MCAST

Creating difficulty levels with reinforcement learning in a strategy game

Georg Grech

Supervisor: David Deguara

June - 2023

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

AI	Artificial Intelligence
FSM	Finite State Machine
RL	Reinforcement Learning
MDP	Markov Decision Process
DL	Deep Learning
DRL	Deep Reinforcement Learning
DDA	Dynamic Difficulty Adjustment

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 [4].

Another example of citations [4, 5].

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]

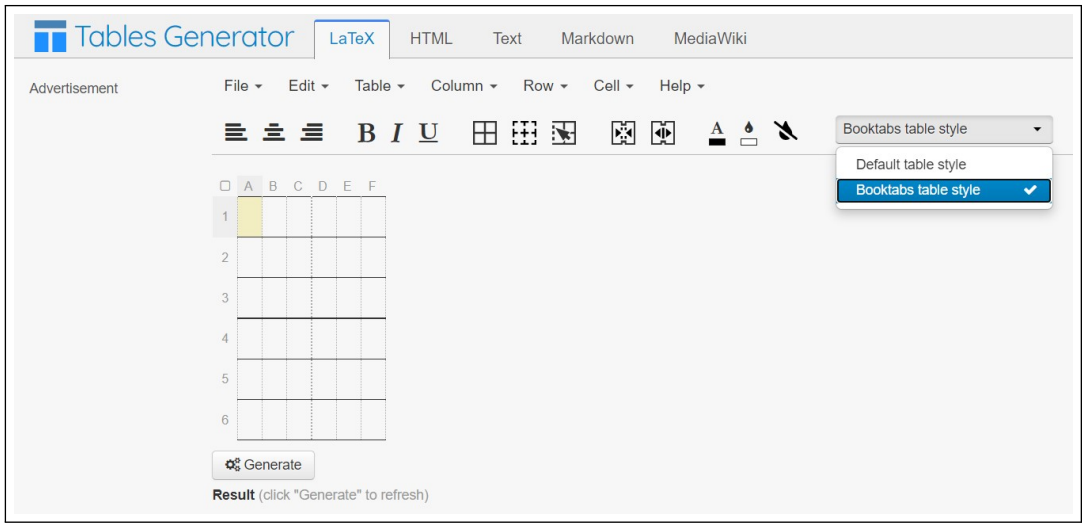

Figure 1.1: *Create a Booktabs style table*

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

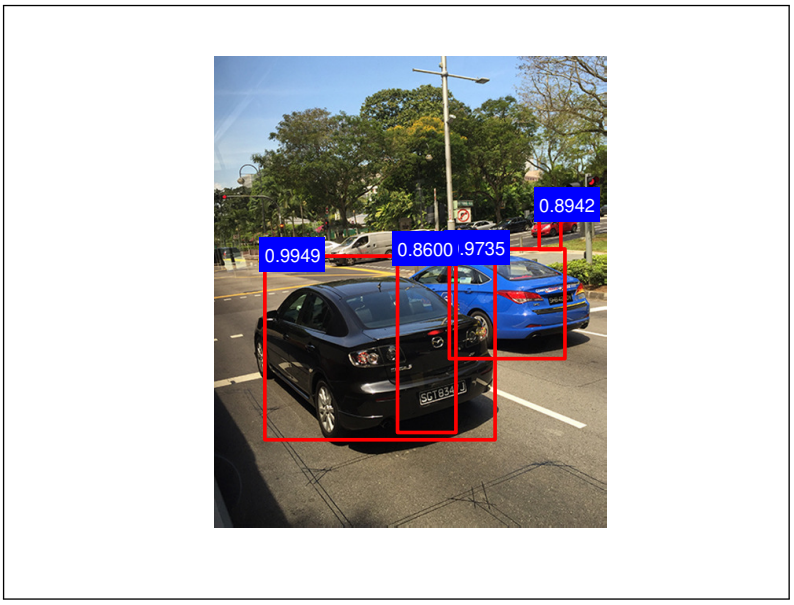

Figure 1.2: *Bounding-box example of cars.*

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 Overview

The main purpose of a literature review is to show the reader that the Student studied and analyzed viewpoints of other researchers on the problem under consideration. A literature review is not just a summary of the books read but rather a thorough analysis of other viewpoints on the problem being analysed. Percentage amount of words in section: 25 % of Dissertation.

2.2 Player Experience and Game Balance

In video game development, a critical aspect to the success of a video game is a carefully developed and well put-together Player Experience. Mainly through the method of playtesting, game developers will seek to refine in-game systems and features to shape the Player Experience, making it as enjoyable as it can be [6]. There are multiple facets that can affect the player experience (Elaborate). One such critical aspect is difficulty. The perfect level of difficulty will keep players engaged and ensure that they will not feel anxious, a result of excessive difficulty, or boredom, the usual by-product of a task of insufficient difficulty [1]. Therefore, feedback from playtesting is also used in video game development to refine difficulty until the game considered optimally balanced [7].

The difficulty of a task, however, can be perceived differently between people depending on their level of skill [1]. Therefore, for a task to always maintain

a perfect level of difficulty, the challenge presented by a task must increase or decrease relative to the player's skill. This was described in Csikszentmihalyi's "Flow Model", shown in Figure 2.1, the relationship between player skill and the task's challenge to maintain the perfect level of difficulty being known as the "Flow Channel" [1]. In an attempt to cater for this, some developers opt to create multiple difficulty settings, for instance the traditional "Easy", "Medium", and "Hard" settings. This, however, does not account for any changes in an individual player's skill level, which potentially increases as the player learn the game mechanics, making a previously challenging difficulty setting become too easy [8].

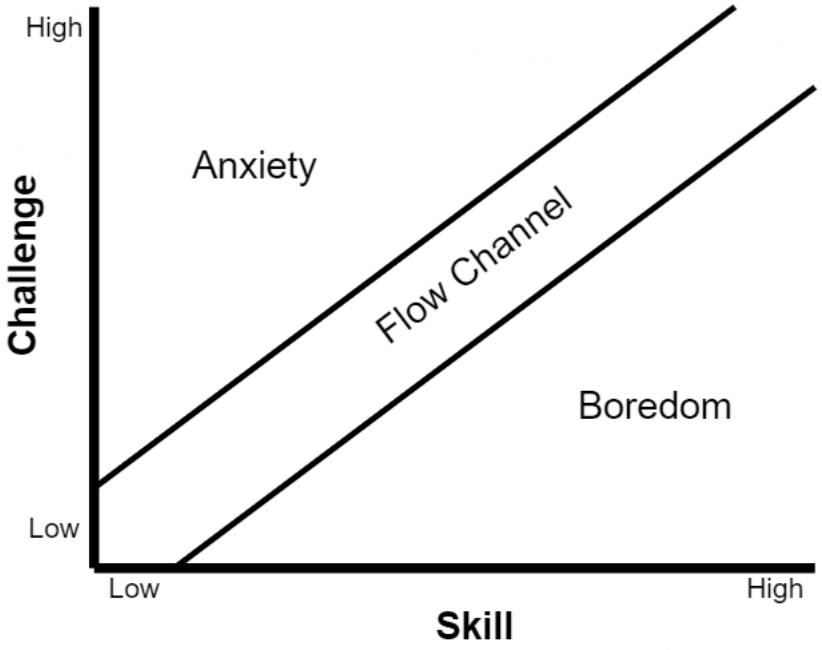

Figure 2.1: *Csikszentmihalyi's flow model diagram [1]*

2.3 Artificial Intelligence in Games

The use of Artificial Intelligence (AI) has become a prominent feature in games [9]. Well-designed implementations can serve to enhance player experience, and in turn add to the game's commercial value [10]. Most popularly, this implementation is done in the form of non-playable characters and opponents. These AI game opponents are developed to play games with two core objectives in mind, to play well, focusing on giving the player a challenge, and/or to play believably, with the AI emulating a human player to provide a more believable and immersive experience [10].

2.3.1 Finite State Machines

The complex and dynamic nature of video games requires that intelligent in-game opponents, often referred to as "agents", be made to take decisions [11]. After developers decide what actions an agent can decide to carry out, it is imperative that a process is developed that allows the agent to make appropriate decisions. Rabin [12] describes the Finite State Machine (FSM) as the most common software pattern that addresses this problem. FSMs describe a relationship between several states, transitions between states that are described by a condition in order to elicit a state change, and actions that are followed within each state [13], Figure 2.2 showing an example of this.

Citation changed after description, slight rewording might necessary

FSMs are simple to implement and debug on smaller scales. However, they can become increasingly difficult to understand and debug as more scales and conditions are added [14]. For more dynamic decision making, their inflexibility

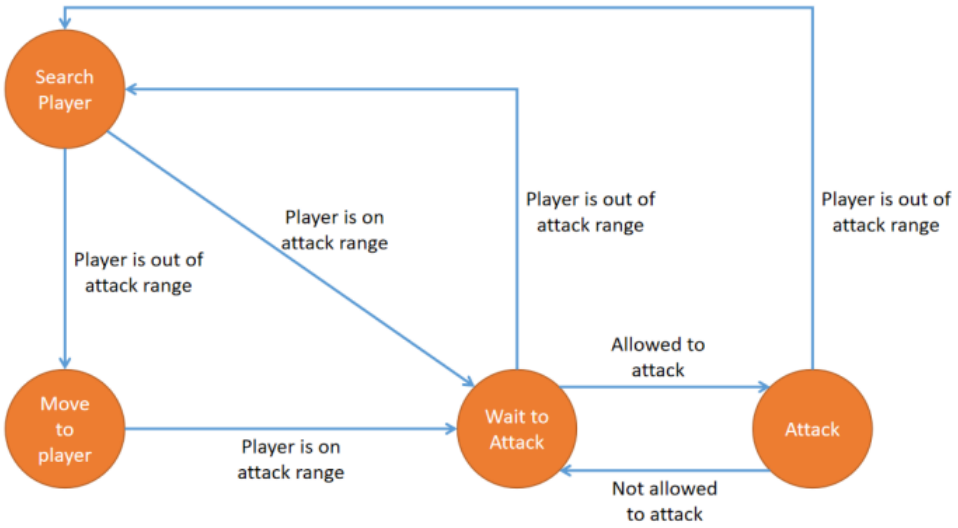

Figure 2.2: *An example of a FSM used by an enemy soldier in the game Khalid ibn Al-Walid [2]*

makes them unfavourable, and they depict predictable behaviour which players tend to be able to figure out [15].

2.4 Machine Learning

2.4.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning that takes much of its inspiration from biological learning systems. As such, it most closely emulates the learning process of humans and animals out of all machine learning methods [3]. RL concerns itself with how intelligent agents learn from interacting with the environment around them [3]. The most common solution RL systems use to solve this problem is the Markov Decision Process (MDP) [3]. The most important elements of MDPs are states, actions, and rewards [16]. On each step of interaction an MDP-based RL agent is fed an input through which it observes the state of its environment [17]. While in a state, there are several actions

available to an agent. Unlike an FSM, however, these actions aren't taken when certain conditions are satisfied. Instead, the agent is free to attempt any action available. [17]. Following the action taken, the agent is rewarded or penalised, through which the agent learns the optimal actions to take in a state [16]. After the reward is given, the agent is notified about the new state of the environment, and the cycle repeats [10]. Thus, through this repeated process of training, an RL agent becomes gradually more intelligent, as it prioritizes taking the decisions that maximize its reward [3].

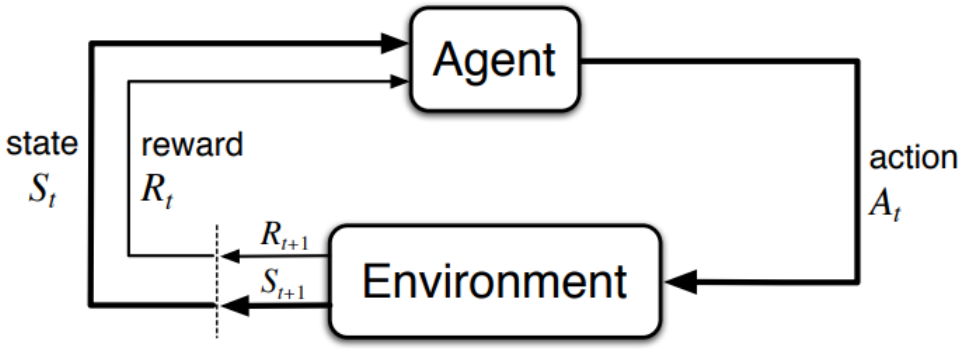

Figure 2.3: *The agent–environment interaction in reinforcement learning. [3]*

RL Terms to possibly elaborate on:

- Markov Decision Process (MDP)

- Temporal Difference (TD)

- Q-Learning

- Proximal Policy Optimization (PPO)

2.4.2 Reinforcement Learning in Games

Maybe skip this section and explain about DRL in games?

2.4.3 Deep Learning

Deep Learning (DL) is another method of Machine Learning that is based on the use of Neural Networks [18]. Neural Networks are systems that are based off of biological brains such as those of humans and animals. In these systems, an artificial neuron receives information processed by other neurons before it, processes it further, and passes it on the the next neuron to continue processing [19]. This layered approach continues until the system ends up with a single output [19]. DL, therefore, can learn from an existing unprocessed data set [20]. DL algorithms process this raw data using multiple processing layers to find structures and patterns [21]. This process is automatic, not requiring any sort of prior data processing to find and extract features [21].

2.4.4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a method of Machine Learning that combines RL and DL. DRL uses the DL approach to process a large quantity of raw, complex data and break it down into simpler data to be used by the RL agent [22]. This processing done by a Neural Network allows the RL agent to learn directly from a complex input such as a visual signal composed of pixels [23].

2.4.5 Deep Reinforcement Learning in Games

Agents using DRL have had impressive results in video games. A demonstration of the performance and range a DRL agent can achieve was seen in a 2013

study where a DRL agent was trained to play several Atari 2600 games, using the raw on-screen pixels as observations [24]. In the games of *Breakout*, *Enduro*, and *Pong*, this agent managed to score higher on average than an expert human player [24].

This level of performance have made DRL agents a logical choice to serve as opponents against human players in more competitive games. This includes real-time fighting games such as *Super Smash Bros. Melee* [25] and *Blade & Soul* [26]. In scenarios such as these, where two players directly interact with each other, a technique known as "self-play" may be adopted [27]. With self-play, an agent learns by playing against past versions of itself, trying to achieve an improved performance [27]. Using this technique, both fighting game agents were able to achieve favourable results against competitive players [25] [26]. However, it is unlikely that a DRL agent will learn how to combat all possible player strategies during training, and therefore may be easily defeated with certain unconventional strategies, as was observed by [25].

DRL has now been tried in competitive strategy games. One of the most notable examples in recent years is OpenAI Five in *Dota 2*. OpenAI Five was also trained using a self-play technique. After the completion of training, it defeated the world champion team at an Esports game [28]. Despite this success, there is the significant drawback of training time required for an agent to reach this level of intelligence playing complex strategy games. For instance, *Dota 2*'s level of complexity required OpenAI Five to use a model with over 150 million parameters, resulting in 180 days of training time, spread out over a total of 10

months [28].

2.4.6 Unity ML Agents

Unity is an immensely popular game engine, used by amateur and professional developers alike [29]. Its popularity has led many Unity developers to attempt to implement modern technologies, some of these implementations being published as plugins and packages to aid other developers [30]. Machine Learning, including RL and DRL, is one such technology developers have attempted to implement. The most popular package for this is the open-source Unity ML-Agents package, initially developed and published in-house by Unity Technologies [31].

The Unity ML-Agents package has been used successfully for the development of intelligent agents as opponents. A 2021 study, for instance, used the package to develop an AI opponent in a fighting game [32]. This agent was trained by observing health and stamina points of itself and its opponent, and being rewarded for actions that get it closer to winning [32]. Another study using Unity ML-Agents sees an agent being trained to play the popular board game *Connect-4* [33]. This agent was successful in learning strategies for effective play, managing to win against a human player [33]. (Issues with ML-Agents)

2.5 Dynamic Difficulty Adjustment

A solution that game developers have opted to use in order to make video game difficulty conform to Csikszentmihalyi's Flow Model [1] and cater to players of differing skill levels is that of Dynamic Difficulty Adjustment (DDA). DDA is the process of making automatic adjustments to a video game in real-time to modify

its level of difficulty, based on the player's abilities [34]. With these adjustments, an optimal level of difficulty is attempted to be found for every individual player regardless of how their skills fluctuate, keeping them in the Flow Channel and negating boredom or frustration [35].

Many players have been shown to enjoy DDA over static difficulty settings. A 2017 study, for instance, tested a dynamically adjusted Match-3 game on a group of participants. These participants reported a higher level of engagement compared to a focus group that played a version of the same game with a static difficulty [36]. Participants using DDA willingly played more rounds of the game and had an increased gameplay duration [36]. Players, however, have been shown to enjoy a good challenge more than winning, therefore DDA systems should be careful not to exaggerate when lowering difficulty. This was pointed out by [37], who carried out an evaluation between five versions of an AI opponent in a real-time strategy game, two of which static, three of which using dynamic difficulty. In this study, two dynamic versions of the opponent were ranked as the most enjoyable by players. While the static versions did not rank as high, they were higher than the third dynamic version, which had a special condition that lowered its difficulty in a manner that always let the player win if they were seriously struggling [37]. This exaggerated lowering of difficulty made the opponent feel boring, even if its DDA system was very similar to one of the high-ranking dynamic opponents [37].

2.5.1 Determining Player Skill

To develop an effective DDA system, developers must take care to choose the variables that best determine the player's skill level, although these can vary greatly between games. In Hunicke's implementation in a game developed in the *Half-Life* engine, the Hamlet DDA system was used to monitor the game's core inventory, including health, shielding, ammunition, and weapons, with adjustments in the level being carried out partly based on fluctuations of items in the inventory [7]. When in a combat encounter with an enemy, the DDA system viewed rapid decreases of items in the player's inventory as a sign of struggle, and therefore the system intervened to carry out necessary level modifications to aid the player [7]. (Mention more DDA parameters).

2.5.2 Player Skill in Strategy games

Perhaps instead of "Strategy Games", talk about competitive Player vs Opponent games

Player Skill in Strategy games

2.5.3 Game Modification

After developers decide on the ideal parameters for measuring skill level, it is then to determined what changes will be carried out in-game to accordingly adjust difficulty. A manner this can be accomplished is through the supply of helpful in-game items, an example of which can be seen in Hunicke's implementation, where in response to the system detecting the player struggling, a health pack may spawn somewhere within the scene, as well as health packs and ammunition having a higher chance of being dropped by an eliminated enemy [38].

Modification of in-game opponents can also prove effective as can be seen in Chowdhury and Katchabaw's study [39]. In their implementation of DDA in a variant of Pac-Man, adjustments are made to the attributes of enemies, modifying their movement speed and area of vision, among others.

2.5.4 Applications of RL/DRL for DDA

There has been thought to combine RL and DRL methods with DDA, modifying agent intelligence to achieve an enjoyable game balance.

Chapter 3: Research Methodology

3.1 Research Pipeline

Implement Pipeline, rewording some things to make reference to it

3.2 Prototype Game Outline

A prototype game was developed to implement and test Reinforcement Learning for our research. The game was created using the Unity game engine version 2021.1.22f1. The core gameplay consists of traversing a map in a top-down setting, filling the player's inventory by gathering resources and exchanging them at their base for points. Resources on the map are spawned procedurally with random positioning and being of a random type. The game also procedurally spawns obstacles on the map to provide the player a challenge in getting near a potentially valuable resource.

The main challenge presented to the player was through an AI opponent which could take the same actions as the player. The objective for both the human player and the AI opponent is to gather as many points as possible before a timer of five minutes runs out. The game ends when the timer finishes, the player winning if they manage to gather more than the AI opponent, otherwise resulting in a loss if the opponent has a greater score or a draw if both are equal.

For the purpose of this research in analysing difficulty, the difficulty setting

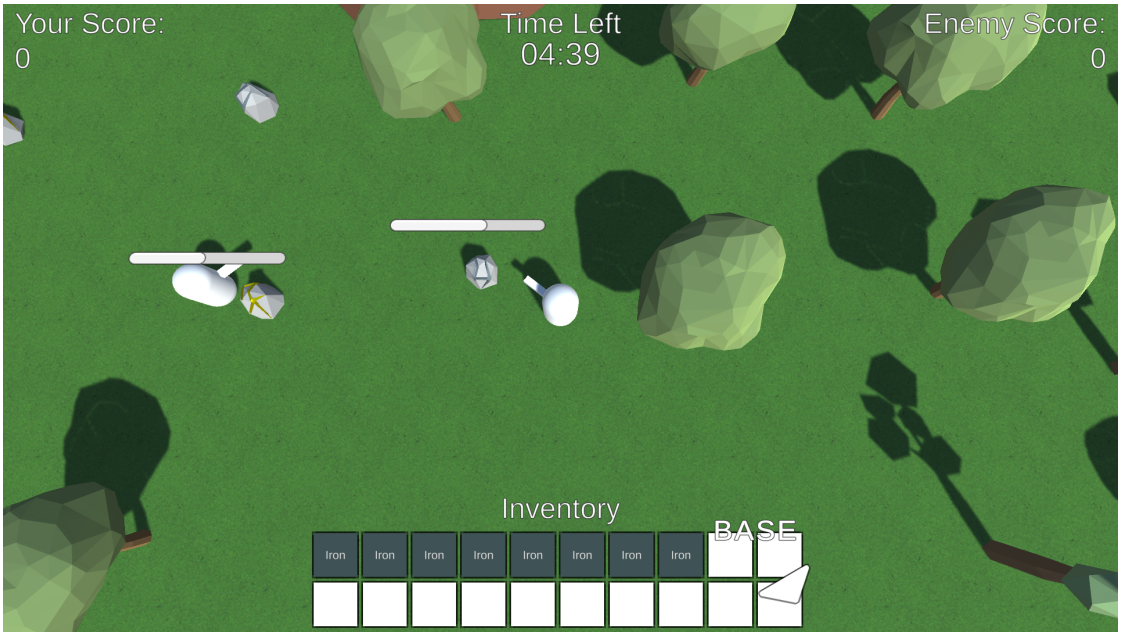

Figure 3.1: *Screenshot of gameplay showing the player and opponent gathering resources*

could be chosen via the main menu before starting a game. Players are given a choice between three difficulty levels: Easy, Medium, and Hard. This setting modifies the agent by switching between versions with different training durations.

3.2.1 Resources

There are three types of resources that can spawn in the game. These are wood, iron, and gold. The spawned Resource objects contain an amount of items which, after interaction, one-by-one fill the player or agent's inventory until the resource depletes or their inventory is full. This process is illustrated in Figure 3.2.

Resource objects of different types are visually distinct, as can be seen in Figure 3.3. They also have different chances of spawning.

Resource Type	**Percentage Chance to Spawn**
Wood	75%
Iron	20%
Gold	5%

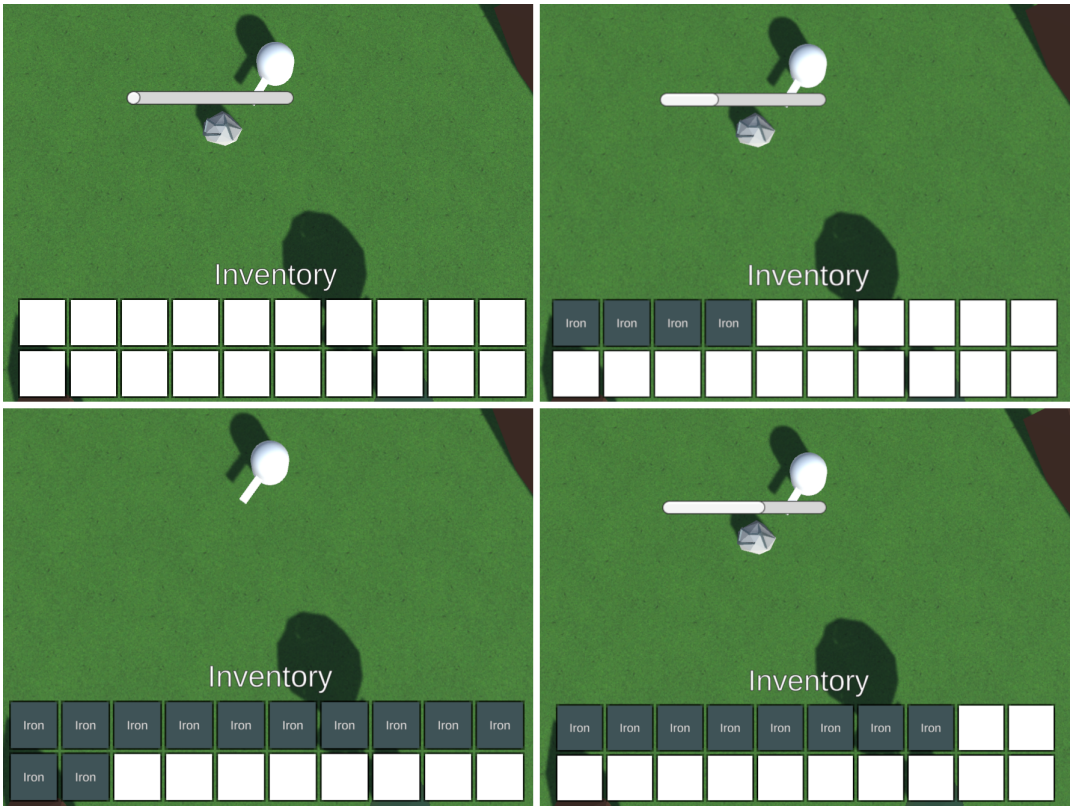

Figure 3.2: *Clockwise from top-left, the player gathering a resource and filling their inventory*

Resource types have distinct differences in time required to gather, inventory space taken, and worth in points. Per inventory item, the resource types have the following attributes.

Resource Type	**Score**	**Gather Time**	**Inventory Space Taken**
Wood	5	1.5	1
Iron	15	3	4
Gold	30	5	5

The resource objects, taking account all of their contained items, have the following attributes.

Resource	**Drop Amount**	**Total Score**	**Total Gather Time**	**Total Inventory**
Wood	4	20	6	4
Iron	3	45	9	12
Gold	2	60	10	10

Figure 3.3: *Types of resources found in the game, as shown by the in-game tutorial panel*

3.3 AI Opponent

Our main goal when developing the AI opponent was that it should play the game intelligently to provide a challenge to the player. Therefore, it should have some level of understanding of when best to gather a resource, the optimal resource to gather, and when best to return to base.

The functionalities of the opponent were created using several AI-related packages. Unity ML-Agents was used to implement DRL for decision making. Unity Navmesh and Navmesh components were used for pathfinding and distance calculation when taking decisions.

Elaborate on version

3.3.1 Agent Observations

To understand its environment, our agent is supplied with variables to observe via the Unity ML-Agents CollectObservations method, seen in Figure 3.4. All of the observations are normalised between 0 and 1 . These are:

- Game Time left

The game duration left, normalised by diving the seconds left with the maximum game seconds.

- Agent Inventory

  Inventory available to the agent, normalised by dividing by the max inventory size.

The agent also observes variables related to a selected resource. These are:

- Normalised Distance from Agent
- Normalised Distance from Base
- Normalised Inventory space taken by resource
- Resource Type

The process of resource selection and how these variables are normalised will be described later on.

```
public override void CollectObservations(VectorSensor sensor)
{
    try
    {
        if (targetTransform != null)
        {
            sensor.AddObservation(targetPDistanceNormalised);
            sensor.AddObservation(targetBDistanceNormalised);
            sensor.AddOneHotObservation((int)targetType, numOfTypes);
            sensor.AddObservation(targetInvAmountLeftNormalised);

        }
        sensor.AddObservation((float)enemyPlayer.inventoryAmountFree / enemyPlayer.maxInventorySize);
        sensor.AddObservation((float)enemyPlayer.gameManager.timerSecondsLeft / enemyPlayer.gameManager.timerTotalSeconds);
    }

    catch
    {
        Debug.Log("Exception caught in observations");
    }
}
```

Figure 3.4: *The agent's CollectObservations method*

3.3.2 Agent Actions

The agent's logic was based off two main actions, GatherResource and ReturnToBase. Both actions are similar in functionality. The process begins by giving the agent a destination which it will attempt to move towards. The process then suspends until the destination has been reached. Once it has been detected that the destination has been reached, the agent proceeds to interact with the object at its destination. The process once again waits until the interaction has concluded, after which the agent decides on the next action to take. This process is illustrated in Figure 3.5

The main difference between the GatherResource and ReturnToBase actions are the destinations. The destination in GatherResource is a resource that the agent has decided it wants to attempt to gather, while in ReturnToBase the destination is the enemy's home base, where the agent attempts to deposit the resources in its inventory.

3.3.3 Decision Process

Mention old system before scrapping due to limitation?

Using features provided by the Unity ML-Agents package, a process was built to that allows the agent to observe the attributes of nearby resources and decide between the previously described actions, either returning to base or selecting a resource and gathering it. The entire process is described below and visualised in Figure 3.6.

- **Selecting nearby resources**

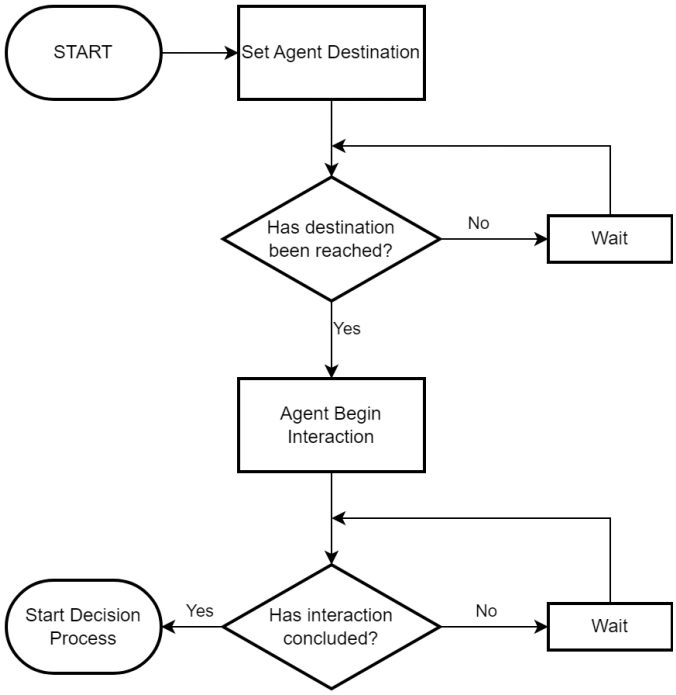

Figure 3.5: *Flowchart of process taken by GatherResource and ReturnToBase actions*

To reduce computational power when gathering information about resources, the agent is limited to only selecting from resources in the immediate vicinity. This limiting was achieved by using a Unity OverlapSphere, saving the colliders of resources within a set range of the agent. If no resources are found within range, the size of the OverlapSphere is increased.

- **Resource attribute collection**

Once the agent finds resources in the vicinity, it goes through them and saves their attributes. This includes the two distance values: how distant the resource is from the agent, and how distant the resource is from the base. To take into account the procedurally generated obstacles, Navmesh distance was used, giving a more accurate indication of

the length of the path the agent would have to travel. The furthest and closest of these distances are saved to be used for observation normalisation later on. The type of resource is also saved, along with the inventory space gathering this particular resource would take.

- **Updating Observations**

The next step in this process is to update the agent with the appropriate observations. Before setting the observations, attributes need to be normalised between 0 and 1. In the case of distances, this is achieved with the following equation:

$$ND = \frac{ResourceDistance - MinDistance}{MaxDistance - MinDistance} \tag{3.1}$$

The resulting normalised distance ND may never be less than 0 or greater than 1.

$$0 \leq ND \leq 1 \tag{3.2}$$

$ND = 0$ indicates that this resource is the closest, while $ND = 1$ indicates that it is the furthest of the scanned resources.

Amount of inventory taken by the resource is normalised using the agent's maximum inventory size. The agent's observations are finally updated with the normalised attributes.

- **Requesting Decision**

After updating the agent's observations, the agent calls the Unity ML-

Agents method RequestDecision to let the agent decide between three discrete actions it can take:

- Gather this resource

  The agent begins the GatherResource action, setting its destination to the resource currently being observed and attempts to gather it.

- Return to base

  The agent begins the ReturnToBase action, setting its destination to its base and attempts to deposit what it has collected till that point.

- Observe next resource

  The agent iterates through the next resource in the list and repeats the steps of updating the observations and calling RequestDecision. If the end of the list has been reached, the agent starts going through the list anew.

3.3.4 Reward and Penalty setup

Mention past versions before final rewards/penalties config?

For our research, it was decided to use a simple approach for rewards and penalties, with focus on rewarding the agent for score obtained, while also teaching the agent to avoid unnecessary wastes of time.

This was achieved with the following setup:

- Reward for depositing at base

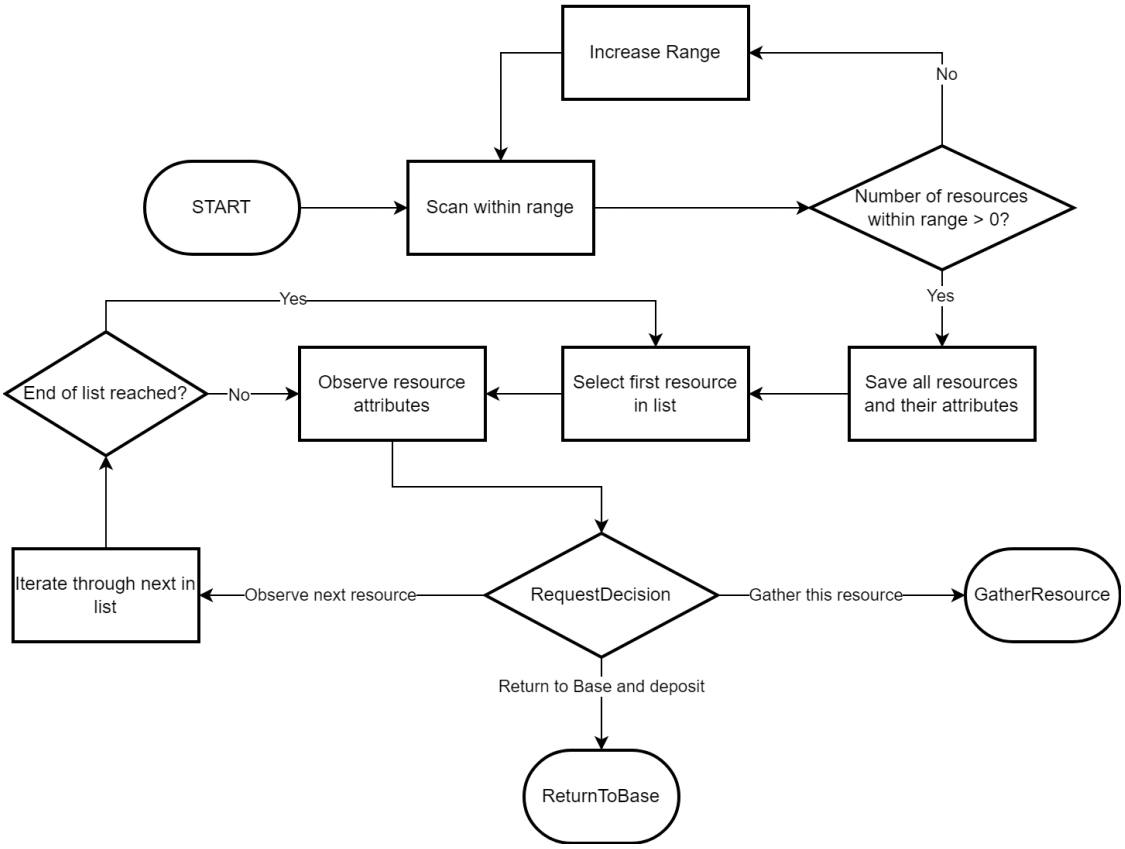

Figure 3.6: *Flowchart of agent's observation and decision process*

A reward was given for every resource item deposited at the agent's base. The reward given was based on the amount of the points the resource is worth, resulting in the following reward values for each respective type.

Resource Type	**Points worth**	**Agent Reward**
Wood	5	0.25
Iron	15	0.75
Gold	30	1.5

- Repeated penalty for walking

The agent was repeatedly penalised during moments where it was not interacting with any resource or its base, during which it was likely traveling. The weight of this penalty was 0.05 given every second.

This penalty was implemented to motivate the agent to minimise unnecessary travel distance.

Before settling on this system, two other reward/penalty setups were implemented and tested on this agent.

- Version 1

  Rewarding for base deposit, same as latest implementation, but without any penalties. This simple system was tried to give the agent a clear goal of obtaining the largest possible amount of points.

- Version 2

  Rewarding for base deposit, but also penalising based on the amount of inventory left empty when deciding to deposit at base. This was attempted to motivate the agent to fill its inventory before depositing at base, minimising number of trips to base.

3.3.5 Agent Training

Training the agent was done on a version of the game without a human-controlled character and their base. The time scale during training was quicker than real-time to allow for faster training, being set to the Unity ML-Agents default of twenty times faster than real-time. Training episodes ended after five minutes to simulate a real game, scaled according to the time scale, and the level reset with newly spawned resources and obstacles to allow for a new episode. The configuration file for our training included a *checkpoint_interval* parameter, allowing

models to be extracted without need for the researcher's intervention.

Tensorboard was used to assess agent performance during training, allowing the researcher to view cumulative reward per episode, as well episode length. Tensorboard also allowed us to log custom variables. Therefore, the agent's score at the end of every episode was logged, and this was used when deciding which extracted models to carry out further evaluation on.

3.3.6 Model Selection

The agent's training runs were halted once its average score per episode stopped increasing. It was then decided at which points to select the three models for the difficulties by observing the score. The models used for easy and hard were chosen at the points where the scores were the lowest and highest respectively. The medium difficulty model was selected at the midpoint between the lowest and highest score.

Include training diagrams for the three versions, marked at extraction points

Version 1

Training for Version 1 was halted after 145 hours of training, equal to over 557,600 training steps.

Models for the difficulty settings were extracted at the following points.

Difficulty Setting	**Steps**	**Time**	**Average Score**
Easy	1.5K	19 minutes	255
Medium	13.5K	9.5 hours	287
Hard	490K	133 hours	319

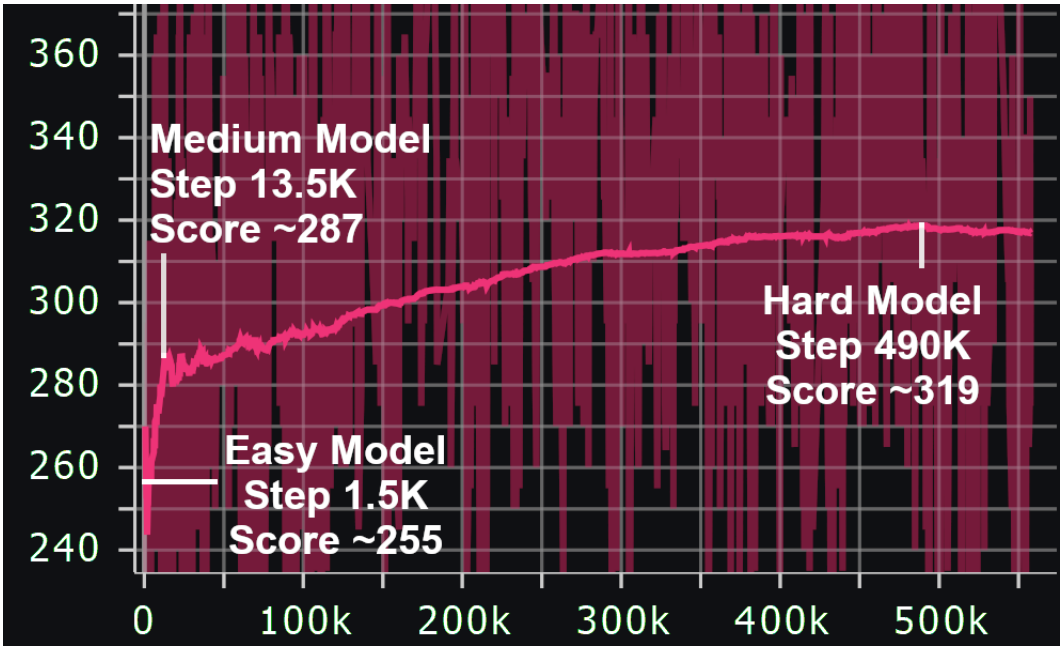

Figure 3.7: *Tensorboard graph of Version 1 agent's average score throughout training, marked with model extraction points*

Version 2

Version 2 was halted after 40 hours and 185,800 steps of training, during which it achieved a maximum average score of 310. Models were not extracted for evaluation for this agent, due to poorer performance compared to the previously developed Version 1.

Version 3

Training for Version 3 was halted after 16.5 hours and 105,900 steps of training. Models for the difficulty settings were extracted at the following points.

Difficulty Setting	**Steps**	**Time**	**Average Score**
Easy	1.5K	23 minutes	224
Medium	12K	2.7 hours	283
Hard	93K	14 hours	328

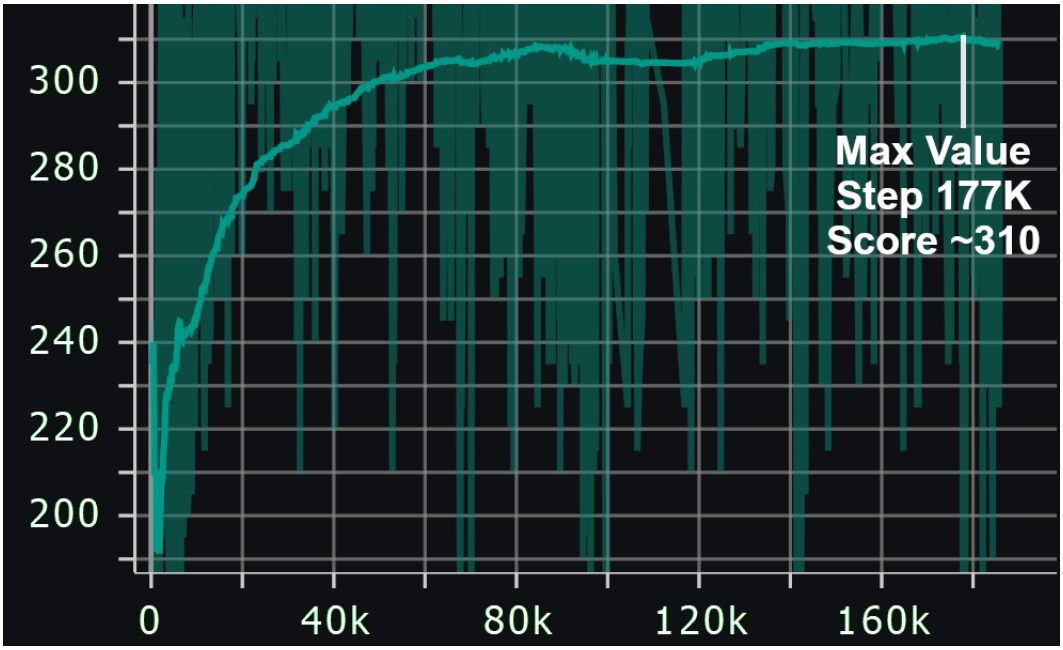

Figure 3.8: *Tensorboard graph of Version 2 agent's average score throughout training*

3.3.7 Model Evaluation

To evaluate the performance of models in real-time and examine differences between them, another level was created similar to the training level. Our evaluation level also lacked the player character and base, but ran in real-time.

After loading the models we wished to evaluate into our project, the evaluation level was run. On running, an agent is assigned the first model we loaded into the project, then plays a set amount of five-minute rounds, keep track of the score on each round. After these rounds are played, the average score of the agent using that particular model is calculated and saved, and the process repeats using the next loaded model, until all the models that require evaluation are used. The results of all rounds and the average score per model were saved in a local file within project.

When evaluating the models for each of our runs, it was decided to have

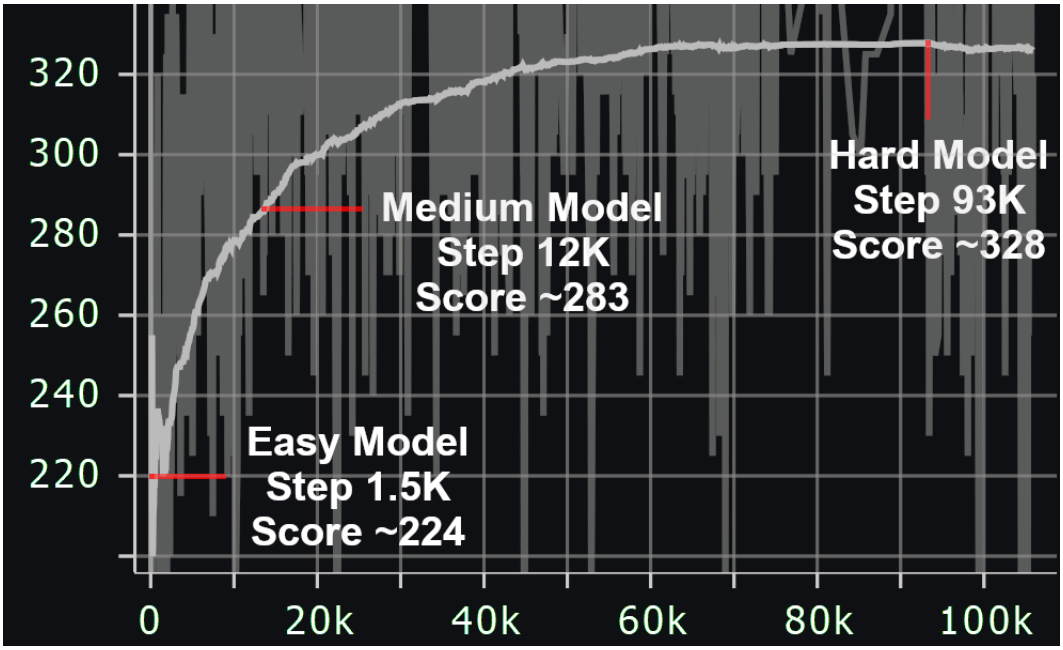

Figure 3.9: *Tensorboard graph of Version 3 agent's average score throughout training, marked with model extraction points*

each model play four games before calculating the average, minimising the impact caused by the randomness of resource and obstacle spawning.

Include agent evaluation diagrams for different runs (here or in results?)

Version 1

When running the evaluation level with the version 1 models, the following scores were achieved for the extracted models

Model	Round 1	Round 2	Round 3	Round 4	Average Score
Easy	340	460	365	340	376.25
Medium	410	425	450	370	413.75
Hard	415	450	420	490	443.75

The difference between models for this agent was promising, but it was decided to develop more versions in an attempt to achieve even better scores. This prompted the creation of versions 2 and 3 of the agent.

Version 3

The following scores were achieved when running the evaluation level with the extracted models for version 3 of our agent.

Model	Round 1	Round 2	Round 3	Round 4	Average Score
Easy	390	290	400	380	365
Medium	405	445	385	405	410
Hard	425	440	470	475	452.5

Due to the higher average score on the hard model, as well as the overall greater difference between the average scores of the three models compared to version 1, it was decided to use the models from this agent for the final difficulty settings.

3.4 Data Collection

It was decided to use a mixed-methods approach for this research. Quantitative data was extracted from gameplay sessions to analyse agent and player performance, as well as qualitative data from participants giving detailed opinions on their final player experience.

3.4.1 Quantitative Metrics

To understand the performance of the agent against a human player, we created a class named *DataLogger* to keep track of all important statistics regarding the game. After a five-minute game concluded, all statistics were saved as files in a custom path set from the main menu.

The files saved per game are the following:

- Player Log
- Enemy Log
- Game Summary
- Performance Summary

Player Log and Enemy Log

The player and enemy logs show a record of all noteworthy events taken by the player and enemy respectively during the game. Logged events are successful resource interaction, specifying the type of resource, and successful base depositing. Entries in the log are listed in the chronological order in which the event occurred, and even include a precise date and time of occurrence.

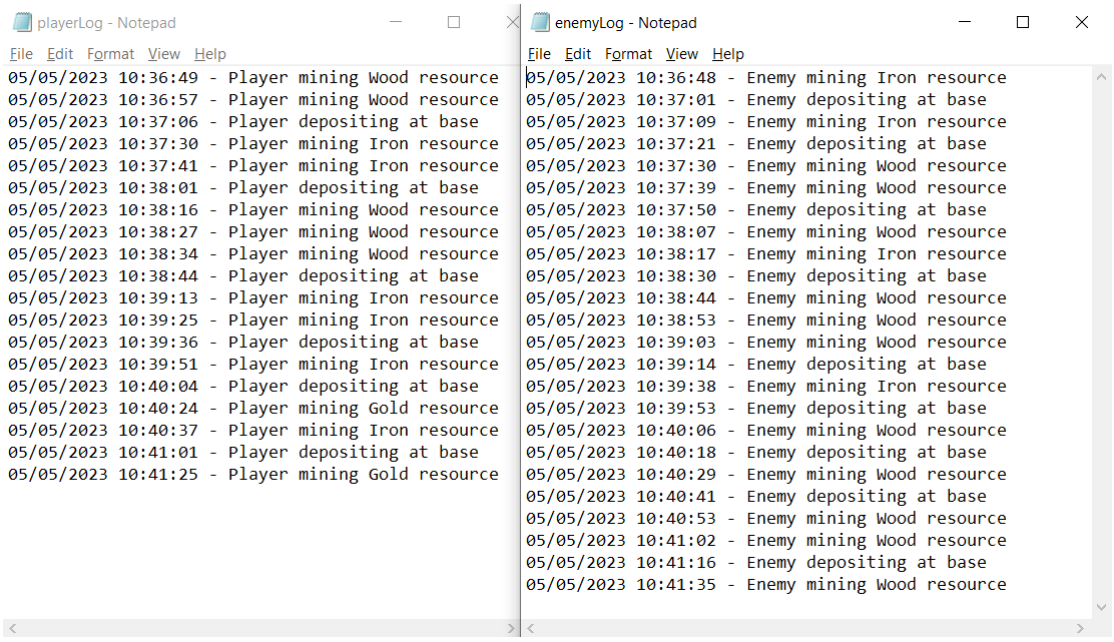

Figure 3.10: *Logs showing both player and enemy actions during the duration of a game*

Game Summary

The game summary shows all important numerical data about the performance of both the enemy and player during the game. This includes keeping track of how many times respectively the player and enemy carried out noteworthy events such as resource interactions and base depositing. It also saves the time in seconds they spent travelling, i.e not interacting with any object, as well as the final scores. Outside of numerical data, the game difficulty is additionally saved in this file.

	Player	Enemy
Wood resource interactions	5	11
Iron resource interactions	6	4
Gold resource interactions	2	0
Base deposit interactions	6	9
Time Spent Travelling (seconds)	140	95
Final score	385	380
Difficulty	Easy	

Figure 3.11: *Game Summary file showing important game information*

Performance Summary

The performance summary gives an indication of the computer's performance when running the game by keeping track of relevant metrics during playtime. These metrics are the CPU usage in percentage as well as total memory used by the game in megabytes. These are saved every second until the end of the game.

Elaborate on how this data will be processed for evaluation

Time (seconds)	Cpu Usage (%)	Memory Usage (MB)
1	21.48	77
2	7.62	78
3	14.26	78
4	31.45	79
5	19.53	79
6	9.18	79
7	11.33	80
8	12.7	79
9	10.35	79
10	9.77	79

Figure 3.12: *Performance Summary file showing computer performance metrics in the first ten seconds of a game*

3.4.2 Testing Session and Qualitative Data

To gather data, a testing session was hosted with nine volunteering participants. All participants were given a letter of information detailing the aims of the study, what they were expected to do, and how their data would be used. All participants signed a consent form agreeing to partake in the study.

Each participant was assigned a computer already loaded with the prototype game. They were given between twenty and thirty minutes to play the prototype, being asked to read the tutorial before starting their first game. Participants were also asked to play through each difficulty setting at least once, but were free to do so in whatever order they pleased. They were also free to replay a difficulty setting within their allocated time if they so wished.

All games played during the testing session generated the previously described files with the quantitative metrics. Directly after the allocated play-time, the researcher collected these files from every computer.

Afterwards, the participants were invited to attend a focus group to the discuss their experience playing the prototype. Discussion was guided by questions asked by the researcher regarding things like their player experience and their perception of the game's difficulty, but participants were allowed to speak freely and discuss amongst themselves. All discussion was recorded and transcribed for use in this study.

Chapter 4: Analysis of Results and Discussion

4.1 Introduction

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 Final Scores

A breakdown of scores per game on each difficulty level, comparing between them, followed by opinions of players about the differences between difficulty levels.

4.2.1 Easy

4.2.2 Medium

4.2.3 Hard

4.2.4 Player Perception of Difficulty

4.3 Strategies During Gameplay

4.3.1 Agent strategies

An analysis of agent strategies between difficulty levels

4.3.2 Player strategies

An analysis of how player strategies/performance changed throughout play time, elaborating on DDA angle.

4.4 System Performance

From our extracted data on system performance, we were able to study the changes in CPU and Memory usage per each difficulty during the game duration.

4.4.1 CPU Usage

Calculating the average CPU usage of every participant per difficulty, it was found that CPU Usage did not see any considerable changes throughout game time.

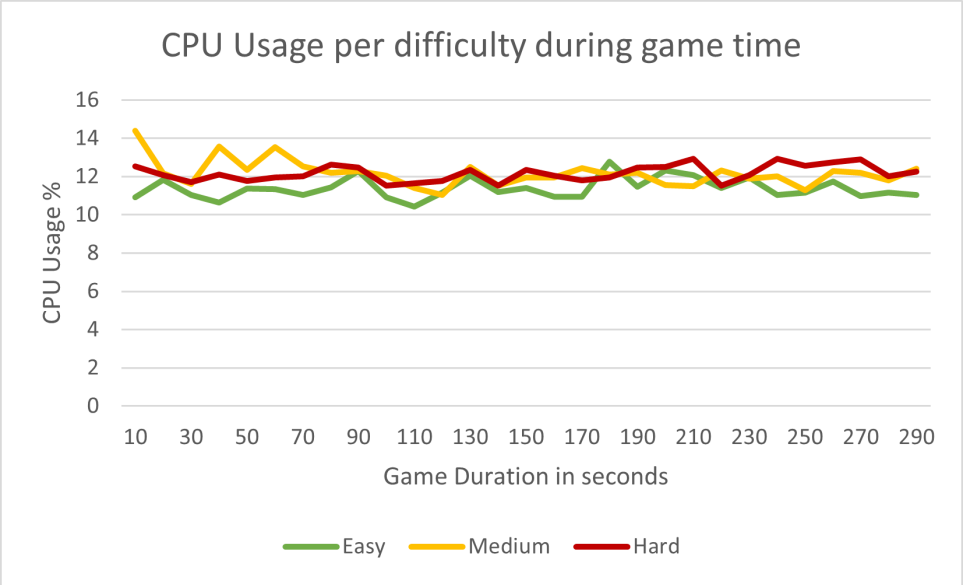

Figure 4.1: *Average CPU Usage during game time*

Additionally, there were no distinct differences in CPU usage between difficulties. Averaging all values throughout game time per difficulty, the following result was achieved.

Easy	**Medium**	**Hard**
11.44%	12.18%	12.15%

At most, the difference between these values is 0.74%, which can be considered negligible.

4.4.2 Memory Usage

Analysing average memory usage per difficulty, it was found that the all difficulties had a gradual increase in memory usage throughout game time. This increase, however, proved to be negligible. The difference was, at most, 1.73MB on the Easy difficulty.

Calculating the average memory usage throughout game time, the following result was achieved.

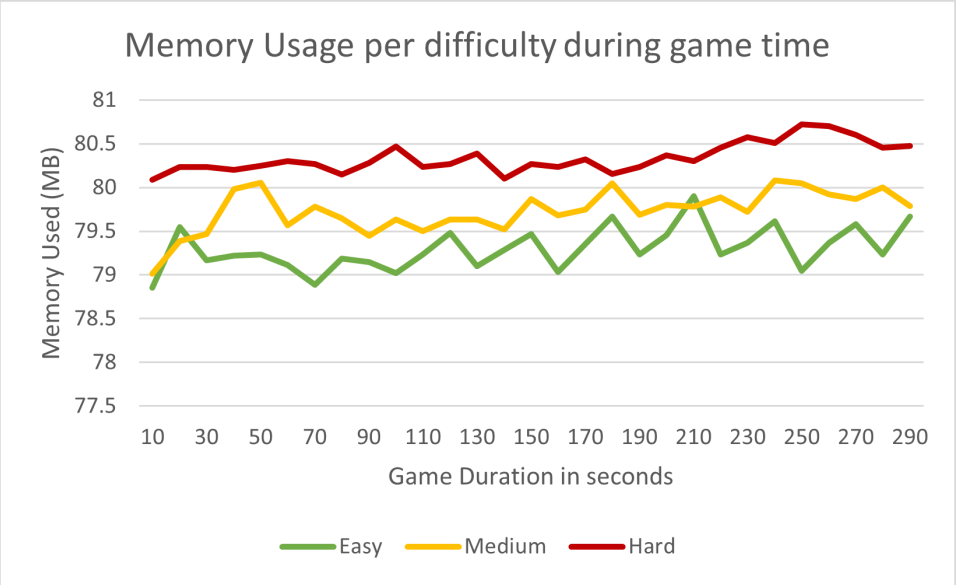

Figure 4.2: *Average Memory Usage during game time*

Easy	**Medium**	**Hard**
79.26 MB	79.7 MB	80.32 MB

The biggest difference was between the easy and hard difficulties at 1.06 MB. This difference, however, can also be viewed as negligible.

Overall, these results indicate that there is no significant difference in system performance between models of different training duration.

4.5 Flaws and Limitations

Elaborating on errors and flaws found by players during testing, as well as limitations of current implementation

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] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper-Perennial, 1991.

[2] K Fathoni, R Y Hakkun, and H A T Nurhadi. Finite state machines for building believable non-playable character in the game of khalid ibn al-walid. *Journal of Physics: Conference Series*, 1577(1):012018, Jul 2020.

[3] Richard S Sutton and Andrew Barto. *Reinforcement learning : an introduction*. The Mit Press, 1998.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[5] Leslie Lamport. *LaTeX: a Document Preparation System*. Addison Wesley, Massachusetts, 2 edition, 1994.

[6] Judeth Oden Choi, Jodi Forlizzi, Michael Christel, Rachel Moeller, MacKenzie Bates, and Jessica Hammer. Playtesting with a purpose. *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, Oct 2016.

[7] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology - ACE '05*. ACM Press, 2005.

[8] Joy James Prabhu Arulraj. Adaptive agent generation using machine learning for dynamic difficulty adjustment. *2010 International Conference on Computer and Communication Technology (ICCCT)*, Sep 2010.

[9] Anubhav Anand and Ajay Kumar. The rise of artificial intelligence in video games. *International Research Journal of Modernization in Engineering Technology and Science*, 4(7):2768–2771, Jul 2022.

[10] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018.

[11] Qiyue Yin, Jun Yang, Kaiqi Huang, Meijing Zhao, Wancheng Ni, Bin Liang, Yan Huang, Shu Wu, and Liang Wang. Ai in human-computer gaming: Techniques, challenges and opportunities. *arXiv:2111.07631 [cs]*, 14(8), Aug 2022.

[12] Steve Rabin. *Introduction to game development*. Course Technology Cengage Learning, 2010.

[13] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation*. Pearson, 2014.

[14] Mat Buckland. *AI game programming by example*. Wordware ; Lancaster, 2004.

[15] Risto Miikkulainen, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. Computational intelligence in games. In Gary Y. Yen and David B. Fogel, editors, *Computational Intelligence:*

Principles and Practice. IEEE Computational Intelligence Society, Piscataway, NJ, 2006.

[16] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[18] Amitha Mathew, P. Amudha, and S. Sivakumari. Deep learning techniques: An overview. In Aboul Ella Hassanien, Roheet Bhatnagar, and Ashraf Darwish, editors, *Advanced Machine Learning Technologies and Applications*, pages 599–608, Singapore, 2021. Springer Singapore.

[19] Kevin Gurney. *An introduction to neural networks*. Ucl Press, Cop, London, 1997.

[20] Lei Chen. *Deep Learning and Practice with MindSpore*. Springer Singapore, 2021.

[21] Nicole Rusk. Deep learning. *Nature America*, 13(1):35, Jan 2016.

[22] Le Lyu, Yang Shen, and Sicheng Zhang. The advance of reinforcement learning and deep reinforcement learning. *2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, Feb 2022.

[23] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, nov 2017.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[25] Vlad Firoiu, William F. Whitney, and Joshua B. Tenenbaum. Beating the world's best at super smash bros. with deep reinforcement learning. *CoRR*, abs/1702.06230, 2017.

[26] Inseok Oh, Seungeun Rho, Sangbin Moon, Seongho Son, Hyoil Lee, and Jinyun Chung. Creating pro-level ai for a real-time fighting game using deep reinforcement learning. *IEEE Transactions on Games*, 14(2):212–220, 2022.

[27] Anthony DiGiovanni and Ethan C. Zell. Survey of self-play in reinforcement learning, 2021.

[28] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[29] John K Haas. A history of the unity game engine. 2014.

[30] Maxwell Foxman. United we stand: Platforms, tools and innovation with the unity game engine. *Social Media + Society*, 5(4):205630511988017, oct 2019.

[31] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[32] Adi Aiman Bin Ramlan, Azliza Mohd Ali, Nurzeatul Hamimah Abdul Hamid, and Rozianawaty 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)*, pages 96–100, 2021.

[33] Nirmal Baby and Bhargavi Goswami. Implementing artificial intelligence agent within connect 4 using unity3d and machine learning concepts. *International Journal of Recent Technology and Engineering (IJRTE)*, 7(6S3):193–200, Apr 2019.

[34] Muhammad Iftekher Chowdhury and Michael Katchabaw. Bringing auto dynamic difficulty to commercial games: A reusable design pattern based approach. In *Proceedings of CGAMES'2013 USA*, pages 103–110, 2013.

[35] Mohammad Zohaib. Dynamic difficulty adjustment (dda) in computer games: A review. *Advances in Human-Computer Interaction*, 2018:1–12, Nov 2018.

[36] Su Xue, Meng Wu, John Kolen, Navid Aghdaie, and Kazi A. Zaman. Dynamic difficulty adjustment for maximized engagement in digital games. In

Proceedings of the 26th International Conference on World Wide Web Companion, WWW '17 Companion, page 465–471, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[37] Johan Hagelback and Stefan J. Johansson. Measuring player experience on runtime dynamic difficulty scaling in an rts game. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 46–52, 2009.

[38] Robin Hunicke and Vernell Chapman. Ai for dynamic difficulty adjustment in games. *Challenges in game artificial intelligence AAAI workshop*, 2, 01 2004.

[39] M.I. Chowdhury and M. Katchabaw. Improving software quality through design patterns: A case study of adaptive games and auto dynamic difficulty. *13th International Conference on Intelligent Games and Simulation, GAME-ON 2012*, pages 41–47, 01 2012.

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)
```