MCAST

Dynamic Difficulty adjustment aided by Reinforcement Learning Agents in Real Time Strategy games. Also Second Line

Joshua James Polanszky

Supervisor: Ms Maureen Cristina

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) Multimedia 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 (name of dissertation tutor –Title, Name and surname)

. . . .

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

The list of people that the Student would like to thank on the completion of the dissertation. For example ‘Mr Name Surname, who supported me during my dissertation work as my tutor’.

Abstract

This section should clearly state what the study is about, summarizing how it was carried out and what the results were. References are not to be included in the abstract. It should present only the essentials of the work in general.

Keywords: Dissertation, keywords.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

NN	Neural Network
ML	Machine Learning
DL	Deep Learning
FCN	Fully Convolutional Network
CNN	Convolutional Neural Network

Chapter 1: Introduction

In this section, **you**, the Student, are expected to state clearly:

(a) the 'problem' or 'question' being researched;

(b) why this topic was chosen;

(c) what motivated the you to choose this topic;

(d) why did you investigate the topic the way you did;

(e) what problem did the you wish to explore;

(f) what is the context for the research?

Percentage amount of words in section: 10 % of Dissertation*

1.1 Sub-chapter One

Background goes here. Also you can put in some references .

Another example of citations.

Here is a sample of table in Table 1.1

Age Groups	Frequency	Percent	Valid Percent	Cumulative Percent
16-20 years	**100**	**98.2**	**98.2**	**95.2**
21-25 years	**5**	**4.8**	**4.8**	**100.0**
Total	**105**	**100.0**	**100.0**	

Table 1.1: *Age of Participants*

Use `\newpage` to force start a new page.

A very quick way to create tables in a point & click environment is to use an online table generator[1]

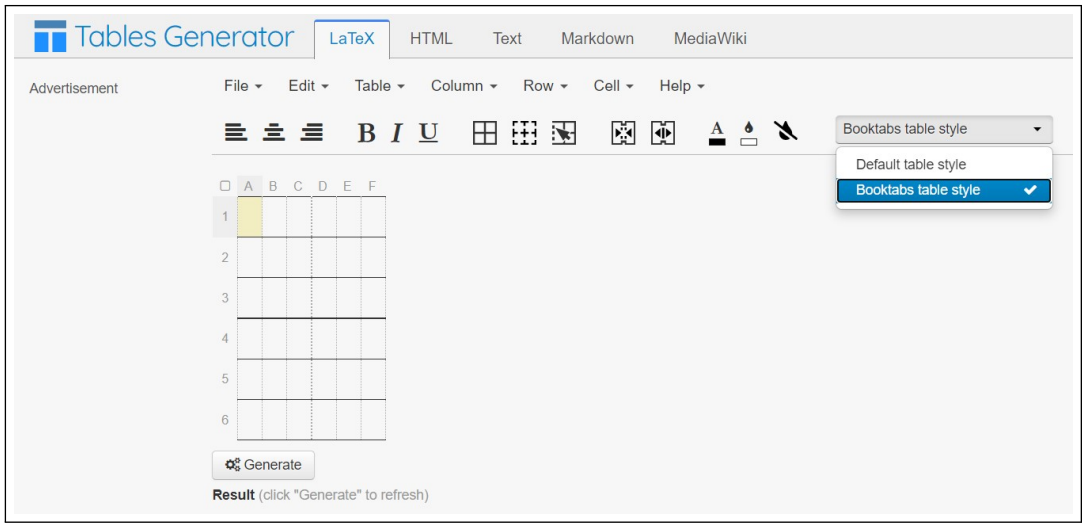

Figure 1.1: *Create a Booktabs style table*

Use `\enquote` for double-quotes. “This is a sample quote.”

Also can try to refer to this image in Figure 1.2. Notice that the `.eps` and `.pdf` format vector graphs are favoured, because:

1. they can be zoomed-in to check the detail.
2. text in such formats are search-able.

Try to insert a math equation as in Equation 1.1. If you wanna try the in-line mathematical, here is a sample $\alpha = \pi \cdot \frac{1}{\Theta}$.

$$e^{ix} = \cos x + i \sin x \tag{1.1}$$

When mention some file formats can use `music.mp3`, `latex.pdf`, etc.

[1] https://www.tablesgenerator.com

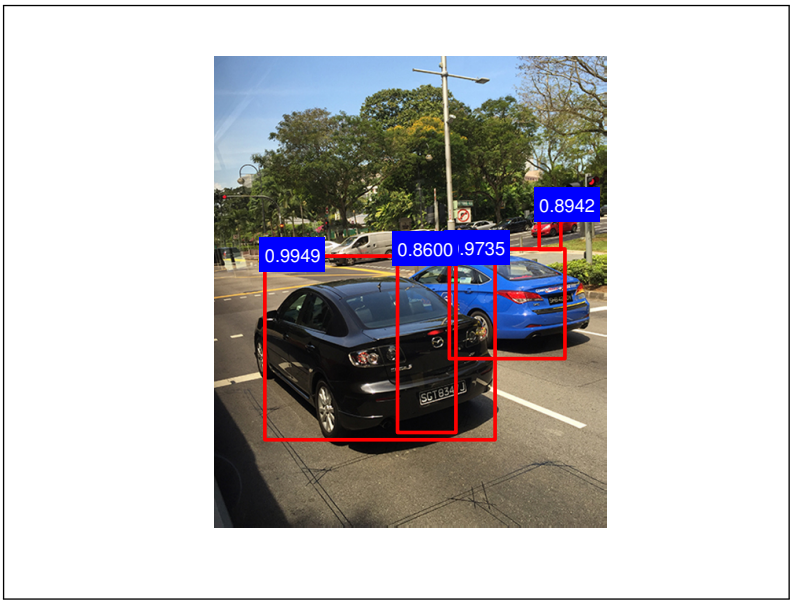

Figure 1.2: *Bounding-box example of cars.*

1.2 Sub-chapter Two

This is a todo note which appears in the margin

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of de Finibus Bonorum et Malorum (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, “Lorem ipsum dolor sit amet”.., comes from a line in section 1.10.32.

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum

is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

1.3 Sub-chapter Three

This todo note appears in line with the text. Todo notes are a great tool to leave comments or notes for self and can then be simply commented out.

Vivamus eget odio tellus. Nam libero augue, eleifend molestie est eu, tempus vehicula magna. Sed dignissim imperdiet urna, non viverra risus blandit in. In lacinia aliquet leo, ut interdum ipsum ornare sed. Praesent ac fringilla justo. Vivamus pharetra non ipsum eget semper. In hac habitasse platea dictumst. Donec neque lectus, ultricies id massa posuere, sodales interdum ante. Sed et nunc vitae urna ornare porttitor nec vitae urna. Curabitur sit amet luctus urna. Nulla porta malesuada rutrum. Pellentesque vitae velit sed odio tincidunt iaculis. Nulla facilisi.

1.4 Sub-chapter Four

To create a bulleted list you need to make use of the ”itemize” environment. A LaTeXenvironment is a special section within a page where items as placed. An environment always has a “\begin” and a “\end”. Items you can place in such an environment include:

- enumerate – used to create numbered/letter lists
- itemize – creates a bullet list
- figures – to add images or figures (.eps is the recommended format
- tables – you can create them in tablesgenerator.com then copy the LaTeX code and paste it.
- equations – for those really neat looking mathematical formulae
- this list is not exhaustive...

1.5 Sub-chapter Five

The following are some of the most common commands used during your writing. Some characters are reserved and cannot be directly written as in a normal word processor but need to be "escaped". Then there are other useful commands such as adding a footnote, inserting a new line (to begin a new paragraph after a blank line, adding labels to items for cross-referencing, etc.

- The following are special characters which require a \ before each character so the character is reproduced on the output. The tilde and exponent are even more special.
  - \#
  - \$
  - \% – comment
  - \&
  - _
  - ~– type \textasciitilde
  - ^– type \textasciicircum
- \textbf – bold font
- \textit – italics
- \underline{words to underline}

- \emph – emphasized text. If used within an italicized text, the output is normal font. If used by itself, text in its argument is italicized.

An example highlighting the use of \emph.

- An example of emphasized text with *these words emphasized* within normal font.

- *This is an example where* these three words *are emphasized within italicized font*

Chapter 2: Literature Review

2.1 Chapter Introduction

Video games are constantly evolving to provide more engaging and immersive experiences to players, be it by improved graphics fidelity, better story telling, or by increasing the challenge. When it comes to the balancing difficulty to player skill, traditional difficulty techniques often fail to properly adapt to the vast range of player skill levels, and the difference on how fast certain players learn. Dynamic Difficulty Adjustment (DDA), along with Reinforcement Learning (RL) agents emerge as solutions to adapt game difficulty in real time based on player performance, along with decreasing predictability and increasing the challenge players face, enhancing player immersion, motivation and increasing replayability [2] [3] [4]. This section explores and discusses previous works and their results regarding DDA and RL agents in video games to identify the current state of the art and to provide a foundation for the research conducted in this thesis.

2.2 Dynamic Difficulty Adjustment (DDA)

Dynamic Difficulty Adjustment (DDA) refers to a game design approach where a set of systems dynamically alter the difficulty of a game through modifying game parameters such as AI behaviour, player and enemy stats, or environmental factors in real time to match player's skill. These systems target maintaining

player engagement to be in line with Csikszentmihalyi's Flow Model, also known as the "flow state", a mental condition that is characterised by its ability to keep players deeply immersed and focus on gameplay, avoiding boredom or frustration [2] [5] [6] [7].

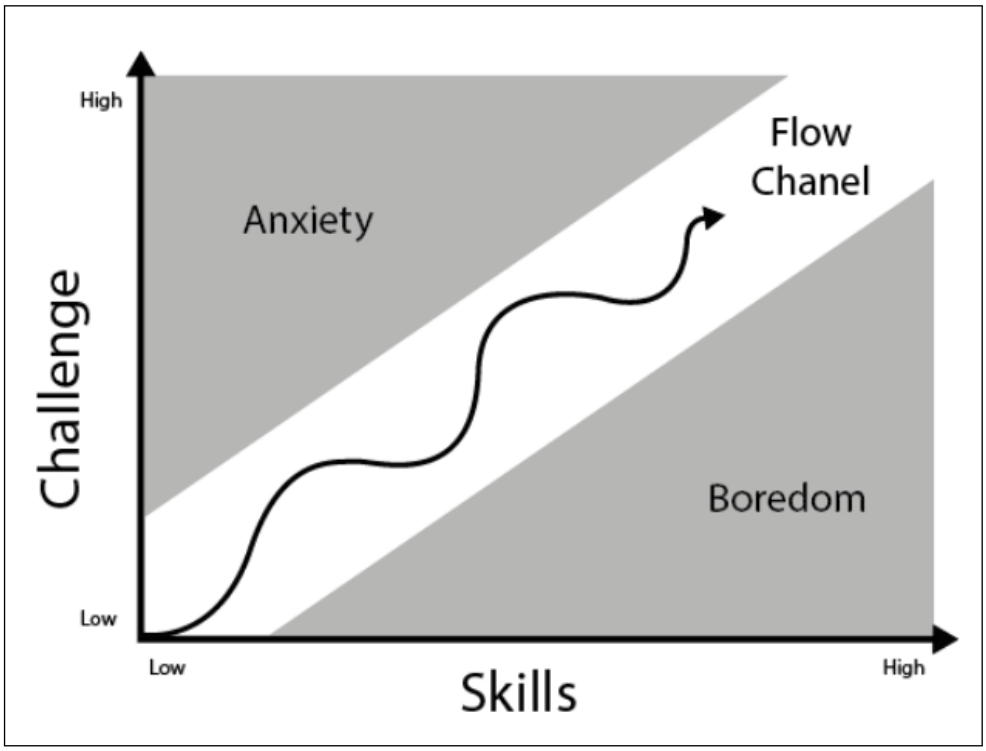

Figure 2.1: *Flow State. [1]*

One of the fundamental aspects of DDA is its adaptability to player performance, which allows it to better accomodate the diverse range of player skill levels and learning speeds, avoiding the frustration that can occur with traditional fixed difficulty settings. Research has shown that several DDA implementations, such as ramping (rDDA) and player-controlled adjustments (pDDA), can significantly improve player experience. And recently, these approaches on DDA have been combined with AI techniques and algorithms to create more sophisticated and responsive systems that can adapt to player performance in real time, such

as Deep Reinforcement Learning (DRL), Fuzzy Logic, Decision trees and Monte Carlo Tree Search (MCTS) [4] [**?**].

2.2.1 Determining Player Skill

Determining player skill is a crucial aspect of DDA, as if the DDA system is inaccurate, it can lead to the opposite effect, where the player will become more bored or frustrated, and can even lead to player's leaving the game. To develop effective DDA systems, it is crucial that developers take care to choose the variables that best determine the player's skill level. In the work done by Hunicke 31, the Hamlet DDA system was used to closely monitor the core inventory, including the players health, shielding, ammunition, and weapons, adjusting the difficulty of the level based on the player's items. In a combat encounter, the DDA system was developed to see rapid depletion of items as a sign of player struggle, and therefore the system would intervene by modifying the level to aid the player 31.

2.3 Static Game Difficulties

Currently, most games use static difficulty settings, where players choose a static difficulty level before starting the game. There are many reasons for this, such as the simplicity of implementation, the ability to control the player experience and progression, and that in the past players did not like the instability of dynamic difficulty systems, which with older algorithms did not have access nor the ability to process the complex data needed for DDA to work in these types of games [4]

2.3.1 Fixed Difficulties

These fixed difficulties rely on pre-set levels (e.g., easy, medium, hard) chosen by players before playing, and suffer from being rigid and unable to adapt to player performance, skill levels or evolving gameplay dynamics [4] [6].

2.3.2 Difficulty Selection

With difficulty selection, players can change the difficulty during gameplay, helping in reducing the frustration or boredom that comes from fixed difficulties, and helps in keeping players engaged for longer periods [4] [6]. However, this approach still requires players to manually adjust the difficulty, which can be cumbersome and disrupt the flow of the game, and are still rigid in nature, leading to the difficulty always being either too easy or too hard [4] [6].

2.4 Dynamic Difficulty Adjustment Compared with Static Difficulties

As discussed, dynamic systems outperform static ones when it comes to keeping to Csikszentmihalyi's Flow Model, helping reduce player boredom and frustration while enhancing both player immersion and replayability, thus keeping players engaged for longer periods [2] [4] [5]. The main drawbacks of DDA come from its more complex development process, the need for more data and computational resources, and the potential for it to become more boring and frustrating than fixed difficulties if not implemented correctly, and given an adequate amount of real-time data [4] [6].

2.4.1 Game Immersion and replayability

Studies such as those done by [3] [4] [5] [6] [7] confirm that DDA significantly improves player immersion by keeping players in a balanced challenge zone, keeping them both motivated and better entertained, resulting in a better player experience. Furthermore, the dynamic nature of these systems motivate players to revisit games, as the change in difficulty and challenges they present the player scale with the player's skill, keeping the game feeling fresh and engaging where fixed difficulties could not [2] [4] [5] [6] [1].

2.5 AI with DDA Algorithms

Studies have also shown that in recent time, AI has played a pivotal role in improving on the implementation of DDA systems by allowing better decision making and data processing based on player performance metrics. Such models are Reinforcement Learning (RL), as shown in the work done by [3] [5] [8], fuzzy logic [**?**], Monte Carlo Tree Search (MCTS) [9] [10], and Decision Trees [11], also known as Behaviour trees. These different AI algorithms have helped enable more nuanced and responsive DDA implementations, offering players what feels like a more tailored experience [4] [12] [**?**].

2.6 Adaptive Gameplay AI Algorithms

2.6.1 Deep Reinforcement Learning (DRL) Algorithm

RL algorithms, and in turn Deep Reinforcement Learning (DRL) algorithms use neural networks to enable AI agents to learn and adapt to the situation presented

to them. This is done through multiple methods, with the most common being Proximal Policy Optimisation (PPO), the State Action Reward State Action (SARSA) and Q-learning algorithms, which have been used to great effect in games like [13] [14]. They work by placing the agent in a controlled environment, where they learn through trial and error, being rewarded for good decisions and penalised for poor ones. This causes the agent to learn the optimal strategy for the situation presented to them, and can be used not only as opponents or allies in games, but also as a way to dynamically adjust game difficulty in real time, particularly in fighting games and other high-interaction environments, as seen in the works done by [3] [5] [8].

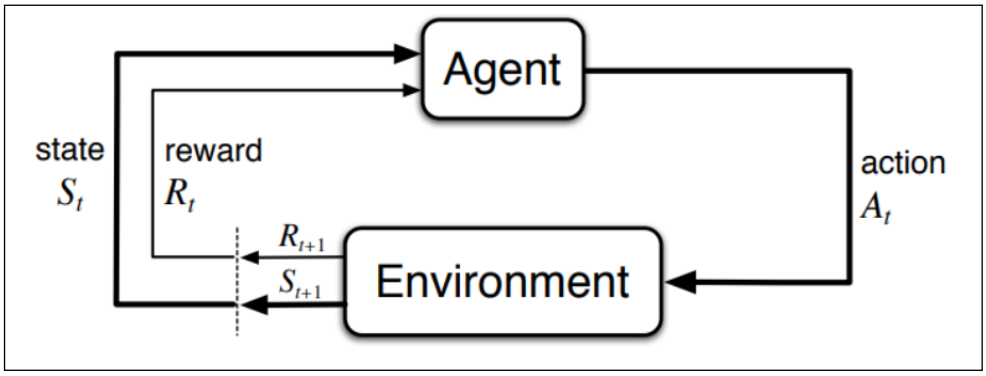

Figure 2.2: *Agent training environment interaction in RL. [2]*

2.6.2 Fuzzy Logic

In the study done by Chrysafiadi, et al. [**?**], Fuzzy logic systems were employed to dynamically adjust game difficulty to match the player's skill level, with the goal of maintaining player engagement while teaching them HTML programming. Fuzzy logic systems are particularly effective in such real-time adjustments, as they require less training data when compared to ML models. Moreover, trhese

systems offer immediate real-time adjustments to better align with player performance. The results of this study show that player dropouts decreased significantly, and player engagement and motivation increased, and even performed better than other groups, proving that fuzzy-based DDA can enhance educational outcomes and help keep players in the Flow State with reduced data [**?**].

2.6.3 Monte Carlo Tree Search (MCTS)

MCTS works by exploring different game states in the form of simulations to optimise decision-making, and is done by building a search tree iteratively, balancing exploration and exploitation to find the best move [12] [9]. In the work done by Demediuk, et al. [9], MCTS was used to enhance player engagement by modifying AI strategies to maintain balanced matches, with variants like Reactive Outcome-Sensitive Action Selection (ROSAS). ROSAS is reactive in nature, which can make it feel unnatural, whilst Proactive OSAS (POSAS), which is more proactive and can feel more natural, at the cost of precision. In the work done by Soemers, et al. [10], different MCTS optimisations were made, such as Progressive History (PH) and N-Gram Selection Technique (NST), where PH biases the selection step towards actions that have been successful in the past, and NST biases the play-out step towards actions that have been successful in the past, with the results showing that when combined these greatly improve the algorithm's performance. Another improvement was Deteministic Game Detection (DGD), which modifies MCTS behaviour to optimise for deterministic games by using alternative evaluation strategies, which was effective when combined with other enhancements [10]. What these studies show, is that when it comes to

more complex and deterministic games, MCTS can be a powerful tool to enhance player engagement both as an opponent, as well as a tool to dynamically adjust game difficulty.

2.6.4 Decision & Behaviour Trees

Decision trees are popular tools in data analysis, and are used for classification and prediction due to their intuitive structure, which include nodes (root, internal, leaf) and branches [15] [16]. The framework around how decision trees work when it comes to games varies, as not only can it reduce code repetition in game development, both for complex and simple games [17], but its structure allows for easy interpretation and visualisation of the algorithm's decision making process, making them particularly useful for adapting game features dynamically, as seen in the work done by [11]. Their approach involved examining two distinct approaches, one using predefined behaviour trees of varying difficulties, and another employing a single tree with dynamic adjustment of probabilities to alter difficulty. The first approach struggled when it came to flexibility, as it could not adapt to player performance, which was in stark contrast with the second approach, which was able to optimise behaviour probabilities and display superior adaptability, helping to keep the player in the Flow State, and maintain player engagement [11]. Along with this, decision trees have been shown to be even more effective when combined with other algorithms, such as RL, which help greatly in improving their efficiency and scalability in complex gaming environments, or even with state machines in more simple environments [18] [11].

2.7 Procedural Content Generation

Procedural Content Generation (PCG) is a technique used in game development to create game content algorithmically, leveraging algorithms to dynamically create and add content such as levels, environments, and items to games that both help in reducing development time and costs, but also increase replayability and can easily be combined with DDA systems to create more engaging and immersive experiences [6]. When paired not only with DDA, but also with AI agents, such as those created by [2], PCG was used to constantly modify the best possible action that the player and the agent could take, which in turn not only forced the player to adapt to the changing environent, but also helped keep the player feeling challenged and engaged, avoiding frustration since the opponent was also having to adapt in a human like way, helping in keeping the player in the Flow State.

2.8 Unity ML-Agents and how they work

Unity ML Agents is a toolkit that enables developers to more easily create and integrate intelligent agents within Unity environments using Machine Learning (ML) techniques. The package allows users to transform any scene into a training enviroment where agents can be trained using algorithms such as RL, Imitation Learning and neuroevolution, by allowing agents to observe their surroundings, take actions and receive rewards, as well as specifying the behaviours that these agents can take [19] [20].

2.8.1 Finite State Machines

Traditionally, and even to this day the most common way to implement AI in games is through Finite State Machines (FSM), which are used to describe a relationship between a set of states, where transitions between these states are determined by condition, with actions being performed based on the current state [2]. FSMs are useful in simple environments, however as the number of actions and states are limited, However as the number of states increase, so does the complexity of the FSM, making it difficult to manage, debug and maintain [2].

2.8.2 RL Agents

As mentioned in an earlier section, RL agents are AI agents that learn through trial and error, being rewarded for good decisions and penalised for poor ones. RL-Agents trained using Unity ML-Agents have more access to the environment and its data, and this has been used in the work done by Grech [2], where he trained an RL-Agent to act as an opponent in a Real Time Strategy (RTS) game, taking multiple stages of the agent in training for the different difficulties. There were also different RL-Agent models such as the Proximal Policy Optimisation (PPO) and others that were used to act as enemies in fighting games [13], bosses [14], and even for racing games [21]. In these works, along with those done by [22] [23] [19], it was revealed that RL-Agents have shown many times to be an effective way to make a more unpredictable, life-like and challenging opponent for player, significantly boosting their experience and engagement.

2.8.3 RL Agents compared with Finite State Machines

The main differences between FSMs and RL-Agents are their development time and costs, as FSM's are easier and cheaper to implement, but are then limited in their complexity and scalability, where as RL-Agents take more time and resources to develop, but solve the issues with FSMs being their complexity, scalability, and the simple fact that when a player fights an FSM enough, they will learn the patterns and states, making the game predictable and boring [2] [13] [14]. This life-like aspect to RL-Agents can, and have been shown to greatly improve game replayability and immersion, keeping the player in the flow state for longer and providing a superior experience to the player [2] [13] [14]. With all of this however, it is important to not that if an RL-Agent is not implemented or trained correctly, it can lead to the opposite effect, where an opponent is too difficult, leading to player frustration, or too easy, leading to player boredom [2] [13] [14].

2.8.4 Practical Applications

As mentioned in the previous sections, RL-Agents have been used in a variety of games, from RTS games to fighting games and now even racing games. The practical applications of RL-Agents in games are vast, and as hardware capabilities improve, and our understanding and training techniques improve, the potential for RL-Agents in games is almost limitless. The ability to create more engaging, immersive and challenging opponents for players, as well as allies, and the ability to easily integrate them with DDA systems, PCG and other game development

techniques to make games more replayable and engaging is a powerful tool for developers to create better games [2] [22] [23] [19] [14]

2.9 Chapter Overview

This chapter has provided an overview on the current state of the art in Dynamic Difficulty Adjustment (DDA) and Reinforcement Learning (RL) agents in video games, and how individually they have been used to improve upon more static techniques, such as fixed difficulties and Finite State Machines (FSMs). It has shown that integrating AI techniques has helped in improving the accuracy and adaptability of DDA systems, and the latest advancements in RL-Agents, including the Unity ML-Agents package, have shown just how revolutionary these techniques are at improving player engagement and immersion, as well as game replayability when these techniques are implemented.

Chapter 3: Research Methodology

3.1 Chapter Introduction

3.2 Conceptual Framework

Figure 3.1 outlines the methodology pipeline for this study's research, with it being divided into 4 major sections. The first section consisted of decided on the prerequisites and tools required for the development of the prototype, along with developing the traditional game mechanics and State Machine opponent. The second section consisted of setting up the training scene and Unity mL-Agents plugin, training, and evaluating the RL-Agent. The third section involved developing the data collection and developing the DDA system using behaviour-trees, which were decided over the other options due to their ease-of-use, and how well they are at representing the AI's decision making process. Finally, the prototype was evaluated by the participants, with the data collected being analysed and compared to previous research to answer the research questions.

3.3 Prerequisites and Tools

Before starting development on the prototype, a github repository for the project was created, and Jetbrains Rider was used as the IDE for the project. The prototype was developed using Unity 6000.0.31f1, and utilised several Unity packages, assets and plugins, which are listed below:

Along with the built-in Unity packages, the project also used the Unity ML-

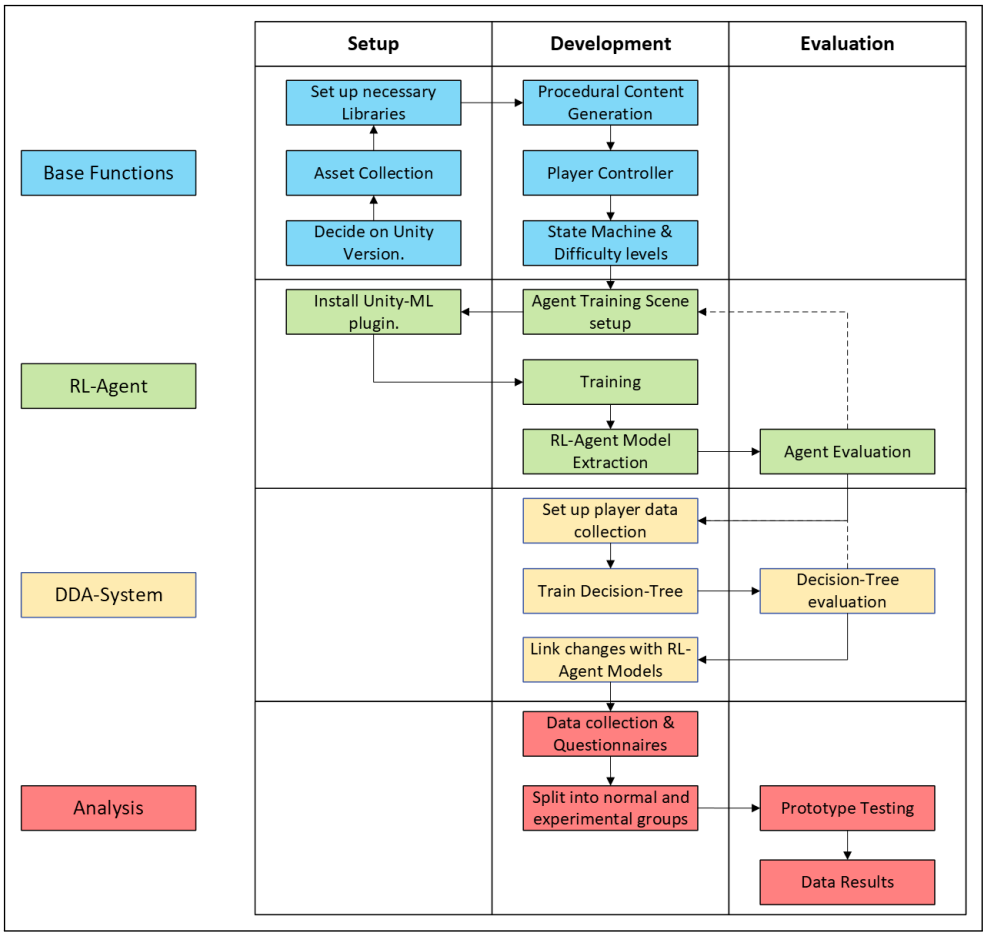

Figure 3.1: *Diagram representing the methodology pipeline.*

Agents package which was cloned off the Unity ML-Agents github repository, and the Unity ML-Agents Toolkit, which was cloned off the Unity ML-Agents Toolkit github repository, and then imported into the project as a Unity package along with the ML-Agents Extensions packages. The version of ML-Agents used in the prototype was version 22, the latest version at the time of development.

3.4 Prototype Outline & Implementation

A Prototype game was developed to implement and test the RL-Agent and DDA system for their effectiveness in creating a dynamic and adaptive game experience. The core gameplay will consists of traversing the game world in a third

person perspective, with the goal to mine and deposit as many ores as possible. There are 3 main mechanics which for the core gameplay loop of the game, the scoring system, the mining system, and the inventory system. The mining system is responsible for the player's ability to mine ores, the scoring system is responsible for tracking the player's score and awarding them points based on the types or ores deposited, and the inventory system is responsible for forcing the player to carefully think about their inventory space and which ores to mine and when to go deposit. The ores and terrain are procedurally generated, with some small obstacles which can force the player to take a slightly longer path.

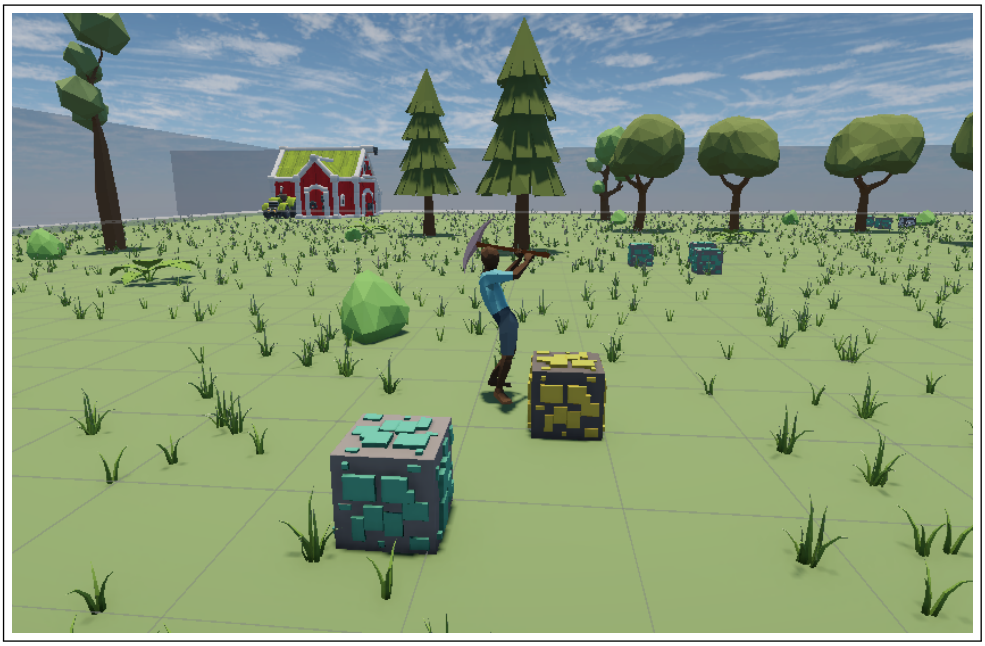

Figure 3.2: *Screenshot of gameplay showing the gathering of resouces.*

The main challenge the player will face is the AI opponent which has the same goal and capabilities of the player. The objective for both of them is to gather as many points as possible before the game ends. If the player has more points at the end, they win, if its the same, it is a draw, and if the AI has more points, the player loses. The AI opponent will be different for each stage of the

test, with the first being a State Machine, and the second being an RL-Agent. More information regarding the test stages and how data will be gathered and analysed will be discussed in a later section.

3.4.1 Ores

There are three types of ores in the game, Copper, Silver, and Gold. Each ore is visually distinct, as can be seen in Figure **??** and have a different chance of spawning, as listed in Table 3.1.

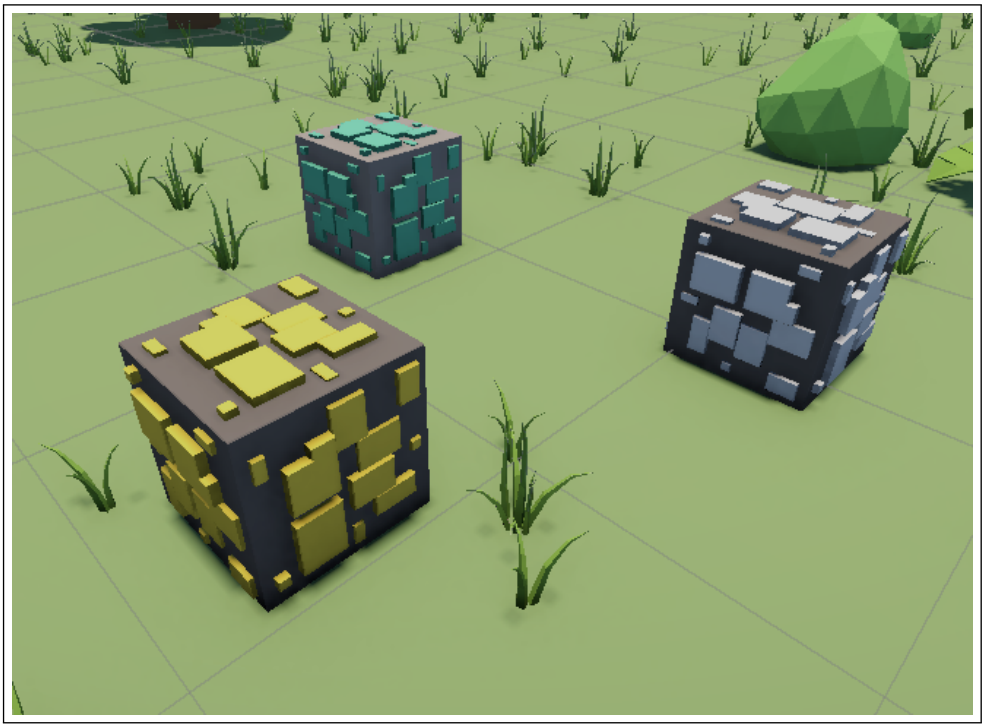

Figure 3.3: *Screenshot showing the different ores.*

Difficulty	Copper Spawn %	Silver Spawn %	Gold Spawn %
Easy	60%	30%	10%
Medium	70%	25%	5%
Hard	75%	20%	5%

Table 3.1: *Chance of spawning for each ore.*

They also take different amounts of time to mine, inventory space, and points

awarded when deposited, as listed in Table 3.2.

Resource Type	Score	Mining Time (swings)	Inventory Space Taken
Copper	2	3	1
Silver	5	5	3
Gold	10	8	5

Table 3.2: *Ore statistics.*

3.4.2 Traditional State Machine

The State Machince will be a simple AI opponent using the Unity navmesh system to navigate the game world. The AI will have 3 states, a mining state, a depositing state, and a travelling state. After completing a task, the AI will use a Physics OverlapSphere to search for the best ore within the vicinity, and once the best option is found, the AI will navigate to the ore and mine it. The speed at which the AI travels, and the radius it searches for ores will be adjusted based on the difficulty selected, allowing for a faster and more effective opponent on higher difficulties. Once the inventory is full, the AI will navigate to the closest deposit point to deposit the ore and gain points.

3.4.3 RL Agent

The Reinforcment Learning Agent will be trained using the Unity ML-Agents plugin, and will use Proximal Policy Optimisation (PPO) as the training algorithm, since it is the most commonly used algorithm due to its effectiveness at training agents in complex and continuously changing environments, the main RL training algorithm for the Unity ML-Agents framework, and the one used in the works done by [2], [13], [21], and [19]. The agent will then be placed instead

of the State-Machine as the opponent the player will face. More information on the training and evaluation will be discussed in a later section.

3.4.4 DDA System

This section will be created over time while the prototype and DDA system are being developed and tested.

3.5 RL-Agent opponent

3.5.1 Agent Observations

To understand the game enviroment, the agent is supplied with several variables to observe via the Unity ML-Agents plugin. These variables are all normalised numerical values between 0 and 1, and are as follows:

- The type of ore currently being obesrved.
- How much of the inventory the ore will take up.
- The agents normalised position from the ore.
- The player's normalised position from the ore.
- The deposit point's normalised position from the ore.
- The normalised amount of inventory space left.
- The normalised amount of time left.

```
public override void CollectObservations(VectorSensor sensor)
{
    // Collect ore type observation
    sensor.AddOneHotObservation((int)oreType, range: numOfTypes);

    // Collect normalised inventory space taken observation
    sensor.AddObservation((float) GameData.InvStorageQty[oreType] / GameData.MaximumInvQty);

    // Collect normalised distances
    sensor.AddObservation(agentDistanceNormalised);
    sensor.AddObservation(playerDistanceNormalised);
    sensor.AddObservation(depositDistanceNormalised);

    // Collect inventory space observation
    sensor.AddObservation((float)GameData.MachineData.TotalInventory / GameData.MaximumInvQty);

    // Collect time left observation
    sensor.AddObservation((float)GameData.TimeLeft / GameData.InitialTime);
}
```

Figure 3.4: *Screenshot showing the Agent's observations.*

3.5.2 Agent Decisions & Actions

The Agent has 3 decisions it can make, and 2 actions it can take. The decisions are between travelling to and mining the ore being observed, travelling to the deposit point, or observing the next ore. The travel to & mining action involves setting the ore's position as the NavmeshAgent's destination, waiting until the agent arrives, and then mine the ore. Sometimes due to obstacles and the agent's pathfinding, the agent may get stuck, which the program will detect very quickly by requesting the agent to decide on another ore. While this is not ideal, it is an uncommon bug with the NavmeshAgent. The other action the agent can take is travelling to the deposit point, which acts similarly to the travel to & mining action, but instead of mining, the agent will just observe and make another decision once it arrives.

3.5.3 Agent Rewards & Penalties

A simple reward and penalty system was used to train the agent, since the enviroment is simple and the agent's goal is clear. The focus was on rewarding the agent for mining and depositing ores, and penalising them for travelling for too long.

3.5.4 Agent Training

3.5.5 Model Selection & Evaluation

3.6 Data Collection

It was decided to use a mixed-methods approach for the evaluation of the prototype, with the majority of the data being quantitative. This was chosen to better match [2] & [3]'s approaches, allowing for an in-depth comparison and analysis of results, while allowing for a more detailed understanding of the player's experience and recommendations. Quantitative data will be collected through Unity Analytics, allowing data to be extracted from gameplay sessions, as well as close ended questionnaires. Quantitative data will be collected through open ended questions in the questionnaire.

3.6.1 Quantitative Data

To better understand the performance of the agent and DDA system against a human player, and to compare the two against State Machines, important statistics throughout the game will be tracked, and saved to a database. This data will in-

clude the player and enemy action logs, the game summary, and the performance summary.

Player and Enemy Action Logs

These logs, as seen in **??** will show all note-worthy actions taken by the player and the enemy, detailing the time, action and any relelvant information, such as the time left and score, and will be shown in chronological order. This data will be used to analyse the player's and enemy's behaviour and decision making throughout the game.

Game & Perfomance Summary

The game summary will show some of the less note-worthy actions taken by the player and enemy, such as time spent mining and travelling. It will also show the final score and the difficulty selected. Due to the DDA system in place, some of the statistics will include logs of difficulty adjustments, and the final difficulty the DDA system selected. The performance summary will be used to capture the game's performance by tracking metrics such as frame-rate, CPU Usage and memory usage.

3.6.2 Qualitative Data

To better understand the player's experience, as well as some pain points of the game, the questionnaires will have some open ended questions, asking the player to better describe their experience, and to provide feedback and any recommendations that they might have. This data will help expand upon the metrics gathered

from the quantitative data, and will be used to better answer the research questions.

3.6.3 Participants

To really gauge the effectiveness of RL-Agents and DDA systems in creating a dynamic and adaptive experience, a large diverse group of participants will be required. By having players of different ages and experience, the data collected will be more varied and allow for a more in-depth analysis of the data. This is inline with the research being conducted, as the goal is to identify places where AI is worth implementing, such as more difficult competitive games, or games with a more casual audience.

3.6.4 Tests

The tests will be conducted in two stages. First, a close-ended questionnaire will be taken by the participants to gather basic information anonymously, such as age and gaming experience. Once complete, the participants will be split into two groups, ensuring that both groups have a similar distribution of age and experience, and they will both play the traditional segment of the prototype, with traditional difficulty settings and a State Machine as the opponent. In the second stage, the first group will then play against the RL-Agent, whereas the second group will play against the RL-Agent with the DDA system in place. After each playtest, the participants will be given a questionnaire to describe their experience and provide feedback, with both closed and open-ended questions. As mentioned earlier, data from the playtests will also be gathered through Unity Analytics.

3.7 Evaluation

The data collected from the playtests and questionnaires will be analysed and cross-referenced to find any patterns or correlations between the data. The quantitative parts of the data will be analysed using T-Test analysis, and the qualitative data will be analysed using thematic analysis. The results will then be compared to previous research, and then be used to answer the research questions, and to prove or disprove the hypothesis.

3.8 Chapter Overview

Chapter 4: Analysis of Results and Discussion

4.1 One

This section includes critical discussion about the Student's findings and shows how these findings support the original objectives laid out for the dissertation, which may be partially or fully achieved, or even exceeded. The Student may also include new areas of an investigation prompted by developments in the research dissertation. Above all, it is required to present strong arguments which show how findings may offer a valid contribution to the development of the subject of the selected research area or issues related to it. Percentage amount of words in section: 25 % of Dissertation

4.2 Two

4.3 Three

Chapter 5: Conclusions and Recommendations

5.1 One

In this chapter, the Student has to evaluate the significance of the work done and give recommendations for any further investigations. Percentage amount of words in section: 20 % of Dissertation.

5.2 Two

5.3 Three

List of References

[1] G. K. Sepulveda, F. Besoain, and N. A. Barriga, "Exploring Dynamic Difficulty Adjustment in Videogames," in *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, Nov. 2019, pp. 1–6, arXiv:2007.07220 [cs]. [Online]. Available: http://arxiv.org/abs/2007.07220

[2] G. Grech, "Creating Difficulty Levels with Reinforcement Learning in a Strategy Game," Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2023.

[3] A. Mercieca, "Evaluating Player Appreciation towards Dynamic Difficulty Adjustment managed through Artificial Intelligence," Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2023.

[4] J. P. Attard, "Analysing Player Motivation when using Dynamic Difficulty Adjustment in Games," Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2021.

[5] K. Mifsud, "Utilizing Dynamic Difficulty Adjustment for improving player experience," Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2021.

[6] L. Laus, "Dynamic Gaming Difficulty Calculation: A Study On The Impacts Of Procedural Difficulty Calculation On Player Performance And Im-

mersion In Video Games." Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2022.

[7] C. Vang, "The Impact of Dynamic Difficulty Adjustment on Player Experience in Video Games," *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, vol. 9, no. 1, Mar. 2022. [Online]. Available: https://digitalcommons.morris.umn.edu/horizons/vol9/iss1/7

[8] T. Zheng, "Dynamic difficulty adjustment using deep reinforcement learning: A review," *Applied and Computational Engineering*, vol. 71, no. 1, pp. 157–162, Sep. 2024. [Online]. Available: https://www.ewadirect.com/proceedings/ace/article/view/15374

[9] S. Demediuk, M. Tamassia, W. L. Raffe, F. Zambetta, X. Li, and F. Mueller, "Monte Carlo tree search based algorithms for dynamic difficulty adjustment," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. New York, NY, USA: IEEE, Aug. 2017, pp. 53–59. [Online]. Available: http://ieeexplore.ieee.org/document/8080415/

[10] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sep. 2016, pp. 1–8, arXiv:2407.03049 [cs]. [Online]. Available: http://arxiv.org/abs/2407.03049

[11] K. Sejrsgaard-Jacobsen, "DYNAMIC DIFFICULTY ADJUSTMENT USING BEHAVIOR TREES," Ph.D. dissertation, Aalborg University, Jun. 2011.

[12] T. Vodopivec, S. Samothrakis, and B. Ster, "On Monte Carlo Tree Search and Reinforcement Learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, Dec. 2017. [Online]. Available: https://jair.org/index.php/jair/article/view/11099

[13] A. A. Bin Ramlan, A. M. Ali, N. H. Abdul Hamid, and R. Osman, "The Implementation of Reinforcement Learning Algorithm for AI Bot in Fighting Video Game," in *2021 4th International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR)*. Batu Pahat, Malaysia: IEEE, Sep. 2021, pp. 96–100. [Online]. Available: https://ieeexplore.ieee.org/document/9567749/

[14] L. A. P. Metz, "An Evaluation of Unity ML-Agents Toolkit for Learning Boss Strategies," Ph.D. dissertation, Reykjavík University, Sep. 2020.

[15] Y.-y. SONG and Y. LU, "Decision tree methods: applications for classification and prediction," *Shanghai Archives of Psychiatry*, vol. 27, no. 2, pp. 130–135, Apr. 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/

[16] I. D. Mienye and N. Jere, "A Survey of Decision Trees: Concepts, Algorithms, and Applications," *IEEE Access*, vol. 12, pp. 86 716–86 727, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10562290/

[17] A. Vuong, "Artificial Intelligence - Modularity with Behavior Trees," *Technical Library*, Jan. 2020. [Online]. Available: https://scholarworks.gvsu.edu/cistechlib/357

[18] K. McQuillan, “A Survey of Behaviour Trees and their Applications for Game AI A Survey of Behaviour Trees and their Applications for Game AI.” [Online]. Available: https://www.academia.edu/33601149/A_Survey_of_Behaviour_Trees_and_their_Applications_for_Game_AI_A_Survey_of_Behaviour_Trees_and_their_Applications_for_Game_AI

[19] U. Raut, P. Galchhaniya, A. Nehete, R. Shinde, and A. Bhoite, “Unity ML-Agents: Revolutionizing Gaming Through Reinforcement Learning,” in *2024 2nd World Conference on Communication & Computing (WCONF)*. RAIPUR, India: IEEE, Jul. 2024, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/10692314/

[20] “ML-Agents Overview - Unity ML-Agents Toolkit.” [Online]. Available: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/

[21] R. Berta, L. Lazzaroni, A. Capello, M. Cossu, L. Forneris, A. Pighetti, and F. Bellotti, “Development of Deep-Learning-Based Autonomous Agents for Low-Speed Maneuvering in Unity,” *Journal of Intelligent and Connected Vehicles*, vol. 7, no. 3, pp. 229–244, Sep. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10695161/

[22] M. Borg, “Investigating the use of machine learning in real-time strategy games.” Ph.D. dissertation, Malta College of Arts Science and Technology (MCAST), Paola, Malta, Jun. 2020.

[23] D. Zhasulanov, B. Marat, K. Erkin, R. Omirgaliyev, A. Kushekkaliyev, and N. Zhakiyev, “Enhancing Gameplay Experience Through Reinforcement

Learning in Games," in *2024 IEEE 4th International Conference on Smart Information Systems and Technologies (SIST)*, 2024, pp. 175–180.

Chapter A: Introduction of Appendix

Interview summaries, sample questionnaires, and references should be placed in this section. For easier referencing, figures, tables, graphs, photos, diagrams, etc., should be inserted within the main text such as the literature review, the experimental process or procedure, the results and discussion chapters. Appendices are usually used to present further details about the results. Appendices may be a compulsory part of a dissertation, but they are not treated as part of the dissertation for purposes of assessing the dissertation. So any material which is significant to judging the quality of the dissertation or of the project as a whole should be in the main body of the dissertation (main text), and not in appendices.

Chapter B: Sample Code

You can share your GitHub link. Below shows how to insert highlighted source code from the source file.

```
# This function takes an integer n as input and returns the sum of numbers from 1 to n
def find_sum(n):
    sum = 0
    for i in range(1, n+1):
        sum += i
    return sum

# Ask the user for input and call the function
n = int(input("Enter an integer: "))
result = find_sum(n)

# Output the result
print("The sum of numbers from 1 to", n, "is", result)
```