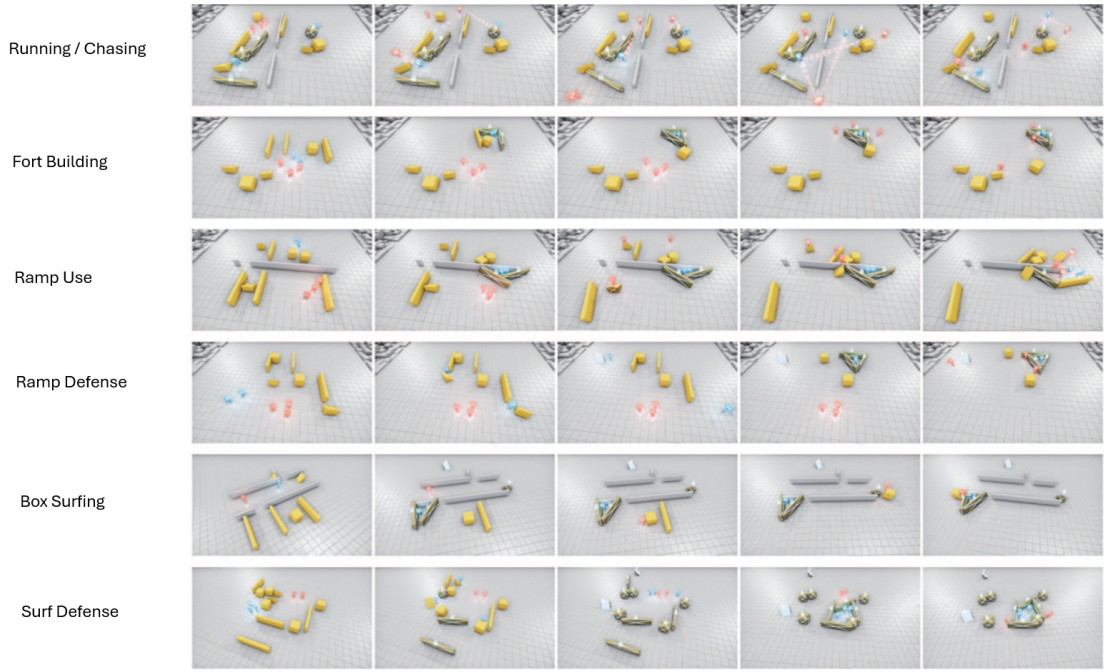


Figure 4.51: *Each image, from left to right, illustrates the progression of each emergent strategy (Relates to 2.8) [2]*

Figure 4.52: *The two emergent behaviours*

Figure 4.51 shows the emergent behaviour progressions for the base paper, this includes:

- Box Surfing
- Surf Defense

Whereas the results for this paper shown in figure 4.52 shows emergent behaviours in respective order which include:

- Hider agent running off the map when not in an enclosed space
- Agents able to pass through walls

Similarly, this answers RQ3:' Do Deep Reinforcement Learning Agents (DRLA) exhibit emergent behaviours, particularly during curriculum learning with the introduction of external objects?' Both studies observe emergent behaviours which arose from complex interactions within a game environment. The behaviours by Owen Baker et al. (2021) were more fluid and adaptive, in contrast the emergent behaviours observed in this research suggest more systematic and unintended interactions with the game's physics and environment, highlighting the potential technical flaws and game engine exploits.

Chapter 5: Conclusions and Recommendations

The results provide strong evidence that DRLAs can be leveraged as an effective tool, revealing emergent behaviours that highlight both expected and unintended interactions within the environment. Through various training phases, the agents exhibited complex adaptation strategies, such as environment exploitation, learning to counterbalance adversarial strategies, and uncovering unintended game engine vulnerabilities.

Overall, these findings highlight the potential use of DRLAs for provoking emergent behaviours, offering insights into unintended mechanics that could affect game balance and player experience within game domains. Moreover, this study successfully answered the hypothesis positively: 'Deep Reinforcement Learning Agents (DRLA) will exhibit emergent behaviours throughout their training process, mainly during curriculum learning with the introduction of external objects' in which all three research questions were also answered successfully. Throughout the whole experiment, the agents managed to play in a fluid and balanced way while finding exploitable mechanics, which provoked the development of emergent behaviours.

Future work could enhance this methodology by incorporating more diverse and dynamic environments to assess agent adaptability, improving model interpretability through explainability techniques, and refining reward structures to mitigate unintended exploits. Additionally, exploring transfer learning for better general-

ization and studying human-agent interactions to evaluate player experience will further ensure responsible and effective deployment of DRLAs in game environments. Beyond gaming domains, emergent behaviours play a crucial role in fields such as robotics, automation, cybersecurity, healthcare, and urban planning. Therefore, the insights gained from this study could be applied across various domains to address complex, real-world challenges as mentioned in the research boundaries of this study.

List of References

[1] P. L. P. de Woillemont, R. Labory, and V. Corruble. Automated play-testing through rl based human-like play-styles generation. *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, 18(1), 2022.

[2] B. Baker et al. Emergent tool use from multi-agent auto-curricula. In *2022 IEEE 3rd Global Conf. Adv. Technol. (GCAT)*, Bangalore, India, Oct. 7–9 2022.

[3] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent complexity via multi-agent competition. In *Proc. Int. Conf. Learn. Representations (ICLR)*, 2018.

[4] J. Guo. Deep learning in long-short stock portfolio allocation: An empirical study. *arXiv preprint arXiv:2411.13555*, Oct. 2024.

[5] S. Jagtap, S. Gadiwan, A. Thomas, and S. Mishra. Multi-agent reinforcement learning - implementation of hide and seek. In *2022 IEEE 3rd Global Conf. Adv. Technol. (GCAT)*, pages 1–6, Bangalore, India, Oct. 2022.

[6] M. Apte, K. Kale, P. Datar, and P. R. Deshmukh. Dynamic retail pricing via q-learning: A reinforcement learning framework for enhanced revenue management. In *Proc. 1st IEEE Int. Conf. AIML-App. Eng. Technol. (ICAET-25)*, 2025.

[7] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén. Augmenting automated game testing with deep reinforcement learning. In *Proc. 2020 IEEE Conf. Games (CoG)*, pages 600–603, 2020.

[8] IBM. What is machine learning? IBM Cloud Education, Jul. 2020.

[9] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[10] H. Jang and Y. M. Kim. Remp: Reusable motion prior for multi-domain 3d human pose estimation and motion inbetweening. In *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, 2025.

[11] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, Dec. 2020.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, Aug. 2017.

[13] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[14] M. Jaderberg et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[15] OpenAI. Openai five defeats dota 2 world champions. `https://openai.com/blog/openai-five-defeats-dota-2-world-champions`, Apr. 15 2019.

[16] M. Jaderberg et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[17] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 2137–2145, 2016.

[18] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *Proc. Int. Conf. Learn. Representations (ICLR)*, 2019.

[19] N. Haber, D. Mrowca, S. Wang, L. Fei-Fei, and D. L. Yamins. Learning to play with intrinsically motivated, self-aware agents. In *Adv. Neural Inf. Process. Syst.*, volume 31, pages 8388–8399, 2018.

[20] J. Perolat, J. Z. Leibo, V. Zambaldi, C. Beattie, K. Tuyls, and T. Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *Adv. Neural Inf. Process. Syst.*, pages 3643–3652, 2017.

[21] I. E. Skoulakis and M. G. Lagoudakis. Efficient reinforcement learning in adversarial games. In *Proc. IEEE 24th Int. Conf. Tools Artif. Intell.*, 2012.

[22] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao. A survey of deep reinforcement learning in video games. Dec. 2019.

[23] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *Proc. Int. Conf. Learn. Representations (ICLR)*, 2017.

[24] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, May 2020.

[25] Z. Li, J. Wu, X. Ling, T. Luo, Z. Rui, and Y. Wu. The seeds of the future sprout from history: Fuzzing for unveiling vulnerabilities in prospective deep-learning libraries. In *Proc. 47th Int. Conf. Softw. Eng. (ICSE)*, 2025.

[26] M. Rashki. Probability-informed machine learning. *arXiv preprint arXiv:2412.11526*, Dec. 2024.

[27] B. Goertzel. Actpc-chem: Discrete active predictive coding for goal-guided algorithmic chemistry as a potential cognitive kernel for hyperon & primus-based agi. *arXiv preprint arXiv:2412.16547*, Dec. 2024.

[28] S. Sun, K. M. Schmidt, and T. R. Henry. Don't let your likert scales grow up to be visual analog scales: Understanding the relationship between number of response categories and measurement error. *arXiv preprint arXiv:2502.02846*, 2025.

[29] Qualtrics. What is a likert scale? definition, examples, and use cases. `https://www.qualtrics.com/blog/likert-scale/`, 2024.

Appendix A – Environmental Setup / Configurations

Table 1: *Environmental setup / Configuration in order*

Title	Command / Description
Python Installation	Done through Microsoft Store
Python Environment Installation	Python -m venv [Env Name]
Activate Environment	cd venv\Scripts\activate
Update Python Package Manager	python -m pip install –upgrade pip
Install PyTorch	`https://download.pytorch.org/whl/torch_stable.html` with pip install torch==[PyTorch Version]
Install ML Agents Package	pip install mlagents
Check ML Agents installation	mlagents-learn –help (There will be a list of help commands)
Install Protobuf	pip install protobuf==[Protobuf Version]
Install Packaging	pip install packaging
Install Onnx	pip install onnx
Install ML-Agents Package in Unity	In a Unity Project: Window ¿ ML Agents ¿ Install

Used Versions:

- Unity Editor Version: 2022.3.20f1
- Python Shell Version: 3.9.5
- Pip Version: 24.0
- PyTorch Version: 2.2.1+cu121
- ML-Agents Version: 0.30.0
- Protobuf Version: 3.20.3
- Packaging Version: 24.0
- Onnx Version: 1.15.0

Appendix B – Potential Problems

Title	Description	Solution
Error with command ‘pip install mlagents‘	Pip changes the way it resolves dependency conflicts	Use a different dependency resolver: ‘pip install mlagents –use-feature=2020-resolver‘
Error with Protobuf package when confirming ML-Agents install	Compatibility issue with newer protobuf versions	Lower the Protobuf version
External Code Editor path does not exist	Unity might not have a default IDE chosen	Unity ¿ Edit ¿ Preferences ¿ External Tools ¿ External Scripts
Filename too long on Git	File names may exceed Windows limit	‘git config –system core.longpaths true‘

Table 2: *Potential Problems Encountered*

Appendix C – Yaml Parameters

Title	Description
Trainer Type	Type of RL algorithm used
Batch Size	Samples processed together in one optimization step
Buffer Size	Stores past experiences during training
Learning Rate	Step size towards minimizing error
Beta	Strength of entropy bonus for exploration
Epsilon	Controls PPO clipping range
Lambda	GAE discount factor for advantage estimation
Num Of Epoch	Training dataset passes per update
Shared Critic	Whether actor and critic share the same network
Learning Rate Schedule	How learning rate evolves during training
Beta Schedule	How beta evolves during training
Epsilon Schedule	How epsilon evolves during training
Normalize	Whether inputs are normalized
Hidden Units	Number of neurons per hidden layer
Num Of Layers	Number of hidden layers
Vis Encode Type	Visual data encoding before passing to network
Goal Conditioning Type	Type of goal conditioning
Deterministic	Whether agent acts deterministically
Gamma	Discount for future rewards
Strength	Value placed on extrinsic reward
Keep Checkpoints	Number of saved checkpoints
Checkpoint Interval	Frequency of checkpoint saving
Max Steps	Total training steps
Time Horizon	Steps considered for reward estimation
Summary Frequency	How often summaries are generated
Threaded	Whether training uses multi-threading

Table 3: *Yaml Parameters*

Appendix D – Hardware Used

Title	Model
Control Processing Unit	Intel i7-9700K 3.60GHz
Graphical Processing Unit	Nvidia GeForce RTX 4070
Random Access Memory	32GB

Table 4: *Hardware Used*

Appendix E – Questionnaire Section

Link to emergent behaviour video: `https://youtu.be/wbwNKbcPvdA`

The video below shows a real-time event of the seeker(in red) going through the wall. This is done due to mathematical errors in the code that allows the seeker to go through by placing himself at the perfect angle.

Hider(In turquoise) tries to stop the seeker from passing through, once through it results in a chase

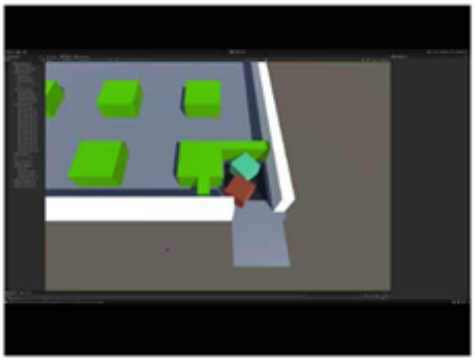


Figure 1: *Emergent behaviour video close-up*

Questionnaire Questions

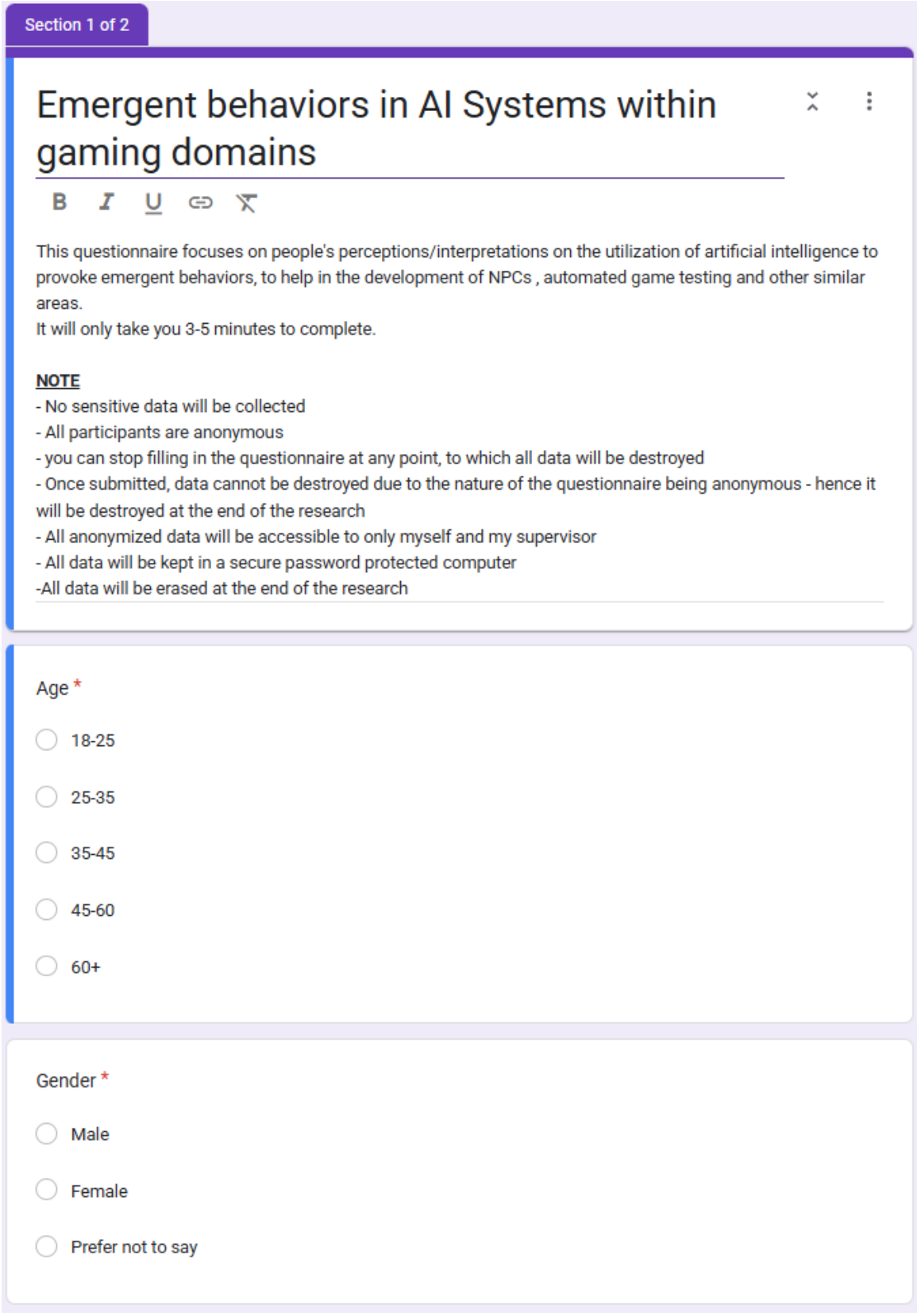


Figure 2: *Questionnaire Image_1.png*

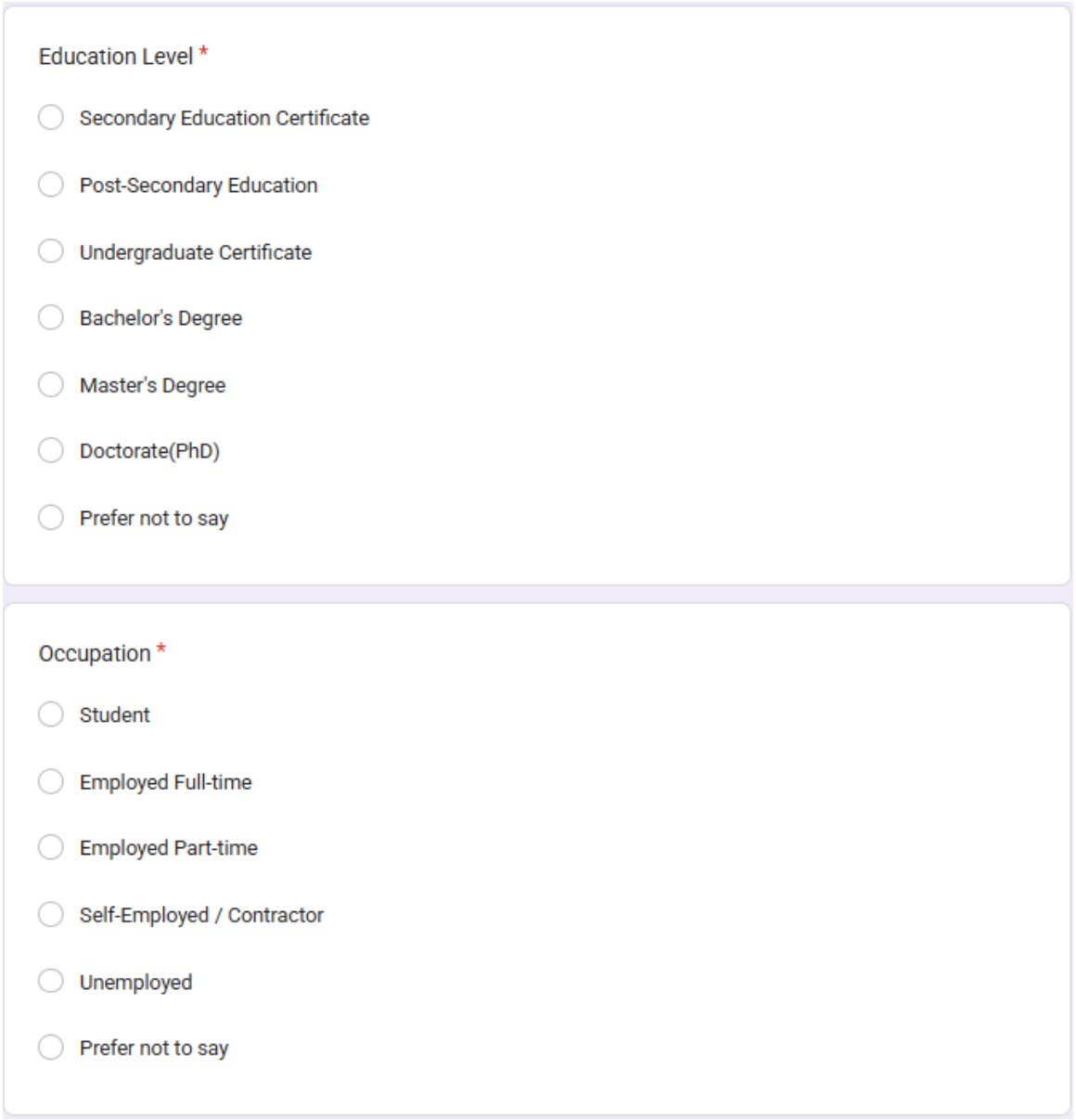


Figure 3: *Questionnaire Image_2.png*

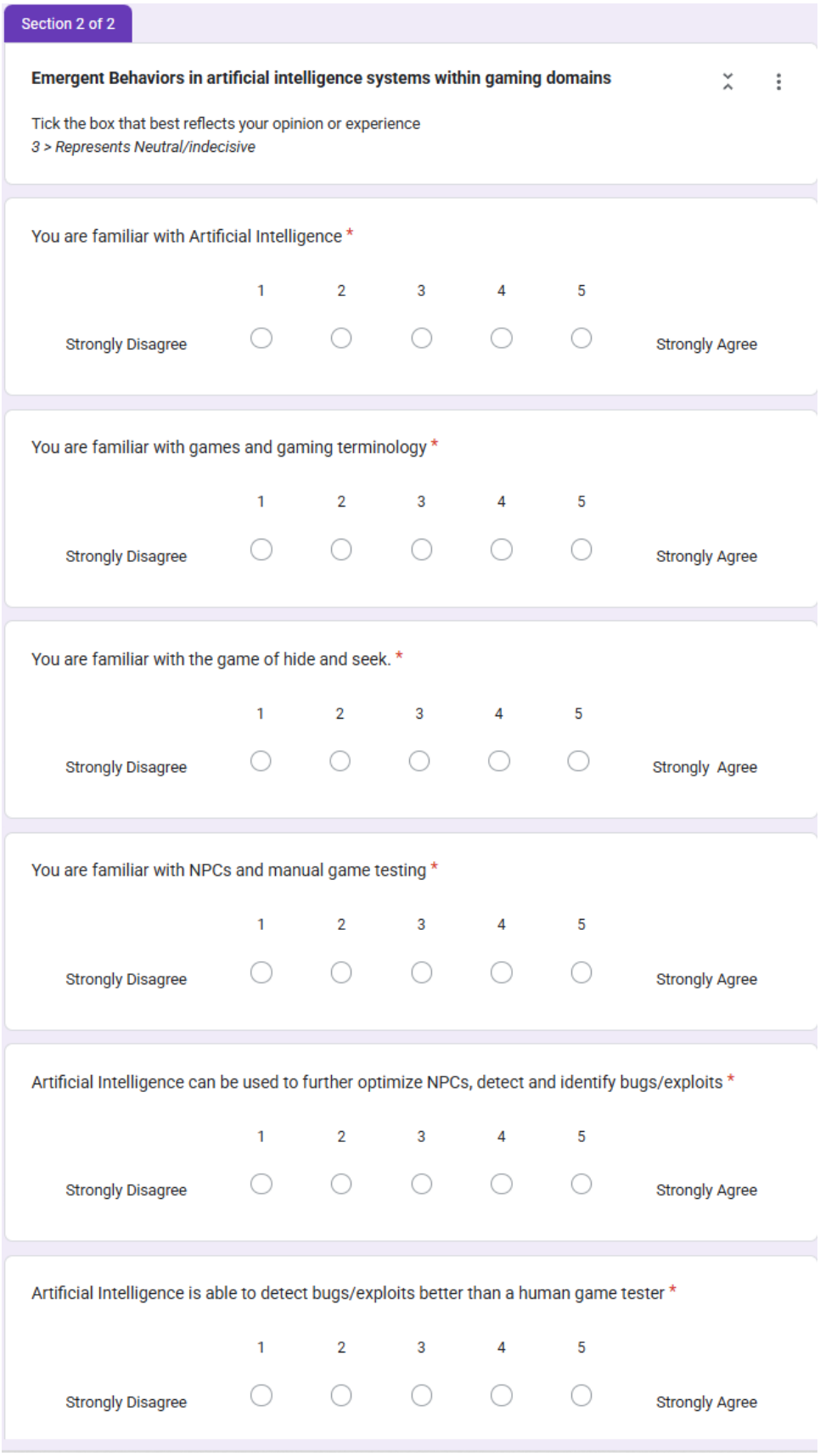
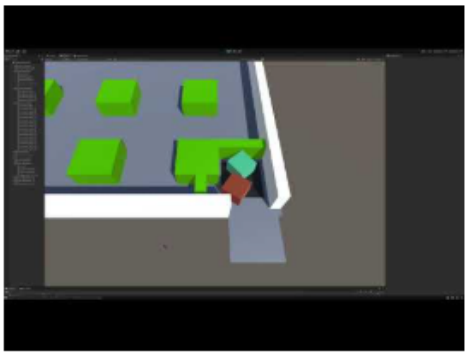


Figure 4: *Questionnaire Image_3.png*

The video below shows a real-time event of the seeker(in red) going through the wall. This is done due to mathematical errors in the code that allows the seeker to go through by placing himself at the perfect angle.

Hider(In turquoise) tries to stop the seeker from passing through, once through it results in a chase



The behavior is strategic and intentional, not random *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The seeker is taking advantage of unintended game mechanics *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The displayed behavior shows intelligence *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The agents would behave differently in a more complex environment *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Figure 5: *Questionnaire Image_4.png*