

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Adaptivity in Single Player Video Games

João Augusto dos Santos Lima



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Jacob

Co-Supervisor: Zafeiris Kokkinogenis

July 23, 2021



# **Adaptivity in Single Player Video Games**

**João Augusto dos Santos Lima**

Mestrado Integrado em Engenharia Informática e Computação

July 23, 2021



# Abstract

Players typically play video games to have fun and enjoy the moments they create during the gaming session. Each game gives the player a unique gaming experience, differentiating them from an ever-growing video game market. Additionally, each player has a specific genre of games that they enjoy. Even in the same game, different players have distinct objectives of what they seek in the gameplay.

Furthermore, games try to compensate for the discrepancy of skill found between players by allowing them to choose a level of difficulty. However, this method depends on the player's ability to self judge and self assign themselves to a difficulty on a game they may never have played before. Consequently, this assignment may lead to a poor gaming experience, not allowing the player to enjoy all the creative content available in the game.

This dissertation tries to present a machine learning approach to game adaptivity in order to maximize the player's gaming experience. Firstly, to solve this problem, a player simulation needs to be created to have sufficient data points to train the reinforcement learning adaptivity system. Then, with the adaptivity system, the adapted game will change its content depending on the playing user to improve the gaming experience and game flow.

Although the player simulation final results did not achieve the desired results for it to be incorporated in the adaptivity system, it showed potential for future exploration in this topic. Furthermore, the adaptivity system successfully altered the game elements depending on the user, which delivered a different experience from the original game. The game adaptivity system can also be adapted to different games and gaming elements to allow all games to deliver their created experience.

**Keywords:** Video Game, Game Adaptivity, Machine Learning, Player Profiling, Player Type, Adapted Game, Game Flow, Reinforcement Learning

**Category:** Human-centered computing, Human-computer interaction (HCI), Interaction design, Computing methodologies, Machine Learning



# Resumo

Os jogadores normalmente jogam video jogos para se divertirem e aproveitar os momentos que criam durante a sessão de jogo. Cada jogo oferece ao jogador uma experiência de jogo única, diferenciando-o de um mercado de video jogos em constante crescimento. Além disso, cada jogador tem um gênero específico de jogos de que gosta. No mesmo jogo, diferentes jogadores têm objetivos distintos que procuram durante o jogo.

Além disso, os jogos tentam compensar a discrepância de perícia encontrada entre os jogadores, permitindo que eles escolham um nível de dificuldade. No entanto, este método depende da capacidade do jogador de se auto-avaliar e se auto-atribuir a uma dificuldade num jogo que talvez nunca tenha jogado antes. Consequentemente, esta atribuição pode levar a uma experiência de jogo sub-ótima, não permitindo que o jogador desfrute de todo o conteúdo criativo disponível no jogo.

Esta dissertação tenta apresentar uma abordagem de machine learning para a adaptabilidade do jogo, com o fim de maximizar a experiência de jogo do jogador. Primeiramente, para resolver este problema, uma simulação de jogadores precisa ser criada para ter dados suficientes para treinar o sistema de adaptabilidade de reinforcement learning. Depois, com o sistema de adaptatividade, o jogo adaptado mudará seu conteúdo dependendo do usuário que está jogando para melhorar a experiência de jogo e o fluxo do jogo.

Embora os resultados finais da simulação do player não tenham alcançado os resultados desejados para que fosse incorporada com o sistema de adaptatividade, a simulação mostrou potencial para exploração futura neste tópico. Além disso, o sistema de adaptabilidade alterou com sucesso os elementos do jogo dependendo do usuário, o que proporcionou uma experiência diferente do jogo original. O sistema de adaptação do jogo também pode ser adaptado a diferentes jogos e elementos de jogo para permitir que todos os jogadores obtenham a experiência criada.

**Keywords:** Video Game, Game Adaptivity, Machine Learning, Player Profiling, Player Type, Adapted Game, Game Flow, Reinforcement Learning

**Category:** Human-centered computing, Human-computer interaction (HCI), Interaction design, Computing methodologies, Machine Learning





# Acknowledgements

I would like to thank everyone that had the patience to heard my rambles and talk about the whole dissertation process, even during these challenging times. A special thanks to my family and girlfriend, who had the most patience in the world and gave their time to help and support me. I also have to thank my supervisors for helping me develop and achieve a good project for this chapter of my studies.

João Lima



*“The way I see it, every life is a pile of good things and bad things.  
The good things don’t always soften the bad things, but vice versa, the bad things don’t always  
spoil the good things and make them unimportant.”*

11th Doctor by Sydney Newman, C. E. Webber and Donald Wilson



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context and Motivation . . . . .	2
1.2	Objectives . . . . .	3
1.3	Implemented Solution . . . . .	3
1.4	Contributions . . . . .	4
1.5	Document Structure . . . . .	4
<b>2</b>	<b>State of the Art Review</b>	<b>7</b>
2.1	Game adaptivity . . . . .	7
2.1.1	Background . . . . .	7
2.1.2	Player types . . . . .	10
2.1.3	Game elements . . . . .	12
2.1.4	Player profiling . . . . .	14
2.1.5	Game Flow . . . . .	15
2.1.6	Applications . . . . .	16
2.1.7	Summary . . . . .	17
2.2	Machine learning . . . . .	18
2.2.1	Supervised Learning . . . . .	21
2.2.2	Unsupervised Learning . . . . .	22
2.2.3	Reinforcement Learning . . . . .	23
2.2.4	Imitation Learning and Inverse Reinforcement Learning . . . . .	25
2.2.5	Player simulation . . . . .	26
2.2.6	Curriculum Learning . . . . .	27
2.2.7	Online Learning . . . . .	27
2.2.8	Summary . . . . .	28
<b>3</b>	<b>Reinforcement learning based approach for game adaptivity</b>	<b>29</b>
3.1	Preliminary work . . . . .	29
3.2	General adaptivity system methodology . . . . .	30
3.3	Selected Game . . . . .	31
3.4	Game modification . . . . .	37
3.4.1	Life System . . . . .	37
3.4.2	Time system . . . . .	40
3.4.3	Additional modifications . . . . .	40
3.4.4	Summary . . . . .	41
3.5	Player simulation . . . . .	41
3.6	Adaptivity System . . . . .	44
3.7	Summary . . . . .	47

<b>4</b>	<b>Development and workflow of game adaptivity system</b>	<b>49</b>
4.1	Life system . . . . .	49
4.2	Time system . . . . .	50
4.3	Player simulation . . . . .	51
4.4	Adaptivity system . . . . .	59
4.4.1	Personality system overview . . . . .	60
4.4.2	Adaptivity system overview . . . . .	67
4.5	Summary . . . . .	73
<b>5</b>	<b>Results and evaluation</b>	<b>75</b>
5.1	Player simulation . . . . .	75
5.1.1	Experiments performed . . . . .	79
5.1.2	Experiments overall results and analysis . . . . .	143
5.2	Adaptivity system . . . . .	147
5.2.1	Adaptivity agent results . . . . .	147
5.2.2	Adaptivity agent overall results and analysis . . . . .	155
5.2.3	Adapted game results . . . . .	155
5.3	Summary . . . . .	162
<b>6</b>	<b>Conclusions</b>	<b>169</b>
6.1	Limitations . . . . .	170
6.2	Future work . . . . .	170
<b>A</b>	<b>Personalities</b>	<b>173</b>
<b>B</b>	<b>Questionnaire</b>	<b>185</b>
<b>C</b>	<b>Player simulation results</b>	<b>191</b>
<b>D</b>	<b>Adaptivity agent results</b>	<b>193</b>
<b>E</b>	<b>Additional questionnaire results</b>	<b>195</b>
	<b>References</b>	<b>201</b>

# List of Figures

2.1	Platform game Celeste [44]	8
2.2	HEXAD scale [67]	12
2.3	Choice during the game Heavy Rain [54]	14
2.4	Machine learning areas. Source: Big data and machine learning for Businesses by Abdul Wahid [73]	20
2.5	A Taxonomy of RL Algorithms by [1]	25
3.1	Gameplay image with two stationary saws	33
3.2	Gameplay image with a stationary spike	33
3.3	Gameplay image with falling enemies	34
3.4	Gameplay image with several moving saws	34
3.5	Gameplay image that requires a jump over a hole of water	35
3.6	Gameplay image in a platform level	35
3.7	Sections 0-8 and starting section	38
3.8	Sections 9-15	39
3.9	Movement values in Unity inspector	41
3.10	GAIL suggested network architecture	43
3.11	Expert demonstration data collection architecture suggestion	44
3.12	Adaptivity system proposed structure	47
4.1	Examples of the heart display	50
4.2	Unity inspector section restart position values	50
4.3	Timer values in Unity inspector	50
4.4	Timer display during gameplay	51
4.5	Player camera gameplay vision	53
4.6	Image Synthesis package script	53
4.7	Final colour segmented image example	54
4.8	Camera sensor script for agent's object	54
4.9	Code for the architecture of the network for GAIL	55
4.10	Demonstration recorder script for player agent's object	56
4.11	Ray cast script values in Unity editor	57
4.12	Ray cast vision during gameplay	58
4.13	New starting section	59
4.14	Starting section collision blocks	60
4.15	Unity personality scriptable object example	61
4.16	Unity personality scriptable object coins minimum and maximum value example	62
4.17	Unity personality scriptable object type of enemy example	63
4.18	Unity personality scriptable concentration values example	64

4.19	Special section . . . . .	69
4.20	Display of the type of round during gameplay . . . . .	71
5.1	Example of the training <i>.yaml</i> file . . . . .	78
5.2	Player simulation experiment 1 results . . . . .	81
5.3	Player simulation experiment 2 results . . . . .	82
5.4	Player simulation experiment 3 results . . . . .	84
5.5	Player simulation experiment 4 results . . . . .	86
5.6	Player simulation experiment 5 results . . . . .	87
5.7	Player simulation experiment 6 results . . . . .	90
5.8	Player simulation experiment 7 results . . . . .	91
5.9	Player simulation experiment 8 results . . . . .	93
5.10	Player simulation experiment 9 results . . . . .	95
5.11	Player simulation experiment 10 results . . . . .	97
5.12	Player simulation experiment 11 results . . . . .	98
5.13	Player simulation experiment 12 results . . . . .	101
5.14	Player simulation experiment 13 results . . . . .	103
5.15	Player simulation experiment 14 results . . . . .	104
5.16	Player simulation experiment 15 results . . . . .	106
5.17	Player simulation experiment 16 results . . . . .	108
5.18	Player simulation experiment 17 results . . . . .	109
5.19	Player simulation experiment 18 results . . . . .	112
5.20	Player simulation experiment 19 results . . . . .	115
5.21	Player simulation experiment 20 results . . . . .	116
5.22	Player simulation experiment 21 results . . . . .	118
5.23	Player simulation experiment 22 results . . . . .	120
5.24	Player simulation experiment 23 results . . . . .	122
5.25	Player simulation experiment 24 results . . . . .	124
5.26	Player simulation experiment 25 results . . . . .	127
5.27	Player simulation experiment 26 results . . . . .	128
5.28	Player simulation experiment 27 results . . . . .	130
5.29	Player simulation experiment 28 results . . . . .	133
5.30	Player simulation experiment 29 results . . . . .	135
5.31	Player simulation experiment 30 results . . . . .	136
5.32	Player simulation experiment 32 results . . . . .	138
5.33	Player simulation experiment 33 results . . . . .	140
5.34	Player simulation experiment 34 results . . . . .	142
5.35	Player simulation experiment 35 results . . . . .	144
5.36	Adaptivity agent experiment 1 results . . . . .	149
5.37	Adaptivity agent experiment 2 results . . . . .	150
5.38	Adaptivity agent experiment 3 results . . . . .	152
5.39	Adaptivity agent experiment 4 results . . . . .	153
5.40	Adaptivity agent experiment 5 results . . . . .	154
5.41	Adapted vs non-adapted section shown . . . . .	158
5.42	Adaptivity agent experiment 5 results . . . . .	159
5.43	Difference between non-adapted and adapted version for each section . . . . .	160
5.44	Total time spent in each section . . . . .	160
5.45	Jumps in each section . . . . .	161
5.46	Average velocity in each section . . . . .	161



5.47	Difference between non-adapted and adapted version for each game . . . . .	162
5.48	Captured coins at the end of a game . . . . .	162
5.49	Total time spent playing one game . . . . .	163
5.50	End score of each game . . . . .	163
5.51	Average velocity in each game . . . . .	164
5.52	Correctly identify the adaptivity round . . . . .	164
5.53	Average GEQ Core module component scores . . . . .	165
5.54	Adapted vs non-adapted negative affect . . . . .	165
5.55	Adapted vs non-adapted positive affect . . . . .	166
5.56	Average questionnaire response . . . . .	167
B.1	Questionnaire . . . . .	186
B.2	Questionnaire . . . . .	187
B.3	Questionnaire . . . . .	188
B.4	Questionnaire . . . . .	189
B.5	Questionnaire . . . . .	190
E.1	Adapted vs non-adapted challenge . . . . .	195
E.2	Adapted vs non-adapted competence . . . . .	196
E.3	Adapted vs non-adapted flow . . . . .	196
E.4	Adapted vs non-adapted sensory and imaginative immersion . . . . .	197
E.5	Adapted vs non-adapted tension/annoyance . . . . .	197
E.6	Adapted game questionnaire responses . . . . .	198
E.7	Non-adapted game questionnaire responses . . . . .	199
E.8	Gender of the questionnaire responses . . . . .	199



# List of Tables

2.1	Comparison of applications of adaptivity . . . . .	18
2.2	Player simulation examples . . . . .	27
4.1	Section's coins and chests . . . . .	70
4.2	Section's enemies . . . . .	71
4.3	Section's completions times in seconds . . . . .	72
4.4	Section's completions average speeds in <i>Unity units</i> . . . . .	73
4.5	Section's completions jumps . . . . .	74
5.1	Player simulation experiment 1 configuration . . . . .	79
5.2	Player simulation experiment 2 configuration . . . . .	80
5.3	Player simulation experiment 3 configuration . . . . .	83
5.4	Player simulation experiment 4 configuration . . . . .	85
5.5	Player simulation experiment 5 configuration . . . . .	88
5.6	Player simulation experiment 6 configuration . . . . .	89
5.7	Player simulation experiment 7 configuration . . . . .	89
5.8	Player simulation experiment 8 configuration . . . . .	92
5.9	Player simulation experiment 9 configuration . . . . .	94
5.10	Player simulation experiment 10 configuration . . . . .	96
5.11	Player simulation experiment 11 configuration . . . . .	99
5.12	Player simulation experiment 12 configuration . . . . .	100
5.13	Player simulation experiment 13 configuration . . . . .	100
5.14	Player simulation experiment 14 configuration . . . . .	102
5.15	Player simulation experiment 15 configuration . . . . .	105
5.16	Player simulation experiment 16 configuration . . . . .	107
5.17	Player simulation experiment 17 configuration . . . . .	110
5.18	Player simulation experiment 18 configuration . . . . .	111
5.19	Player simulation experiment 19 configuration . . . . .	113
5.20	Player simulation experiment 20 configuration . . . . .	114
5.21	Player simulation experiment 21 configuration . . . . .	117
5.22	Player simulation experiment 22 configuration . . . . .	119
5.23	Player simulation experiment 23 configuration . . . . .	121
5.24	Player simulation experiment 24 configuration . . . . .	123
5.25	Player simulation experiment 25 configuration . . . . .	125
5.26	Player simulation experiment 26 configuration . . . . .	126
5.27	Player simulation experiment 27 configuration . . . . .	129
5.28	Player simulation experiment 28 configuration . . . . .	131
5.29	Player simulation experiment 29 configuration . . . . .	132

5.30	Player simulation experiment 30 configuration . . . . .	134
5.31	Player simulation experiment 32 configuration . . . . .	137
5.32	Player simulation experiment 33 configuration . . . . .	139
5.33	Player simulation experiment 34 configuration . . . . .	141
5.34	Player simulation experiment 35 configuration . . . . .	143
5.35	Adaptivity agent PPO configuration . . . . .	147
5.36	Adaptivity agent SAC configuration . . . . .	148
A.1	Personality 1 - Experienced player that wants to collect coins . . . . .	174
A.2	Personality 2 - Experienced player that just wants to go fast . . . . .	175
A.3	Personality 3 - Casual player that wants to collect coins . . . . .	176
A.4	Personality 4 - Casual player that wants to go fast . . . . .	177
A.5	Personality 5 - Casual player that plays with care . . . . .	178
A.6	Personality 6 - Casual player that has trouble with moving enemies . . . . .	179
A.7	Personality 7 - Casual player that has trouble with platforming sections . . . . .	180
A.8	Personality 8 - Inexperienced user playing the game for the first time . . . . .	181
A.9	Personality 9 - Inexperienced player that plays the game with care . . . . .	182
A.10	Personality 10 - Inexperienced hyperactive player . . . . .	183

# Abbreviations

ABWPL	Adjusted Bounded Weighted Policy Learner
AI	Artificial Intelligence
A3C	Asynchronous Advantage Actor-Critic
CNN	Convolutional Neural Networks
CT	Computational Thinking
DDA	Dynamic Difficulty Adjustment
GAIL	Generative Adversarial Imitation Learning
GEQ	Game Experience Questionnaire
IMPALA	Importance Weighted Actor-Learner Architecture
LSTM	Long Short-Term Memory
NPC	Non-playable character
PPO	Proximal Policy Optimization
RL	Reinforcement learning
SAC	Soft actor-critic
SVM	Support Vector Machines



# Chapter 1

## Introduction

Video games are one of the many ways that we can get entertainment. Although physical games provide entertainment, they require much time to set up and a lot more time playing, creating long playthroughs that most players do not have time for. Digital games are an advantage to people that prefer a more digital world and faster play. Video games can be played anywhere and anytime with most of our daily tools, like a personal computer or a personal phone. They deliver a unique experience by creating environments and aesthetics that can only be achieved on synthetic ambients. Also, video games provide the user with more options for interaction with the digital environment, making the world richer.

Entertainment is not all we can achieve by playing video games. Some games have the objective to teach users using different techniques. These different types divide video games into categories, educational and entertainment. However, the entertainment branch is an ever-growing industry of games. According to game statistics [43], the game industry was a 17.68 billion dollars industry in 2016. By 2021, it is expected that the number of players will come close to 2.8 billion players. Although the game industry is more focused on entertainment, most games feature both fun and learning elements.

Furthermore, with vast amounts of players, the market is also filled with different games, each with unique gameplay aspects. Depending on the gameplay, games can be categorised differently, for example, strategy, casual, action games and much more depending on the game's intention. Even if different games are classified in the same genre, they still can provide different experiences and features, making them stand out from each other. Each game is distinct because they have different visual elements, gameplay features and mechanics and many other details that create a unique gaming experience.

All of these varieties make players prefer a specific game genre, depending on their character and personality. Users may search for games that offer a more intense experience, with extensive interactions and constant action. Others prefer gameplay that is more strategic and slower to give them time to think and elaborate a well-fought plan. However, not everyone has the same skill and

dexterity while playing. Different players have different abilities to play, even if it is a game they enjoy. Moreover, to allow every player to have a satisfying and rewarding experience, they must play at a suitable skill level to provide a comfortable experience.

## 1.1 Context and Motivation

Games should deliver their optimal gameplay experience to any player, ensuring that everyone can get an optimal game experience. One way of guaranteeing that all players can have an appropriate challenge is by allowing the choice of game difficulty. Although this approach is not ideal since each player is self-evaluating, this method is widely used within the gaming industry. Nevertheless, games should try to adapt their content to fit the user playstyle and skill level. The adaptivity can create more extended playtime and recurring playthroughs by the same player. Furthermore, it can also bring new players to the game by broadening their content to as many players as possible. In this context, adaptivity should be considered during the development phase and even in already implemented games. It can help create a more popular game by pleasing a good portion of the players while also helping with the game's longevity. Additionally, it can create a more stable player base with more dedicated players and help the game distinguish itself from the competition by being more player-focused.

An adaptivity system can make associations between the game content and the player's personality to typically maximise the game flow, consequently also solving the correct association between the game difficulty level and the player skill's level. For this to work, the system must first profile the player using machine learning techniques. It will distinguish users across various skill levels and personal preference on the game objectives and measure different gameplay strategies and patterns applied during gameplay. After this, we can alter different game aspects to fit the diverse player base. These game aspects can range from game mechanics to the environment the player can interact with.

Game adaptivity is not exclusive to the objective of maximising the game flow. A game developer can use game adaptivity to what fits their ultimate goal. An example of a different use for adaptivity is in serious games. The goal may vary depending on the game's objective, but some serious games focused on education aim to maximise the students' learning. The user's profiling is different, in this case, since educational games want to measure how much the player is learning instead of how much he is having fun. The content to be adapted also differs since the primary objective is to immerse the user in a learning environment.

Adaptivity in video games is an engaging topic in the ever-growing current industry, where new techniques and implementations are still being investigated and improved. Its primary focus is to improve the user's experience by delivering a self-adaptive product.



## 1.2 Objectives

The following objectives were created to guide the research work and the creation of the proposed system:

- Research the field of Game Adaptivity, find the most used types of algorithms, and what type of content can be altered to satisfy the player.
- Research the key elements (player actions and performance metrics) to estimate the player's personality.
- Research state of the art machine learning algorithms that can be used to solve game adaptivity and player simulation.
- Design a machine learning framework of game adaptivity that can be used in most single player games.
- Develop the framework using a specific game and evaluate the results delivered by the prototype.

All of these objectives will help guide the research and development phase of this project. In the first phase, state of the art research should give more context to the topic and the problem. This phase will help define the architecture and the prototype of the adaptivity system to be developed and implemented in the second stage. The third phase is relevant to the analysis of the result delivered by the prototype created and the evaluation of the adaptivity to prove that the system can help improve the gaming experience. If the results are not satisfying or do not achieve the expected results, the prototype should go over the three phases and improve the previously designed solution using different approaches and algorithms.

## 1.3 Implemented Solution

As a result of the work done during this thesis, a prototype of an adaptivity system was created. This prototype was developed using an open-source game [11]. The game is an infinite runner, platform game, where the player can move his character to the left, right and jump. The game's main objective is to go as far as possible and collect as many coins as possible without losing all lives.

The implemented prototype game included some changes to the game to help with the adaptivity system and create an overall better experience by giving the player more play options. These changes include a three life system and a game timer. Moreover, these differences allow the player to create different tactics and strategies when approaching the obstacles. All the other game elements remain mainly unchanged, with some exceptions to challenges that were too difficult or too annoying to overcome.

The game is divided into distinct sections that represent different challenges. These sections are used in the adaptivity system, where the adaptivity agent selects the next section for the player

to engage. The objective of the adaptivity is to select the most indicated section depending on the previously seen gameplay of the same user. The section chosen aims to maximise the players' gameflow and the players' preferred objectives (go as far as possible and collect coins) during all the game sessions.

## 1.4 Contributions

The dissertation contributes by creating a prototype of an adaptivity system in a game. Although this system is specific to this game and this genre, it can be modified and used in other games. This prototype is also accompanied by a state of the art review of game adaptivity and machine learning methods to implement the adaptivity. The review of game adaptivity includes the elements that can be changed and how they affect each personality type to improve the game flow and the gaming experience. Furthermore, there is also a review of the state of the art machine learning methods to implement adaptivity.

Although many state of the art approaches to game adaptivity focus more on dynamic difficulty adjustment, this dissertation creates a approach by combining the concept of game flow and state of the art reinforcement learning algorithms to create an adaptivity system. This approach includes a prototype of the implementation of adaptivity and suggestions on how to implement and modify this work to any game. In addition, it also includes recommendations for further investigation and improvement.

Furthermore, this dissertation also includes some investigation and experimentation regarding the topic of player simulation for the creation of artificial players to train the adaptivity agent. Most of the state of the art regarding player simulation is more connected with a machine's ability to play the game to an equal or better level than a human to get to the end of the level. On the other hand, to train the adaptivity system, the player simulation needs to behave as closely as possible to the players' movements and actions, which this thesis explores.

Finally, the prototype user test results can give more insight into the usage of adaptivity in a game while also providing more information and conclusions on what needs to be changed and iterated in future works and implementations. Additionally, the results of the player simulation can also be used for further iteration and investigation on the topic.

## 1.5 Document Structure

In this current chapter (chapter 1), the topic of Game Adaptivity in Single Player Video Games is introduced. The following five chapters are divided into the state of the art review, proposed system architecture and planning of the prototype, development of the prototype, evaluation of the adaptivity system, and the work's conclusions.

In the chapter state of the art review (chapter 2), game adaptivity, applications, and other relevant topics will be explored, including player types, game elements, player profiling and gameflow.

Furthermore, the topic of machine learning and its application to solving the problem of adaptivity will also be reviewed, as well as some discussions on player simulation.

In the following chapter (chapter 3), a proposed solution to the problem will be presented. Additionally, it will include system architecture and requirements for the validation of the implemented solution. It will also contain more detail about the prototype's implementation and the steps that need development.

In the fourth chapter (chapter 4), the prototype developed will be described in more detail, including the technologies used, the game alterations made, and the adaptivity system implemented. This chapter will also include a detailed description of the player simulation, the technologies used and its implementation in the game context.

The next chapter (chapter 5) will present the evaluation system created inside the game and how the collection of the data was made. Next, the data collected will be presented and analysed, and conclusions will be made regarding the observations made during the data analysis to evaluate the capability of the adaptivity system.

The final chapter (chapter 6) will contain some conclusions about this dissertation work and proposals for future works and implementations to improve the adaptivity system prototype created.



## Chapter 2

# State of the Art Review

In this chapter, there will be an analysis of the various state of the art topics and preliminary work relevant to this project's development to fulfill the objective described in 1.2. Firstly, section 2.1 will explore the definition and purpose of games and game adaptivity. It will then explore the different player types and scales, what elements can be adapted in a game, how to profile the player, gameflow and its meaning to the player and game, and some application of adaptivity in recent works. Furthermore, in section 2.2 the topic of machine learning and its various applications to game adaptivity will be explored, with a more in-depth analysis of RL for its importance on the project's development.

### 2.1 Game adaptivity

In this section, several topics relating to game adaptivity will be explored. These topics include the background behind game adaptivity and how it is usually implemented in the industry; the player's types, and what scales exist to categorise the different players; the game elements that can be adapted and changed, and how do they affect the gameplay; player profiling techniques, what actions to observe and how to measure them; and the game flow, how can it be implemented in a game and how does it influence the player. Finally, some applications of game adaptivity are going to be explored.

#### 2.1.1 Background

Video games are a digital way to provide entertainment to the player. The user can interact with the computer using input devices, changing what they see in the output device. According to [58], games are mainly composed of skill, strategy, and randomness elements. Games of skill imply that the player needs to have the ability to overcome the problems. This skill can come from successive playthroughs in the game, acquiring knowledge of the game's specific mechanics that give him the advantage needed to solve the challenging problems. The game skill also depends on the user's

mental and physical state, for example, the ability to have proper reaction times. An example of a game that requires skill is Celeste[44] (Figure 2.1). This game is considered a platform game where the user must navigate the level using platforms to reach the end. The game is mainly a skill game since every jump and dash is time-based, and it requires a lot of skill and dexterity from the player. Games that contain strategy elements depend on the ability of the user to make decisions. These decisions significantly impact a specific event in a game or even in the game outcome. Although the decisions can be related to skill, the strategy is related to the user's ability to think and develop specific and even innovative ideas to solve a problem. Games that include strategy elements include strategy games, puzzle games and other more specific categories. Strategy games include games where the user needs to solve several problems to satisfy a game condition to win. Puzzle games are more inclined to solve only one puzzle; however, they include several levels to solve, each with increasing difficulty. An example of a strategy game is StarCraft [13], which is considered a real-time strategy game, where the user needs to manage resources, create an army and defeat the enemy. Games that include randomness or chance elements are games where the game's outcome depends on random factors that the user has no control over. This type of uncontrollable outcome can impact the player's enjoyment of the game[65]. [58]



Figure 2.1: Platform game Celeste [44]

In general, games should not be so easy for the user that they may cause boredom, but also they should not be too challenging to cause stress and overload the player [58]. However, some games are specially made to be challenging [19], specialising in a type of player that enjoys being able to complete a more complicated scenario [21], while alienating other players that might find the difficulty too overwhelming. This difference is mainly caused because each player has different preferences to what difficulty motivates them more [21]. The main focus of entertainment games is to provide the player with the sensation of fun[41]. Like frustration and boredom, some metrics

can be used to measure an approximated level of the fun the player currently has. Frustration can be associated with more challenging levels, where the player feels like this is an unfair challenge to him. Boredom can be associated with lighter levels where the user feels that the task given does not stimulate his skill over the game. However, both metrics can also be associated with the opposite level of skill. Frustration can also occur with lower skill levels, where the user feels like he has to do this task to move forward through the game. On the other hand, boredom can be associated with more skill demanding levels, where the user feels like the task in hand may be too time-consuming or implying much work to be completed, even if it represents a more significant challenge.

One way to tackle different difficulties is to allow the player to change the skill level themselves. The option to choose the difficulty can happen at the start of the game or even during gameplay. One easy way to divide the player's skill level would be to introduce three levels: beginner, average skill, and expert skill [45]. These levels allow the player to choose what they think is the most appropriate difficulty to fit their current skill. A player can learn game mechanics more easily and evolve their skill if placed in their correct play level. However, the learning process changes the player's skill, needing a change in the game skill levels [45]. Nevertheless, there only exist three levels that the user can choose in this case, making it hard to distinguish the differences between the enumerated skills. Moreover, even if the game offered more options to choose from, the gameplay would feel disconnected from the user since he would regularly be wasting time changing his skill level to match his preference instead of playing the game [17]. Also, there is a possibility of a wrong choice of the level. If the challenge is too high and the task too complicated, the player would feel frustrated, making him quit the game and choose a more rewarding task. On the contrary, if the challenge is too easy, the user may not learn, causing boredom [45]. Most of the information used to design the different levels is limited to market research and pre-production playtesting, pre-release usability and post-release maintenance. This limitation can sometimes be caused due to specific demographics, making the design not player distinct.

Game adaptivity is the game's ability to adapt itself to fit the players, their type and their individual needs[41]. Dynamic difficulty adjustment, a subsection of game adaptivity, tries to adapt the game difficulty to fit the player's performance, creating a balance between the player's ability and the challenge proposed by the game[58]. One example of adjusting the difficulty is in the game of Mario Kart [51]. This game's AI, usually called Rubber Band AI, makes the game's opponent stay close to the user. If the player is too far ahead of everyone, the adversary will gain increased speed to catch up to the player. On the contrary, if the player falls too far behind, the opponent will slow down to give the player a chance. Furthermore, there also exists an item system in the game. If the player is in one of the last positions of the race, the items it will receive are of higher quality. However, if he is in the first three positions, the player's items are lower tier. This example is one type of dynamically adjusting the game difficulty without the player having to specify any settings. This automatic change allows the player to avoid feeling stressed or bored when faced with different tasks [52].

For this adaptation to happen, the program needs to know the skill demonstrated by the player.

This information can come from recorded data before the game happens, in different gameplay sessions or, if possible, in different games. It will then allow the game to perform adaptivity during the loading time. If the objective is to change the game during the gameplay, real-time information about the player and the environment must be recorded [58]. Some examples of elements to be annotated that can then be used to adapt the difficulty [52]:

- The environment the player is interacting with.
- The difference between the score of the players.
- The last actions that were taken by the user in the game.
- The movement the player is performing in the game if applicable to that game.
- Information about past games the user has played.

However, most of these dynamical difficulty adjustment applications are based on specific heuristics and probabilistic methods that relate the different information obtained with the different difficulty levels created by the developers [52]. Even in games where NPCs are present, the different skill levels only change the life and damage values, making them more powerful, but not cleverer [60]. NPCs also need to change their behaviour and adapt their skill based on the player's behaviour during the gameplay [19].

Dynamic difficulty adjustment mainly focuses on balancing the skill of the player with the difficulty of the game. Game adaptivity tries to adjust its game elements to fit the player personality to make the gaming experience unique and personal. In general, games need to become more unpredictable and player-centric. There needs to be a better capture of the player's true intentions for playing that game. These intentions need to be captured by analysing the different characteristics and emotional states during the gameplay to be then used to ensure the player's immersion in the game. [41]. Usually, game adaptivity alters game elements that are not entirely differentiable from regular gameplay. However, when a player notices the change, through repeated gameplay[19], they may find the adaptivity an unfair element, affecting their engagement in the game [58], or they may also find it appropriated, creating a richer experience [19]. There is also a possibility that the player may abuse the system created by, for example, losing on purpose, indicating a wrong level of skill [58].

Game adaptivity can have a positive impact on the game industry. According to [6], player-centric adaptation can increase player loyalty and enjoyment with the game. The adaptivity will allow the game to have increased longevity and become more commercially reliable.

### 2.1.2 Player types

A player type can be associated with the behaviour the user demonstrates during gameplay. The behaviour shown is also what makes each person different. Groupings of players' type can be created to describe better personalities observed in games through the player's psychographic



interests and game behaviours [2]. [2] enumerates several user taxonomies that currently exist and can be used:

- Bartle Taxonomy [8] divides the user into achievers, explorers, socialisers, killers. Achievers categorise players who like to accomplish goals; explorers include users that like to discover the environment; socialisers include players that enjoy having interactions with others in the game; killer represents players who like to disrupt the functioning of the game and like to go beyond what is expected to do in the game.
- Yee's MMORPG user motivations [75] is an iteration from the Bartle taxonomy.
- Four Fun Keys [39] mainly focuses on categorising emotions. They can be divided into four groups: easy fun, hard fun, people fun and serious fun.
- Demographic Game Design model [9] divides the user into conquerer, manager, wanderer and participant.
- Demographic Game Design model 2 [10] is an extension of the previous work. It divides the player into four types: logistical, tactical, strategic and diplomatic.
- BrainHex [50] divides the player into seven categories: seeker, survivor, daredevil, mastermind, conqueror, socialiser and achiever.
- HEXAD [67] divides the user into six types: philanthropists, socialisers, free spirits, achievers, players and disruptors.

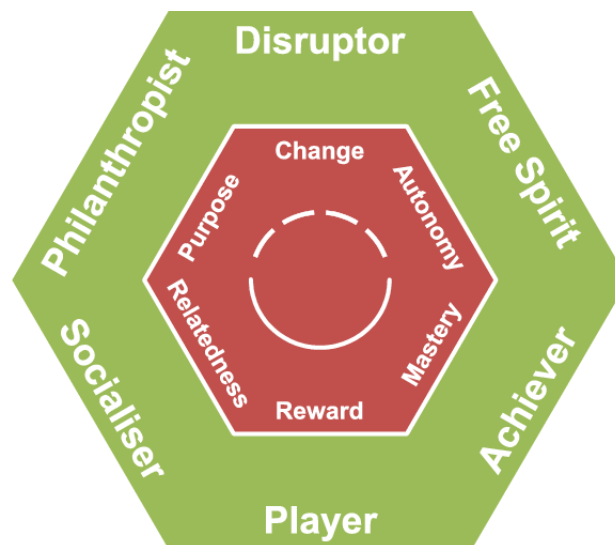
The taxonomies described above serve as a baseline to classify player type in games. One player does not have to be in a specific group but can be part of multiple simultaneously. [2]

From each player type, associations can be made between them and game elements [2]. These associations will enable the game to tailor its content to fit each player. The player's personality modifies its representation of player types and consequently changes its preference in the different game genres [67]. According to [67], the HEXAD scale (Figure 2.2) can have associations between its categories and specific game elements, which might be relevant for the topic of game adaptivity. The following list will further describe the different associations [67]:

- The philanthropists are associated with the players that do not mind performing actions that do not benefit themselves. They relate to the principle of altruism. Game elements like collection, trading, gifting, knowledge sharing and administrative roles better relate to this personality type.
- Socialiser players prefer to perform actions that serve as interaction with others to create social bonds. Elements like guilds, teams, social networks, social comparison, social competition and social discovery benefit this type of player.

- Free Spirits is the type of player that values freedom and self-expression. It can be associated with exploratory tasks, nonlinear gameplay, easter eggs, unlockable content, creation tools and customisation.
- Achievers are the type of player that likes to complete challenges. Elements like certificates, learning new skills, quests, levels, progression, and boss battles include what they primarily seek.
- Players are the type that aims at earning rewards, no matter what the activity is. Elements preferred by this type are points, rewards, prizes, leaderboards, badges, achievements, virtual economy and games of chance.
- Disruptors include players that tend to disrupt the system and enjoy testing the limits of the game. Elements like innovation platforms, voting mechanisms, development tools, anonymity and anarchic gameplay favour this player type.

These player types are an essential piece in game adaptivity since they allow the developers to identify and define game elements for the specific personality. Machine learning algorithms can use these elements to perform adaptivity.



© Andrzej Marczewski 2016 (CC BY-NC-ND)

Figure 2.2: HEXAD scale [67]

### 2.1.3 Game elements

As described in the previous chapter, several game elements can be modified to adapt the game to fit a specific player type. Several examples of games that adapt their game content to fit the game behaviour can be found already implemented in the industry. According to [41] some of these games include Max Payne [59], where the main element being adapted is the players aim assistance. This game mechanic adaptation allows less skilled players to have a slightly better

performance in the game; PES 08 [38] changes the enemy's AI strategy depending on the player's strategy; Left 4 Dead [34] changes the game world generation, the narrative and the events depending on the player's progress and responses to the task presented; Heavy Rain [54] changes the narratives depending on the decision performed by the player in the story (Figure 2.3). Elements of narrative and their adaptation can approximate what the player is searching for in terms of storytelling [41].

According to [6], several elements of the game can be used in the topic of game adaptivity. The following list enumerates some elements presented by the author that have the most impact when implementing an adaptive system:

- Space adaptation. It is referent to how space can adapt to the behaviour of the player. In a terror game, space evolution can include more scenes or environments the player tries to avoid.
- Mission, task or quest. The use of objectives can steer a player in a specific direction (in a narrative sense or by merely orienting the player) and rewarding them by completing the tasks.
- Character adaptation. This can include enemy AI that tries to adapt to the player strategies or behaviour, consequently making the world feel more real and alive.
- Mechanics adaptation. Change in game's aim assistance, the character speed or other specific games mechanics can help create a more adaptable scenario for the player.
- Narrative. The use of narrative and the influence the player can have on them, creates a better experience for the user, making him feel more immersed in the world and in control of the story.
- Music and sound. The correct use of this element is crucial to create a feeling of immersion, flow and engagement. The different kinds of music, with variations in tone, structures and rhythm can induce different feelings during gameplay.
- Difficulty scaling. The use of elements that create a balance between players. These can include randomness, enemy strategies and difficulty, and more pacing to create faster or slower gameplay.

Although these enumerated items do not include every possible element that can be implemented in a game, their use, combined with the player's personality, can create more relevant user experiences. Furthermore, if these elements are adapted to suit the player type, they will be more immersed in the gameplay, creating a more prominent feeling of enjoyment and game flow.



Figure 2.3: Choice during the game Heavy Rain [54]

#### 2.1.4 Player profiling

A critical step to perform game adaptivity is to create a model of the player profile. Demographic information about the user produces some knowledge that can be used to profile the player. However, this data is insufficient to create a personalised experience and can lead to stereotypes in the player models [19]. Another way of obtaining player information is by requesting the completion of a questionnaire. The use of a questionnaire can be seen in the article [19], where the author uses an immersive experience questionnaire to obtain the total immersion felt by the users while playing the adapted game. Furthermore, the author also uses the player score to profile the player. Although the use of questionnaires can provide more insight into the gameplay's experience, it also breaks the game's immersion by interrupting the game [21]. One way to resolve this problem is by integrating the questions in the narrative of the game. This method can help maintain the game's continuity while obtaining information about the player experience [21].

Although questionnaires are commonly used to obtain specific and detailed information about the player, other techniques can also be applied. They include the modelling techniques of the player's actions, tactics, strategies and player profile [6]. Player actions can be directly observed from his behaviour in the game. Several player actions performed in a short period form player tactics. In the same style, several player strategies performed over multiple games compose the player strategies. The applications of these three aspects are motivated by the player's profile and personality, making different players have different actions [6]. The primary idea behind modelling a player profile is to allow the system to know who the player is, their desires and motives, and the user behaviour in that specific instant [6]. However, these techniques require much more observations across various scenarios to be useful, making them more expensive and time-consuming [71]. Also, the game's systems must be sufficiently complex while the observations have to be

retrieved in short intervals during gameplay, and can sometimes be partially completed [6].

One example of using the modelling techniques can be seen in the work [71], where the author uses the player movement, conversation with NPC of the game and other action like interactions with objects to create a model of the player. They also used a questionnaire at the end of the play session. In another work [42], the player model is highly dependent on its gameplay, like his movement through the level. Although user profiling is usually made by observing its behaviour during gameplay, other observations may also be considered. In the work [33], the user profile is also made using the heart rate during gameplay. [41] also comments that body expressions and facial expressions can give more insight into the current user's emotions.

In general, player profiling can contribute to the game's adaptivity as it helps to measure and create a user profile model [41]. This information can help to personalise the gameplay, improving the overall user experience and game flow in the game [33].

### 2.1.5 Game Flow

Flow can be defined as:

“It provided a sense of discovery, a creative feeling of transporting the person into a new reality. It pushed the person to higher levels of performance, and led to previously undreamed-of state of consciousness. In short, it transformed the self by making it more complex.” [18]

Gameflow is the feeling of being concentrated on a demanding and appropriated gaming task while enjoying it [65]. According to [65], the task must be on the same skill level as the player and have explicit goals to guide progress while giving feedback for the player's actions. If these tasks are correctly implemented, it will give the player a sense of control, creating a sense of immersion. The game flow experience by the player can lead to a sense of enjoyment while playing.

According to [66], there are eight core elements of game flow: concentration, challenge, player skills, control, clear goals, feedback, immersion and social interaction.

- **Concentration.** Games should provide different incentives that the player wants. Games should also keep the player's attention on high workload tasks they feel worth completing, motivating their perceptual, cognitive and memory skills. For example, games with high-quality graphics, animations, audio, cutscenes, and tasks can improve concentration [65].
- **Challenge.** Games should provide the player with challenges and tasks on the same level as the player and evolve with the player's evolving skills. Furthermore, challenges can be related to the quality of the enemies and their artificial intelligence during gameplay [65].
- **Player Skills.** Games should encourage players to learn new skills and evolve their gameplay by providing adequate learning challenges while rewarding the user's actions.

- **Control.** Players should be able to control their characters, the actions they can freely perform, and how they can impact the environment they are playing. They are mainly performed using the game input devices, for example, keyboard or controller. Their simplicity and ease of use are crucial to creating a good sense of control in the game [65].
- **Clear Goals.** Goals should be explicitly presented to the player.
- **Feedback.** Players should receive feedback for the actions they take and the tasks they complete.
- **Immersion.** The player should lose the sense of awareness over their external environment, being entirely dedicated to the game. The sense of immersion can also be facilitated by the element of narrative in the game. A more compelling story, interaction with the elements and NPC's of the game, can better connect the player with the game [65].
- **Social Interaction.** Games should provide the player with options to socially interact with other players.

It is worth noting that not all the elements must be created or implemented in a game to give a sense of enjoyment. Some of these elements may also not apply to every game or are trivial to implement, but the general use can increase the global game flow experience. The sense of flow can also be estimated since they can be related to the task the player is completing [65]. The use of questionnaires can help to measure the sense of flow felt during gameplay. The game experience questionnaire[32] has three main modules, the first two to measure how the player felt during the gameplay experience and the last one to understand how the player felt after the session ended. The first module is critical since it estimates the immersion, flow, competence, positive and negative affect, tension, and challenge the player felt during the gaming session. Another questionnaire that also measures flow is the Flow for Presence Questionnaire[57], where the user is asked if he experienced any described flow type during their task.

### 2.1.6 Applications

In the article [58], the author acknowledges that game balancing using dynamical difficulty adjustment requires a large amount of data to solve this problem. To generate more data, the author creates agents with different skills and strategies and mixes them to create different games. The agents utilise RL to preserve the ability to learn from past play and create innovative strategies. RL is commonly used to create AI of enemy players or agents that can solve and play games. Some examples include [58] the algorithm created by OpenAI [14], who trained RL agents that played against each other and were able to win against a professional player of Dota 2 [70], and AlphaZero[64] who was able to play with excellent accuracy games of chess, shogi and go.

Adaptivity is mainly used in multiplayer games. These include the use of systems like difficulty adjustment, matchmaking, asymmetric role and skill and aim assistance that can improve the overall user experience with the games [19]. The main reason behind this is the algorithm's

ability to balance all the player's different skill levels in the same match and between matches. This balancing creates a more even match between the players, leading to more significant equity and potentially fun.

According to [42], the author uses automatic generation of content, specifically game world generation, to provide a more adaptive approach. With the help of a tool that allows the game designers to specify what types of elements and content correspond to each user's personality, the author can create an adaptive system that automatically generates game worlds that fit each personality type. It uses variables like ramp usage, obstacle avoidance, floor edge avoidance, power-up selection, AI defence and AI offence, to measure the player's skill during gameplay.

In [21], the author refers that sensors can capture more complex emotions that can then be used to create a player model. However, their use can be conditioned since they can not be affordable by the average user or are solely too uncomfortable to wear during gameplay sessions. Finally, it used in-game dialogue to evaluate the player's emotional state and then used a function mapping to apply the game adaptivity to each feeling.

Game adaptivity can also be applied to the context of educational games. They provide a tool to teach and educators to keep their students motivated and engaged while acquiring knowledge [29]. An educational game's primary focus is to provide students with a tool to gain education during a gaming environment, making a difference between entertainment games by not focusing on the challenge [41]. Fun, interactivity, challenges and immediate rewards can also be fun in educational games. Instead of focusing on the topic to learn, like in traditional education, the focus is on the gameplay, making indirect learning activities while maintaining the primary rewards from entertainment games [53]. One way of calculating the player's performance, in this case, is by using past question has a reference for measuring the metric. With this information, one can associate the metric measured (player's performance) with potential future questions [53].

A comparison of some state of the art game adaptivity applications can be observed in the table 2.1.

### 2.1.7 Summary

As described in this section 2.1, key elements must be analysed to implement a player-centric game adaptivity correctly. In section 2.1.2 and 2.1.3, several player types were described and the elements each player prefers. The game elements play an essential role in adaptivity since they are to be altered, and they affect how the player experiences the game. To measure the player type, section 2.1.4 explored the use of questionnaires and observation of player behaviour. Section 2.1.5 described the meaning of gameflow and how it can affect the gaming experience. The game flow describes the primary stimuli that the players search during gameplay, making it essential for both the game's development and adaptivity implementation. Although game flow is composed of many different elements, only a few apply to game adaptivity. In section 2.1.6, some applications of game adaptivity were analysed.

Game adaptivity is typically implemented as a dynamical difficulty adjustment with simplistic implementations and observations. Even when adaptivity is implemented in a more complex game,

Table 2.1: Comparison of applications of adaptivity

Article	Type	Player profile/ Observations	Modifications/ Actions	Algorithm	Objective/ Reward	Notes
[21]	DDA	Frustration and boredom	Platform mechanics	Basic	Balance frustration and boredom	-
[33]	Adaptivity	Age and heart rate	Spawning mechanic	Basic	Balance effort	-
[19]	DDA	Immersion (immersive experience questionnaire)	Game timer	Basic	Increase immersion	-
[53]	Education	Past questions	Next questions and tips	Basic	Education	-
[52]	DDA	Distance between players, difference in health, last action	Move the character	RL-PPO	Fair opponent	Simulated players
[29]	Education	Mouse situation, potential score	Cat movement, tips	Basic	CT	-
[58]	DDA	Win/lose ratio	Move wall	RL-ABWPL	Balance ratio	Simulated players
[42]	Adaptivity	Usage of game elements	World generation	Basic	User experience	-
[45]	DDA	Level, score and health	Skill level	SVM + K-means	Correct skill level	-

their implementation is generally limited to the association between the player type identified by specific actions and the game elements that need to be altered. This association is usually implemented manually, leaving room for an exploration of machine learning approaches.

## 2.2 Machine learning

Machine learning is the use of computational algorithms, enabling a computer to learn from past experiences and make correct predictions. Any type of digital data can be used to train and test the algorithm. However, this data must have sufficient quality and size for the adequate functioning of the predictions. This constraint creates a dependency on the data used, relating the predicament of teaching with data analysis and statistics. [48]



A large variety of tasks can employ the use of machine learning. As revealed by [48], these algorithms can be used for text or document classification, natural language processing, speech processing applications, computer vision applications, computational biology applications and many other specialised applications like learning to play games. Implementing machine learning is not limited to these topics and can be adapted to solve any problem correctly.

According to [24], data quality assessment should be considered before building the model used for machine learning. Although this project does not iterate over the data quality topic, one must always consider data analysis and try to solve their problems to promote better performance and consistency over the built system. It is also worth noting that the system's performance is an indirect measure of data quality since it is evaluated using a different subset of the data that does not correspond to the data used to teach the algorithm. On the contrary, data analysis can significantly impact the model's performance; for example, the presence of outliers during the training phase can cause instability in the final model. [24]

Some dimensions of data quality that can be considered during the data analysis include [63]:

- Timeliness is related to the age of the data and if it is appropriated with the task proposed.
- Consistency is related to the data format that must correspond with all the data, wherever it is stored.
- Accuracy is related to how well the data reflects the real-world values.
- Completeness is related to the data's ability to represent sufficient complex information to the task employed.
- Duplication is related to unwanted duplication present in the dataset.
- Consistent Representation is concerned with the format of the data and the preservation of it.

The dimensions enumerated can serve as a baseline, but are not limited to it. Other data quality dimensions can also be considered [63]. To meet data quality, the user does not need to meet every single dimension. However, the knowledge of them can help to improve quality. On the contrary, quality issues can arise from data entry errors, redundancy duplicates, contradictory values and inconsistent data [63].

In terms of machine learning tasks, they can be divided into three major groups: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Supervised learning is a subset of machine learning where the objective is to build a model capable of learning to predict an output, where the data used for training is labelled. In other words, there exists a mapping between the input variables and the output variable. Inside the supervised learning category, we can have classification and regression tasks. The main difference refers to the output variable. In classification, the problem is to assign a category or a limited set of items to the output correctly. In contrast, regression is used when the output represents a real value. [48]

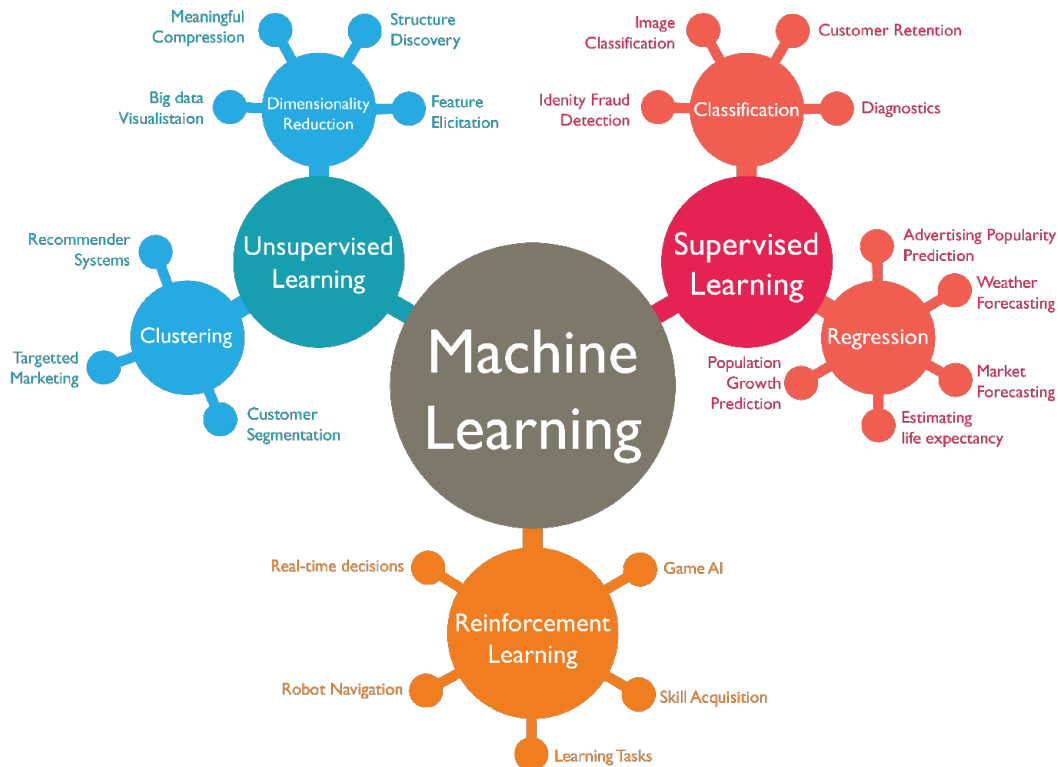


Figure 2.4: Machine learning areas. Source: Big data and machine learning for Businesses by Abdul Wahid [73]

On the other hand, unsupervised learning is used to solve tasks where the main objective is to find hidden data structures, creating data groups. This type of machine learning uses unlabeled data to train the algorithms. Unsupervised learning mainly consists of two tasks, clustering and dimensionality reduction. The clustering task is the partitioning of the data into a homogeneous subset of items (clusters). These clusters contain items that are similar to each other within a given filter. Dimensionality reduction consists of transforming a set of data into one that has a lower-dimensionality representation. [48]

Finally, reinforcement learning can also be considered a subset of machine learning. Using this learning scenario, the training and testing phases are mixed. The model actively interacts with the environment by taking actions while receiving a corresponding reward. The objective of the learner is to maximise the reward function. To achieve this, he must actively be exploring new actions to take or exploiting the actions already taken with the information obtained from them. [48]

Although only three machine learning subsets were referred to, others can also be included for more specific and intricate learning scenarios. Such as semi-supervised learning, transductive inference, online learning, active learning.

### 2.2.1 Supervised Learning

Supervised learning's basic idea is to learn a mapping between the input variables to an output variable. It will then create a model of this mapping that can classify the unlabelled data. To create the model, it first needs to learn from annotated training data to understand the relationship between the input variables and output. The name supervised comes from the fact that the model only trains with the labelled dataset. The learning of the model only stops until it can make correct predictions with an adequate performance level.[49]

Several methods of the application of supervised learning exist and are available to solve many problems. Some of these include: support vector machines, neural nets, logistic regression, naive bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees and boosted stumps. More information about these algorithms and an empirical comparison can be found at [16]. Although this project's main focus does not involve an in-depth analysis of the methods, some basic knowledge about their functionality and their advantages and disadvantages are essential for potential implementation.

However, can these machine learning methods be used to solve problems in the topic of game adaptivity? One use of the algorithm SVM can be found in [45] where the author tries to adjust the game difficulty. This algorithm learns using data from an "offline" phase, where the user automatically sets his difficulty, and then the information about the game is logged. In an "online" phase, which corresponds to the unlabelled data where a new user plays the game, the first step is to determine the player's type, using a K-means algorithm [26] then the learned model(SVM) classifies the data, giving the corresponding difficulty level(label).

Support vector machine (SVM) is a supervised learning algorithm that can classify nonlinear problems. It is based on statistical learning theory that can use different kernel functions to map the input variables to a high dimensionality feature space. This algorithm can be used to solve either classification or regression problems. [27]

One of the main limitations of [45], referred to above, is that the output is only a variable corresponding to the player's difficulty. Although choosing a difficulty level is lifted from the developers' work, there is still a limitation to how complex and detailed the creator can make each skill level. In one of this thesis initial works [35], the problem of single output was analysed, and multi-output regression was used to solve this problem.

A problem faced by supervised machine learning algorithms is that the mapping is limited to giving one unique output variable to several input variables. One approach to try to solve this problem is the use of multi-target prediction. According to [72] several approaches could be made regarding multi-target prediction. One can use individual models for each target label or use a specific model to perform multi-output while considering the relation between the different variables. Although these approaches may be more intensive in memory usage, they still allow a possible solution to the problem. It is also worth noting that in [35], multi-target aims to solve the game adaptivity problem. The labels are related to the game design; more specifically, they correspond to what the user can interact with, modifying the game's final game experience.

As referenced [35], supervised learning algorithms can be used to solve game adaptivity. However, as the author mentioned, some assumptions were made about the user. One of these assumptions is the user's inability to progress and learn from gameplay, effectively denying the possibility of evolving and becoming better. Additionally, it also indicates that the predictions made are limited to the game's start and end. This problem limits the game's adaptation by generalising the gameplay that lasted several seconds (ideally, the adaptation should be made in real-time). These limitations could be solved using an online learning approach, although they are not explored in the article. It is also worth noting that the algorithm was trained with data that had satisfaction above a predefined value.

### 2.2.2 Unsupervised Learning

Unsupervised learning is related to the algorithm's ability to find hidden patterns in a dataset where it may appear to be unstructured noise [22]. In supervised learning, the dataset has a mapping between input variables and the output variable. The output corresponds to what the algorithm can find in the real world. In contrast, unsupervised learning receives unlabeled data. This data can be used to build representations of the input and can be used for decision making and predicting future inputs [22]

Although this learning scenario can be decomposed into more specific tasks, clustering is the only one to be considered and reviewed. Clustering is the technique that groups the raw data received into clusters. The clusters are created from hidden patterns in the data. Items inside one cluster are similar to each other and different from other cluster's items [62]. An example of an algorithm that creates clusters of data is the k-means algorithm. K-means is a non-deterministic unsupervised learning algorithm that produces fast and simple numeric clustering results. It iterates over every item, calculating the distance between, commonly using the Euclidean distance, and then creates new optimal cluster centres [62].

The clustering algorithm can be used with the game's observations as input and player types clusters as output. The use of the K-means algorithm is used on the work [45], where the author uses the clusters created by the algorithm to find what the new player type is. After having the corresponding type, the difficulty adjustment can be applied. In this case, the author uses an SVM algorithm to create a correspondence between the player type and the difficulty level to be adjusted.

The use of unsupervised learning can solve the problem of game adaptivity. However, the player personality is being generalised to one player type and one game difficulty. This generalisation creates a less player-centric experience needing a more complex system that can associate different game elements with each observed action player while maximising the players game flow.

### 2.2.3 Reinforcement Learning

One of the subsets of machine learning is reinforcement learning. Reinforcement learning utilises machine learning algorithms to create an agent that can learn from a world and act on it to achieve the desired objective. However, instead of providing information on how to solve the task, the agent is put in the world, and with enough trial and error, it will learn the goods and bad rewards for each action in each state [56]. One example of reinforcement learning can be observed when a user plays for the first time the game Super Mario [46]. In the first session of gameplay, the player tries the game's different control and movement options. He will try to move the character, and as soon as he leaves the initial screen, an enemy will appear, and it will start to stroll towards the player. If the player does not jump over the enemy, the character will die, and the game will start over. The player's interaction with the character and the enemy incentivises forward movement since the screen does not move to the left side and avoiding enemies by jumping over them. This gameplay can be seen as reinforcement learning since the player is rewarded with each action that he takes, such as the screen moving to the right, which signifies a positive reward since the player is presented with progression and new challenges. Additionally, the player can also lose the game to an enemy, representing a negative reward since the player is brought back to the start of the game.

For reinforcement learning to work in a machine learning manner, several elements should be defined, including the agent, the environment, the states, the actions and the reward [56]. The agent can be viewed in the example above as the user who plays the game. He is the one who determines how to play the game and what are the best decisions to take [56]. The environment is where the agent lives and can interact with [56]. In the example above, it can be seen as the level of the game, the world where the character plays. The state is an instant in the environment, and each state can be obtained by acting on the world [56]. Each state represents the complete information about the environment, and the agent can make observations about the world to receive information [1]. If the agent can observe the whole state of the environment, the environment is fully observed [1]. On the other hand, the environment is partially observable if the observation only represents a portion of the world's information [1]. In each state, the agent can perform an action, which will lead to an update in the environment and a new state where the agent can then perform another action [56]. An action space defines what actions the agent can take on the environment [56]. In the example above, the player has only a finite amount of moves that he can make. These include moving the character to the right, to the left, jump and run. In this case, the action space is discrete since all the agent's actions are discrete [56]. In contrast to this, if the world has infinite possible actions, the action space is continuous [56]. An example of this is a racing game, where the agent can decide what speed he wants the car to go at or how much the car needs to rotate in degrees.

Furthermore, based on the action taken on the environment, the agent receives a corresponding reward [56]. In a real-world scenario, the reward corresponds to a treat a dog receives for performing a correct action. In machine learning, the reward is represented by a number, and its value depends on the environment the agent lives in. The agent's objective is to maximise the sum of the

rewards, denoted as return, over the environment states starting from the initial state to the final state, called an episode [56]. In the example above, an episode starts from the initial position until the player reaches the end of the level, located in the far right, or when the user dies to an enemy and is forced to start over. An episode can also be called a trajectory and is denoted by all the states achieved by performing actions at each timestep or state with an associated reward [56]. The agent decides what action to take in each state based on the current policy [56]. In the first episode, the agent policy is randomly initialised, and the agent performs random actions on the environment, receiving a corresponding reward based on the action taken [56]. After several episodes, the agent will eventually learn the best action to take in a specific state [56]. If the policy decides to perform a specific action in a state, the policy is deterministic [56]. Furthermore, if the policy instead maps a probability distribution over the action space, the policy is stochastic [56]. Instead of performing the same action when the agent is in a specific state, as in the deterministic policy, the stochastic policy creates a probability of performing each action of the action space [56]. The optimal policy is the policy that chooses the correct action in each state and, consequently, maximises the agent's return (sum of rewards) [56]. To prevent the return from reaching an infinite value, a discount factor is introduced to decrease the reward for future actions [56]. This factor value defines how vital the immediate and future rewards are [56].

Reinforcement learning can be divided into model-based and model-free RL (Figure 2.5). A model-based approach can be implemented if the algorithm can access the environment model [1]. The agent knows the probability of moving to a new state based on the current state and action (transition probability) and the reward associated with this transition [56]. On the model-free approach, the agent does not know the environment model dynamics [56]. There exist two primary methods in model-free learning: policy optimisation and Q-learning. Policy optimisation algorithms are typically on-policy; they decide based on one policy while improving the same policy to obtain the optimal policy [56]. Methods like A2C/A3C[47] and PPO[61] are examples of policy optimisation algorithms. Policy optimisation algorithms are stable and reliable principled methods that try to optimise the reward. Q-learning algorithms are mainly performed off-policy; these algorithms use two policies, one behaviour policy and a target policy [56]. The behaviour policy decides the actions on the environment while the target policy is improved [56].

Algorithms like DDPG[40] and SAC[25] are algorithms that interpolate between policy optimisation and Q-learning. They use parts of both approaches to create more reliable and stable algorithms. [1]

The application of reinforcement learning in-game adaptivity can be seen in the work [58], where the author uses an RL game master to control the game and to apply the game adaptation. However, this work only uses the win ratio of the players to train the model. In another work [52], the author uses reinforcement learning(PPO algorithm) to train a game adversary to learn the game and match the player's skill to provide a fair match. In both of these works, the RL agent's reward was to create a state of game balance, or in other words, this is an application of dynamic difficulty adjustment.

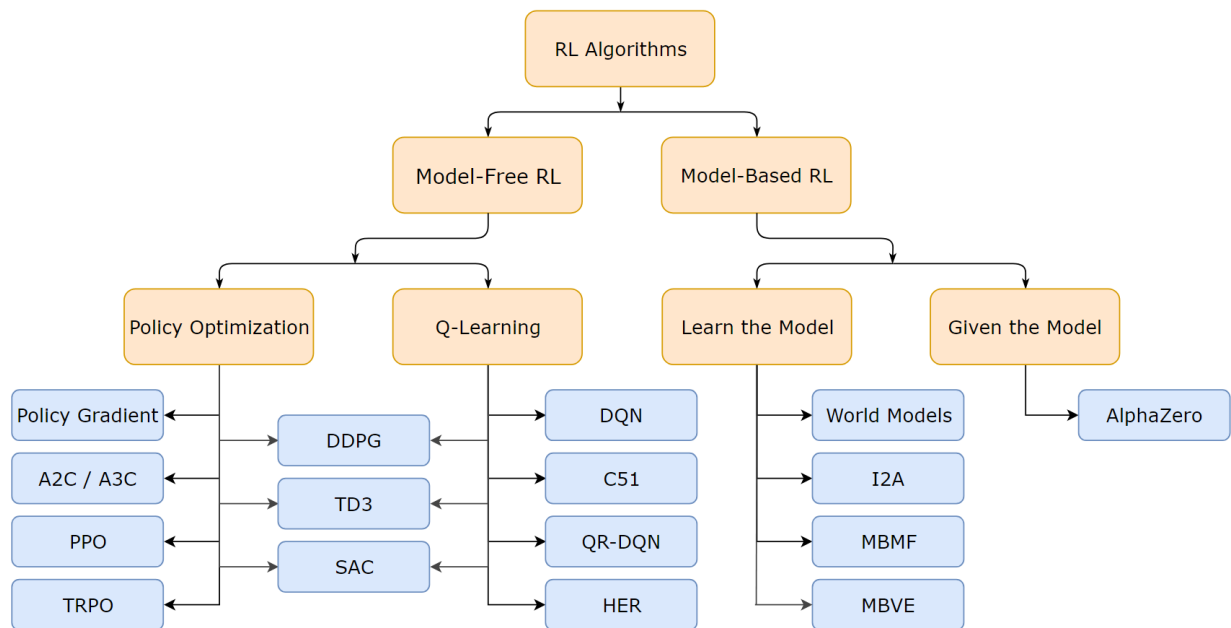


Figure 2.5: A Taxonomy of RL Algorithms by [1]

## 2.2.4 Imitation Learning and Inverse Reinforcement Learning

Imitation learning is an algorithm that learns how to perform from demonstrated behaviours without receiving the reward [30]. Instead of the agent being put directly into the environment, the agent collects expert demonstrations and then learns how to perform in a supervised learning manner by minimising the loss function [56]. Imitation learning allows the agents to know how to perform in each state presented in the world. However, if the expert training demonstrations do not include a state, the agent will not be able to act since the expert never experienced that state [56]. One algorithm that can implement imitation learning is Behaviour Cloning [68]. The implementation of this algorithm can also be found at Unity ml-agents [69], where developers can implement and adapt this algorithm to work with their games. The implementation of this algorithm also includes a method to record expert demonstrations from the players. Furthermore, the framework ml-agents also includes other algorithms to supplement behavioural cloning to mitigate its flaws. One of the available algorithms is GAIL (Generative adversarial imitation learning)[28], which implements inverse reinforcement learning.

The main objective of reinforcement learning is to find the optimal policy in a specific environment that maximises the return [56]. For this to work, the environment needs to reward the agent depending on the current state and the action chosen. However, the reward in an environment is too troublesome to define, for example, in a pedestrian simulation, the agent might be rewarded for walking in the walkway and avoiding other people, but he might also need to watch for other obstacles to avoid and other problems that might need to have a positive or negative reward associated to them. Inverse reinforcement learning tries to learn the reward function from expert demonstrations [56]. After learning the reward function for that environment, the agent can

then be trained to learn the optimal policy using a reinforcement learning algorithm [56]. One algorithm that implements inverse reinforcement learning and is also available in the framework Unity ml-agents[69] is GAIL[28]. This algorithm is based on a supervised machine learning algorithm GAN(Generative adversarial network)[23]. GAN consists of two networks: a generator and a discriminator[23]. The generator generates new data points based on the distribution of the input dataset [56]. The discriminator then distinguishes between the actual data points that hold the ground truth and the generated ones[56]. The algorithm GAIL works similarly, where the generator generates a new policy based on the distribution of the expert policy, and the discriminator determines if the policy belongs to the expert or the agent(generated)[56]. These two networks compete and learn simultaneously to improve their ability to fool/distinguish[56]. Furthermore, the GAIL algorithm does not require large amounts of expert data to train since it can generate data from the environment and train with it.

### 2.2.5 Player simulation

Player simulations are primarily utilised to create agents that can play a game to an equal or higher level than a human being. Examples of player simulation can be found in the table 2.2. In the work[4], an agent is trained using an imitation learning algorithm and with observations of real users playing games. Here the agent can mimic the actual player and even surpass their performance. The architecture of this article work network uses convolutional neural networks to process the video of the expert gameplay. Furthermore, this article presents an imitation learning solution to training agents on games that could not be reasonably played using only simple reinforcement learning algorithms. This solution does not require a complex reward function, which previous attempts had trouble with.

Convolutional neural networks are popularly used machine learning algorithms for computer vision problems[56]. These neural networks help extract meaningful features from images by performing convolutional operations [56]. CNN's are also implemented in Unity ml-agents[69]. In a platforming game like Super Mario[46], using CNN can help describe the current state of the environment instead of defining every possible walkable space, which may prove to be less efficient. The use of CNN can be seen in the work [74], where the author uses a simple three-layer convolutional neural network as input for the algorithm GAIL to translate the character's current position in the game Super Mario [46] using the frames from the gameplay.

The use of a long short-term memory (LSTM)[39] block in a deep neural network can help the algorithm remember old states that happen during that episode and can influence the decision in the current state [56]. In the work [5], the neural network architecture includes an LSTM block to help the agent have a sense of object permanence when placed in the environment. The memory also helps in the final result of the article by allowing the agents to remember where each object they moved or seen were placed.



Table 2.2: Player simulation examples

Article	Algorithm	Observations	Rewards	Actions	Notes
[37]	Basic association between data and action	Information about the football player and the current situation in the game	-	What should be the action for the player	-
[5]	PPO	Position of the agent and information about the world and its obstacles	Hide and seek game, competition for winning	Movement, grabbing and locking	Uses CNN and LSTM
[4]	Distributed A3C RL agent IMPALA	Youtube videos of gameplay	Reward to orient the agent in the game	Movement inside the game	Uses CNN
[74]	PPO and GAIL	Simulated expert gameplay in form of frames	Expert training was done using reward defined from gym environment	Movement of the character	Uses CNN

### 2.2.6 Curriculum Learning

Curriculum learning works by incrementing the agent's difficulty as soon as he can complete the previous difficulty [12]. In the same way, as humans learn in increasingly more complex environments, the agent will be faced with more difficult challenges to help him learn the basics before trying to complete the more demanding challenges [12]. It is also worth noting that curriculum learning can improve training times, but it is not guaranteed to improve the algorithm performance for that specific problem [12]. This is caused because learning depends on the teacher and its selections of problems to combat the student's difficulties [12], which, in the case of machine learning, means that curriculum learning needs to define the specific problems that the agent will need to learn incrementally.

### 2.2.7 Online Learning

Online machine learning is commonly used when the data evolves and changes with time [48]. This approach will allow the model to dynamically adapt to the new occurring patterns in the time-based data. For example, when a user needs assistance for motor skill and human activity, online learning can help the system adapt to the user's real-time behaviour [76]. This problem can be adjusted to game adaptivity, where the user's behaviour during the game is recorded, allowing the algorithm to adapt continuously. Furthermore, it can also be applied for several playthroughs

or gameplay sessions, where the data between games is continuously recorded and used to train the algorithm. These approaches can assist the supervised learning algorithm to adapt to the evolution of the player.

According to [20] an online learning algorithm combines both estimation and optimisation. Similarly to RL algorithms receiving a reward from their actions, the online learning approach also gets a reward correspondent to the decision. Its objective is to minimise the regret or to maximise the cumulative reward. The main difference between online learning and RL is that the RL algorithm needs to understand the world's rules from the rewards it receives from performing actions. In contrast, online learning is used to solve a problem, that is known, using the sequential data received.

### **2.2.8 Summary**

In section 2.2, machine learning approaches to game adaptivity were explored and explained. Although some limitations were found on some approaches, reinforcement learning still showed the most potential to solve this thesis's topic. However, it is typically employed to adapt the game difficulty to the players, and its usage for improving the game flow and game experience is minimal, which this dissertation explores. Also, imitation learning and inverse reinforcement learning were described in section 2.2.4 since they will be used to develop this project. Additionally, in section 2.2.5 player simulation was defined, and additional techniques that can be used in conjunction with reinforcement learning were described. Finally, online learning is briefly described, although its application to this project is minimal or non-existent because of its similarity and replaceability with reinforcement learning.

## Chapter 3

# Reinforcement learning based approach for game adaptivity

This chapter focuses on the project description that is going to be developed to fulfill the objectives described in section 1.2. It will describe what should be implemented and the overall structure. This chapter combines the state of the art review elements to create a concrete approach to game adaptivity. Firstly, the preliminary work's game adaptivity approach will be explained and its work's method will be detailed. Then, the selected game will be presented, and the game elements will be displayed. After this, the proposed modifications to create a more diverse set of strategies for the players will be discussed. The final two sections will present the proposed player simulation and the adaptivity system to be implemented.

### 3.1 Preliminary work

In the preliminary work of this thesis [35], multi-target and reinforcement learning was explored to solve the problem of game adaptivity. The game breakout[3] was recreated to be used during the training of the algorithm. It is a simple game, where the player moves a horizontal paddle to block the ball from exiting the play zone through the bottom. The ball can hit the sides and the top of the play-area to bounce. The ball is also used to destroy the group of horizontal bars that are presented to the player.

Since this project's development was short, real players could not be asked to play the game to train the algorithm, so a simulation of the players needed to be done. Six personalities were simulated to represent real players: newbie, gifted newbie, experienced, competitive, fast learner and risky. The main difference between these personalities are their actions per minute, reaction time, paddle safety distance (how much will the player let the ball get to the edge of the paddle) and the movement heuristic. These elements are related to the game's input and impact the gameplay, differentiating the different player types.

This project was done using Unity[69], Unity ml-agents, and Stable Baselines3[55]. The game was recreated in Unity, and the implementation of reinforcement learning, more specifically,

implementation of proximal policy optimisation algorithm (PPO) [61], was done using Stable Baselines3. Unity ml-agents was used to create the machine learning agent and create a connection between the unity environment and the python environment.

For the algorithm's training, one episode was composed of ten games played by one personality (the personalities were used alternately between episodes). At the start of a game, the content and mechanics were modified, and when the game finished, the observations were made, and the reward was calculated. The action space was composed of the game content and mechanics, including the brick height, paddle speed, ball speed, paddle length and ball size. The reward or satisfaction was calculated using four questions of the game experience questionnaire[32]: content, skillfulness, occupied and difficulty. The importance of each of these questions depended on the player's personality. The observation space is composed of:

- The gameplay duration.
- How much the paddle travelled.
- The amount of ball hits with the paddle.
- The number of ball bounces.
- The number of bricks not destroyed.
- The amount of wins and loses.
- The type of player.
- The player's actions per minute.
- The player's reaction time.
- The player's paddle safety.
- The four interrogations of the questionnaire.

This project suggests that the problem of game adaptivity could be solved using reinforcement learning. However, the work showed some limitations and future work that could be explored. For example, the game itself is simplistic, making the player's actions during gameplay limited. Additionally, the paddle's movement and the game's statistics make the player's profiling very limited. This problem could be explored better in a relatively more complex game. Furthermore, the use of handcrafted simulated players somewhat limited the variety of different player types. Also, the algorithm suffered from an insufficient amount of data, needing a longer training time.

## **3.2 General adaptivity system methodology**

This dissertation focuses on creating an adaptivity system that can be used in almost every game. However, a lot of the systems that are going to be developed are specific to one game. Therefore, to create this adapted game, several steps and definitions need to be created.

Firstly, a game needs to be selected since the adaptivity system is very much dependent on the game's elements. Each game is unique, and as such, the implementation of the adaptivity system is also different.

After the selection of the game, a player simulation system can be developed. An adaptivity system could use this player simulation if it is not possible to gather enough data to train the adaptivity agent. If the game has access to a lot of real player data, the adaptivity agent can train with this data. In this dissertation, since the player's data would not be enough, the player simulation system needed to be created. This system collects gameplay demonstrations from real users and then uses an imitation learning algorithm (for example, GAIL and/or behavioral cloning) to train different agents for each user's demonstration. The states, demonstrations and actions are all dependent on the game selected.

If the player simulation does not prove to be an ideal representation of the user, a more straightforward system can be created to represent the potential players of the game, for example, a system to represent the different personalities that would play the game and generate the results and reward at each state.

Then, the player simulation can be used to generate games for the adaptivity agent to train with. However, the adaptivity agent will need to receive a reward for the actions performed. These rewards should represent the gaming experience and game flow that the user would feel at that state of the game. The rewards should be calculated using a few questions relating to the game elements and their experience at each state. Additionally, other algorithms like inverse reinforcement learning can be explored to represent the reward function further.

The observations of the adaptivity agent should consist of metrics and values of the playing user, which represents their strategy and decisions performed in the current or the past gameplay sessions. After observing the environment's current state, the agent should decide how the game element should be modified. These game elements that compose the action space depend on the game selected and should be picked based on their potential to modify the gaming experience (some of these elements are described in section 2.1.3). The agent should be trained with a state of the art reinforcement learning algorithm, for example, PPO, SAC or other relevant algorithms.

After the creation of the adaptivity system, the adapted game should be tested with real users. The users should play the adapted and non-adapted game and then respond to a questionnaire, which will serve to confirm that the adaptivity system impacted the gaming experience and game flow. Additionally, other game metrics, which depend on the selected game, can also be collected to observe the adapted game's impact on the play session.

### 3.3 Selected Game

The first step to implement adaptivity is to select a game that can be modified to implement an adaptivity system. As mentioned in the section 3.1, the game selected for the preliminary work was Breakout[3], a relatively simple game. In this dissertation, a more complex game will be chosen to explore the different types of observations, rewards, and actions to implement. However,

developing a game from the ground up will not leave enough time to implement a well thought out game, and for this reason, an open-source game will be selected. The game Red Runner was selected from a list of games available in GitHub <sup>1</sup>, for being an infinite runner platform game with simple expandable mechanics and its easy to read and modify code.

As mentioned, the game is a platformer game similar to Super Mario[46], where the game will present the player with some jump challenges and enemy avoidance. Additionally, the game is also an infinite runner, meaning that another objective of the game is to go as far as possible to the right to achieve a new max record. The max records are persistent through multiple gameplay sessions to compare its performance to previous attempts.

A critical feature of any platform game is the movement of the character. The game Red Runner offers a well-defined movement set. The player can move the character to the right, and the left with the “A” and “D” key, respectively, or the arrows keys. The character starts with a walking speed and then accelerates in a given direction to a max running speed. To reach max speed, the player has to keep holding the button down for several seconds, which the game explores with some challenges. To overcome some obstacles that would kill the user, the player can jump over them using the “W” key of the up arrow key. Although, the jump not only serves to avoid these enemies but also to jump and climb to higher places. During a jump, the character’s speed is maintained over the whole duration, and the player can also change direction without losing most of that speed. If a player hits a wall, the character will stop and will need to accelerate again. Additionally, if the player hits a ledge, the runner will lose a substantial amount of momentum, but he will not fall over the edge most of the time. Finally, the runner can also perform a roll movement, which will allow him to gain a small amount of speed. This gain is more significant than if he had run over the same distance.

Since the game is an infinite runner, the game has a procedural generator system that creates an infinite running space for the player. For this, the game is composed of sixteen different and distinguishable sections. Each section represents a different challenge, with some parts consisting of platforming and others with more complex levels with hidden coins. Furthermore, each section can be correlated with different types of gaming personalities, each providing a particular level of enjoyment to what a player is looking for. The sections are composed of several elements and mechanics that affect the player gameplay, which include:

- **Coins and chest.** The coins can be seen in the starting section 3.7a and can be collected if the player passes through them. If the coins are collected, they will be added to a global coin count for the game. The coins are persistent throughout the playthroughs of the game and are never reset to a value of zero. On the other hand, a chest contains several coins inside and can be opened by walking near them. They can be found on three levels and are mostly hidden from player sight. They provide around three to five coins that will jump from the chest when open and can be collected by the player in the same way regular coins are collected.

---

<sup>1</sup><https://github.com/topics/unity3d-games>

- **Stationary objects.** Several stationary objects are present in the game, including saws and spikes. These represent a deadly challenge to the player and are overcome mainly by jumping over them or avoiding walking near them. Furthermore, these objects do not represent an immediate danger to the player but require some attention and some strategy to approach them. They can be seen in figures 3.2 and 3.1.

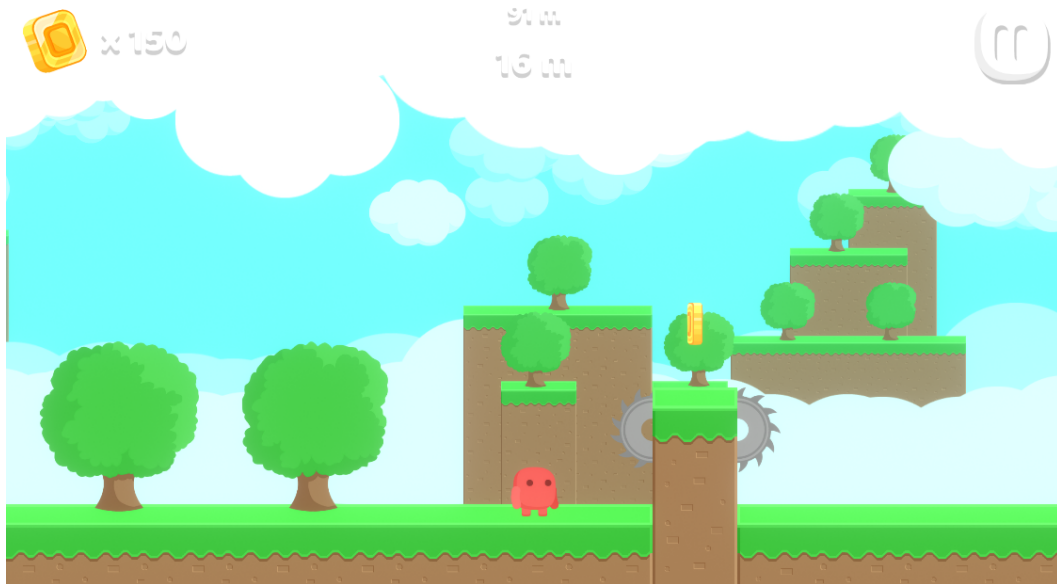


Figure 3.1: Gameplay image with two stationary saws



Figure 3.2: Gameplay image with a stationary spike

- **Moving objects.** The moving objects in the game are enemies to the player and are represented by moving saws and moving blocks with spikes around them. The moving blocks can

travel vertically and horizontally, while the moving saws generally have a visible path to let the player know where they will move to. Both these enemies represent an immediate danger to the player if he is directly in their path. These challenges can usually be bypassed by waiting or jumping over them. The figures 3.3 and 3.4 show examples of moving enemies.

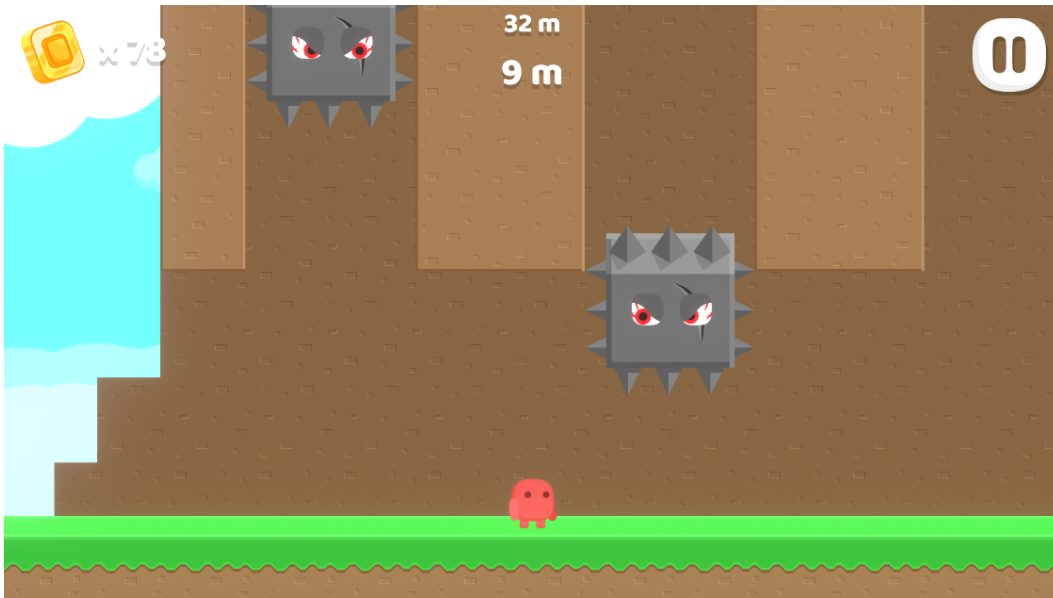


Figure 3.3: Gameplay image with falling enemies

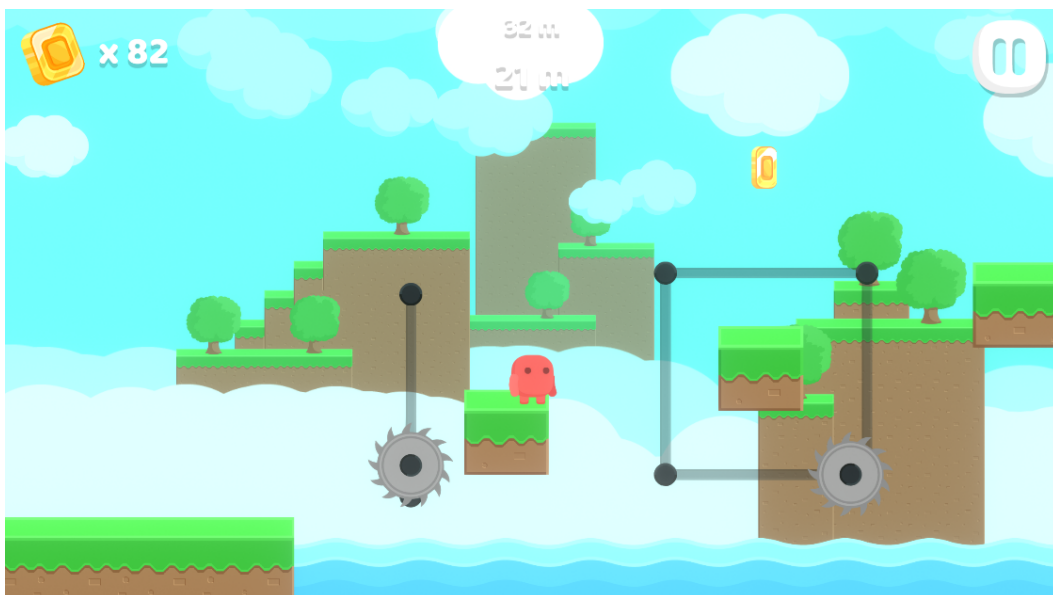


Figure 3.4: Gameplay image with several moving saws

- **Water.** Water is used to create a platforming challenge since the player loses if he touches it (Figure 3.5). In figure 3.6, water can be seen below the player, while platforms are located above to create several consecutive jumps in a platforming section.



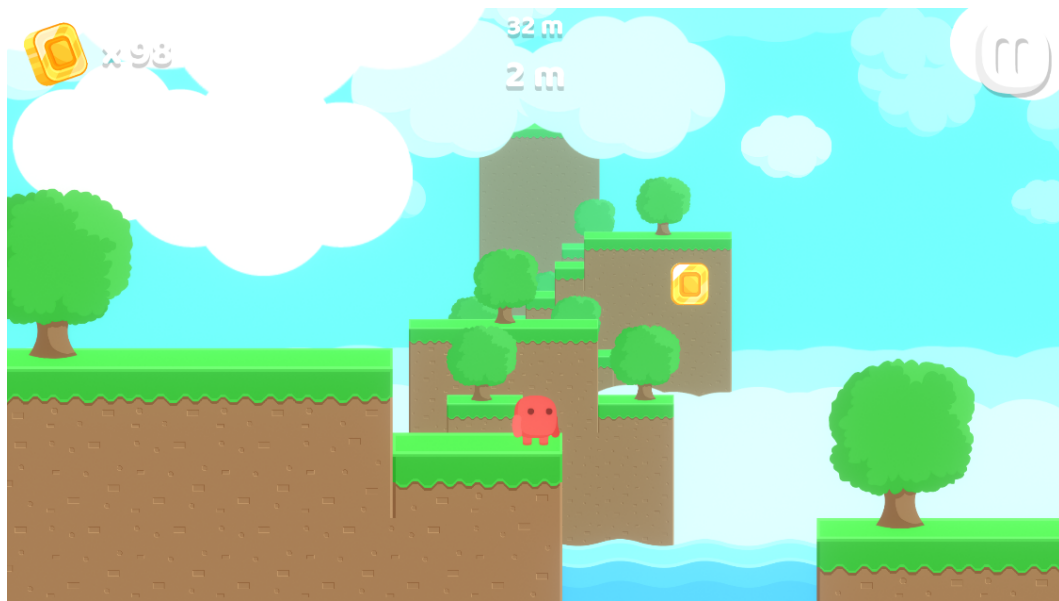


Figure 3.5: Gameplay image that requires a jump over a hole of water



Figure 3.6: Gameplay image in a platform level

All these elements compose the game's main elements that the player must adapt their strategy to complete each section. The following list will explore each section with more detail:

- **Starting Section 3.7a.** The starting section is the first section the player is presented with when starting a new game run. It contains two coins: the first coin is located just to the player's right to incentivise right movement. Here, the player will experiment with the game's movement options until they can collect that coin and move forward. It is also worth noting that the screen cannot move to the left on the starting stage, but the player can fall off the screen and lose the game, which should be fixed in the modification of the game. The

second coin is located right above a water pit, ensuring that the player needs to use the jump movement to overcome this obstacle. Although the design of this section teaches the use of most of the movement options, the player can lose the game prematurely to the water pit, giving him no progress.

- **Sections 0 (Figure 3.7b) and 2 (Figure 3.7d).** Both these sections represent a similar challenge of platforming. They do not require speed for completion, and they always give coins because the player cannot avoid them.
- **Section 1 (Figure 3.7c).** In this section, the player can go over the top or the bottom. At the bottom, the player faces a moving vertical enemy that prevents him from taking the coins straight away. The player can also backtrack in this level to collect all the coins.
- **Section 3 (Figure 3.7e).** This section is the most accessible section of the game, only requiring the player to move right and jump to collect a coin. It does not have any form of enemies, so the player cannot lose the game.
- **Sections 4 (Figure 3.7f),6 (Figure 3.7h),11 (Figure 3.8c).** These sections are similar, requiring a mix of skill at dodging enemies and platforming sections. Section 6 and 11 include moving enemies, requiring more skill at dodging since the enemy are part of the path to complete the section. In section 11, the moving enemies are located in a platforming section, dividing the player's attention to not fall in the pit and watching for the fast-moving saws. In section 6, a chest is located in one of the player's paths, not hidden from the player viewport at any point in this section. The player can also backtrack in these three sections, collecting all the hidden or hard to catch coins.
- **Sections 5 (Figure 3.7g),8 (Figure 3.7j),9 (Figure 3.8a),10 (Figure 3.8b).** These four levels are primarily platforming levels, varying in difficulty level. Section 5 and 8 are classic platforming levels, having water and several small platforms requiring multiple precise jumps. Section 9 is a fast platform level requiring the player to have enough speed to clear the jumps. This level also includes a long jump at the end that requires some backtracking that may also lead to a chest with coins. Section 10 is probably the hardest level in the game, including several moving enemies and platforming challenges. Although the enemies have a defined path, they can change the direction unpredictably, making this section harder than other sections.
- **Sections 7 (Figure 3.7i),12 (Figure 3.8d),13 (Figure 3.8e).** These sections do not include much platforming and only require avoiding enemies and going as fast as possible. Section 12 has a challenging part where the player requires two jumps going at full speed to avoid a static saw. Section 13 also includes a fast-moving saw that chases the player until the end of the section, not allowing the player to stop at any time to think at the cost of losing the game.

- **Sections 14 (Figure 3.8f),15 (Figure 3.8g)** . Both these sections are more relaxed and represent a distinct challenge to the player. In section 14, there is a horizontal moving block enemy which the player must wait to get behind him. Also, in this section, there is a chest located at the top of the section, requiring optional backtracking of the player until he climbs to the very top. In section 15, the player is presented with four vertical moving blocks that the player can wait for them to go back up or just run past them. In addition, one of these enemies is out of the screen, but the design of the cave indicates the space where he will be moving.

All of these sections are randomly selected to be presented next to the player except for the starting section that is always presented to the player at the beginning of the game. Additionally, these sections connect at the same height, so any two sections can connect perfectly.

If the player touches an enemy at any point in time, he will lose the game and is forced to start the game over. Moreover, the coins do not signify anything besides their value being displayed during gameplay. The game itself also includes a save system that stores the information about the max score and coins the player has over the various game sessions. This save system can also be easily expanded to save any other value the developer wants.

Finally, this platform game is made using the game engine unity3d [69]. Unity is a well documented and easy to use cross-platform game engine where almost anything can be done. The engine also gives the options for developers to include additional packages to improve or include something more specific for the product they are creating. One of these packages includes the ml-agents packages, allowing developers to quickly implement machine learning approaches for the game's AI or other purposes. This package was already used in the preliminary work 3.1 and will also be used in this dissertation.

## 3.4 Game modification

The first step to develop the proposed solution is to make some modification to the open-source game. Although most of the game is already developed and could be shipped as a complete game, some problems arise for implementing machine learning. Furthermore, some design decisions relating to the game sections and the movement should need a bit of work to tune or change altogether.

### 3.4.1 Life System

The first modification to be made in the game was for it to include a life system. Initially, when the user touches an enemy obstacle, he will lose the game and is forced to start the level over. A three life system will be implemented to allow the player to have more than one possibility to overcome that section and reach new scores. Each of the three lives can be lost in the same way the player can lose the game by touching the water or the obstacles. However, when the player loses a life, he will be put again at the start of the section instead of at the start of the game and

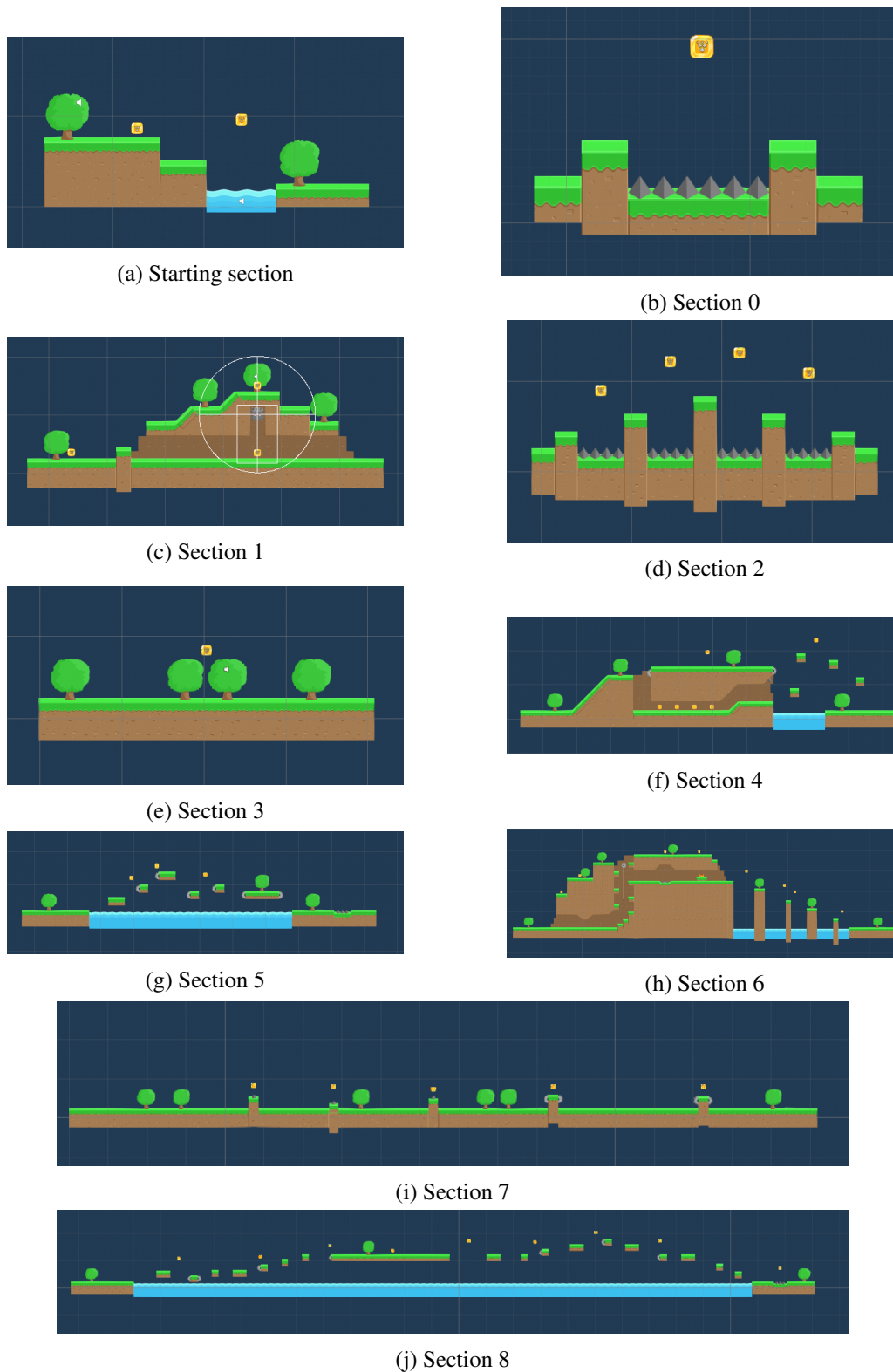


Figure 3.7: Sections 0-8 and starting section

retain the current score. Additionally, if the player loses all three lives, he will lose the game and will be forced to start from the beginning. These lives cannot be restored in any way and are lost

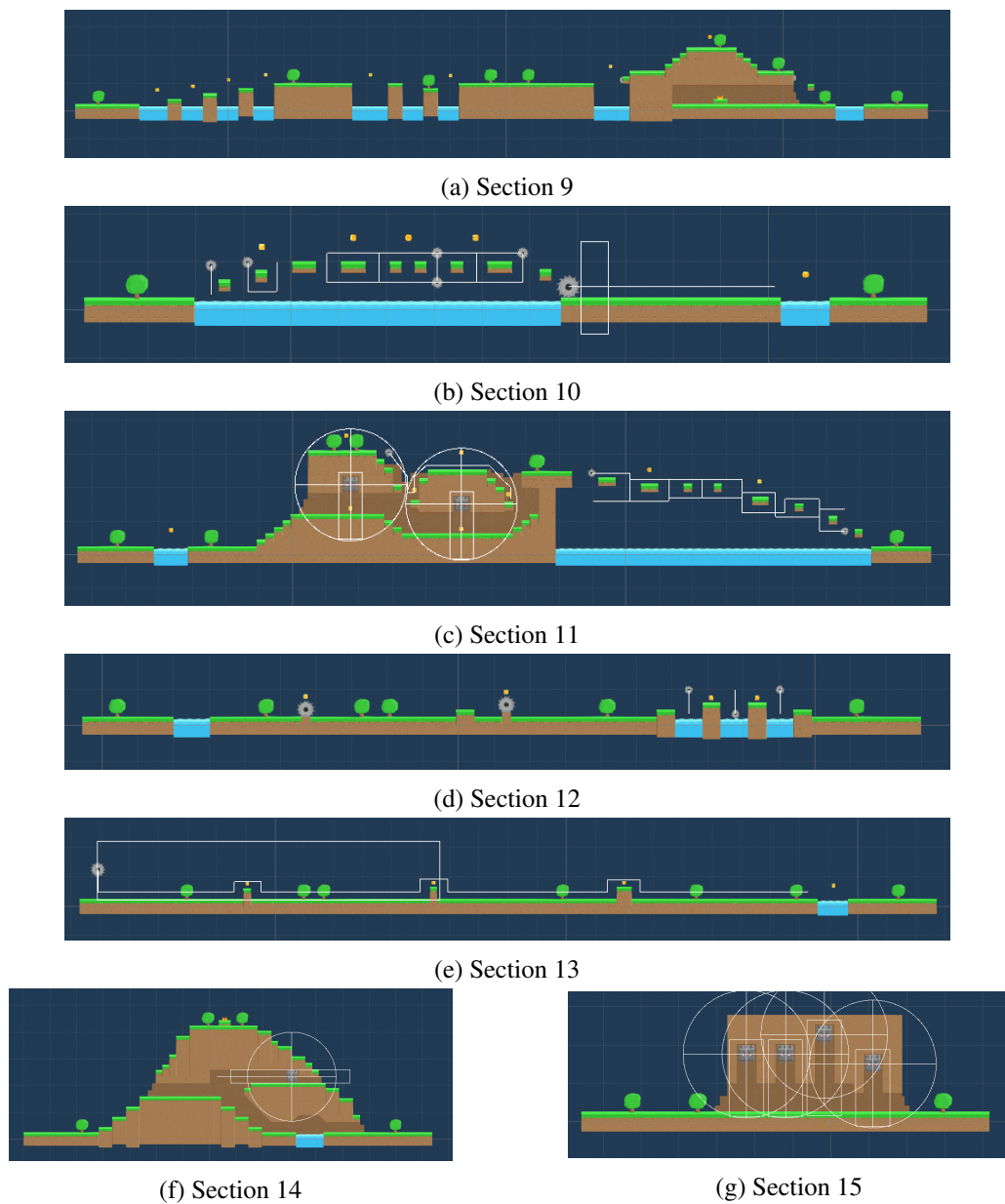


Figure 3.8: Sections 9-15

permanently during that run.

This system should be visible at all times to the player during the gameplay in order for him to have a different strategy when presented with different amounts of lives. Although it may not be guaranteed that the player will have a different strategy when in low lives, some results should be visible. For example, careless gameplay for when the player does not care about their lives and just wants to go as fast as possible with risky decisions; Or more conservative gameplay, in which the player takes more time to make decisions and is more careful of each step it takes when faced with a low amount of lives. This life system can also be seen in other games like in recent Super Mario games [46], where there is a checkpoint-based system with a life system. The player can collect

the checkpoints to indicate where the character will respawn when it dies while also removing one life from the life system. Similarly, the life system modification will allow the player to collect checkpoints by reaching the end of the sections.

### **3.4.2 Time system**

Initially, the game's coins do not have a defined purpose for the game other than collecting them and being displayed on the screen during gameplay. Additionally, if the player does not move or has prudent behaviour not to lose, the game does not end. This situation is not ideal in game design because the player is not pursuing the game objectives. The player should be actively trying to move to the right, and to guarantee that the user has an additional motivation for this, a time system should be implemented.

The time system should be an in-game clock that starts to tick down from the start of the game until the very end. After each run, the clock should reset and start over with the same duration. The total time should be about three to five minutes to allow the player to overcome at least five different sections. Additionally, the coins should also be incorporated with the time system to give them more purpose to their existence. Each coin should give the player, when collected, around one extra second to the time remaining. This timer should also be visible to the player at any time during the gameplay. If the timer reaches the end, the player will lose the game and be forced to restart the game.

This time system aims to give more agency to the coins and pressure the player to reach the furthest right in the fastest time possible. At the start of the run, the player will not have the urge to go fast since he has time to complete the challenges he faces. However, as time reaches its end, the player should feel more pressure, making more risky decisions. On the other hand, if the player already has some experience in the game, he will know that to achieve a new record, he will have to go as fast as possible from the start of the game and collect as many coins as possible.

Finally, the system also contributes to distinguishing different strategies implemented by the player during the gameplay. This modification can help the adaptivity system distinguish and adapt the game to better fit each player's strategy.

### **3.4.3 Additional modifications**

Other problems are also present in the game that should be tweaked or changed altogether. These include changes to the movement system of the game and also the level design of the sections. These modifications will improve the player's gaming experience while also trying to simplify some parts that could be troublesome during the implementation of the machine learning algorithms.

Firstly, the roll movement of the player should be removed to create a more simplistic and more intuitive movement system overall. During the first playthrough of the game, it was not evident what the purpose of the roll movement was, and only after looking at the code the meaning of this movement became apparent. This character revolution should be removed to eliminate

this ambiguity from the player's gameplay, and the overall movement of the character should be reworked to be faster and more fluid. Initially, the character movement feels very sluggish, and the player takes a long time to reach the max velocity needed to overcome some challenges. The movement values of the character can be easily changed in the unity inspector (Figure 3.9), and the code can also be changed to create a responsive movement system. Additionally, this change should also help machine learning by removing one more movement option.



Character Details	
Max Run Speed	9
Run Smooth Time	5
Run Speed	4.5
Walk Speed	1.75
Jump Strength	12

Figure 3.9: Movement values in Unity inspector

Secondly, some sections have challenges that require a significant amount of skill, and even when played correctly, can cause the player to lose, causing frustration. Section 13 (Figure 3.8e) includes a moving saw that chases the player until the end of the same level. This saw moves extremely fast, and if the player did not start the section with enough speed, he would not be able to survive. For this reason, the movement speed of the moving saw should be reduced to allow the player to have an easier time finishing this section.

#### 3.4.4 Summary

These proposed modifications work to improve the gaming experience and solve some problems that the game had. Additionally, they will also help the machine learning algorithms by creating more distinct ways to play the game. Section 3.4.1 explores a new system to give players more opportunities to overcome challenges they have not completed. Furthermore, section 3.4.2 proposes a system to create finite playthrough and give more meaning to players' time and decisions during gameplay. Finally, section 3.4.3 presents additional modifications to the game that will improve some already implemented game system's.

### 3.5 Player simulation

After the game modifications, the next phase is to create a player simulation system similar to the preliminary work. As seen in the preliminary work (sections 3.1), the adaptivity requires a considerable amount of training data to have a good algorithm performance. Even with the simulated players and the time constraint to complete the preliminary work, the algorithm did not train for enough games and needed more episodes to converge to a good result. In the same context, this dissertation will choose to implement a player simulation since real users would not yield a sufficient amount of data.

Instead of creating hardcoded artificial agents, as in the preliminary work, this dissertation will explore the topic of player simulation to copy as closely as possible real players. For this purpose, the algorithm GAIL[28](section 2.2.4) will be used to train the simulated players since it allows for the training of reinforcement learning agents to play games. This algorithm allows the use of recorded expert demonstration of the state and action performed to train the agent more rapidly and imitate the recorded user. Since each user plays the game differently, even with comparable personalities, each player will have a distinct simulated agent associated with it. Essentially, each user will create expert observations from their gameplay, and with these observations, one agent will be trained to copy this user as closely as possible. Then with each trained agent, more games could be played, and the adaptivity system would have enough data train.

Similarly to the preliminary work, the unity ml-agents package should be used to create a link between the game and a python environment. This package will collect all the data related to the games observations and rewards and will transmit this information via a socket connection to the python ml-agents package API. From here, and like in the preliminary work, the python framework Stable Baselines<sup>2</sup> will be used to create the machine learning agent and the gym environment.

The observation and the action the player can take should be defined for good documentation and utilisation of the GAIL algorithm. In terms of observations, they should be composed of gameplay frames, captured and saved as soon as possible. The frame's capture should only contain the essential data, for example, the floor, walls, enemies, coins and chests, to remove cluttering with useless information like the background. Additionally, to provide more precise information about the scene elements, the image should be colour-segmented by the game elements. This segmentation is essential since the game is very colourful, and the same game component can be represented by a different colour, which could confuse the algorithm. Furthermore, to provide more information about the current state of the game, additional observations can be made, for example:

- The current amount of coins collected and the total the player has.
- The current state of the life system and the time system.
- The state of the game's score.
- The position of the player in the section and its velocity.

In terms of the action the agent can choose, these include: doing nothing, moving to the right, moving to the left and jumping, while also being possible to jump to the right and jump to the left. These five movement options are all the actions the agent can decide to use at every frame, and they can be considered discrete actions and labelled from 0 to 5.

For the architecture of the GAIL algorithm, the generator network should be composed of several convolutional layers to process the information from the game's frames, followed by a flatten layer where the other inputs are also padded. Then the rest of the architecture should

---

<sup>2</sup><https://stable-baselines.readthedocs.io/>



follow a standard network design for machine learning. The figure 3.10 shows the idea behind the network architecture. The discriminator should have a similar layout, with the addition of the action chosen by the generator, and the output is if the policy is the ground truth or is the generated one.

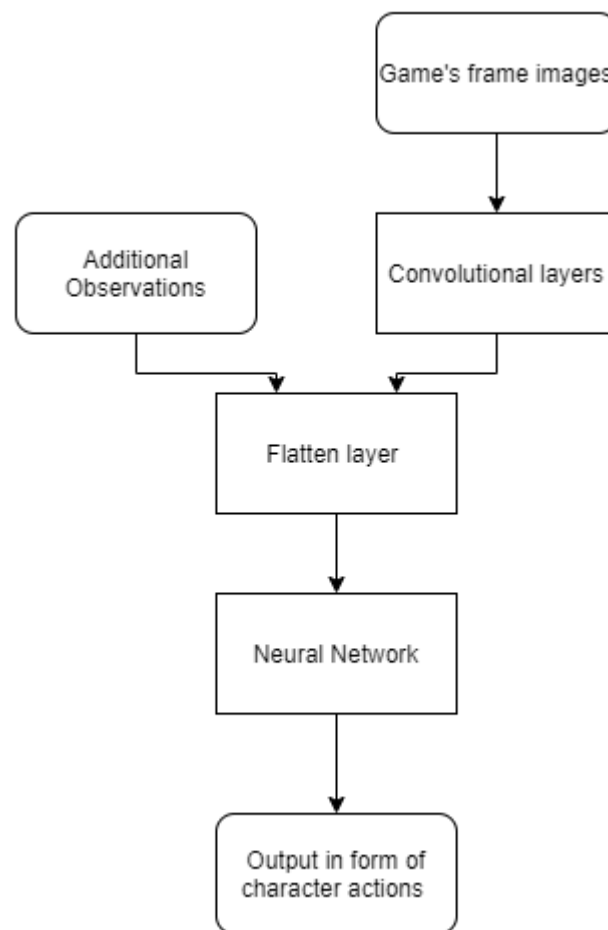


Figure 3.10: GAIL suggested network architecture

After the definition of the Gail algorithm, the performance and efficiency should be tested and evaluated on a small scale to prove its usefulness in creating simulated player's clones. Then, there will be a need to collect expert demonstrations for real players. For this purpose, the following architecture represented in figure 3.11 should be developed to collect all the player data. This data collection model includes a data recording made in python using the python ml-agents package to retrieve the observations and actions the user is making during gameplay. After the gameplay is finished, the data is sent to a centralised server, simple file storage or something more complex to collect the information related to that user. Afterwards, several new agents will be trained using the GAIL algorithm for each of the user data.

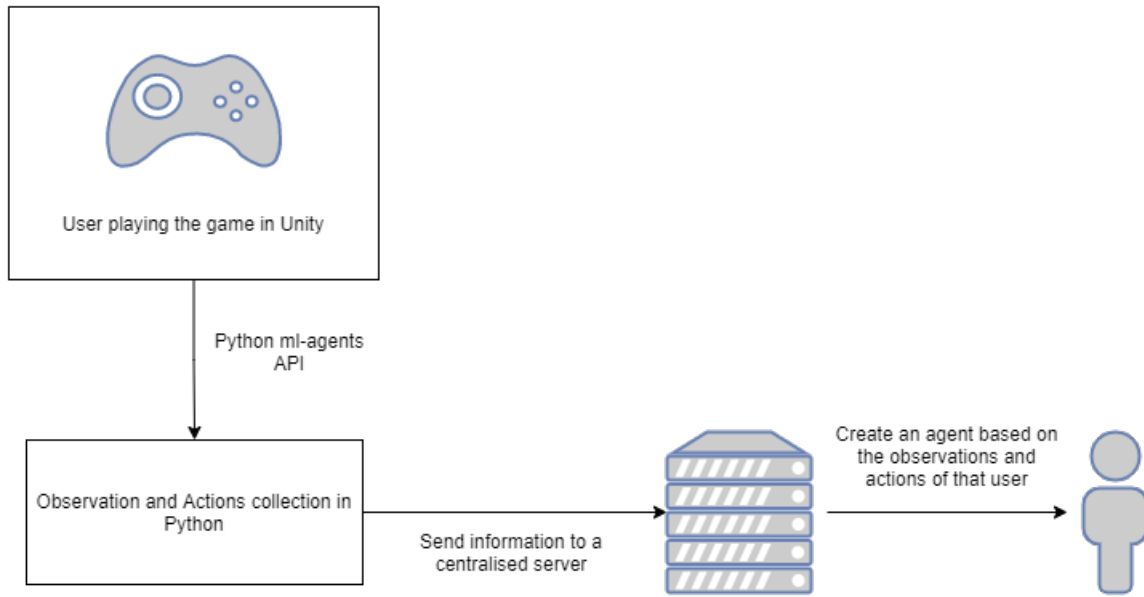


Figure 3.11: Expert demonstration data collection architecture suggestion

### 3.6 Adaptivity System

Before the adaptivity system can be implemented, several problems must be solved, and other definitions and approaches must be defined. The first problem that arises is the reward function for the episodes of the adaptivity algorithm.

Since the adaptivity system is supposed to be trained using reinforcement learning, the observations of the environment will be generated using the simulated players proposed in the previous section (section 3.5). However, these agents only decide what action should be performed at each state and do not give an associated reward for that action. An inverse reinforcement learning algorithm should be applied to help to create a reward function for the algorithm. Since an inverse reinforcement learning algorithm also needs to receive expert demonstrations, these observation-actions states can be generated using the simulated agents or using the user's expert demonstrations. Since the user data should have a relatively small size, an algorithm similar to GAIL (ideally AIRL) should be used. On the other hand, if using the simulated agents to generate the data, a simpler algorithm of inverse reinforcement learning can be used. However, the algorithm's reward is associated with the observation of that state and the correlated action that the agent decided. This reward does not fully represent the user's gaming experience but instead gives a clue of it and can be used in the final reward function.

The reward for the adaptivity system should represent the overall gaming experience the user is feeling during gameplay. Using only the reward generated by the inverse reinforcement learning algorithm is not enough to characterise the gaming experience, and so the real user should give additional feedback on what he thinks improves his experience and what strategies and elements he enjoys the most. For this purpose, a small questionnaire can be introduced during the recording

of the expert demonstration. This questionnaire can appear at the end of each section or at the end of that run. If at the end of each section, the game flow of the player would be disrupted and potentially lead to a bad gaming experience. Therefore, the questionnaire should be present at the end of each run.

This questionnaire should consist of interrogations relating to:

- The various enemies in the game. There should be several questions relating to each type of obstacle: statics, moving and water.
- Relating to what strategies the player prefers to implement. These can include some generic tactics, for example, collecting all coins from a section, going as fast as possible, playing carefully.
- What is their opinion on the different challenges presented by the sections. For example, some sections offer a more platform challenge while others focus more on moving enemies and dodging obstacles.
- What is the preferred objective, collecting coins, having a new score, facing a good challenge or a mix of everything.

The answers to this questionnaire can then be used to create a reward function in conjunction with the reward of the inverse reinforcement learning algorithm. When the simulated agent is playing the game, and a game element appears to the agent, and he interacts with it, a reward can be associated with it depending on the answers to the questionnaire. For example, if the user prefers to collect all coins in the sections. When the simulated agent collects coins from the section, a reward can be associated with it, and even if he collects all the coins in that section, an additional reward can be given. All of these elements will create the reward function to be used in the adaptivity reinforcement learning algorithm.

Furthermore, the reinforcement learning algorithm for the adaptivity system also needs observations and actions defined to work correctly. For this, each observation should be collected at the end of the section and then an action should be decided by the algorithm. When the game ends, the episode should also end, and then a new one should start with the start of a new run. After each episode, the simulated agent should swap to a different user to keep the variety of players, thus allowing each agent to have a chance at playing the game and training with the algorithm.

In terms of observation for the adaptivity system, they should be composed of past and present information:

- The total amount of coins and the coins collected in this game.
- The total amount of chests opened and the currently open in this game.
- The current game time spent in that game.
- The current amount of lives the player has.

- The total amount of lives the player lost and the amount the user lost in this game. Also, the player's total amount of lives lost to each type of enemy, moving, stationary and water, and the amount for this current game.
- The player's best score and the current game score.
- The number of times the player backtracked in the game.
- The number of jumps the player made that game.
- The average velocity of the player in that game and over multiple games.

As referenced in the section 2.1.3, the adaptivity system can alter several game elements to improve the overall gaming experience. Although, these elements need to be adapted for the game in question. In this case, the elements that can be altered are:

- The movement values of the character, to create a different character control experience.
- Alter the number of coins and chests in each section.
- Change the structure of the section.
- Choose the next section the payer will play.
- Change the speed of the moving enemies.
- Alter the number of static enemies in each section.
- Alter the number of lives the player starts with.
- Alter the total amount of time the player starts with.
- Alter how much each coin gives to the timer.
- Alter how many coins each chest gives.

Although all of these options are valid for creating the adaptivity system, for the purpose of this dissertation, the action the algorithm will have available is the choice of the next section. All the other options have their complexity that should be explored, and for simplicity, this decision was made. The action the algorithm is allowed to make is the next section the player will face from the sixteen sections already created (the starting section cannot be selected). Therefore, the action space is discrete, and the actions can be numbered from 0 to 15.

The framework to be used for this adaptivity system should be similar to the preliminary work. For the algorithm, PPO should be used for its stability, which is implemented in the framework Stable Baselines. As in the player simulation (section 3.5), the python ml-agents package should be used to retrieve the observations from the environment. The network architecture can follow a simple multilayer perceptron network since there is no need to handle images in this case. The

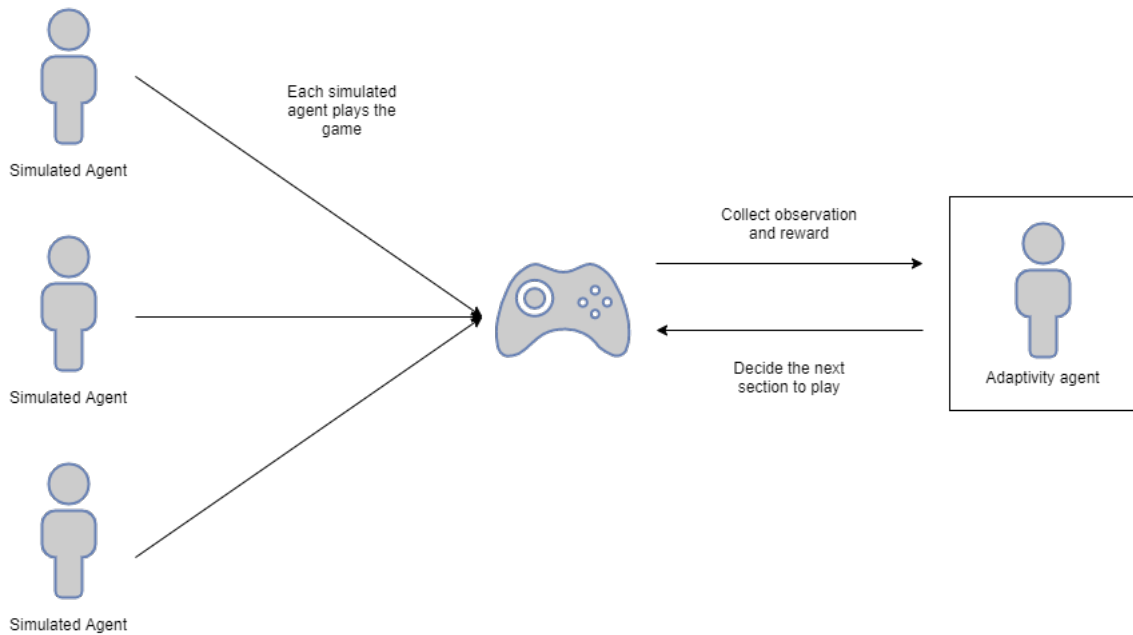


Figure 3.12: Adaptivity system proposed structure

adaptivity system proposed structure can be seen in figure 3.12, which briefly summarises this section.

After the training of the adaptivity agent, the game can then include the adaptivity system and be tested with real users to test its effectiveness at improving the gaming experience and creating the adaptive game.

### 3.7 Summary

In this chapter, the proposed solution was presented, and its implementation was discussed in detail. The first step of the solution is to select the actual game to be implemented. For this purpose, the section 3.3 discussed the game selected and what main elements compose the game. Then, in section 3.4, several modifications were presented to solve some problems that the original game had while also proposing new systems to improve the diversity of strategies. The following section 3.5 discusses the player simulation system to create that will help the adaptivity system since it requires a large amount of data. The final section (section 3.6) presents the proposed adaptivity system, the main focus of this dissertation, and the general structure to be followed.



## Chapter 4

# Development and workflow of game adaptivity system

In this chapter, the developed work will be presented, and the decisions and changes made will be explained to fulfill the objectives described in section 1.2. The final adaptivity system created was incorporated in the game, and the adapted game automatically chooses the next section to be presented to the player.

Section 4.1 will explain how the life system developed work and how it looks during gameplay. The following section (section 4.2) will present the time system implement in the game and how it interacts with the game. In the next section, section 4.3, the player simulation system will be explained, and the principal problems and changes will be justified. Then several solutions and attempts at resolving the problems found will be explored. Finally, the last section (section 4.4) will present a simple solution to solve the problem from the previous section, which will also be used in the adaptivity system. Then, the adaptivity system will be explained, detailing all the essential features implemented to create an adapted game version.

### 4.1 Life system

The development of the life system followed the structure of the proposed solution presented (section 3.4.1). A variable was introduced in the Red Character script to maintain the current amount of lives. This variable is decremented when the user loses to an enemy. A visual representation was then created to inform the player about his current amount of lives. This representation can be seen in figures 4.1a and 4.1b and is illustrated by three hearts representing an individual life. If the heart is filled with red, then the player has one life, and, consequently, with three filled hearts, the player has three lives.

Additionally, after the player loses one of his life, he is respawned at the start of the section. Since all sections have a similar start and can be connected between them, the player is placed at a default x 2 position and y 4 position. These default values (figure 4.2) can also be changed

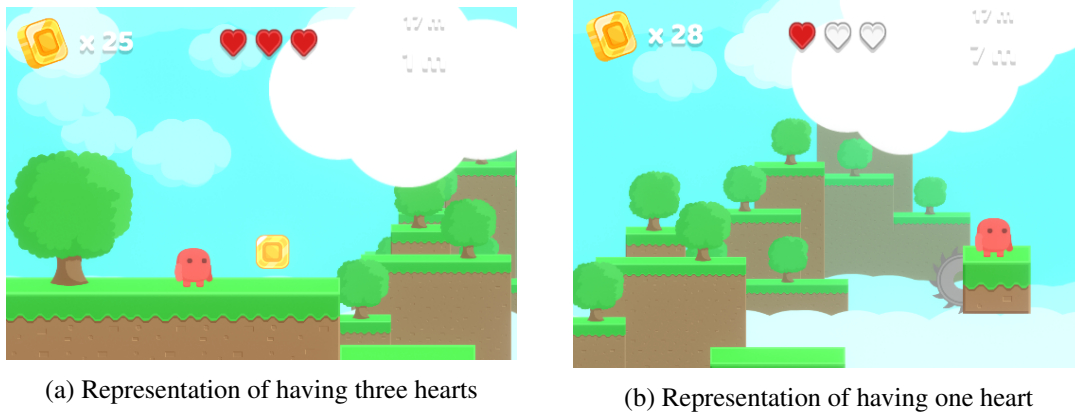


Figure 4.1: Examples of the heart display

if a new section is created. For example, the starting section needs to put the player in a higher position than usual if the player loses, so the values must be x 8 and y 10.

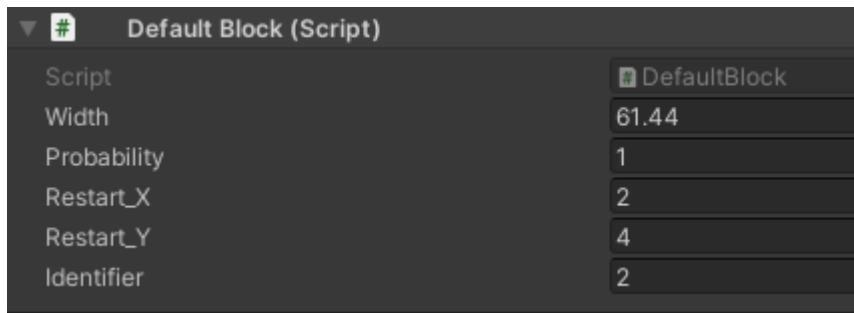


Figure 4.2: Unity inspector section restart position values

## 4.2 Time system

The development of the time system also follows the proposed design in section [proposed time system]. A timer variable was created in the Game Manager script that at each update step of the engine subtracts the delta time (the time passed since the last frame). If this time is less or equal to zero, it will trigger the end of the game. Additionally, if the player collects a coin during gameplay, the variable time will add a value. The starting value of the timer and the amount of time each coin adds can be modified in the Unity Inspector under the object Game Manager. These values can be seen in figure 4.3 and default to 300 and 1, respectively.



Figure 4.3: Timer values in Unity inspector



Furthermore, the timer is also visible to the player during gameplay and can be seen at the top right of the screen. The figure 4.4 also shows how it looks in the game. The design of the timer follows a similar design to the other game elements already created in the game.

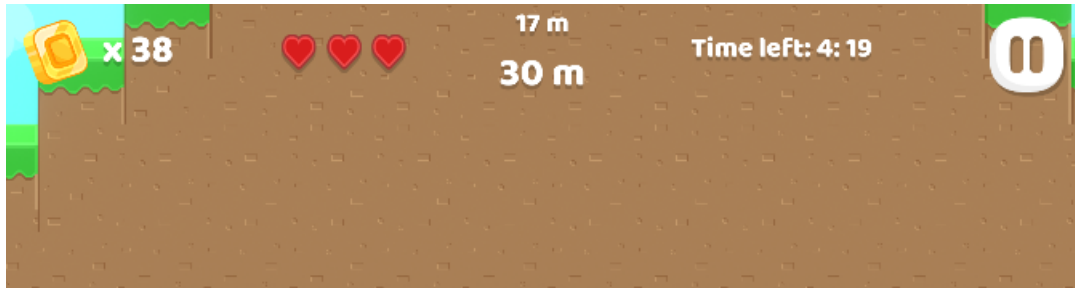


Figure 4.4: Timer display during gameplay

### 4.3 Player simulation

The first iteration of the player simulation tried to follow the proposed solution for the system (section 3.5) as closely as possible. Therefore, after installing the ml-agents package in the Unity editor, a player watcher script was created in a new game object, “Player AI”. In this script, the class extends the Agent class<sup>1</sup> from the ml-agents package and must implement the following methods:

- **Initialize.** This method is called one time when the agent is first enabled.
- **CollectObservations.** This method collects the observation of the agent for a step.
- **OnActionReceived.** This method specifies the action (discrete or continuous) the agent should have in that state.
- **Heuristic.** This method allows for a creation of a custom action the agent can take.
- **OnEpisodeBegin.** This method is called at the start of every episode.

For the player agent class to collect observations, they must first be recorded in variables. Therefore, in the GameManager script, several variables are created to hold the values of the observations. Then, these variables can be used to collect the environment’s information and compose the agent’s observations. The following list describes the simulated agent observations, which follows the proposed observations approximately:

- The player’s total amount of coins collected over the games and the current amount of coins collected in this game.
- The current amount of lives the player has in this game.

<sup>1</sup><https://docs.unity3d.com/Packages/com.unity.ml-agents@2.1/api/Unity.MLAgents.Agent.html>

- The current game score and the best score the player has.
- The current game time.
- The current section identifier and the X position of the character in that section.
- The current character's velocity in the X-axis.

Additionally, the simulated player also needs to collect the game's image. For this collection to work inside Unity, a new camera was created that follows the character the same way as the standard camera. However, this new camera has a black background and filters all the unnecessary game elements, for example, the particle system and the environment aesthetics. Since the original game does not assign each game element to a Unity layer, every section of the game was modified to assign each element to a specific layer. The following list describes the layers created:

- *Ground*. The ground layer includes all the blocks preventing the player from passing through them and not killing the character. These include the ground, the walls and the ceiling.
- *Character*. This layer represents the player's character. Additionally, the layer Fake Player represents a fake player that will be explained below.
- *Reward*. This layer includes all the coins in the level and also the chests.
- *Enemies*. This layer includes all the stationary and moving enemies of each section.
- *Water*. This layer represents the water present in the sections.
- *Path*. This layer represents the paths the moving enemies can take.
- *Trees*. This layer represents the trees, which are aesthetic elements in the sections.

After the creation of these tags, the camera can filter the game elements by their specific layer. The layers selected include the water, ground, fake player, the rewards, the enemies and the path. A fake player was created to simplify the original character since the red runner has animations for running, jumping, and standing. This fake player has the exact position coordinates of the original character at every frame and is not visible in the standard game. The figure 4.5 demonstrates the vision of the camera in the game.

Following the proposed solution, the image then needs to be segmented by colour, by game element. This segmentation was implemented using a helper package<sup>2</sup>, which uses shaders to segment the image by layer. To use this helper package, the script Image Synthesis needs to be included in the camera inspector (figure 4.6). The resulting final image can be seen in figure 4.7. Each layer is assigned a specific colour automatically by the helper package. This segmented image is then collected by a camera sensor script (from the ml-agent package, figure 4.8) and automatically forward to the agent's observation. Additionally, the camera sensor also scales the image and converts it to a grayscale version.

---

<sup>2</sup><https://bitbucket.org/Unity-Technologies/ml-imagesynthesis/src/master/>

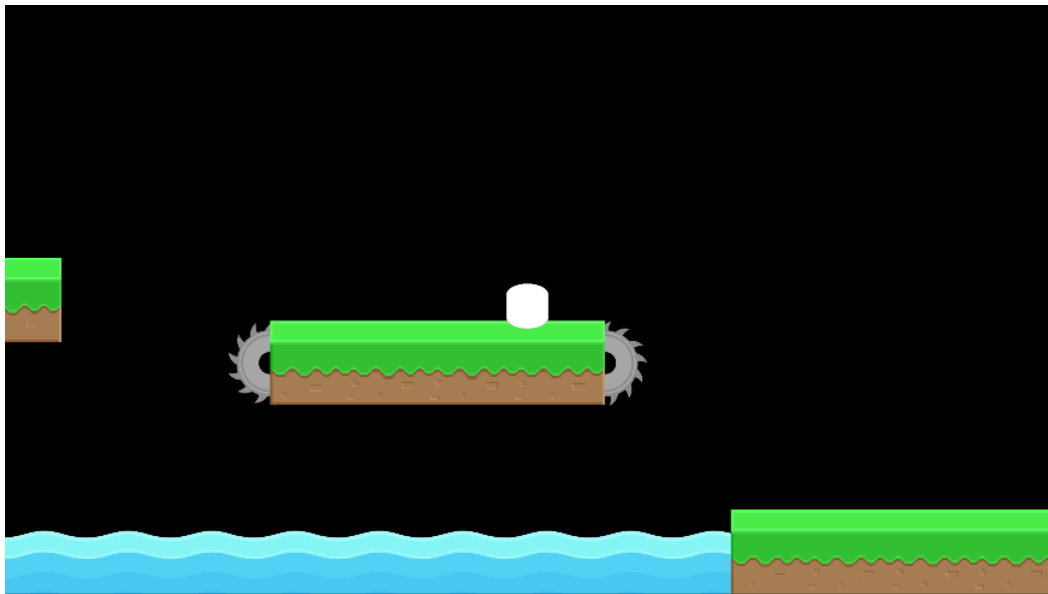


Figure 4.5: Player camera gameplay vision

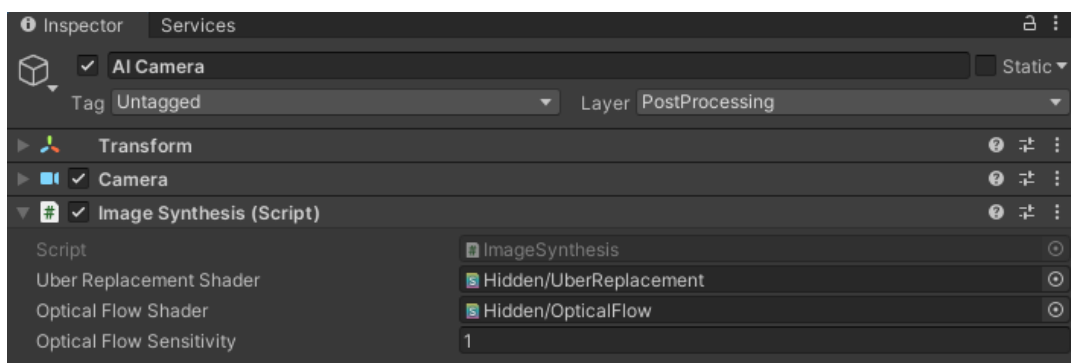


Figure 4.6: Image Synthesis package script

The method *OnActionReceived* was developed to receive one discrete action from six available options: 0 - do nothing; 1 - move to the left; 2 - move to the right; 3 - jump; 4 - move to the left and jump; 5 - move to the right and jump. The movement inputs of the player were initially received by the script Red Character. However, for a correct recording of the real player's input actions, the information is received by the method *Heuristic* in the simulated player script. Furthermore, the original movement system used a cross-platform input method, which caused the character to respond about two frames later. This problem made some precise jumps frustrating since the character would not jump in time. Therefore, the movement input system was modified, and the character movement was automatically translated to a corresponding discrete action. Consequently, this new movement system includes the possibility for the player to perform multiple jumps in a row without releasing the jump button and the ability to change direction in the air without losing speed. Moreover, the movement values were also tweaked to allow the character to accelerate faster.

The next step of the development is the creation of the model and the training environment.

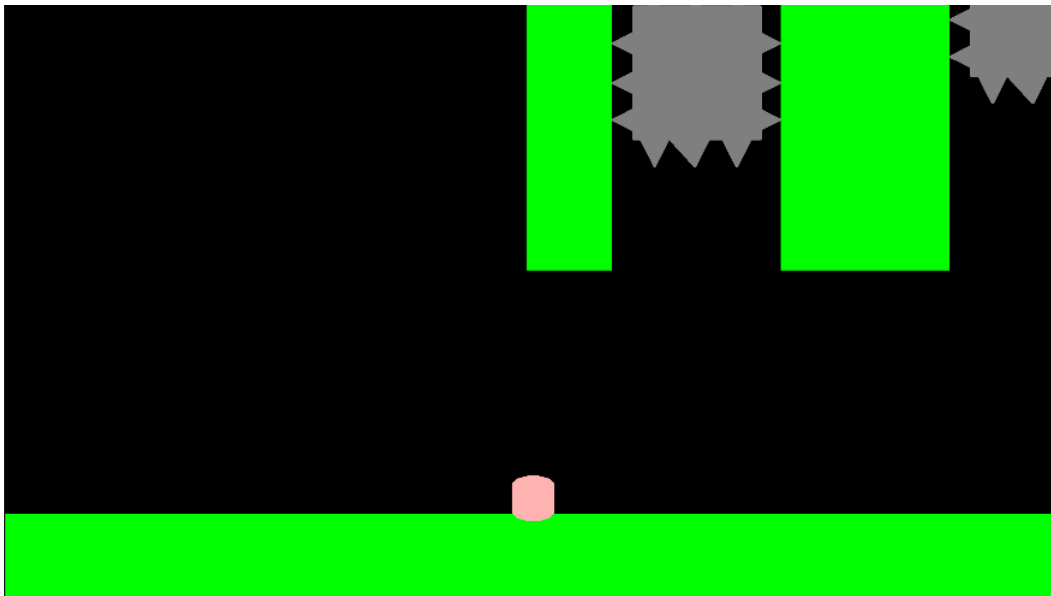


Figure 4.7: Final colour segmented image example

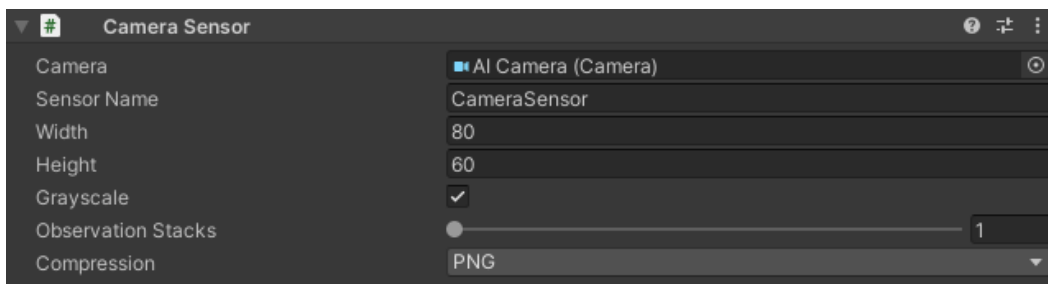


Figure 4.8: Camera sensor script for agent's object

Following what is defined in the proposed solution (section 3.5), the python ml-agent package was used in conjunction with the Stable Baselines package. However, several problems appeared relating to the use of the Stable Baselines package:

- The Unity observations, rewards and actions need to be inside a gym environment<sup>3</sup> class. This class controls the game's episodes and can sometimes conflict with the Unity environment giving undesirable results.
- The algorithm requires a custom network to be defined since the observations are composed of values and images. Furthermore, the GAIL algorithm implementation by the package does not handle images properly.
- The algorithm requires a specific file structure for the expert observations, which is not documented. The recording of these observations also needed to be created by hand since the package did not include an implemented method.

<sup>3</sup><https://gym.openai.com/docs/>

To solve these problems, the Stable Baselines source code needed to be changed or adapted. However, the source code is vast and complex to comprehend, which made it an impossible option for the duration of this dissertation.

Other options were explored, including other open-source implementations of the GAIL algorithm. For example, an adaptation to the repository<sup>4</sup> was created. However, when modifying the network to handle images and observation and output discrete actions, the algorithm showed performance problems and errors that were troublesome to identify. Other problems were also solved using the open-source code, such as recording the expert demonstrations and the definition of neural network architecture. The definition of the neural network can be seen in figure 4.9. This neural network is based on the work presented in the section 2.2.5.

```
def __init__(self, image_shape, state_shape, action_shape):
    super().__init__()
    state_shape = state_shape[0]
    action_shape = action_shape[0]
    h, w, c = image_shape

    self.cnn = nn.Sequential(
        nn.Conv2d(in_channels=c, out_channels=32,
                  stride=4, kernel_size=8),
        nn.ReLU(),
        nn.Conv2d(in_channels=32, out_channels=64,
                  kernel_size=4, stride=2),
        nn.ReLU(),
        nn.Conv2d(in_channels=64, out_channels=64,
                  kernel_size=3, stride=1),
        nn.ReLU(),
        nn.Flatten(),
    )
    self.net = nn.Sequential(
        nn.Linear(in_features=state_shape + 1536, out_features=512),
        nn.ReLU(),
        nn.Linear(in_features=512, out_features=256),
        nn.ReLU(),
        nn.Linear(in_features=256, out_features=128),
        nn.ReLU(),
        nn.Linear(in_features=128, out_features=action_shape),
        nn.Softmax(dim=-1)
    )
```

Figure 4.9: Code for the architecture of the network for GAIL

The best option was using the Unity ml-agents toolkit<sup>5</sup> (and not using only the python API), which is integrated with the unity package. This toolkit allows the use of machine learning algorithms with much ease and without knowing the implementation of the algorithms and is also constantly being updated to enhance the algorithms and their usage by developers. Furthermore, the problems found in the other approach made are not present in this implementation since this

<sup>4</sup><https://github.com/ku2482/gail-airl-ppo.pytorch>

<sup>5</sup><https://github.com/Unity-Technologies/ml-agents>

toolkit is integrated with the Unity engine and is primarily focused on creating artificial gaming intelligence. The already developed class simulated player can also be easily used with this toolkit.

This toolkit allows the usage and training of several machine learning algorithms, including PPO and SAC. In addition, the toolkit also includes several other helpers algorithms and tools to assist in the training of the agents, including:

- Generative adversarial imitation learning algorithm (GAIL)[28].
- Behavioural cloning[68]. This algorithm can be used to help the agent mimic the actions of the expert demonstrations.
- Curiosity reward[36] can be used when the rewards on the environment are infrequent.
- Random Network Distillation[15] can be used when the rewards of the environment are rare.
- Recurrent Neural Networks (using LSTM) to allow the agent to remember past information.
- Self-play[7] allows the agent to learn how to play against versions of itself.
- Curriculum learning. This tool allows the agent to train in a progressively more challenging environment.

For the usage of the imitation learning algorithms, the recording of the expert demonstration was made by including a simple script (*Demonstrations Recorder*) from the ml-agents package in the agent's object, shown in figure 4.10.

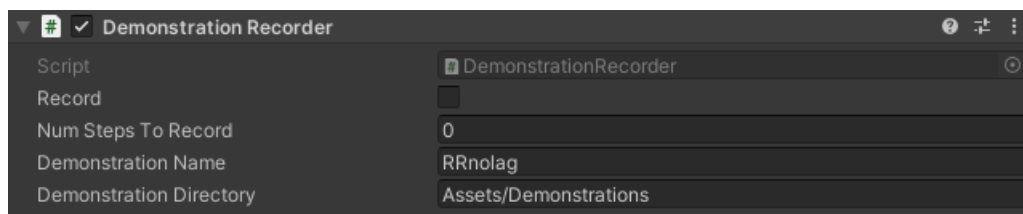


Figure 4.10: Demonstration recorder script for player agent's object

The simulated player agent training was done iteratively using several different configurations and setups available in the ml-agents toolkit. The experiments and results can be found in chapter 5 with a detailed explanation. Furthermore, several changes and modifications were made for assisting and experimenting with the agent's training. These modifications are detailed in the following list:

- **Ray perception sensor.** This sensor is included in the ml-agents Unity package, and only the script needs to be incorporated in the agent's object. This sensor casts a ray cast line from the character and collides with the game elements. Consequently, the game elements need to be classified with a tag for them to be distinguished by the sensor. Additionally, the script also allows the developer to specify which layers the lines should collide with, the total number of lines to ray cast, the angle between them, their length and the size of the collision

sphere of the ray cast. These values can be seen in figure 4.11, and in figure 4.12, an example can be seen of the lines being ray cast during gameplay. Furthermore, this sensor output is automatically included and adapted to be used in the simulated player's observations. This sensor purpose is to help the imitation learning algorithm better understand each state of the environment, such as elements that can be out of the screen but the player has knowledge of them or for situations where the game's frame does not give enough information to describe the current state of the environment.

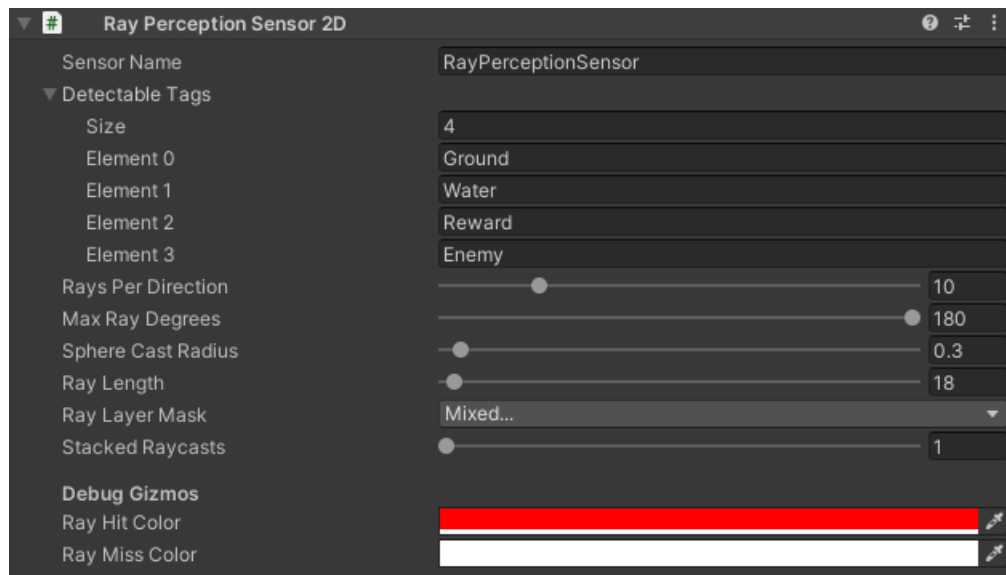


Figure 4.11: Ray cast script values in Unity editor

- **Extrinsic reward.** The purpose of this reward function is to guide the training of the agent. These rewards include:

- A positive reward for moving forward depending on the amount of X position travelled since the last frame. Equation 4.1 represents this positive reward.

$$reward \quad + = \quad \frac{(newXposition - currentscore)}{discount\ factor} \quad (4.1)$$

- A positive reward for completing a section.
- A positive reward for collecting a coin.
- A positive reward for opening a chest.
- A negative reward for a life lost.
- A negative reward for standing still for a specific duration.

These rewards are used in particular experiences, and their values are also modified accordingly.

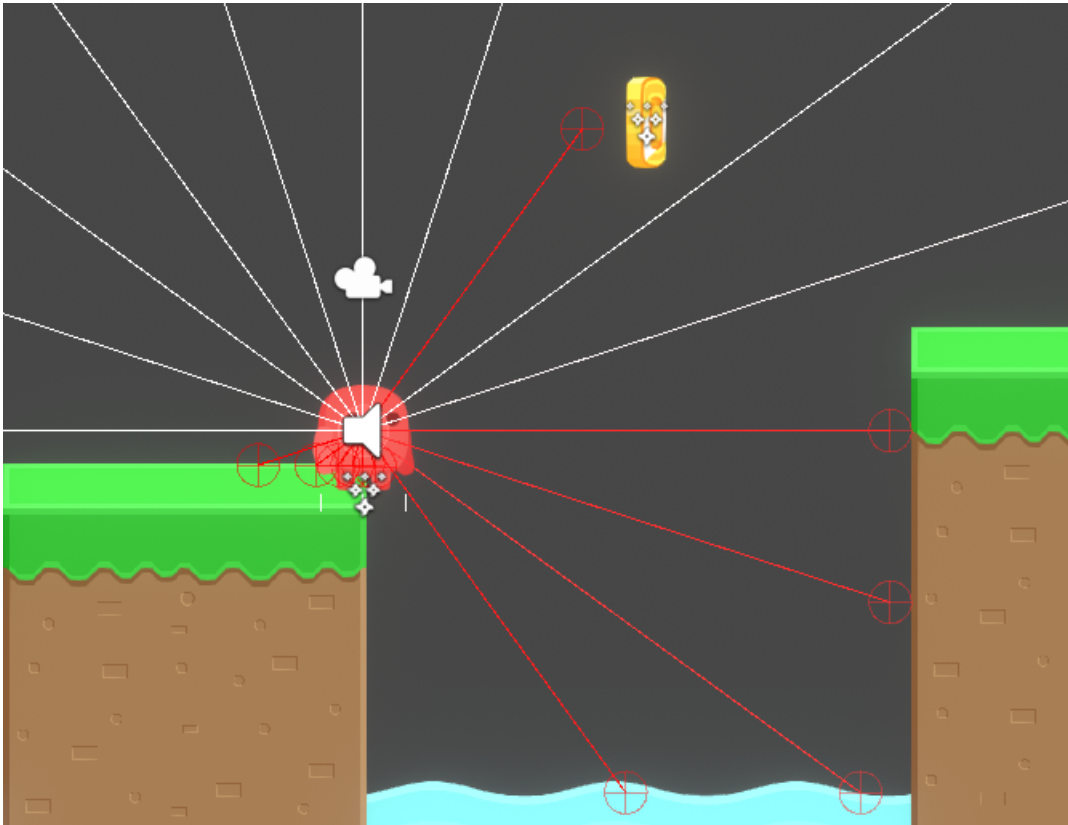


Figure 4.12: Ray cast vision during gameplay

- **Simpler game version.** This modification was included since the agent spent too much time standing still and not progressing, making the training episodes excessively long. Therefore, this game version made the timer have a total of 75 seconds instead of 5 minutes. Additionally, the player only has one life, so the game ends immediately instead of the character being put at the start of the section.
- **Curriculum learning.** For the use of this tool, the game needed to be divided into several different difficulties. In the first iteration, the several sections were combined into four different difficulties:
  - Difficulty 1. Includes sections 0,1,2,3 and 15.
  - Difficulty 2. Includes sections 4,5,7,8 and 9.
  - Difficulty 3. Includes sections 10,12 and 13.
  - Difficulty 4. Includes sections 6,11 and 14.

These difficulties were created based on a personal game design vision after playing the game numerous times, and the training followed the first difficulty to the last, only changing the challenge if the agent was able to receive a particular reward for several episodes. In the next iteration, each section had a difficulty associated, creating an ordered list from



the easiest to the hardest. The following list was created based on a personal game design vision: sections 3, 0, 2, 1, 15, 4, 5, 8, 9, 7, 13, 10, 12, 14, 11, 6.

- **Sections modifications.** Several sections were modified to create more straightforward gameplay and also to facilitate the training of the agent. Some of these modifications are also part of the proposed modifications from section 3.6.
  - The starting section was simplified, and the water was removed because, during the training, the agent lost a majority of times at this section. The figure 4.13 shows how the new starting section looked. Additionally, several collision blocks were added at the left of the original and the new starting section since the agent also lost lives and time going backwards. The figure 4.14 displays these blocks, which are not visible to the player or the agent.
  - The section 13 moving saw movement speed was significantly reduced since this section was tough to complete even when playing perfectly.

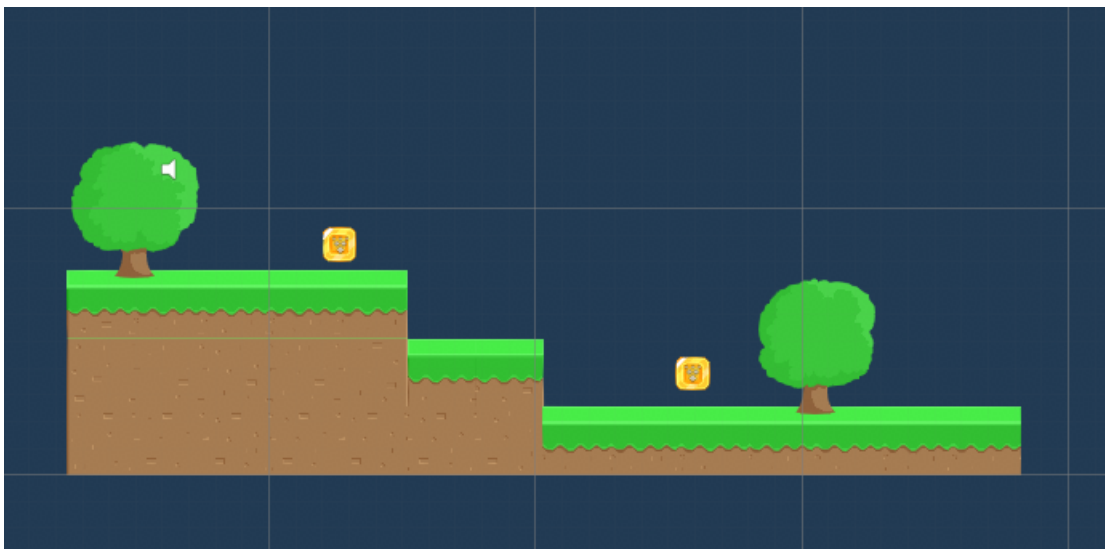


Figure 4.13: New starting section

After all the train sessions and experiments, the simulated player's agent did not prove to be a correct representation of the user's expert demonstrations. Since this agent could not be used to train the adaptivity system, a new approach was used, explained in the next section.

## 4.4 Adaptivity system

Since the player simulation did not yield the expected result, a more simplistic solution needed to be developed. Instead of focusing on developing a simulated player that can play several sections of the game, a simulated personality system was created to produce the results at the end of each section. Essentially, the adaptivity system will need to collect observations at the end of each

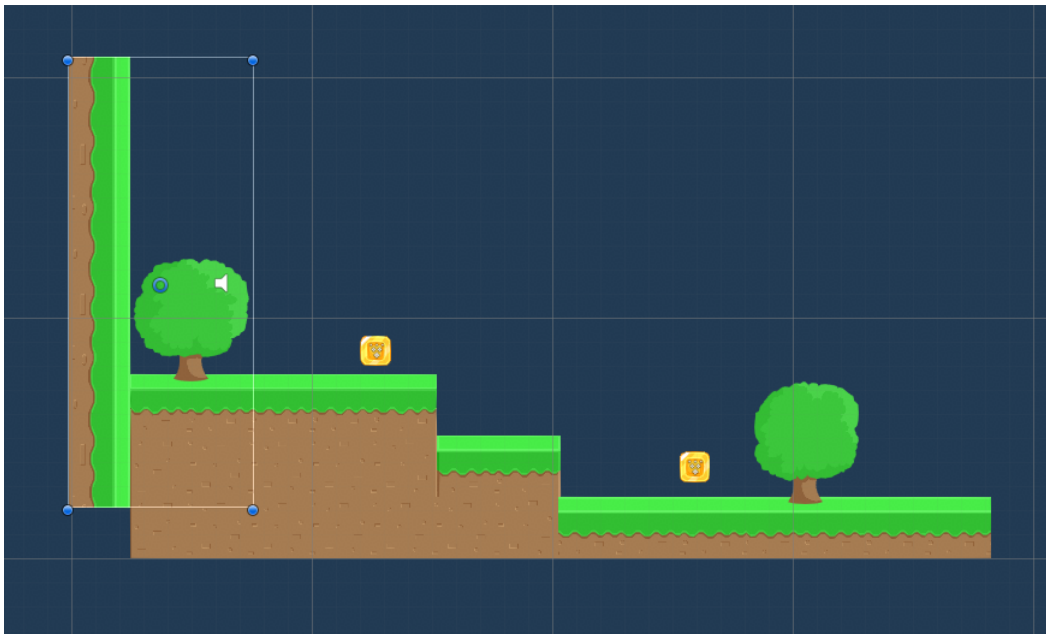


Figure 4.14: Starting section collision blocks

section, so this personality system will generate gameplay values as if that personality played that section.

#### 4.4.1 Personality system overview

The first step in creating the personality system is to build a template personality that can be replicated and modified easily. This template was created using *Unity Scriptable Objects*, allowing the creation of objects with a template format and values that can be quickly changed. The following values, also shown in figure 4.15, compose the structure of each personality object and are used for the observations and rewards of the adaptivity system :

- **Tolerance for repeating sections.** This value indicates how many sections should appear before a section can be shown again. For example, if the first section is 14 and the personality has a tolerance for three levels, the player needs to see three sections different from section 14. After the three sections, the player can tolerate seeing section 14 again.
- **New levels importance.** This value is associated with the value above and signifies how negatively the repeated section impacts the player.
- **Coins importance.** This value indicates how important it is to collect coins for the personality.
- **Coins section.** This variable is composed of sixteen options, one for each section, and each contains a minimum and maximum value of coins the personality can collect in that section, shown in figure 4.16.

Property	Value
Script	PersonalityScriptableObject
Tolerance For Repeting Levels	3
New Levels Importance	30
Coins Importance	0.75
▶ Coins Section	16
Chests Importance	0.75
▶ Chests Section	16
Life Importance	0.5
Life Lost Importance	3
▶ Life Lost Section	16
▶ Life Lost Type Section	16
▶ Time Section	16
Speed Importance	1
Speed Preference	7
▶ Speed Section	16
▶ Jumps Section	16
▶ Backtrack Probability	16
New Score Importance	2
Loss Each Section	3
Concentration Level Preferred	8.5
Concentration Importance	7
▶ Concentration	16
Skill Level Preferred	8.5
Skill Importance	6
▶ Skill	16
Challenge Level Preferred	7.5
Challenge Importance	10
▶ Challenge	16
Immersion Level Preferred	2
Immersion Importance	3
▶ Immersion	16

Figure 4.15: Unity personality scriptable object example

- **Chest importance.** This value indicates how important it is to open chests for the personality.
- **Chest section.** This variable is composed of sixteen options, one for each section, and each contains a minimum and maximum value of chests the personality can open in that section.
- **Life importance.** This value indicates how important it is to maintain lives. Essentially, if the personality has three lives, this importance is given by a factor of three, which is used in the reward calculation.
- **Life lost importance.** This value indicates how much penalty is associated with losing a life. It is used to add a negative reward to the reward function.
- **Life lost section.** This variable is composed of sixteen options, one for each section, and each contains a minimum and maximum values of potential lives the personality can lose in that section.

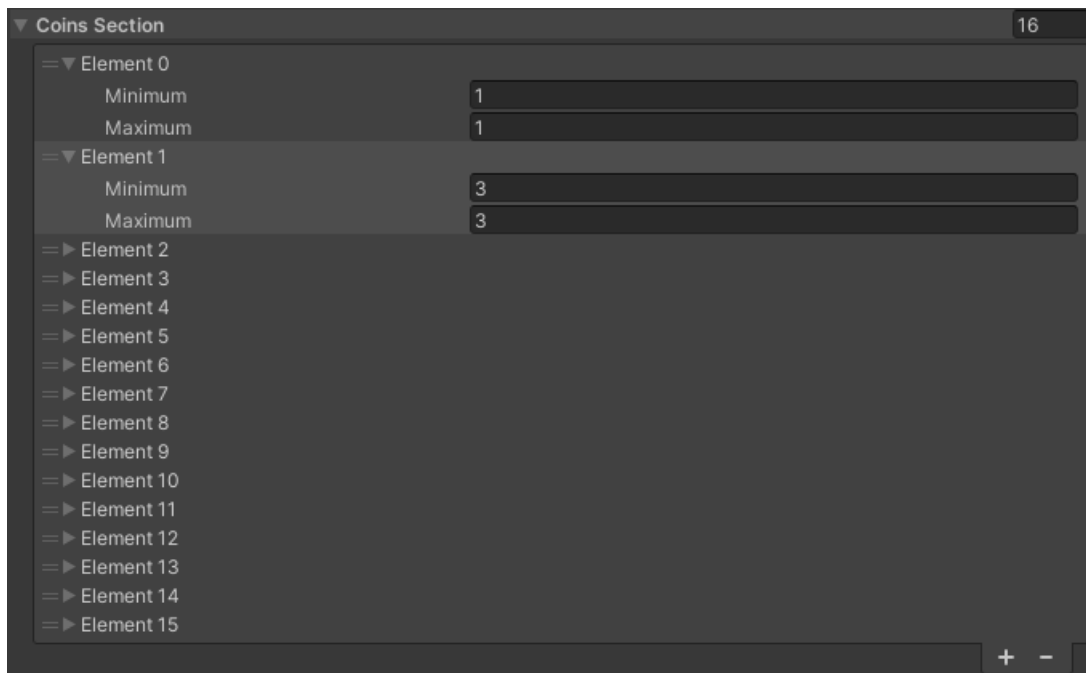


Figure 4.16: Unity personality scriptable object coins minimum and maximum value example

- **Life lost type section.** This variable is composed of sixteen options, one for each section, and each contains three values, one for stationary enemies, one for moving enemies and one for water enemies. These values can be seen in figure 4.17, and they can be any positive number. This variable represents how probable it is for the personality to lose a life to that enemy type. For example, in section 10, the values can be 3 for water, 1 for stationary enemies and 6 for moving enemies, and then a random number is select from 1 to 10 ( $3+1+6=10$ ) and is associated with the corresponding enemy (if the number is [1,3] then he will lose a life to water, if [4,4] then to a static enemy and if [5,10] to a moving enemy).
- **Time section.** This variable is composed of sixteen options, one for each section, and each contains minimum and maximum floating points values for the possible velocity the personality can achieve in that section.
- **Speed importance.** This value indicates how important it is to go at the desired average speed for the personality. This value is used for the reward calculation.
- **Speed preference.** This value indicates what the preferred average speed is for that personality. This value is used for the reward calculation.
- **Speed section.** This variable is composed of sixteen options, one for each section, and each contains minimum and maximum floating points values for the possible speed the personality can perform in that section.

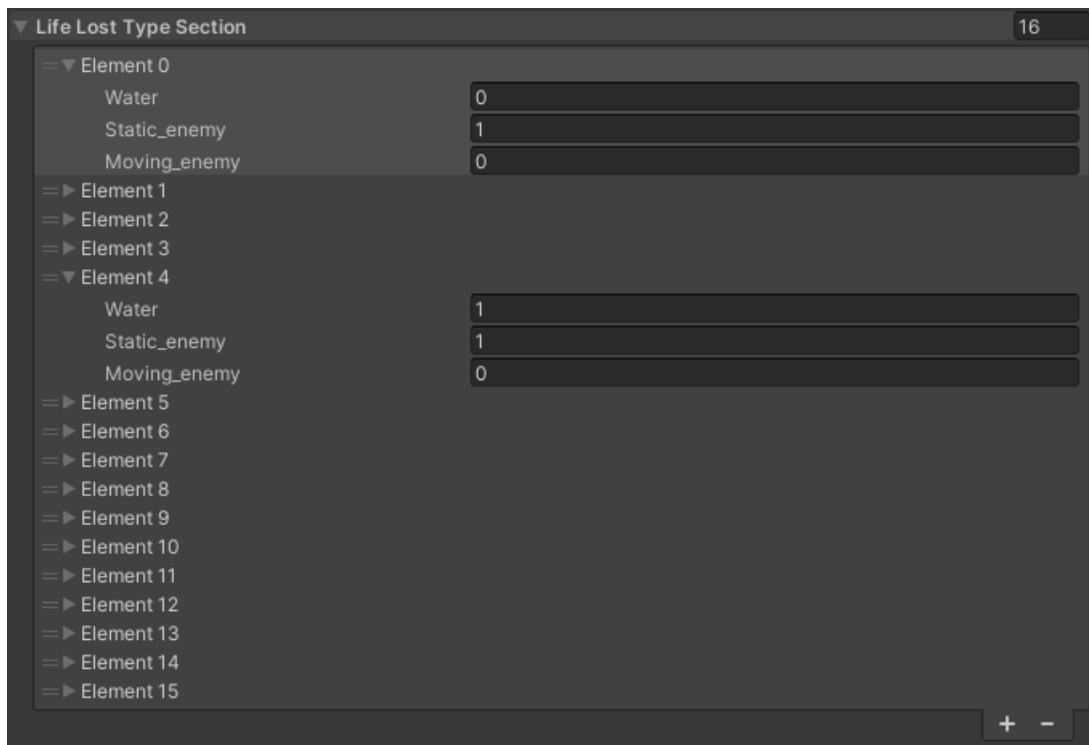


Figure 4.17: Unity personality scriptable object type of enemy example

- **Jumps section.** This variable is composed of sixteen options, one for each section, and each contains a minimum and maximum values of potential jumps the personality can perform in that section.
- **Backtrack probability.** This variable is composed of sixteen options, one for each section, and each contains a percentage change for that personality to perform a backtrack move in the section. In this game, a backtrack is considered when the player goes back 10 Unity units, a sufficient amount of space to be considered a player's decision to move backwards.
- **New score importance.** This value indicates how important it is to achieve a new score for the personality.
- **Concentration.** This variable is based on the game flow concept of concentration (section 2.1.5), and its meaning was adapted for the context of this game and the use in the personality system. This variable signifies the number of things to do in the section relating to the personality's objectives. It is composed of sixteen options, one for each section, and each contains a value from zero to ten representing how much concentration that section contributes to the personality, shown in figure 4.18.
- **Concentration level preferred.** This value indicates what the preferred level of concentration for the personality is. Essentially, this value indicates at what level the concentration must fluctuate between, which is explained better in the reward calculation explanation.

Element	Concentration Value
Element 0	2
Element 1	1
Element 2	4
Element 3	0
Element 4	7
Element 5	4
Element 6	7
Element 7	1
Element 8	10
Element 9	5
Element 10	6
Element 11	9
Element 12	1
Element 13	0
Element 14	7
Element 15	0

Figure 4.18: Unity personality scriptable concentration values example

- **Concentration importance.** This value indicates how important it is to come close to the preferred concentration level.
- **Skill.** This variable is based on the game flow concept of skill(section 2.1.5), and its meaning was adapted for the context of this game and the use in the personality system. This variable signifies the difficulty associated with that section. It is composed of sixteen options, one for each section, and each contains a value from zero to ten representing how much skill that section requires, although all personalities can complete that section.
- **Skill level preferred.** This value indicates what the preferred level of skill for the personality is. Essentially, this value indicates at what level the skill must fluctuate between, which is explained better in the reward calculation explanation.
- **Skill importance.** This value indicates how important it is to come close to the preferred skill level.
- **Challenge.** This variable is based on the game flow concept of challenge(section 2.1.5), and its meaning was adapted for the context of this game and the use in the personality system. This variable signifies the challenges associated with the personality's objectives for each section. It is composed of sixteen options, one for each section, and each contains a value from zero to ten representing how much challenge that section contributes to the personality.
- **Challenge level preferred.** This value indicates what the preferred level of challenge for the personality is. Essentially, this value indicates at what level the challenge must fluctuate between, which is explained better in the reward calculation explanation.
- **Challenge importance.** This value indicates how important it is to come close to the preferred challenge level.

The immersion was also considered but was not included in the personality system since it was too hard to define and distinguish between personalities. These values are used for both the adaptivity system observations and reward function. The reward for the adaptivity system is calculated in the following way:

- The previously completed sections are analysed, and if the current section was seen repeated in the last Tolerance for repeating sections number of sections, then a negative reward is given out according to the equation 4.2.

$$sectionReward - = newLevelsImportance[personality] \quad (4.2)$$

- A random number of coins is generated between the minimum and maximum value for that personality and section. Then a positive reward is given according to equation 4.3

$$sectionReward + = coinsImportance[personality] * randomCoinsSection. \quad (4.3)$$

- A random number of chests is generated between the minimum and maximum value for that personality and section. Then a positive reward is given according to equation 4.4.

$$sectionReward + = chestImportance[personality] * randomChestsSection \quad (4.4)$$

- A random number of lives lost is generated between the minimum and maximum value for that personality and section. Then a negative reward is given according to the equation 4.5. Additionally, for each life lost, an enemy is associated with that life lost and recorded for later use in the adaptivity system observations.

$$sectionReward - = listLostImportance[personality] * randomLiveLostSection \quad (4.5)$$

- If the game ends because the player does not have any more lives or the time has ended, a positive reward is given if the player achieves a new max score according to equation 4.6.

$$sectionReward + = newScoreImportance[personality] \quad (4.6)$$

- A positive reward is given for having lives according to equation 4.7.

$$sectionReward + = currentLives * lifeImportance[personality] \quad (4.7)$$

- A reward is given according to the average speed in that game and between games until that section. If the average velocity is less than the speed preference, a negative reward is given according to equation 4.9. If the average velocity is higher than the speed preference, a positive reward is given according to equation 4.10. The same rewards are given for the

average speed between multiple games, both represented by equations 4.12 and 4.13. These rewards are divided by two since the both values represent similar values and are both used for the reward function.

$$a = \frac{\text{averageVelocityGame}}{\text{speedPreference}[\text{personality}]} \quad (4.8)$$

$$\text{sectionReward} - = \frac{(1 - a) * \text{speedImportance}[\text{personality}]}{2} \quad (4.9)$$

$$\text{sectionReward} + = \frac{a * \text{speedImportance}[\text{personality}]}{2} \quad (4.10)$$

$$b = \frac{\text{averageVelocity}[\text{personality}]}{\text{speedPreference}[\text{personality}]} \quad (4.11)$$

$$\text{sectionReward} - = \frac{(1 - b) * \text{speedImportance}[\text{personality}]}{2} \quad (4.12)$$

$$\text{sectionReward} + = \frac{b * \text{speedImportance}[\text{personality}]}{2} \quad (4.13)$$

- A positive reward is given according to how close the concentration level is to the preferred level. Firstly, an average of the concentration is calculated from the last Tolerance for repeating levels number of levels, with the concentration variable values for each personality for the specific sections. Then the reward is calculated according to equation 4.16.

$$\text{difference} = \text{concenLevelPreferred}[\text{personality}] - \text{averageConcen} \quad (4.14)$$

$$\text{absoluteValue} = \text{Math.abs}(\text{difference}) + 1 \quad (4.15)$$

$$\text{sectionReward} + = \frac{1}{\text{absoluteValue}} * \text{concenImportance}[\text{personality}] \quad (4.16)$$

- A positive reward is given according to how close the skill level is to the preferred level. Firstly, an average of the skill is calculated from the last Tolerance for repeating levels number of levels, with the skill variable values for each personality for the specific sections. Then the reward is calculated according to equation 4.19.

$$\text{difference} = \text{skillLevelpreferred}[\text{personality}] - \text{averageSkill} \quad (4.17)$$

$$\text{absoluteValue} = \text{Math.abs}(\text{difference}) + 1 \quad (4.18)$$

$$\text{sectionReward} + = \frac{1}{\text{absoluteValue}} * \text{skillImportance}[\text{personality}] \quad (4.19)$$

- A positive reward is given according to how close the challenge level is to the preferred level. Firstly, an average of the challenge is calculated from the last Tolerance for repeating levels number of levels, with the challenge variable values for each personality for the specific sections. Then the reward is calculated according to equation 4.22.

$$\text{difference} = \text{chalLevelPreferred}[\text{personality}] - \text{averageChal} \quad (4.20)$$

$$\text{absoluteValue} = \text{Math.abs}(\text{difference}) + 1 \quad (4.21)$$



$$sectionReward + = \frac{1}{absoluteValue} * challengeImportance[personality] \quad (4.22)$$

$$sectionreward = \frac{width}{max - width} * sectionreward \quad (4.23)$$

Furthermore, the final section reward is also scaled to the size of each section, shown in equation 4.23 (max-width is associated with the width of the biggest section of the game). This reward function is used in the adaptivity reinforcement learning algorithm to indicate how appropriate the section selected was.

#### 4.4.2 Adaptivity system overview

The adaptivity system will use the reward defined in the previous section 4.4.1 but will also need to collect observations and perform actions. In terms of observations, the following list described the observations retrieves at the end of each section and used to train the algorithm:

- The max score achieved by that personality. The score is calculated using the width of each section completed, which is expressed in *Unity units*. Additionally, the starting section is added automatically to the score at the start of each episode.
- The current score.
- The current amount of lives.
- The number of lives lost to water over the personality's multiple games.
- The number of lives lost to moving enemies over the personality's multiple games.
- The number of lives lost to stationary enemies over the personality's multiple games.
- The number of lives lost over the multiple games of the personality.
- The number of lives lost in this game.
- The number of lives lost to water in this game.
- The number of lives lost to stationary enemies in this game.
- The number of lives lost to moving enemies in this game.
- The total number of coins collected over the multiple games for that personality.
- The total number of chests collected over the multiple games for that personality.
- The total number of coins collected in this game.
- The total number of chests collected in this game.

- The average velocity in this game.
- The average velocity over the multiple games for that personality.
- The variance of the velocity in this game.
- The standard deviation of the velocity in this game.
- The variance of the velocity over the multiple games for that personality.
- The standard deviation of the velocity over the multiple games for that personality.
- The total number of jumps performed in this game. According to the Jump section variable minimum and maximum values, the jumps are recorded and are added according to equation 4.25. The jumps are added this way since the player will have to retry the level if he loses a life.

$$rounded = \text{Math.roundToInt}(randomJumpsSection * lifeLost * 0.8)] \quad (4.24)$$

$$jumps += randomJumpsSection + rounded \quad (4.25)$$

- The total number of backtracks performed in this game. The backtracks are recorded and added according to the variable values. Additionally, if the player lost a life, the backtrack would be added again for that section.
- The total amount of time passed since the start of the game. The time is recorded and added according to the variable minimum and maximum values. Additionally, since the starting section is not recorded, its time can be considered two seconds, and if the player has lost a life, the time is also added an additional time but by a factor of 0.6.
- The section just completed.

The adaptivity system has sixteen possible discrete actions, each corresponding to a section. The purpose of the adaptivity system is to select the next most appropriate section to be presented to the player. However, since this decision is only taken at the end of each section, the following section would not be present, and the player would be faced with nothing until the section spawned. To solve this problem, a special section was added to fill in this gap, shown in figure 4.19.

According to a personal game designer view, a total of ten personalities were created, representing the potential player base of this game. These ten personalities are described in the following list:

- An experienced player that wants to collect coins. This personality represents a player who has played platforming games several times and likes to go fast and wants to collect all the coins and chests in the section.

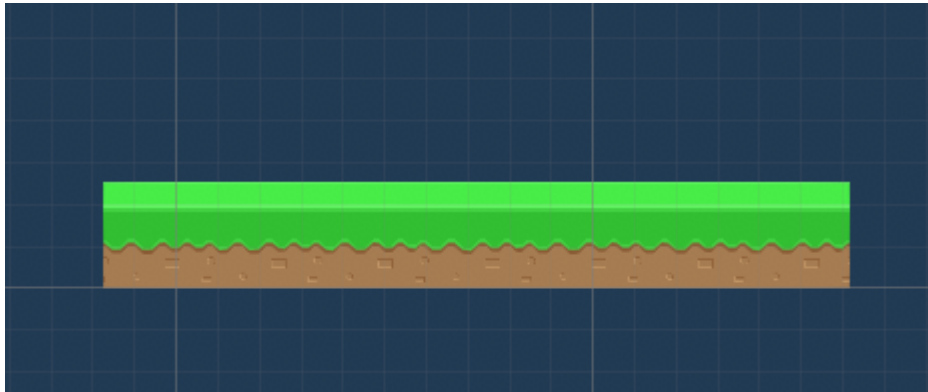


Figure 4.19: Special section

- An experienced player that just wants to go fast. This personality represents a player who has played platforming games several times and wants to go as fast as possible, risking everything for a new score.
- A casual player that wants to collect coins. This personality represents a player who has played platforming games a few times and wants to collect all the coins that he can find.
- A casual player that wants to go fast. This personality represents a player who has played platforming games a few times and wants to go as fast as possible as his skill can accomplish.
- A casual player that plays with care. This personality represents a player who has played platforming games a few times and plays each section with care not to lose his lives. Additionally, this player gives importance to his life and does not pay too much attention to speed.
- A casual player that has trouble with moving enemies. This personality represents a player who has played platforming games a few times but still has trouble completing sections with moving enemies. He does not like being presented with these challenges but does not mind having to complete them from time to time.
- A casual player that has trouble with platforming sections. This personality represents a player who has played platforming games a few times but has trouble completing sections where the main focus is on the ability to platform.
- An inexperienced user playing the game for the first time. This personality represents a player who is playing platforming games for the first time. He does not have many demands for what to find other than an appropriate challenge.
- An inexperienced player that plays the game with care. This personality represents a new player that likes to take things with time and plan their strategy.
- An inexperienced hyperactive player. This personality represents a new player that jumps a lot and plays each section without much care.

Table 4.1: Section's coins and chests

Section	Normal Coins	Backtrack coins	Chests possible coins
0	1	0	0
1	2	1	0
2	4	0	0
3	1	0	0
4	4	2	0
5	3	0	0
6	6	2	3-5 x1
7	5	0	0
8	9	0	0
9	8	chest	4-6 x1
10	5	0	0
11	6	3	0
12	4	0	0
13	5	0	0
14	0	chest	4-6 x1
15	0	0	0

Each of these personality values can be found in the appendix [A](#). The coins, chests, and lives lost values are based on the game elements present in each section and then adapted to each personality. The elements present in each section can be found in the tables [4.1](#) and [4.2](#). Additionally, the time, speed, and jumps values are based on personal gameplay values collected while simulating fast, average, and slow sections completions. These values can be found in tables [4.3](#), [4.4](#) and [4.5](#).

The adaptivity reinforcement learning algorithm follows a similar training structure described in section [3.6](#). In this case, the created personality will be randomly selected to play each game. The algorithm collects the observations and rewards at the end of each section and then decides the next section to be presented to the player. At the end of each game/episode, the following personality to play the game has a 20% probability of resetting his historical values and be considered a new player for this game.

Instead of using the python API for ml-agents, this adaptivity system uses the same ml-agents toolkit as the section [4.3](#). The adaptivity system was created using this toolkit, and several training and experimentation sessions were performed to improve the final adaptivity agent, using both PPO and SAC algorithms. These sessions will be explained in the following chapter. After the training of the adaptivity agent, the game was slightly modified to support an adaptivity system. Additionally, the game changes between an adapted and non adapted version of the game at the start of each run. The first version is randomly selected for each player and is maintained each time the player closes and starts the game again. The first version is identified by the letter "A", and the second version by the letter "B", both displayed on the screen during gameplay, shown in figure [4.20](#).

Finally, to test the performance of the adaptivity system with real players, a questionnaire must be responded to at the end of each run/round. This questionnaire uses the core questions from the

Table 4.2: Section's enemies

Section	Moving enemies	Static enemies	Water
0	0	1	0
1	1	0	0
2	0	4	0
3	0	0	0
4	0	3	1
5	0	7	1
6	1	1	1
7	0	7	0
8	0	8	1
9	0	2	1
10	6	0	1
11	5	0	1
12	3	2	1
13	1	0	1
14	1	0	1
15	4	0	0



Figure 4.20: Display of the type of round during gameplay

Game Experience Questionnaire [31]. The questions must be responded to both round “A” and round “B”. The complete questionnaire can also be found in the appendix B. Additionally, the game also records metrics that can help find a difference between the adapted and non-adapted versions. These metrics are written to a CSV file that the user must upload to the questionnaire. The metrics are recorded by section and game completed and are described in the following list:

- The section passed.

Table 4.3: Section's completions times in seconds

Section	Fast	Normal	Fast while backtracking coins	Slow
0	2	3.5	-	5
1	7	9	16	12
2	6	8	-	12
3	4.6	5	-	7
4	13	17	28	20
5	13.5	17	-	20
6	29	36	40 or 60	40
7	21.5	22.5	-	25
8	38.5	40	-	46
9	38.5	42	47	48
10	24	28	-	32
11	41.5	49	62	55
12	27	30	-	34
13	33	35	-	38
14	18.5	23.5	35	27
15	12	20	-	25

- If it was the adapted or non adapted version of the game.
- The round letter.
- The total amount of coins the player has at that moment.
- The total number of collected coins until that section or during that game.
- The total number of opened chests until that section or during that game.
- The total amount of time passed until that section or during that game.
- The total amount of time gained by collecting coins until that section or during that game.
- The total number of lives lost until that section or during that game.
- The total number of lives lost to water until that section or during that game.
- The total number of lives lost to stationary enemies until that section or during that game.
- The total number of lives lost to moving enemies until that section or during that game.
- The best score at that moment.
- The score until that section or of that game.
- The number of backtracks performed until that section or during that game.
- The number of jumps performed until that section or during that game.

Table 4.4: Section's completions average speeds in *Unity units*

Section	Fast	Normal	Fast while backtracking coins	Slow
0	8.5	4.4	-	4
1	8.8	7	4.5	4
2	5.8	4.5	-	4
3	8.5	8	-	7.5
4	8.7	6.5	3.8	6
5	7.9	6.8	-	6
6	6.4	5.5	4.7 or 3	4.5
7	8.9	8.5	-	7.5
8	7.3	6.7	-	5.5
9	8	7.4	6.5	6.5
10	7.3	6	-	5
11	6.3	5.2	4.2	4.2
12	8.4	8	-	6.5
13	8.9	8.7	-	8
14	8.2	6.7	4.2	6
15	8.9	4.4	-	3.8

- The average velocity until that section or during that game.
- The velocity variance until that section or during that game.

## 4.5 Summary

In this chapter, several implemented systems were explained and detail. Firstly, in section 4.1, the life system developed was explained, and its looks during gameplay were presented. Then, in section 4.2, the time system implemented in the game was described.

In section 4.3, the player simulation system was firstly implemented according to the proposed solution. However, because of several problems, the implementation changed to use the Unity ml-agents toolkit. This toolkit allowed for a more straightforward implementation of the simulated agents. Despite its usefulness and some results showing promising results, the final result could not be used since the agent did not replicate the user actions at an expected level. Finally, in section 4.4, a more simplistic solution to the player simulation was presented and developed. Using this personality system, the adaptivity system could now be trained, and the final adaptivity agent was included in the game for users to test its effectiveness. All the development features and details were presented and discussed for the adaptivity system and the personality system.

Table 4.5: Section's completions jumps

<b>Section</b>	<b>Fast</b>	<b>Normal</b>	<b>Fast while backtracking coins</b>	<b>Slow</b>
0	1	2	-	3
1	1	4	4	5
2	4	5	-	6
3	1	1	-	2
4	6	5	9	7
5	7	8	-	9
6	12	12	14 or 22	12
7	5	5	-	6
8	18	20	-	23
9	13	13	13	16
10	10	12	-	14
11	17	20	25	25
12	8	8	-	12
13	4	4	-	5
14	5	6	14	8
15	0	0	-	1



## Chapter 5

# Results and evaluation

In this chapter, the results obtained from the developed system explained in chapter 4 will be displayed, and an analysis will be made to fulfill the objectives described in section 1.2. In section 5.1, the several experiments performed for the player simulation will be detailed, and at the end, an overall analysis of the data will be made. Then, the results from the training of the adaptivity agent will be shown in section 5.2.1. Furthermore, an analysis of the results will also be made in section 5.2.2. Finally, in section 5.2.3, the adapted game created with the adaptivity agent was tested with real users, and the results obtained will be summarized, and their data analysed.

### 5.1 Player simulation

The first implementation of the player simulation described in section 4.3 did not yield any results since it could not be fully implemented. Therefore, the only results obtained during the development of the player simulation are related to the implementation with the *Unity ml-agents toolkit*.

This toolkit allows the developers to define the algorithm in a *.yaml* file, which will then be used to create and train the algorithm. The toolkit file allows for the following configurations to be defined, which are further explained in the documentation of the toolkit <sup>1</sup>:

- `trainer_type`. This setting allows defining what the main reinforcement learning algorithm to use is. In the case of this dissertation, the main algorithms used are PPO and SAC.
- `max_steps`.
- `hyperparameters->learning_rate`.
- `hyperparameters->batch_size`. This setting represents the number of experiences in each iteration of gradient descent.
- `hyperparameters->buffer_size`. This setting represents the number of experiences that need to be collected before updating the policy model.

---

<sup>1</sup><https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md>

- hyperparameters->learning\_rate\_schedule.
- hyperparameters->PPO->beta. This setting represents the strength of the entropy regularization. This value should be increased if the entropy drops too quickly.
- hyperparameters->PPO->epsilon.
- hyperparameters->PPO->lambda.
- hyperparameters->PPO->num\_epoch.
- hyperparameters->SAC->buffer\_init\_steps.
- hyperparameters->SAC->init\_entcoef.
- hyperparameters->SAC->save\_replay\_buffer.
- hyperparameters->SAC->tau.
- hyperparameters->SAC->steps\_per\_update.
- hyperparameters->SAC->reward\_signal\_num\_update.
- network\_settings->hidden\_units. This setting represents the number of hidden units in the hidden layers of the neural networks.
- network\_settings->num\_layers. This setting represents the number of hidden layers in the neural network.
- network\_settings->normalize. This setting defines if the input values should be normalized.
- network\_settings->vis\_encode\_type.
- network\_settings->memory->memory\_size. This setting represents the amount of memory the agent must keep.
- network\_settings->memory->sequence\_length. This setting represents how long the sequence of experiences should be while training.
- extrinsic->strength. This setting represents the factor to which multiply the reward.
- extrinsic->gamma.
- GAIL->strength. This setting represents the factor to which multiply the reward.
- GAIL->gamma.
- GAIL->demo\_path.
- GAIL->network\_settings.

- GAIL->learning\_rate.
- GAIL->use\_actions. Defines if the discriminator should discriminate based on the performed actions or just the observations.
- GAIL->use\_vail. A variational bottleneck within the GAIL discriminator, which forces the discriminator to learn a more general representation.
- behavior\_cloning->demo\_path.
- behavior\_cloning->steps.
- behavior\_cloning->strength. This setting represents the factor to which multiply the reward.
- behavior\_cloning->samples\_per\_update.

Although the settings listed do not include every configuration available to change, these are the ones defined in the training file (as shown in the figure 5.1), and the default values are used for the settings not defined. Additionally, not all the configuration values will change in the experiments, and the ones who change are generally between the recommended range described by the ml-agents toolkit.

The ml-agents toolkit also allows the use of *Tensorboard* to display the statics during the training sessions, which is further detailed in the toolkit's documentation <sup>2</sup> and the following list:

- Environment/Cumulative reward - Represents the mean cumulative episode reward that should increase during training.
- Environment/Episode Length - Represents the mean length of each episode.
- Policy/Entropy - Represents how arbitrary the decisions of the agent are. This value should slowly decrease during training.
- Policy/GAIL Policy Estimate - Represents the discriminator's estimate for the generated policy.
- Policy/GAIL Expert Estimate - Represents the discriminator's estimate for the expert demonstrations.
- Losses/Policy Loss - Represents the mean magnitude of the policy loss function and should decrease during training.
- Losses/Pretraining Loss - Represents the mean magnitude of the behavioral cloning loss.
- Losses/GAIL Loss - Represents the mean magnitude of the GAIL discriminator loss.

In addition, this tool displays statistics relating to the environment, like the cumulative reward, the losses and the policy. Finally, each experiment is evaluated using the *Tensorboard* statistics graphs and its performance while playing the game.

---

<sup>2</sup><https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md>

```

behaviors:
  Player:
    trainer_type: ppo
    max_steps: 50000000
    hyperparameters:
      batch_size: 128
      buffer_size: 1024
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambda: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory:
        sequence_length: 64
        memory_size: 256
    reward_signals:
      gail:
        gamma: 0.99
        strength: 1
        network_settings:
          normalize: false
          hidden_units: 64
          num_layers: 1
          vis_encode_type: simple
        learning_rate: 0.0003
        use_actions: false
        use_vail: false
        demo_path: ../Assets/Demonstrations/RRreward45minutes.demo
    behavioral_cloning:
      demo_path: ../Assets/Demonstrations/RRreward45minutes.demo
      steps: 0
      strength: 0.8
      samples_per_update: 1024
  torch_settings:
    device: cuda
  engine_settings:
    no_graphics: true
    time scale: 1

```

Figure 5.1: Example of the training *.yaml* file

Table 5.1: Player simulation experiment 1 configuration

Settings	Description
PPO hyperparameters	batch_size=2024, buffer_size=20240, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = true, hidden_units = 512, num_layers = 3, vis_encode_type = simple
Memory	None
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, use_actions = true, use_vail = true
GAIL network settings	normalize = true, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.5, samples_per_update = 0, num_epoch = None, batch_size = None
Demonstrations	One-hour recording composed of 16 episodes and 195280 steps
Observations	Scalar observation + image with colours + ray perception sensor
Training settings	Number of environments = 1, timescale for game = default

### 5.1.1 Experiments performed

The following sections will describe each experiment performed during the development of this dissertation. In addition, each section will present a table with the configuration used for that experiment and explain what changes were made and the results obtained using the *Tensorboard* graphs (all graphs are in the number of steps performed). Each experiment was performed sequentially with a slight change made to test its performance in relation to the other experiments. Furthermore, the results were primarily analysed on how well the agent was able to complete each section since the graphs from the algorithms were sometimes not a correct indication of how well that agent performed.

#### 5.1.1.1 Player simulation experiment 1

This experiment was the first performed and served to test the usage of the GAIL algorithm using the ml-agents toolkit. Unfortunately, the game modifications relating to the sections and the movement system were not yet implemented, meaning there was input lag and the starting section still had the water pit. The table 5.1 shows the configuration used for this experiment. Furthermore, the parameters used for this experiment are based on examples described in the ml-agents toolkit documentation.

Results:

- The agent trained for about 2M steps.
- The final trained agent could not reliably complete the start of the game, losing at the water pit.

Table 5.2: Player simulation experiment 2 configuration

Settings	Description
PPO hyperparameters	batch_size=2024, buffer_size=20240, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = true, hidden_units = 512, num_layers = 3, vis_encode_type = simple
Memory	None
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, use_actions = true, use_vail = true
GAIL network settings	normalize = true, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.5, samples_per_update = 0, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of four games, which corresponds to about 25 minutes of gameplay</b>
Observations	Scalar observation + image with colours + ray perception sensor
Training settings	<b>Number of environments = 8, timescale for game = 20</b>

- The algorithm results (figure 5.2) showed that the episode length is rising over time. However, the policy loss and GAIL loss are inconsistent, jumping the values rapidly.

### 5.1.1.2 Player simulation experiment 2

In this experiment, the agent uses a new set of actions. The agent can decide to perform two actions:

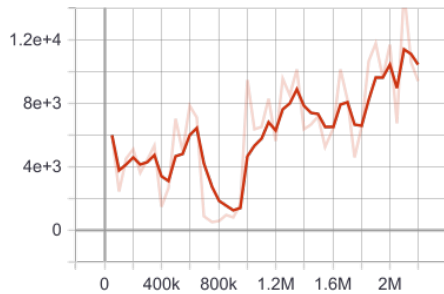
- If he wants to jump or not as a discrete action.
- If he should move to the left or the right or not move, given as a continuous value from  $[-1,1]$ .

The table 5.2 shows the configuration used for this experiment.

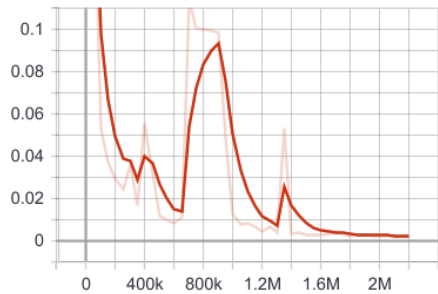
Results:

- The agent trained for about 1.6M steps.
- The final trained agent could not move around the levels or even pass the starting sections.
- The algorithm results (figure 5.3) showed that the policy loss is very inconsistent over the training section.

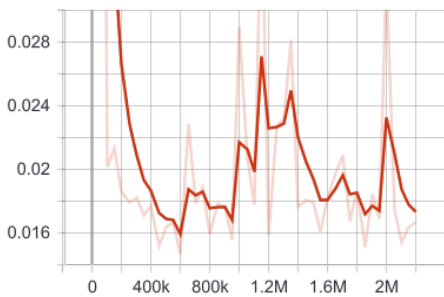
Figure 5.2: Player simulation experiment 1 results



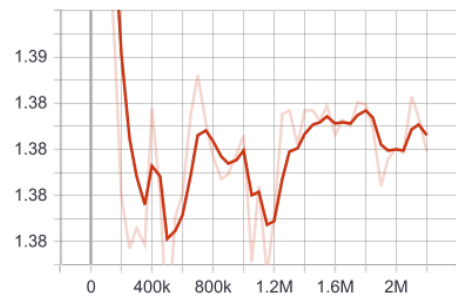
(a) Mean length of each episode over the training steps



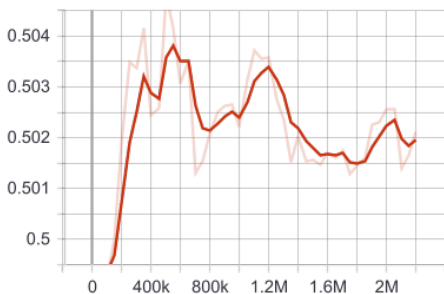
(b) Mean magnitude of the behavioral cloning loss over the training steps



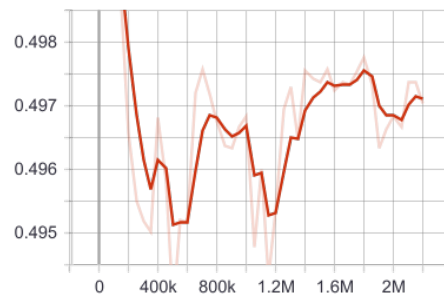
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

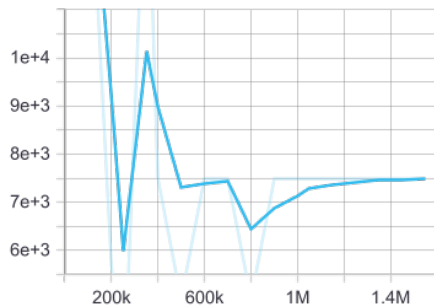


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

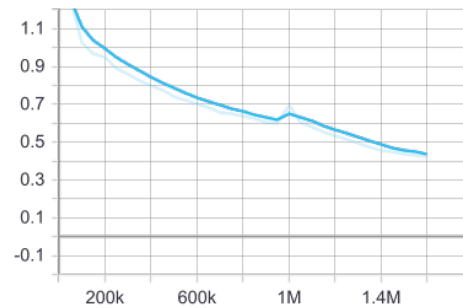


(f) GAIL discriminator's estimate for the policy generated, over the training steps

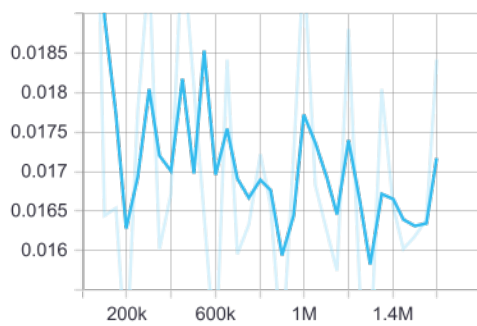
Figure 5.3: Player simulation experiment 2 results



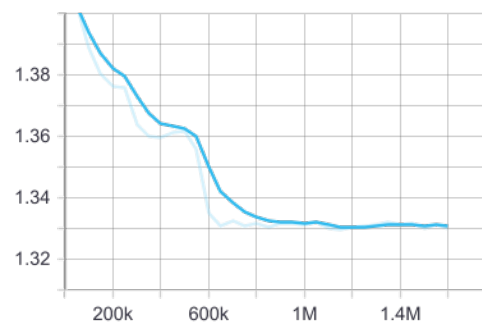
(a) Mean length of each episode over the training steps



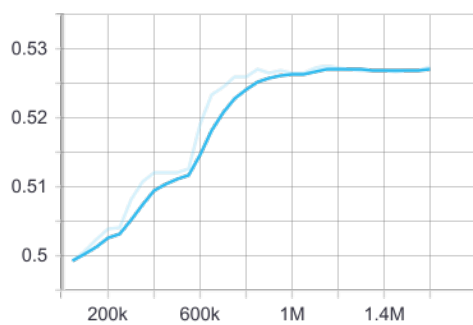
(b) Mean magnitude of the behavioral cloning loss over the training steps



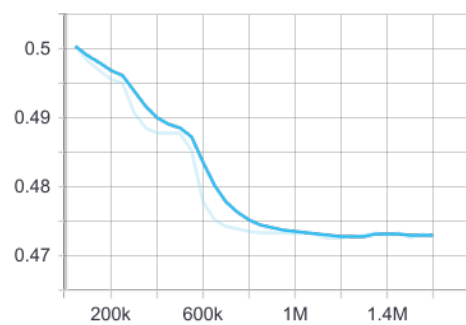
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps



Table 5.3: Player simulation experiment 3 configuration

Settings	Description
PPO hyperparameters	batch_size=2024, buffer_size=20240, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = true, hidden_units = 512, num_layers = 3, vis_encode_type = simple
Memory	None
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, <b>use_actions = false</b> , use_vail = true
GAIL network settings	normalize = true, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.5, <b>samples_per_update = 20240</b> , num_epoch = None, batch_size = None
Demonstrations	<b>Recording of two games, which corresponds to about 15 minutes of gameplay</b>
Observations	Scalar observation + <b>image grayscale</b> + ray perception sensor
Training settings	Number of environments = 8, <b>timescale for game = 1</b>

### 5.1.1.3 Player simulation experiment 3

In this experiment, the agent's actions were reverted to the original actions (1 discrete action, with move and jump options). Furthermore, the images used are grayscale instead of coloured. The table 5.3 shows the configuration used for this experiment.

Results:

- The agent trained for about 2.4M steps.
- The final trained agent could not pass the starting section consistently and would sometimes stop moving in this section.
- The algorithm results (figure 5.4) showed that the policy loss is very inconsistent over the training section.

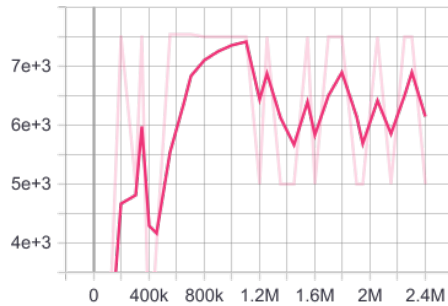
### 5.1.1.4 Player simulation experiment 4

In this experiment, the LSTM was introduced to allow the agent to remember and decide depending on the previous actions. Furthermore, other settings were also changed. The table 5.4 shows the configuration used for this experiment.

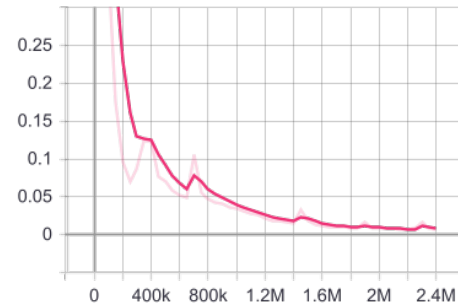
Results:

- The agent trained for about 8M steps.
- The final trained agent did not make any movement with the character.

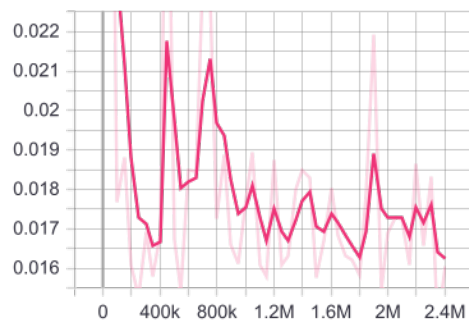
Figure 5.4: Player simulation experiment 3 results



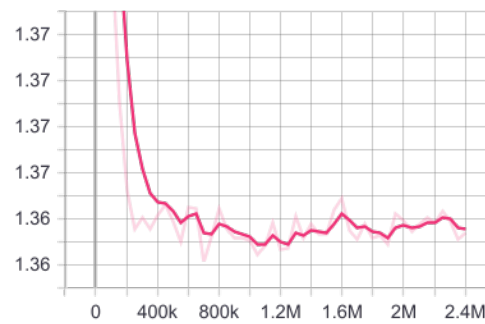
(a) Mean length of each episode over the training steps



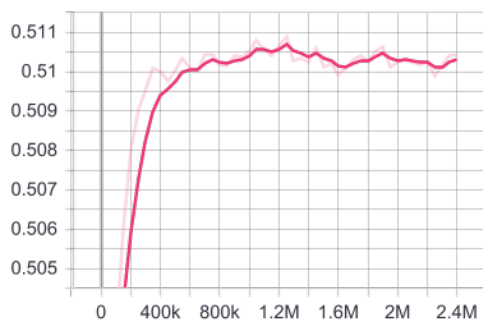
(b) Mean magnitude of the behavioral cloning loss over the training steps



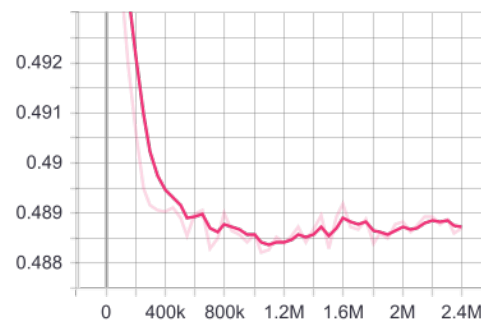
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.4: Player simulation experiment 4 configuration

Settings	Description
PPO hyperparameters	batch_size=2024, buffer_size=20240, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	<b>normalize = false</b> , hidden_units = 512, num_layers = 3, vis_encode_type = simple
Memory	<b>sequence_length = 128, memory_size = 512</b>
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, <b>use_actions = true</b> , use_vail = true
GAIL network settings	<b>normalize = false</b> , hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.5, samples_per_update = 20240, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of six games, which corresponds to about 30 minutes of gameplay</b>
Observations	Scalar observation + image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

- The algorithm results (figure 5.5) showed that the policy loss is somewhat constant and did not decrease.

#### 5.1.1.5 Player simulation experiment 5

In this experiment, the LSTM that was previously introduced was removed. Furthermore, the scalar observations were removed, having only the image and the ray perception sensor. The observations were also stacked, meaning that the observations are composed of two images and ray perceptions. However, the stacking of observations leads to much higher memory usage. Finally, the network size was also reduced. The table 5.5 shows the configuration used for this experiment.

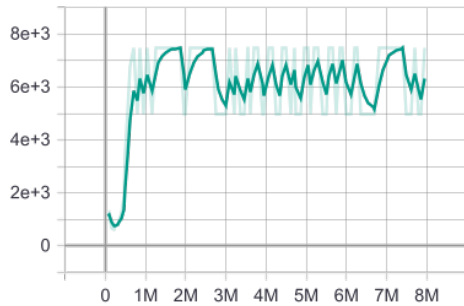
Results:

- The agent trained for about 1.5M steps.
- The final trained agent can pass the starting section but not consistently. Furthermore, the agent has trouble jumping in other sections, stopping when faced with a wall.
- The algorithm results (figure 5.6) showed that the policy loss is inconsistent during the training. Additionally, the entropy is also rising during the training.

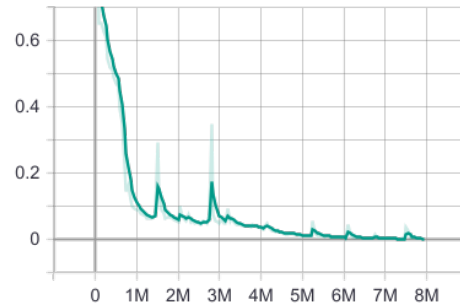
#### 5.1.1.6 Player simulation experiment 6

In this experiment, the LSTM was reintroduced to the agent's network to test its effectiveness with the new observations. Furthermore, the behaviour cloning was also given more strength, giving

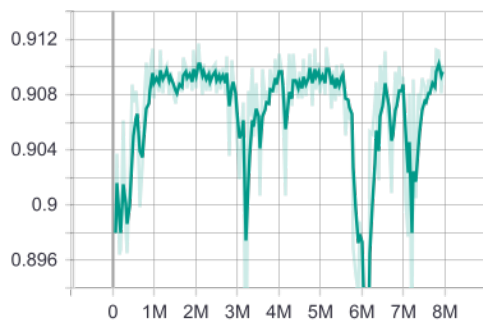
Figure 5.5: Player simulation experiment 4 results



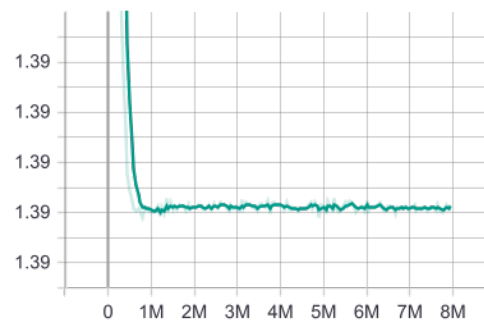
(a) Mean length of each episode over the training steps



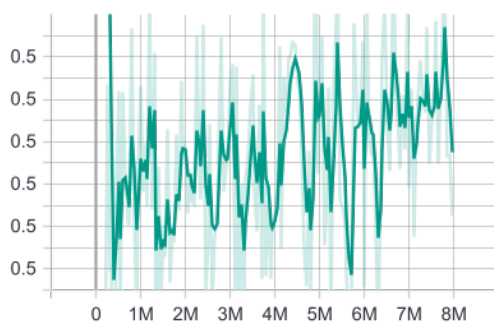
(b) Mean magnitude of the behavioral cloning loss over the training steps



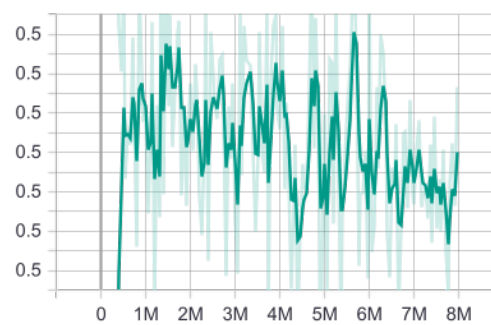
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

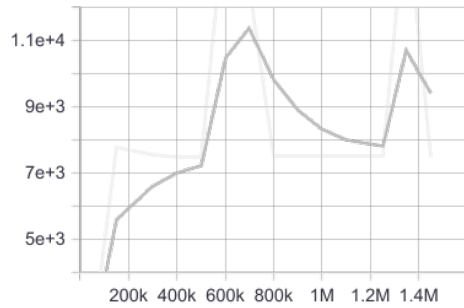


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

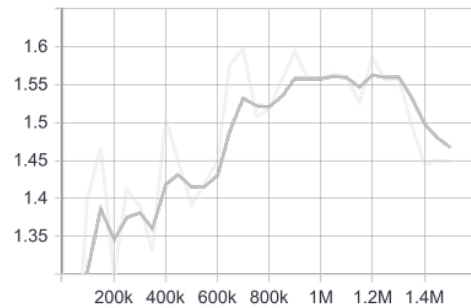


(f) GAIL discriminator's estimate for the policy generated, over the training steps

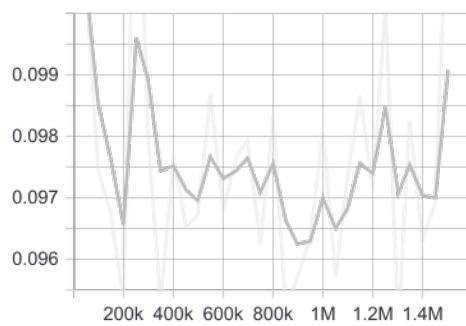
Figure 5.6: Player simulation experiment 5 results



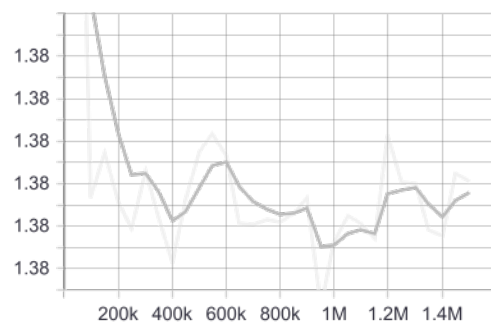
(a) Mean length of each episode over the training steps



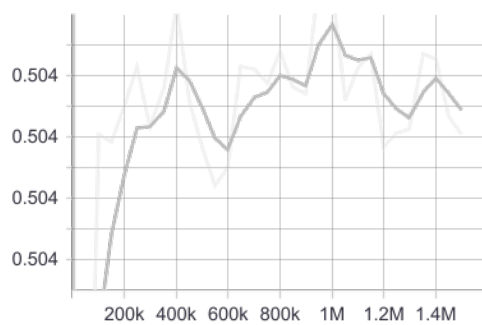
(b) Policy entropy over the training steps



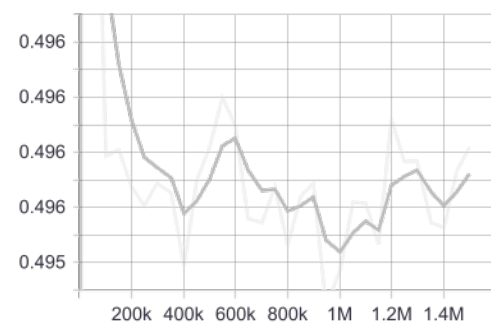
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.5: Player simulation experiment 5 configuration

Settings	Description
PPO hyperparameters	<b>batch_size=64, buffer_size=1280</b> , learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, <b>hidden_units = 256, num_layers = 2</b> , vis_encode_type = simple
Memory	<b>None</b>
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, <b>use_actions = false</b> , use_vail = true
GAIL network settings	normalize = false, <b>hidden_units = 64, num_layers = 1</b> , vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.5, <b>samples_per_update = 1280</b> , num_epoch = None, batch_size = None
Demonstrations	Recording of six games, which corresponds to about 30 minutes of gameplay
Observations	<b>Image grayscale + ray perception sensor</b>
Training settings	Number of environments = 8, timescale for game = 1

this algorithm more importance in training. Finally, some other settings were also tweaked to test their effectiveness. The table 5.6 shows the configuration used for this experiment.

Results:

- The agent trained for about 1M steps.
- The final trained agent can pass the starting section more consistently. However, the agent has some trouble jumping over stairs, often remaining still at the start of them. Additionally, the agent also seems confused about the coins while sometimes avoiding them.
- The algorithm results (figure 5.7) showed that the policy loss is inconsistent, which may be caused by the small training session.

#### 5.1.1.7 Player simulation experiment 7

In this experiment, the LSTM has a smaller size, and the overall training lasted longer. The table 5.7 shows the configuration used for this experiment.

Results:

- The agent trained for about 7M steps.
- The final trained agent seems less stable than the previous experiment, having a more erratic movement.
- The algorithm results (figure 5.8) showed that the policy loss is very inconsistent during the training session. Furthermore, the GAIL expert and policy estimates are near perfect.

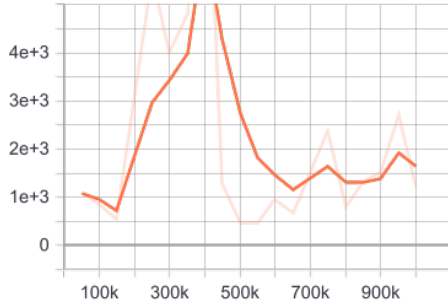
Table 5.6: Player simulation experiment 6 configuration

Settings	Description
PPO hyperparameters	<b>batch_size=128, buffer_size=1024</b> , learning_rate = 0.0003, <b>beta = 0.01</b> , epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, <b>hidden_units = 128</b> , num_layers = 2, vis_encode_type = simple
Memory	<b>sequence_length = 128, memory_size = 512</b>
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.8, samples_per_update = 1024</b> , num_epoch = None, batch_size = None
Demonstrations	Recording of six games, which corresponds to about 30 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

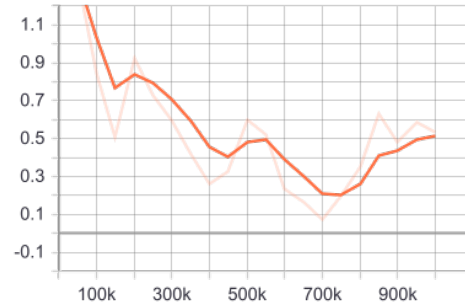
Table 5.7: Player simulation experiment 7 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	<b>sequence_length = 16, memory_size = 64</b>
Reward extrinsic	None
GAIL	gamma = 0.99, strength = 1.0, learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of six games, which corresponds to about 30 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

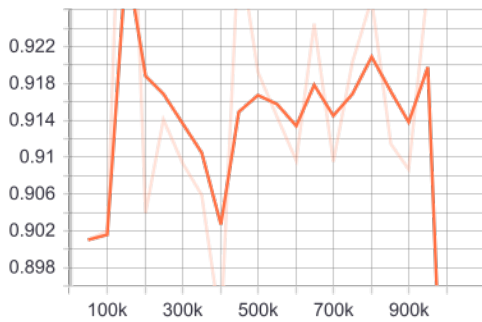
Figure 5.7: Player simulation experiment 6 results



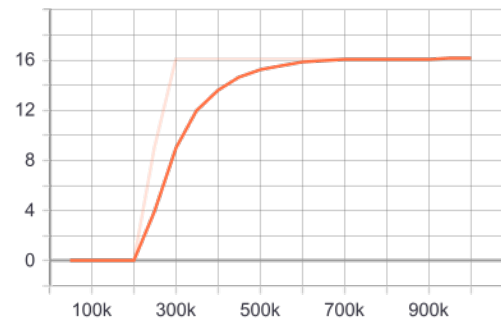
(a) Mean length of each episode over the training steps



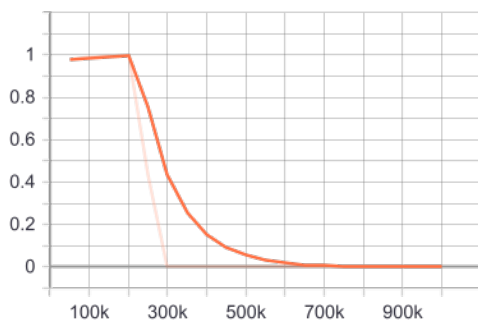
(b) Policy entropy over the training steps



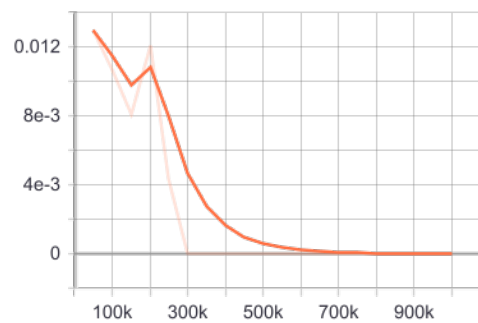
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



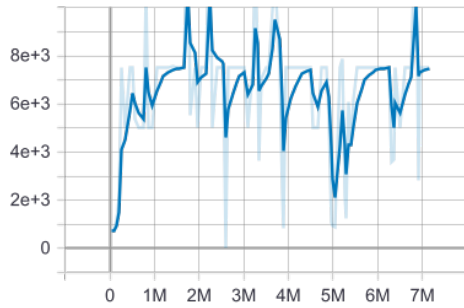
(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



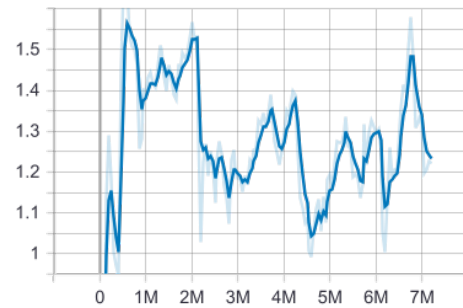
(f) GAIL discriminator's estimate for the policy generated, over the training steps



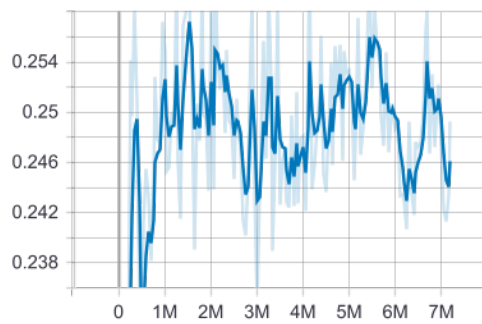
Figure 5.8: Player simulation experiment 7 results



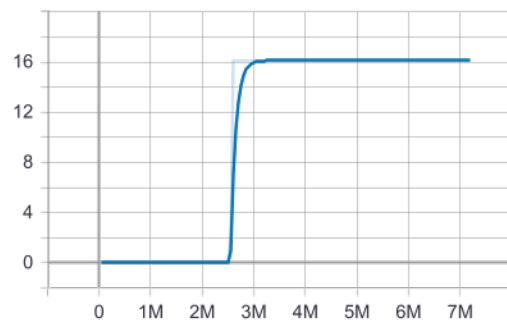
(a) Mean length of each episode over the training steps



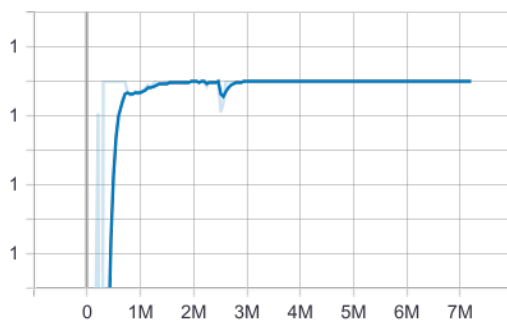
(b) Policy entropy over the training steps



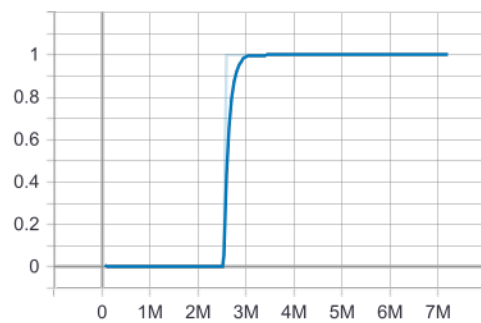
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.8: Player simulation experiment 8 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	<b>sequence_length = 64, memory_size = 256</b>
Reward extrinsic	<b>gamma = 0.99, strength = 0.2</b>
<b>Reward details</b>	<b>if velocity &lt; 1 = reward - 0.0001 per frame, coins = 0.01, moving forward = dx/10000, lose = -1</b>
GAIL	gamma = 0.99, <b>strength = 0.8</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of eight games, which corresponds to about 45 minutes of gameplay</b>
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

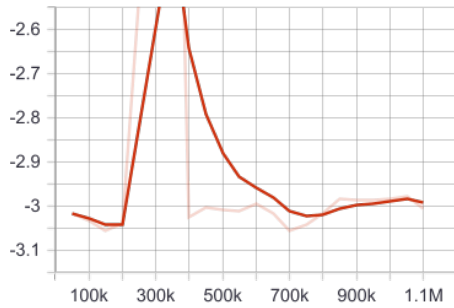
### 5.1.1.8 Player simulation experiment 8

In this experiment, the LSTM is in the middle between the size of the two previous experiments. Furthermore, to help orient the agent to move forward and not stopping so much in the stairs, an extrinsic reward is now given to the agent. Finally, the observation stacking was also removed since the memory usage was too high. The table 5.8 shows the configuration used for this experiment.

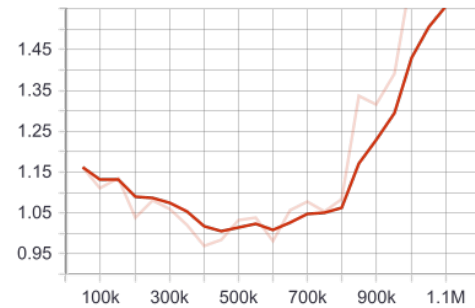
Results:

- The agent trained for about 1M steps.
- The final trained agent is a better version than the previous two agents. However, it still shows some problems relating to the jumps of stairs.
- The algorithm results (figure 5.9) showed that the policy loss seems to be declining over time. Furthermore, the GAIL policy estimate is declining, the expert estimate is perfect, and the Policy entropy over the training steps is rising. Other conclusions are not reliable since the training session is small.

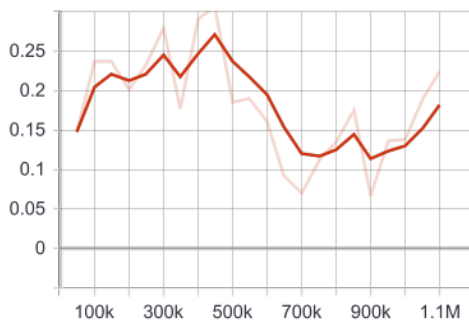
Figure 5.9: Player simulation experiment 8 results



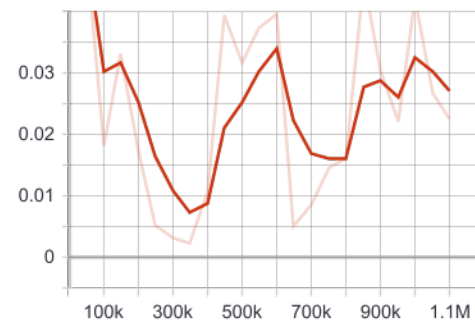
(a) Mean cumulative episode reward over the training steps



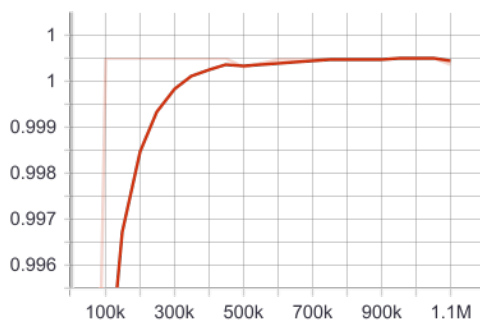
(b) Policy entropy over the training steps



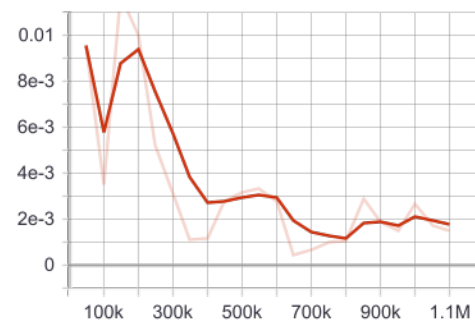
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.9: Player simulation experiment 9 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.5</b>
Reward details	<b>if velocity &lt; 1 = reward - 0.001 per frame, coins = 0.1, moving forward = dx/5000, lose = -10</b>
GAIL	gamma = 0.99, <b>strength = 0.5</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

#### 5.1.1.9 Player simulation experiment 9

In this experiment, the extrinsic reward given to the agent was given more importance, and the GAIL algorithm lost some weight. Furthermore, the reward values were also tweaked. The table 5.9 shows the configuration used for this experiment.

Results:

- The agent trained for about 1.5M steps.
- The final trained agent shows the same problems as the previous one, but his movement is more secure and bold this time.
- The algorithm results (figure 5.10) showed that the cumulative reward is declining over time. However, this training session was cut too soon since the policy loss was declining, and other values were also declining.

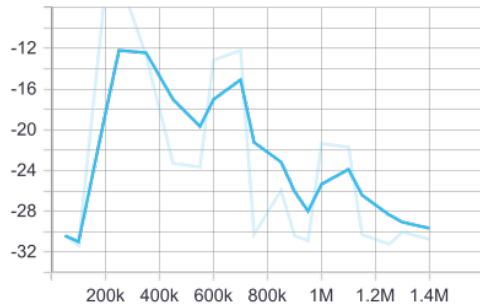
#### 5.1.1.10 Player simulation experiment 10

In this experiment, the curiosity reward was introduced to the agent, with the primary purpose of testing its effectiveness. The table 5.10 shows the configuration used for this experiment.

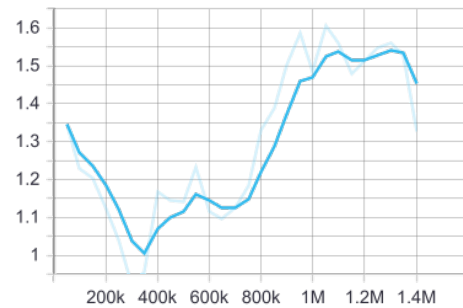
Results:

- The agent trained for about 8.5M steps.

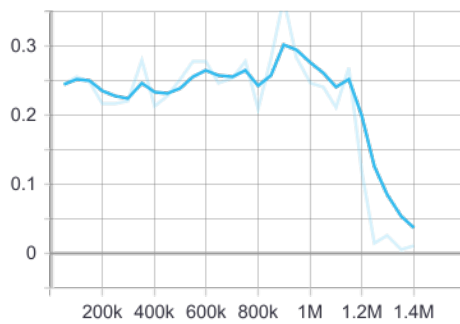
Figure 5.10: Player simulation experiment 9 results



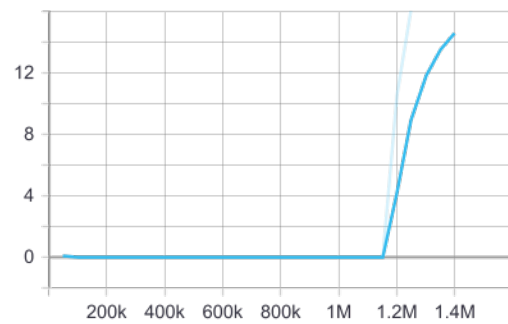
(a) Mean cumulative episode reward over the training steps



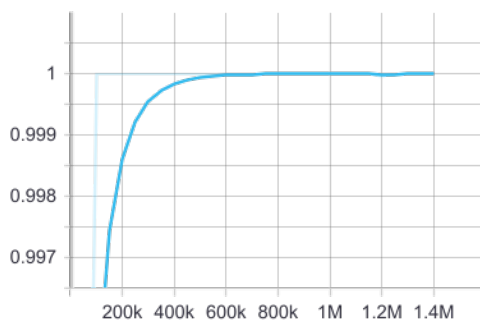
(b) Policy entropy over the training steps



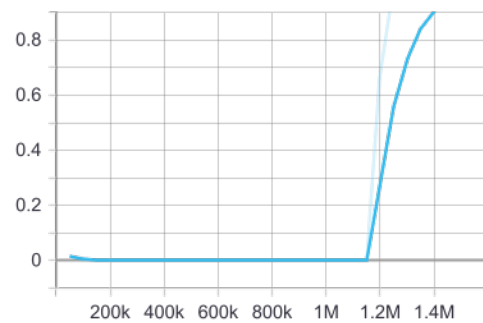
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.10: Player simulation experiment 10 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.4</b>
Reward details	if velocity < 1 = reward - 0.001 per frame, coins = 0.1, moving forward = dx/5000, lose = -10
<b>Curiosity reward</b>	<b>gamma = 0.99, strength = 0.1, hidden_units = 64, num_layers = 1, learning_rate = 0.0003</b>
GAIL	gamma = 0.99, <b>strength = 0.8</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

- The final trained agent has some trouble making some jumps in the game.
- The algorithm results (figure 5.11) showed that the cumulative reward is converging to zero. Furthermore, the curiosity loss is not declining and is also unstable.
- The curiosity reward does not seem to have a positive effect on the agent.

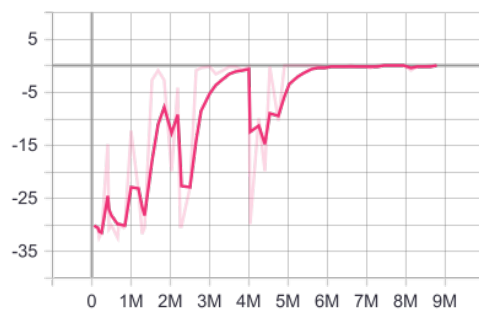
#### 5.1.1.11 Player simulation experiment 11

In this experiment, the curiosity reward introduced in the previous experiment was removed since it did not substantially impact the final agent. Furthermore, some other values were changed to test their effectiveness. The table 5.11 shows the configuration used for this experiment.

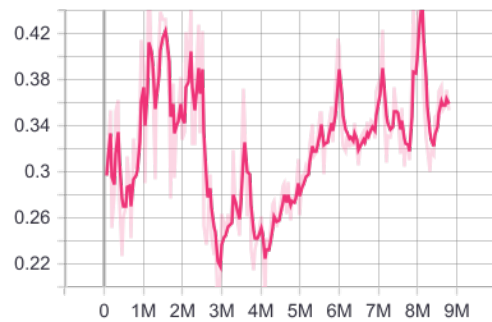
Results:

- The agent trained for about 3M steps.
- The final trained agent can reliably complete some sections, but other sections seem to have more trouble completing, for example, the stair jumping problem.
- The algorithm results (figure 5.12) showed that the policy loss is decreasing. However, the entropy is not decreasing over the training session.

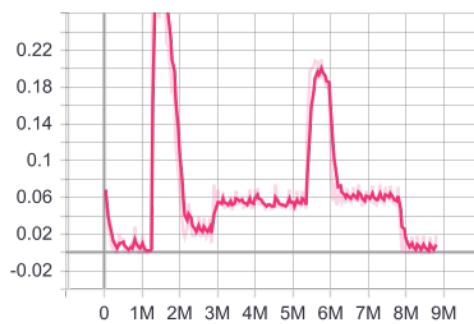
Figure 5.11: Player simulation experiment 10 results



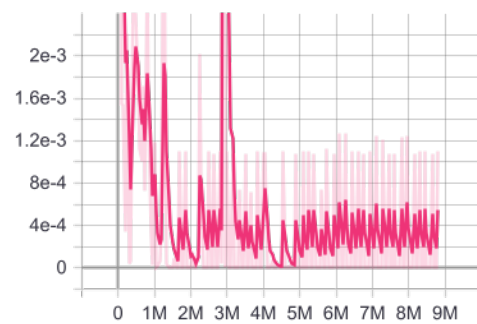
(a) Mean cumulative episode reward over the training steps



(b) Mean magnitude of the policy loss function over the training steps

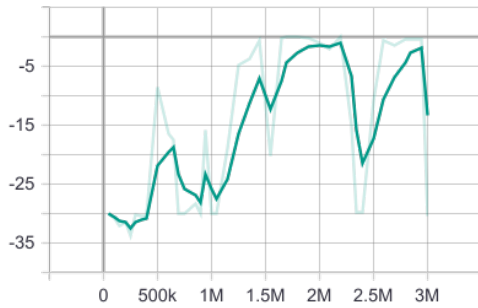


(c) Mean magnitude of the GAIL discriminator loss over the training steps

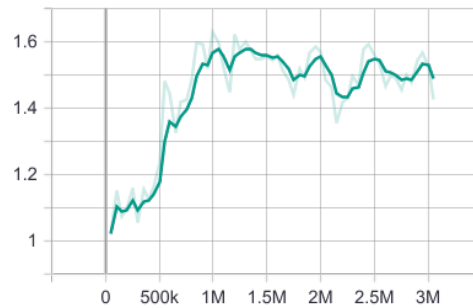


(d) GAIL discriminator's estimate for the policy generated, over the training steps

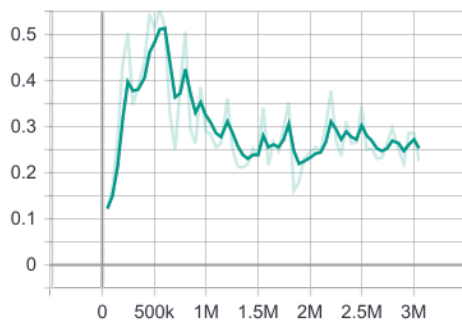
Figure 5.12: Player simulation experiment 11 results



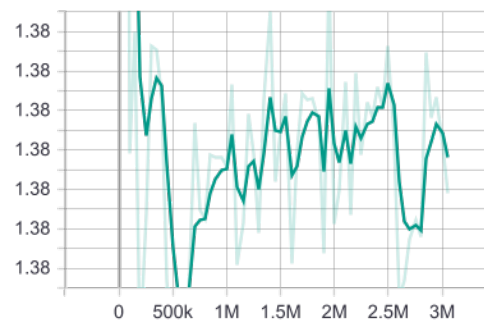
(a) Mean cumulative episode reward over the training steps



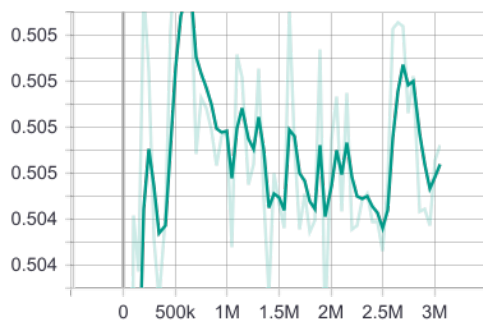
(b) Policy entropy over the training steps



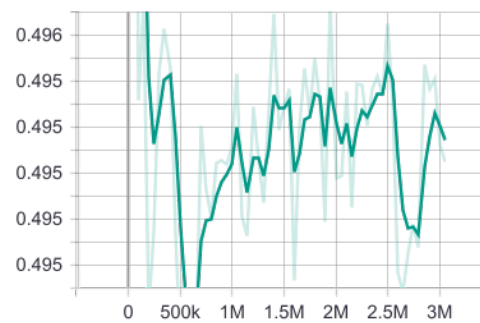
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps



Table 5.11: Player simulation experiment 11 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.35</b>
Reward details	if velocity < 1 = reward - 0.001 per frame, coins = 0.1, moving forward = dx/5000, lose = -10
GAIL	gamma = 0.99, <b>strength = 0.65</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = true</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1

#### 5.1.1.12 Player simulation experiment 12

In this experiment, a simpler game was implemented as described in the section 4.3. The game now lasts 1 minute and 15 seconds, and the agent only has one life. The simpler game will help reduce situations where the agent spent too much time standing still at the stairs obstacle, which sometimes would last until the end of the episode. The table 5.12 shows the configuration used for this experiment.

Results:

- The agent trained for about 3.5M steps.
- The final trained agent shows the same problems as the previous experiment.
- The algorithm results (figure 5.13) showed that the policy loss is decreasing, as well as the entropy. Furthermore, the cumulative reward seems to be increasing.

#### 5.1.1.13 Player simulation experiment 13

In this experiment, the extrinsic reward was removed to test the agent performance using only GAIL and behavioral cloning. The table 5.13 shows the configuration used for this experiment.

Results:

- The agent trained for about 7M steps.

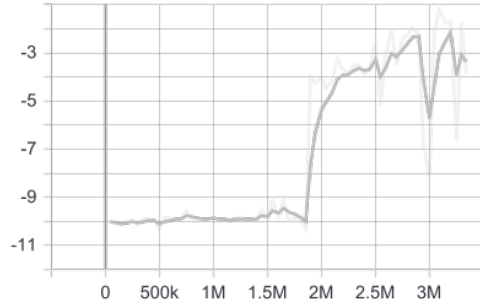
Table 5.12: Player simulation experiment 12 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 0.35
Reward details	if velocity < 1 = reward - 0.001 per frame, coins = 0.1, moving forward = dx/5000, lose = -10
GAIL	gamma = 0.99, strength = 0.65, learning_rate = 0.0003, use_actions = false, use_vail = true
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
<b>Simpler game</b>	<b>1 minute and 15 seconds, 1 life</b>

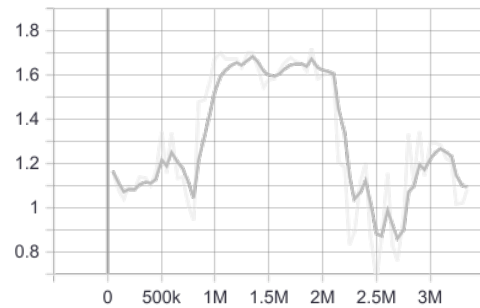
Table 5.13: Player simulation experiment 13 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	<b>None</b>
Reward details	<b>None</b>
GAIL	gamma = 0.99, <b>strength = 1</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life

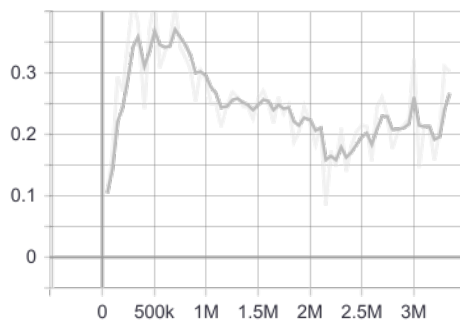
Figure 5.13: Player simulation experiment 12 results



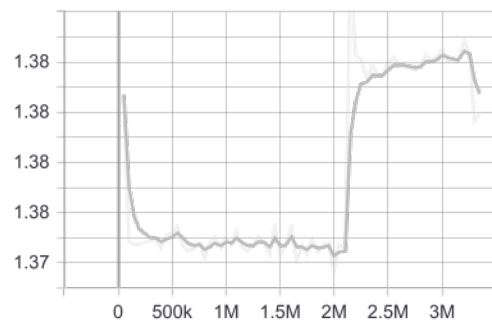
(a) Mean cumulative episode reward over the training steps



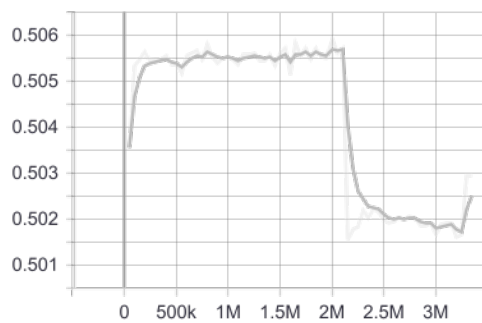
(b) Policy entropy over the training steps



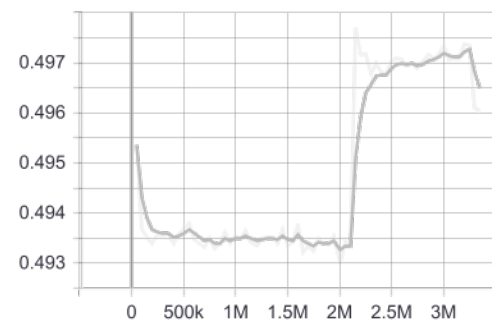
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.14: Player simulation experiment 14 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.35</b>
Reward details	<b>coins = 0.1, moving forward = dx/5000, lose = -10</b>
GAIL	gamma = 0.99, <b>strength = 0.65</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of nine games, which corresponds to about 1 hour of gameplay</b>
Observations	<b>Image grayscale</b>
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life

- The final trained agent shows the problems of jumping stairs.
- The algorithm results (figure 5.14) showed that the policy loss is decreasing. Furthermore, episode length seems to be unstable during the training session.

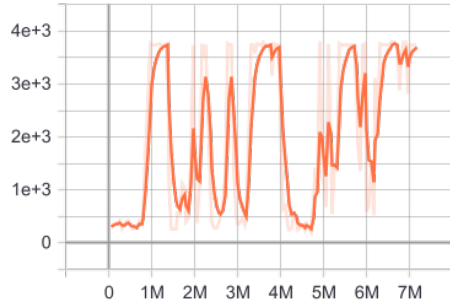
#### 5.1.1.14 Player simulation experiment 14

In this experiment, the ray perception sensor was removed, leading to the observation being composed of only images. Furthermore, the expert demonstration recordings are more extended, and the reward values were changed. The table 5.14 shows the configuration used for this experiment.

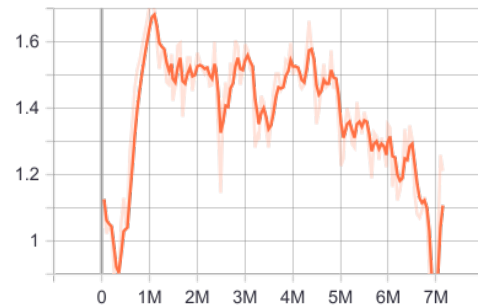
Results:

- The agent trained for about 1.5M steps.
- The final trained agent tries very hard not to lose the game, often not moving forward to not risk its life.
- The algorithm results (figure 5.15) showed that the entropy is increasing, and the policy has an uncertain value. Furthermore, the GAIL policy estimate has a low value meaning that the discriminator has difficulty distinguishing the expert demonstrations from the generated policy.

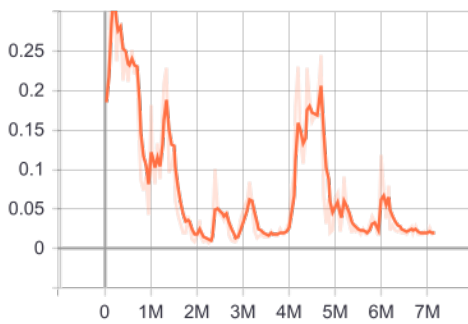
Figure 5.14: Player simulation experiment 13 results



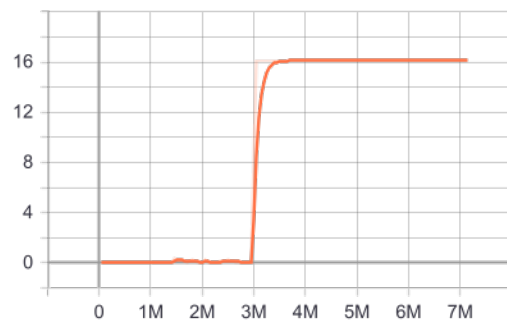
(a) Mean length of each episode over the training steps



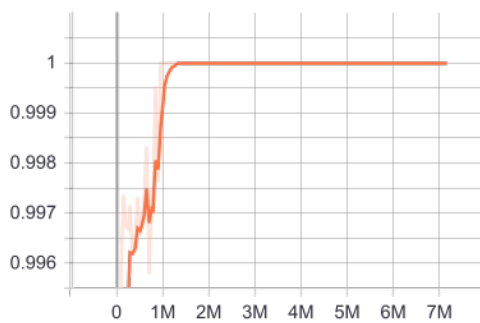
(b) Policy entropy over the training steps



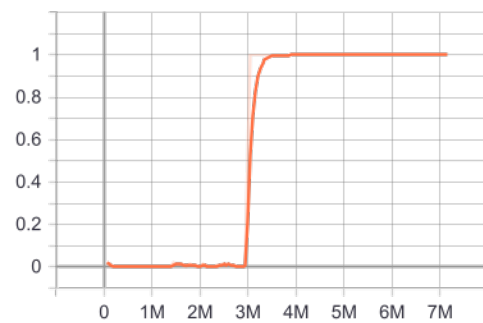
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

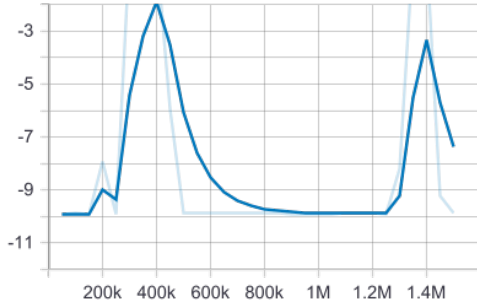


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

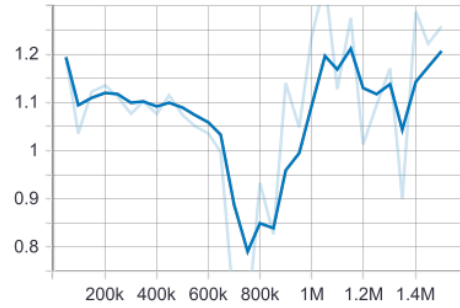


(f) GAIL discriminator's estimate for the policy generated, over the training steps

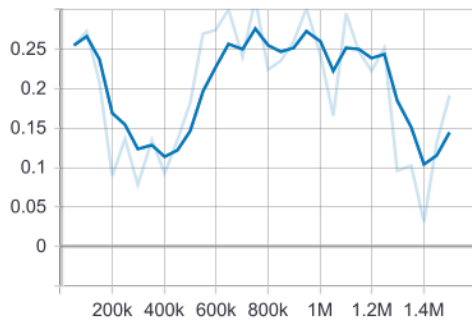
Figure 5.15: Player simulation experiment 14 results



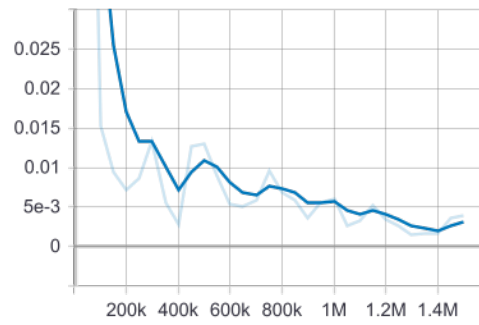
(a) Mean cumulative episode reward over the training steps



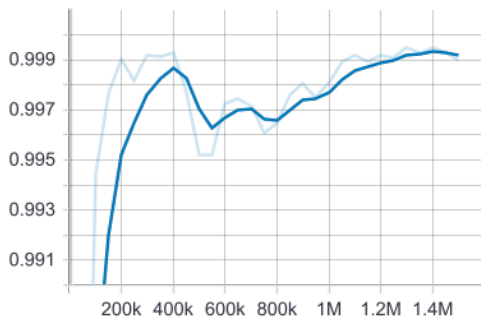
(b) Policy entropy over the training steps



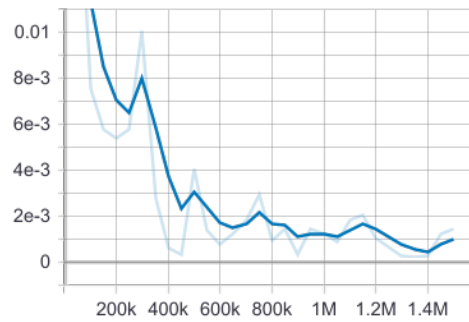
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.15: Player simulation experiment 15 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 0.35
Reward details	coins = 0.1, moving forward = dx/5000, <b>lose = -1</b>
GAIL	gamma = 0.99, strength = 0.65, learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of nine games, which corresponds to about 1 hour of gameplay
Observations	Image grayscale
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life

#### 5.1.1.15 Player simulation experiment 15

In the last experiment, the agent gave too much importance to his life and, consequently, did not move forward. In this experiment, the reward value will be modified to give less importance to losing a life. The table 5.15 shows the configuration used for this experiment.

Results:

- The agent trained for about 4M steps.
- The final trained agent seems to perform worse than the previous experiment.
- The algorithm results (figure 5.16) showed that the policy increased over time, and the entropy did not decrease.

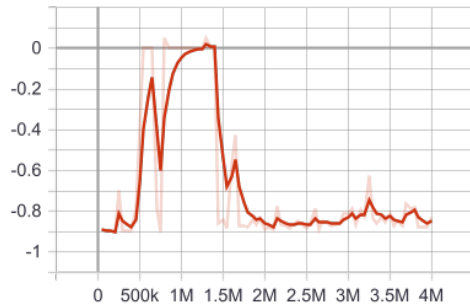
#### 5.1.1.16 Player simulation experiment 16

In this experiment, the agent will give significant importance to the extrinsic reward. The GAIL and behavioral cloning algorithm will have a low strength but will still help with the training. The table 5.16 shows the configuration used for this experiment.

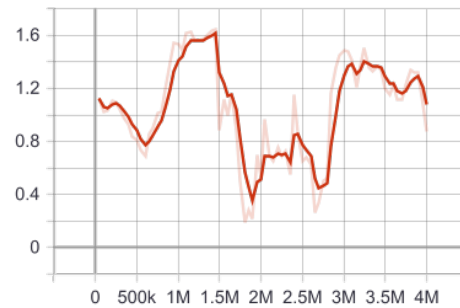
Results:

- The agent trained for about 2M steps.

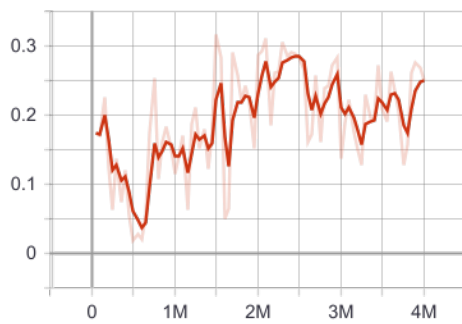
Figure 5.16: Player simulation experiment 15 results



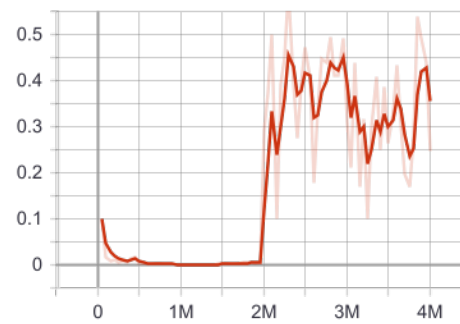
(a) Mean cumulative episode reward over the training steps



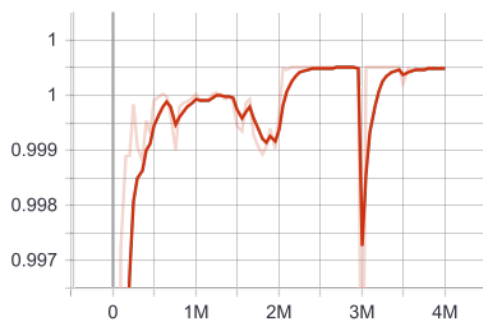
(b) Policy entropy over the training steps



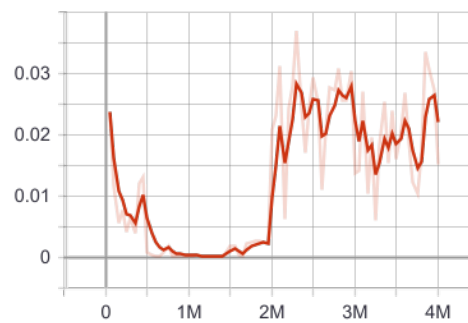
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps



Table 5.16: Player simulation experiment 16 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 1.0</b>
Reward details	coins = 0.1, moving forward = dx/5000, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.05</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.1</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of eight games, which corresponds to about 45 minutes of gameplay</b>
Observations	<b>Image grayscale + ray perception sensor</b>
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life

- The final trained agent has trouble completing the starting section. The previous experiments deal with this better since the imitation learning algorithms have more importance, facilitating and reducing the amount of learning needed to understand how to play the game.
- The algorithm results (figure 5.17) showed that the cumulative reward is converging to 0. Furthermore, the policy loss seems to be unstable.

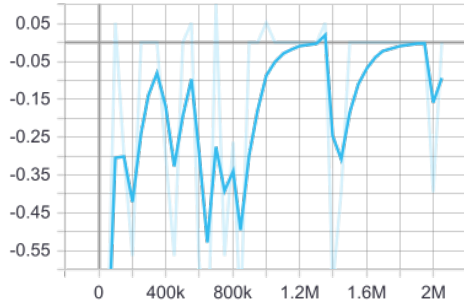
#### 5.1.1.17 Player simulation experiment 17

In this experiment, the extrinsic reward was still given the majority of the importance. Furthermore, the rewards given by the game were modified to test their performance. The table 5.17 shows the configuration used for this experiment.

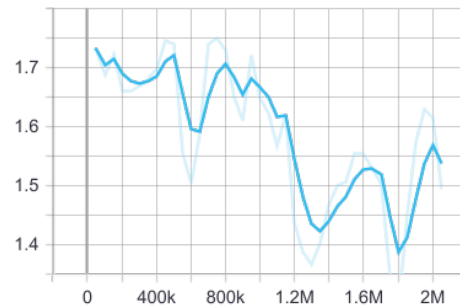
Results:

- The agent trained for about 13M steps.
- The final trained agent can pass some sections more consistently while having trouble with other sections.
- The algorithm results (figure 5.18) showed that the cumulative reward is unstable during the training session, and the policy loss is increasing.

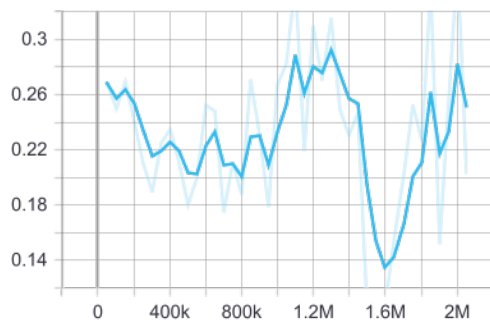
Figure 5.17: Player simulation experiment 16 results



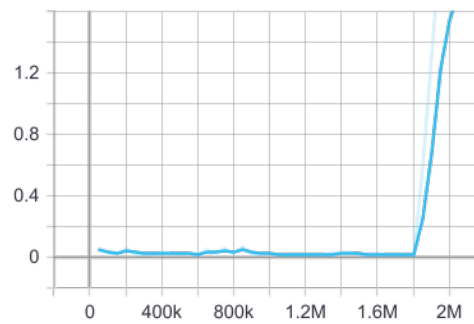
(a) Mean cumulative episode reward over the training steps



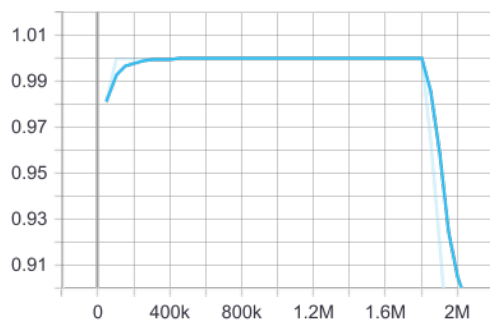
(b) Policy entropy over the training steps



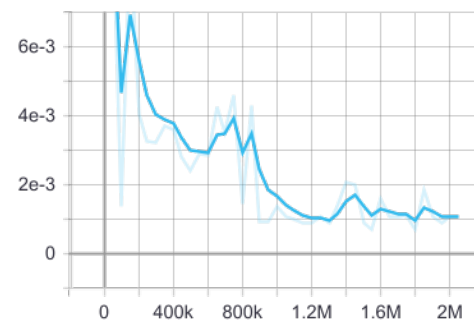
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

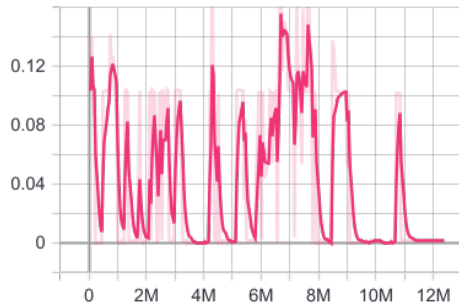


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

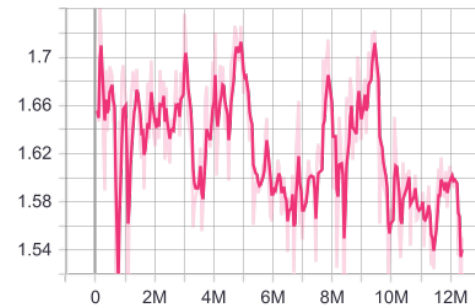


(f) GAIL discriminator's estimate for the policy generated, over the training steps

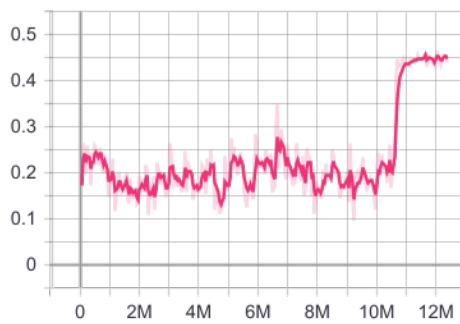
Figure 5.18: Player simulation experiment 17 results



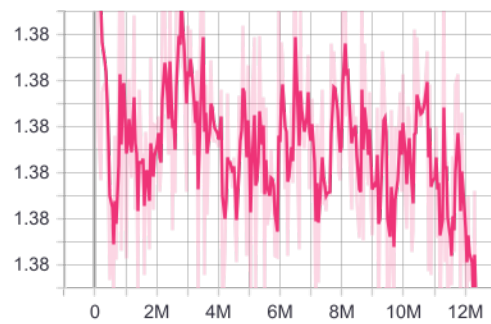
(a) Mean cumulative episode reward over the training steps



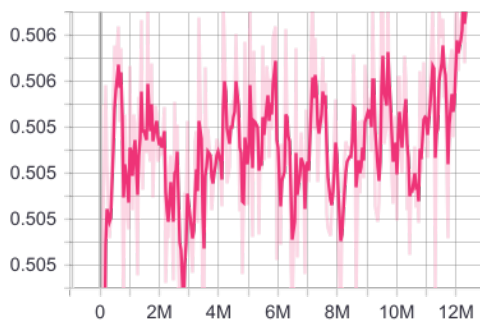
(b) Policy entropy over the training steps



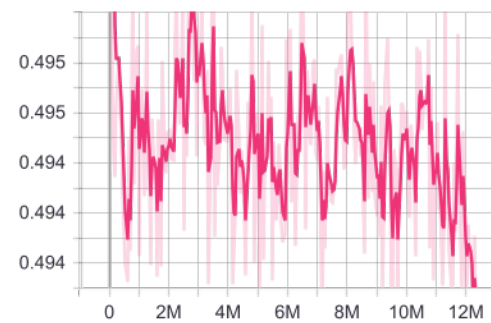
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.17: Player simulation experiment 17 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 1.0
Reward details	<b>coins = 0.1, moving forward = dx/5000</b>
GAIL	gamma = 0.99, strength = 0.05, learning_rate = 0.0003, use_actions = false, <b>use_vail = true</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.1, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life

### 5.1.1.18 Player simulation experiment 18

In this experiment, curriculum learning was introduced to the training session. The sections were divided into four difficulties as described in the section 4.3. The table 5.18 shows the configuration used for this experiment.

Results:

- The agent trained for about 6M steps.
- The final trained agent could not complete the first difficulty of the game since he still has trouble jumping over the stairs.
- The algorithm results (figure 5.19) showed that the policy seems unstable, while the entropy is not decreasing.

### 5.1.1.19 Player simulation experiment 19

In this experiment, the game's reward was changed, including a penalty for standing still and losing a life. The table 5.19 shows the configuration used for this experiment.

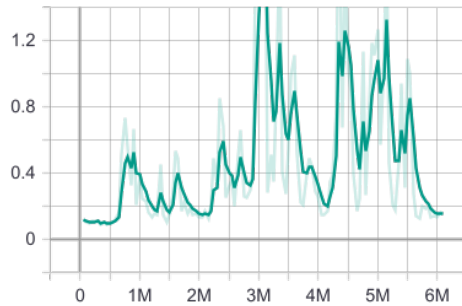
Results:

- The agent trained for about 13M steps.

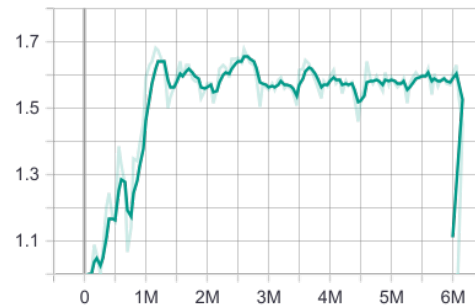
Table 5.18: Player simulation experiment 18 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.35</b>
Reward details	coins = 0.1, moving forward = dx/5000, <b>completing section = 1</b>
GAIL	gamma = 0.99, <b>strength = 0.65</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.8</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
<b>Curriculum learning</b>	<b>4 difficulties with minimum 3 reward</b>

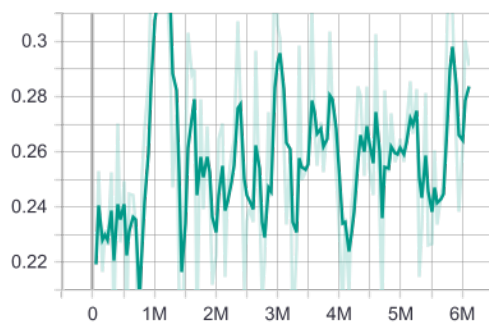
Figure 5.19: Player simulation experiment 18 results



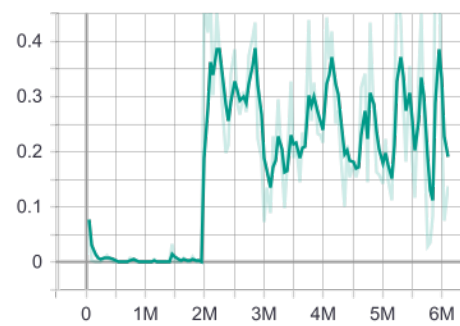
(a) Mean cumulative episode reward over the training steps



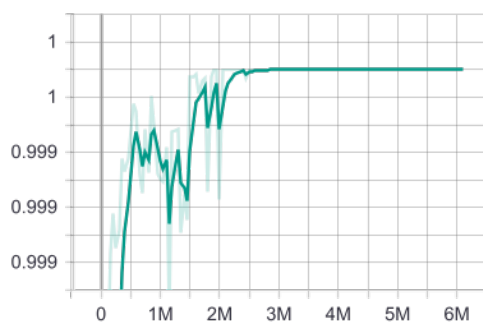
(b) Policy entropy over the training steps



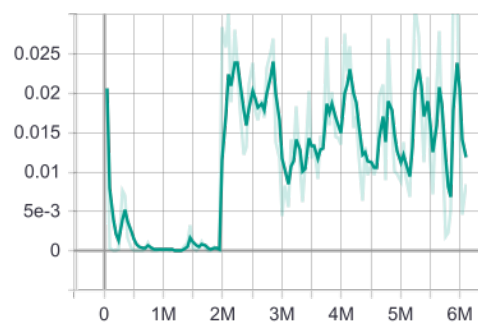
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.19: Player simulation experiment 19 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.65</b>
Reward details	coins = 0.1, moving forward = dx/5000, completing section = 1, <b>lose = -1, standing still &gt;7 seconds = -1 per frame</b>
GAIL	gamma = 0.99, <b>strength = 0.65</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = true</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	4 difficulties with minimum 3 reward

Table 5.20: Player simulation experiment 20 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.01, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, <b>hidden_units = 256, num_layers = 3</b> , vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 0.65
Reward details	coins = 0.1, moving forward = dx/5000, completing section = 1, lose = -1, standing still >7 seconds = -1 per frame
GAIL	gamma = 0.99, strength = 0.65, learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, <b>hidden_units = 128, num_layers = 2</b> , vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>Each section with a minimum 3 reward</b>

- The final trained agent would sometimes stop at the start of the game, not making any movement. Furthermore, the agent has trouble jumping over the stairs.
- The algorithm results (figure 5.20) showed that the entropy decreases, and the policy loss is somewhat constant.

#### 5.1.1.20 Player simulation experiment 20

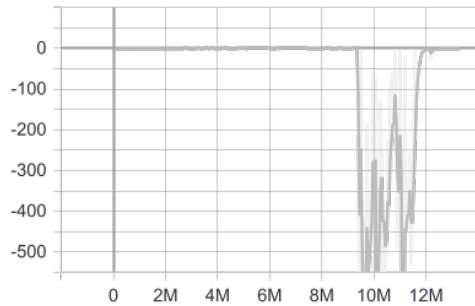
In this experiment, each section was given its difficulty for curriculum learning. Furthermore, the network sizes were increased. The table 5.20 shows the configuration used for this experiment.

Results:

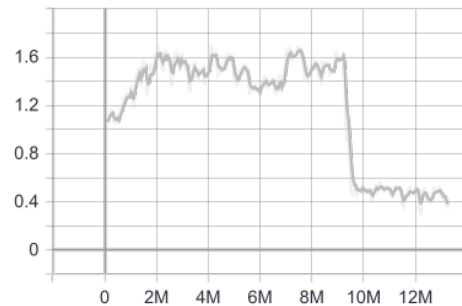
- The agent trained for about 2M steps.
- The final trained agent could not pass the starting section and, consequently, did not change the difficulty.
- The algorithm results (figure 5.21) showed that the entropy increases, and the policy loss is somewhat constant during the training session.



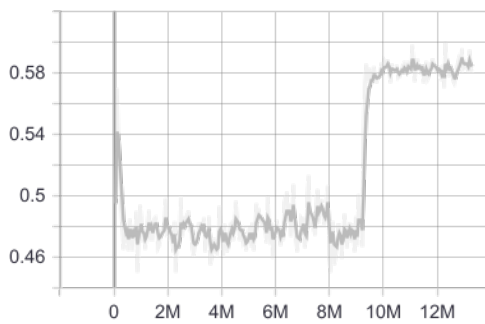
Figure 5.20: Player simulation experiment 19 results



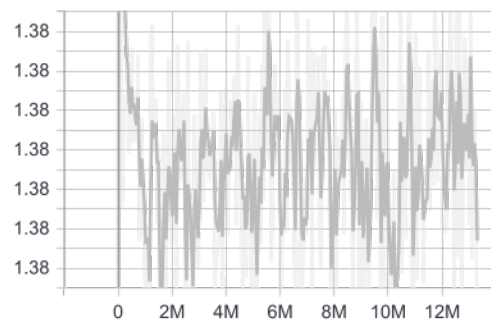
(a) Mean cumulative episode reward over the training steps



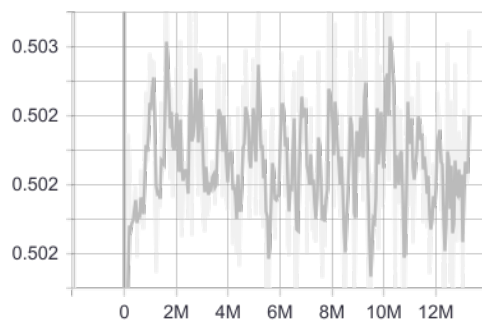
(b) Policy entropy over the training steps



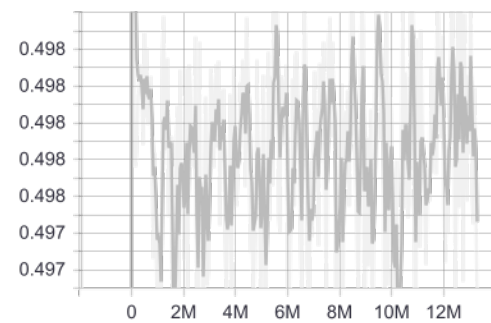
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

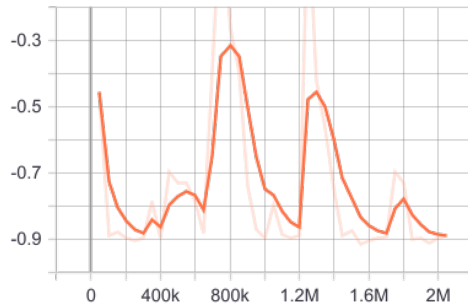


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

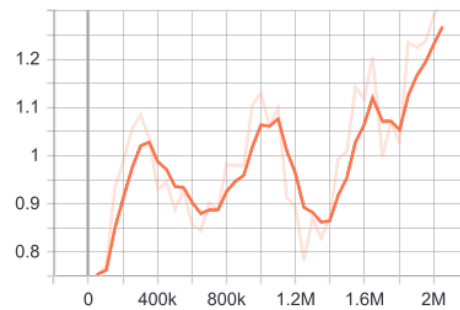


(f) GAIL discriminator's estimate for the policy generated, over the training steps

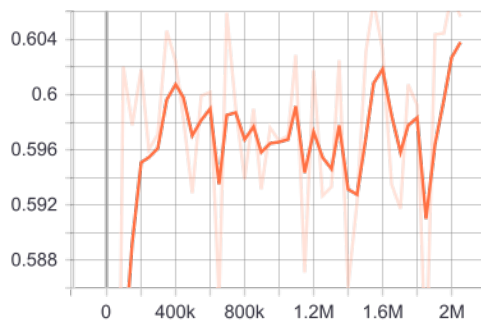
Figure 5.21: Player simulation experiment 20 results



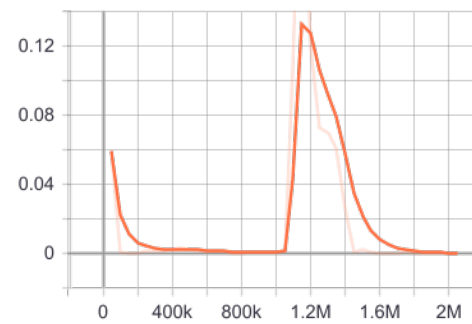
(a) Mean cumulative episode reward over the training steps



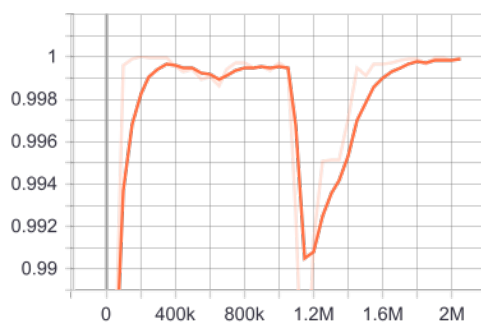
(b) Policy entropy over the training steps



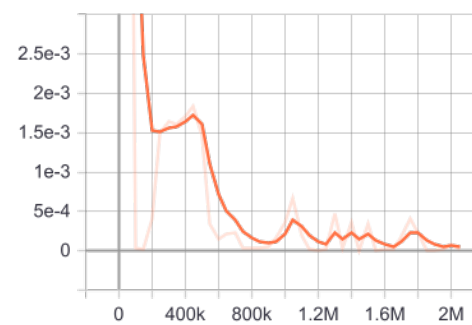
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.21: Player simulation experiment 21 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, <b>beta = 0.005</b> , epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 0.65
Reward details	coins = 0.1, <b>moving forward = dx/250, completing section = 0.3, standing still &gt;7 seconds = -0.05 per frame</b>
GAIL	gamma = 0.99, strength = 0.65, learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	<b>None</b>
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>Each section with a minimum 2 reward</b>

#### 5.1.1.21 Player simulation experiment 21

In this experiment, the rewards were changed, and the minimum reward for passing each difficulty was lowered. Furthermore, behavioral cloning was also removed to test the agent performance without it. The table 5.21 shows the configuration used for this experiment.

Results:

- The agent trained for about 7.5M steps.
- The final trained agent was always jumping and could not complete the starting section.
- The algorithm results (figure 5.22) show that the policy increased and remained constant over the remaining duration. Furthermore, the entropy is unstable during the training session.

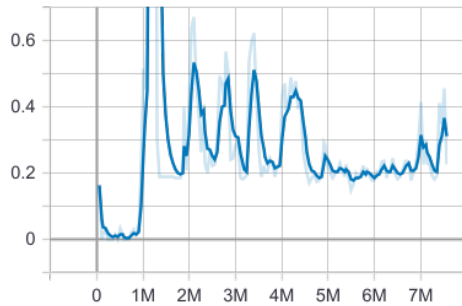
#### 5.1.1.22 Player simulation experiment 22

In this experiment, the reward for losing a life was reintroduced. However, the extrinsic reward importance was significantly lowered. The table 5.22 shows the configuration used for this experiment.

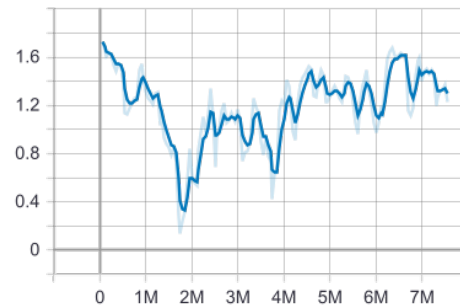
Results:

- The agent trained for about 1.5M steps.

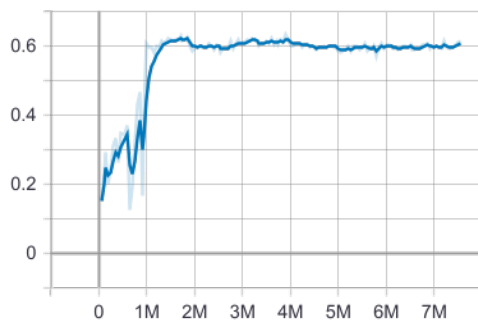
Figure 5.22: Player simulation experiment 21 results



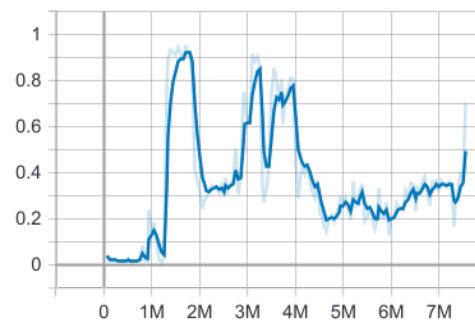
(a) Mean cumulative episode reward over the training steps



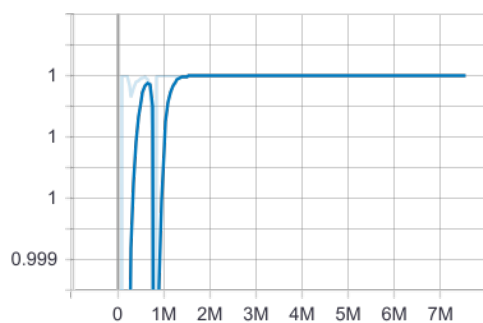
(b) Policy entropy over the training steps



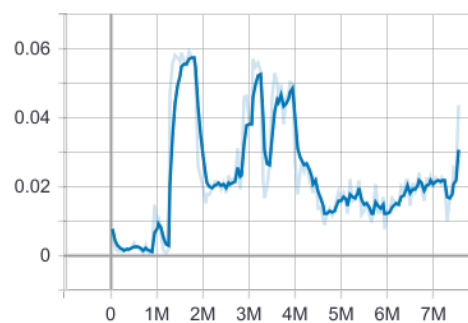
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.22: Player simulation experiment 22 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.05</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, <b>lose = -1</b>
GAIL	gamma = 0.99, <b>strength = 0.95</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = true</b>
GAIL network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

- The final trained agent could not complete the starting section, moving backwards at the start of the game.
- The algorithm results (figure 5.23) show that the policy increased, and the entropy decreased over the training session.

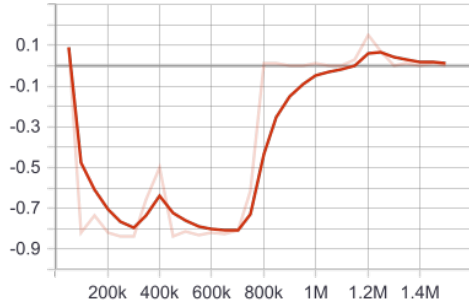
### 5.1.1.23 Player simulation experiment 23

In this experiment, behavioral cloning was reintroduced, and the extrinsic reward was removed. The table 5.23 shows the configuration used for this experiment.

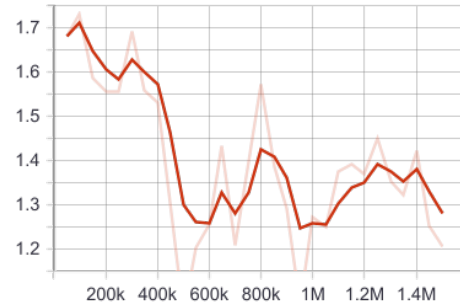
Results:

- The agent trained for about 1.5M steps.
- The final trained agent knows reasonably well how to complete the starting section and the first difficulty. However, the following section difficulty, which requires a jump, could not be completed.
- The algorithm results (figure 5.24) show that the policy loss and the entropy decrease over the training session.

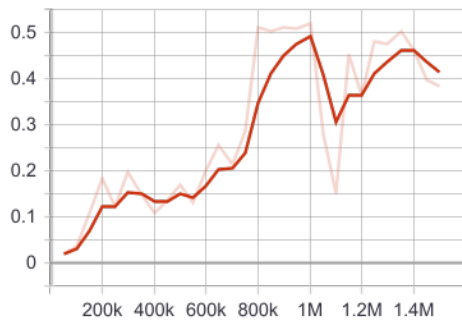
Figure 5.23: Player simulation experiment 22 results



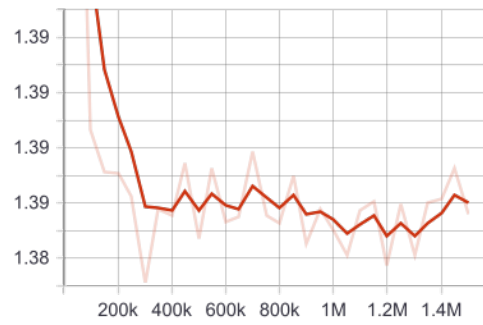
(a) Mean cumulative episode reward over the training steps



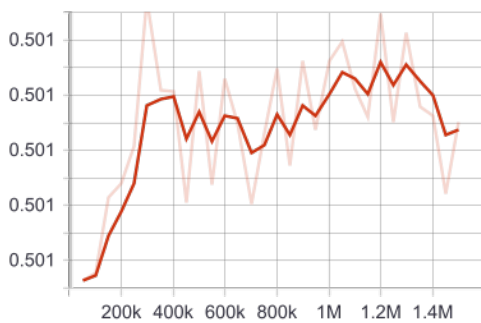
(b) Policy entropy over the training steps



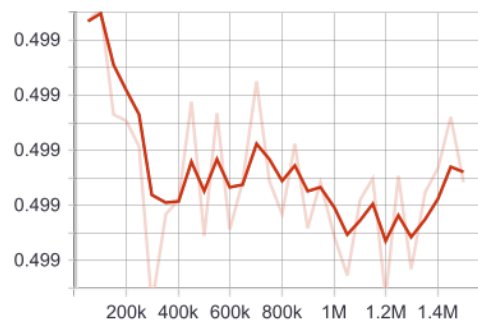
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

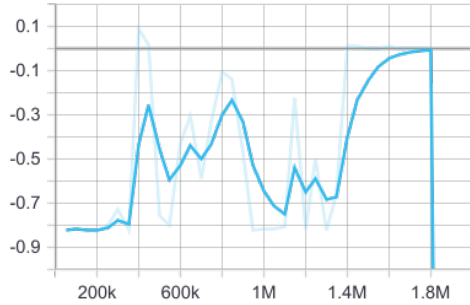


(f) GAIL discriminator's estimate for the policy generated, over the training steps

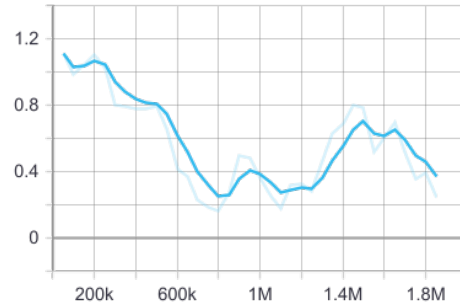
Table 5.23: Player simulation experiment 23 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	<b>None</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 1.0</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	<b>steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None</b>
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

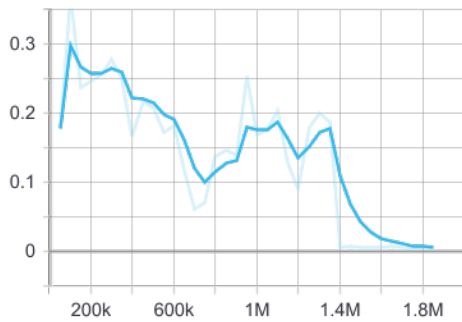
Figure 5.24: Player simulation experiment 23 results



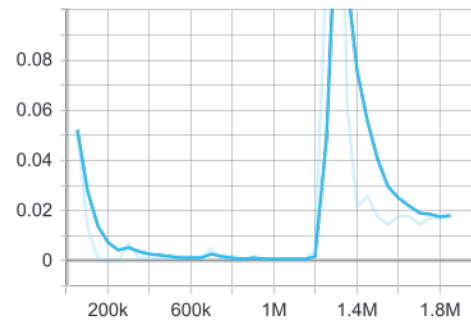
(a) Mean cumulative episode reward over the training steps



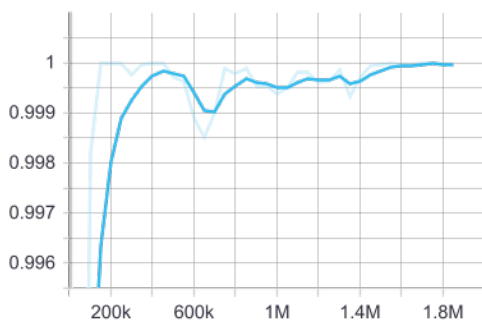
(b) Policy entropy over the training steps



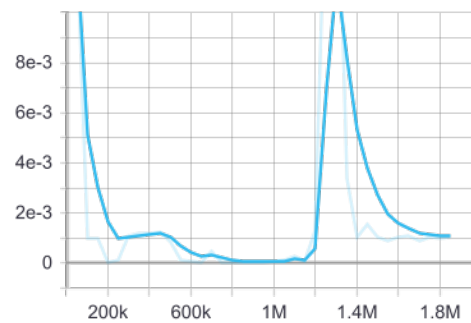
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps



Table 5.24: Player simulation experiment 24 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 1.0</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.005 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 128, num_layers = 2, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of eight games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

#### 5.1.1.24 Player simulation experiment 24

In this experiment, the extrinsic reward was given all the importance, and the other algorithms do not affect the training. The table 5.24 shows the configuration used for this experiment.

Results:

- The agent trained for about 7.5M steps.
- The final trained agent can not complete the starting section.
- The algorithm results (figure 5.25) show that the policy loss increases and remains constant for the rest of the training session. Furthermore, the entropy also seems to be unstable.

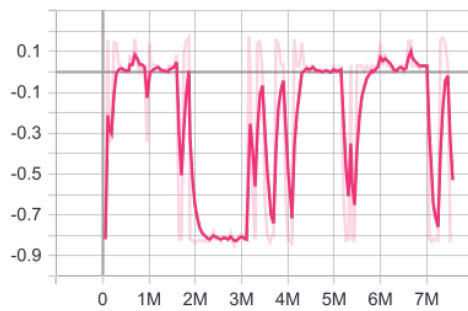
#### 5.1.1.25 Player simulation experiment 25

In this experiment, the movement system was remade as described in section 4.3. Furthermore, the starting section was replaced with the more simplistic version. Finally, the curriculum learning was also removed to test the performance of the new movement system. The table 5.25 shows the configuration used for this experiment.

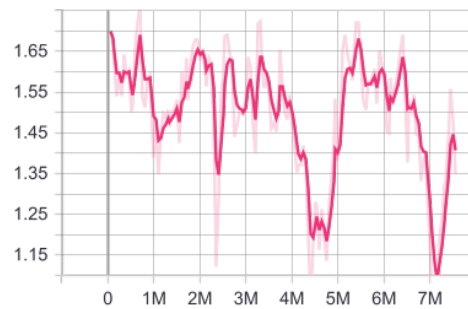
Results:

- The agent trained for about 2.3M steps.

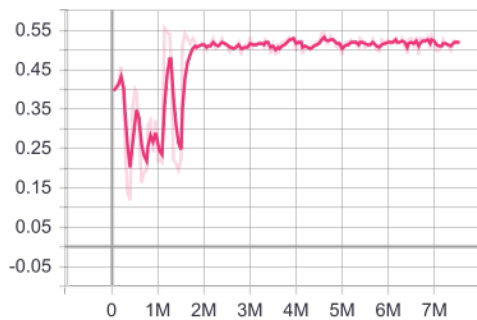
Figure 5.25: Player simulation experiment 24 results



(a) Mean cumulative episode reward over the training steps



(b) Policy entropy over the training steps



(c) Mean magnitude of the policy loss function over the training steps

Table 5.25: Player simulation experiment 25 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.35</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.65</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, <b>hidden_units = 64, num_layers = 1</b> , vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.8</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	<b>Recording of ten games, which corresponds to about 45 minutes of gameplay</b>
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>None</b>

Table 5.26: Player simulation experiment 26 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.15</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.85</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = true</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.9</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	None

- The final trained agent sometimes does not jump over the stairs and some water pits but has overall good results.
- The algorithm results (figure 5.26) show that the cumulative reward falls over the training session. Furthermore, the policy loss increases and the entropy decreases during the training.

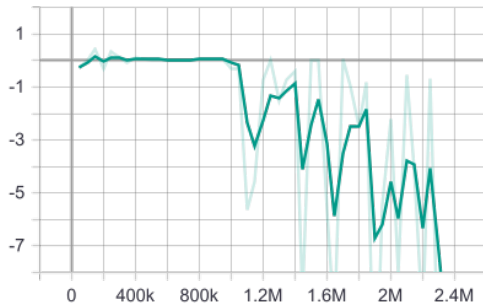
#### 5.1.1.26 Player simulation experiment 26

In this experiment, GAIL and behavioral cloning were given more importance than the extrinsic reward. The table 5.26 shows the configuration used for this experiment.

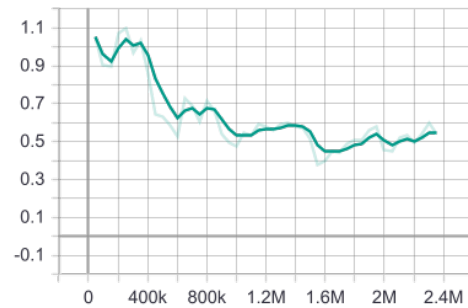
Results:

- The agent trained for about 9M steps.
- The final trained agent has the same problems as the previous experiment but has worse results at completing the sections.
- The algorithm results (figure 5.27) show that the cumulative reward decreases over the training session. Furthermore, the policy loss remains constant, and the entropy decreases during the training.

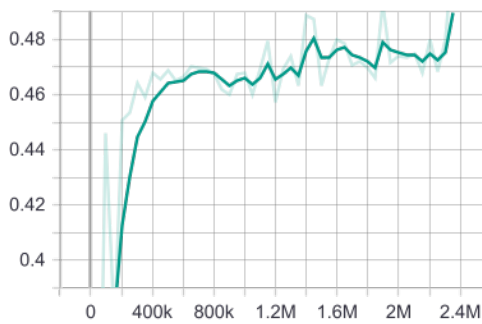
Figure 5.26: Player simulation experiment 25 results



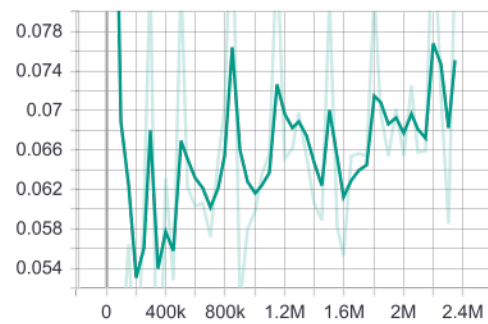
(a) Mean cumulative episode reward over the training steps



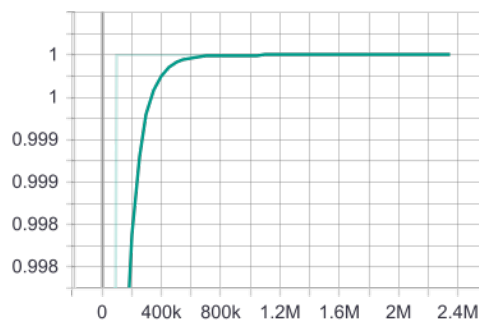
(b) Policy entropy over the training steps



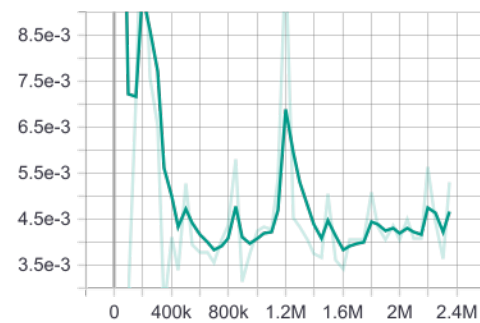
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

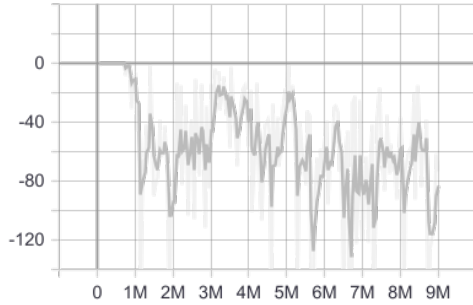


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

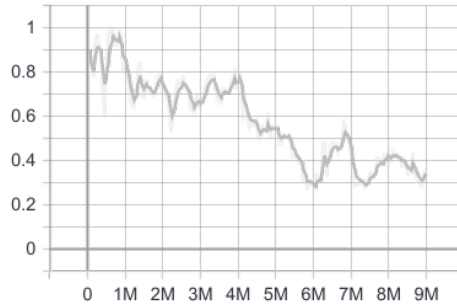


(f) GAIL discriminator's estimate for the policy generated, over the training steps

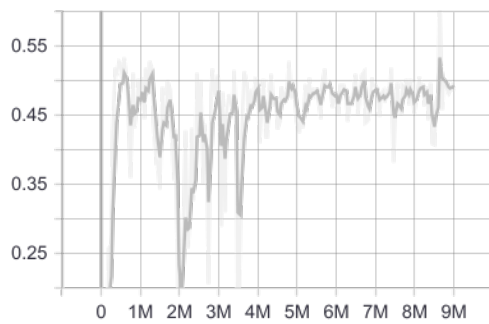
Figure 5.27: Player simulation experiment 26 results



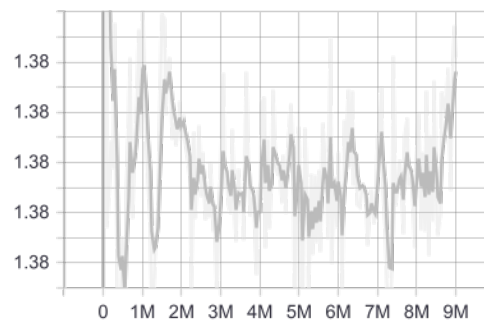
(a) Mean cumulative episode reward over the training steps



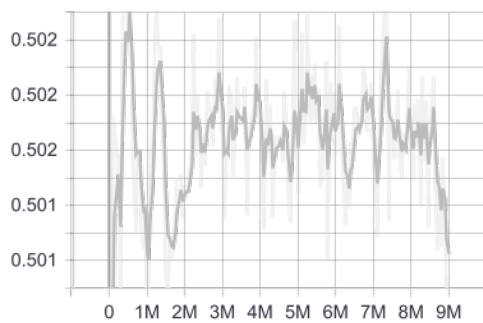
(b) Policy entropy over the training steps



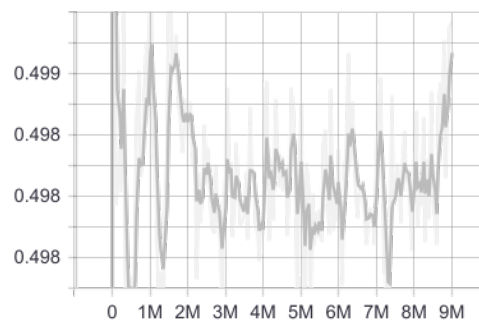
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.27: Player simulation experiment 27 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.9</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.1</b> , learning_rate = 0.0003, use_actions = false, <b>use_vail = false</b>
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.1</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 1
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>Each section with a minimum 2 reward</b>

#### 5.1.1.27 Player simulation experiment 27

In this experiment, curriculum learning was reintroduced with the same settings as the last usage. Furthermore, the extrinsic reward now holds the most value for the algorithm. The table 5.27 shows the configuration used for this experiment.

Results:

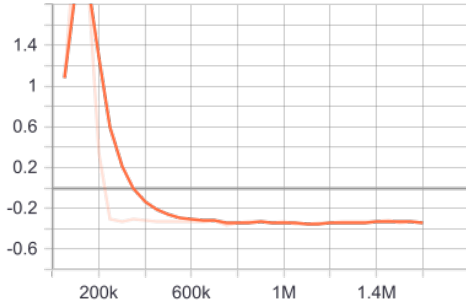
- The agent trained for about 1.5M steps.
- The final trained agent completed the first difficulty of the curriculum learning. However, in the next difficulty, the problem of going up the stairs is evidenced.
- The algorithm results (figure 5.28) show that the cumulative reward falls over the training session. Furthermore, the policy loss increases and the entropy decreases during the training.

#### 5.1.1.28 Player simulation experiment 28

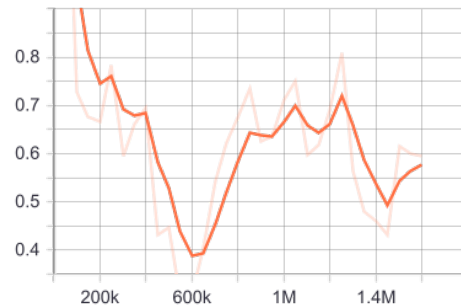
In this experiment, curriculum learning was removed, the strength of each reward was changed, and the training session was much longer. The table 5.28 shows the configuration used for this experiment.

Results:

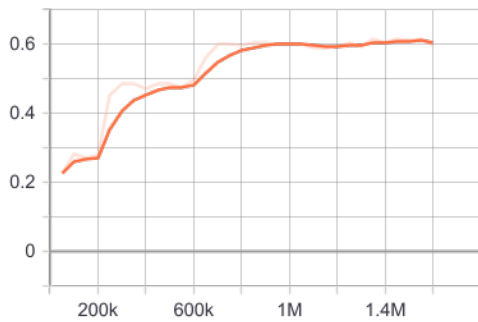
Figure 5.28: Player simulation experiment 27 results



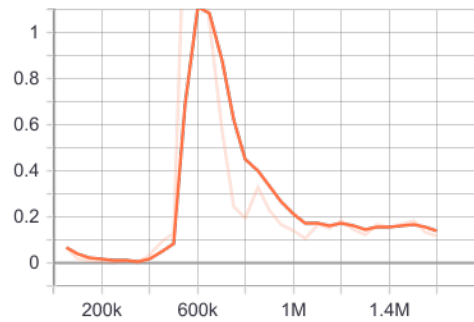
(a) Mean cumulative episode reward over the training steps



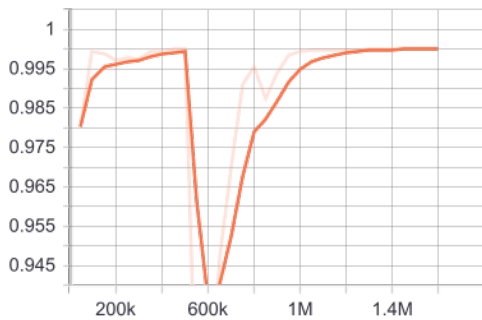
(b) Policy entropy over the training steps



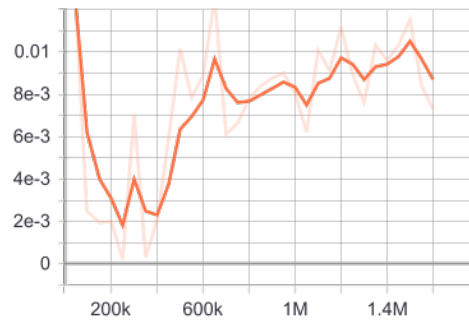
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps



Table 5.28: Player simulation experiment 28 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 1</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.1</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.5</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, <b>timescale for game = 20</b>
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>None</b>

Table 5.29: Player simulation experiment 29 configuration

Settings	Description
Sac hyperparameters	<b>batch_size=128, buffer_size=500000, learning_rate = 0.0003, learning_rate_schedule = constant, buffer_init_steps = 10000, tau = 0.005, steps_per_update = 10.0, save_replay_buffer = false, init_entcoef = 0.05, reward_signal_steps_per_update = 10.0</b>
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 2</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.1</b> , learning_rate = 0.0003, <b>use_actions = true</b> , use_vail = false
GAIL network settings	normalize = false, <b>hidden_units = 128, num_layers = 2</b> , vis_encode_type = simple, memory = none
Behavioral cloning	<b>None</b>
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, <b>timescale for game = 20</b>
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>Each section with a minimum 2 reward</b>

- The agent trained for about 12M steps.
- The final trained agent shows the same problems of not being able to perform jumps over stairs.
- The algorithm results (figure 5.29) show that the policy loss is unstable, and the entropy decreases over the training session. Additionally, the GAIL loss is somewhat constant during the training.

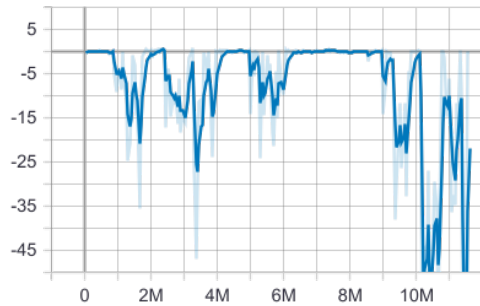
### 5.1.1.29 Player simulation experiment 29

In this experiment, the algorithm SAC was used to test its effectiveness for player simulation. The table 5.29 shows the configuration used for this experiment.

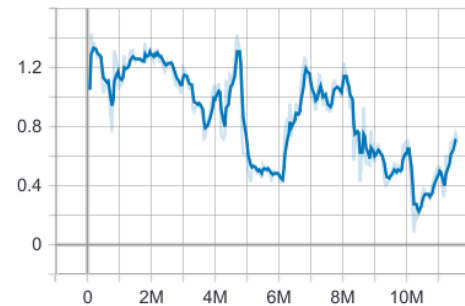
Results:

- The agent trained for about 2M steps.
- The final trained agent was able to complete the first difficulty but was stuck on the second one. However, the overall training session consumed a significant amount of pc memory, which hindered the training.

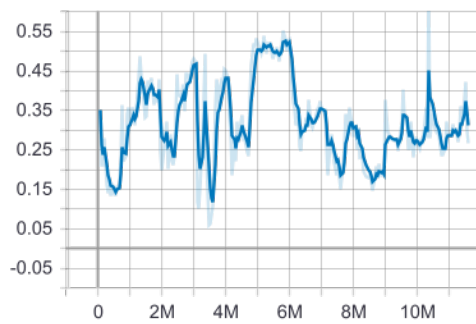
Figure 5.29: Player simulation experiment 28 results



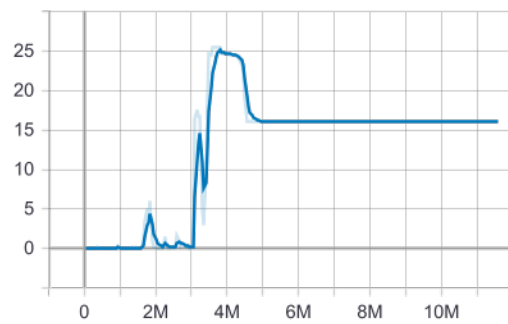
(a) Mean cumulative episode reward over the training steps



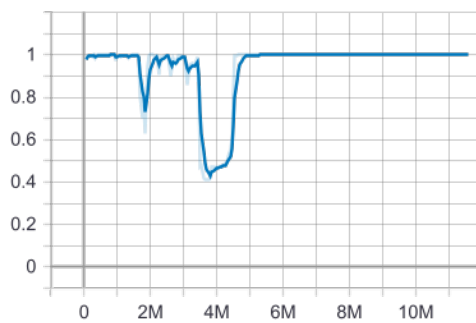
(b) Policy entropy over the training steps



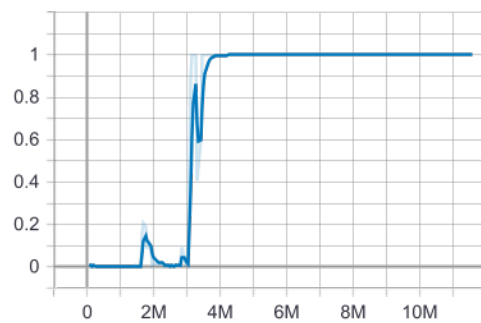
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.30: Player simulation experiment 30 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 256, num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.5</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 1.0</b> , learning_rate = 0.0003, <b>use_actions = true</b> , use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 1.0</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 20
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	<b>Each section with a minimum 2 reward</b>

- The algorithm results (figure 5.30) show that the policy loss increases, and the entropy is unstable during the training session.

### 5.1.1.30 Player simulation experiment 30

In this experiment, the PPO algorithm was used in conjunction with curriculum learning and the use\_actions of GAIL. The table 5.30 shows the configuration used for this experiment.

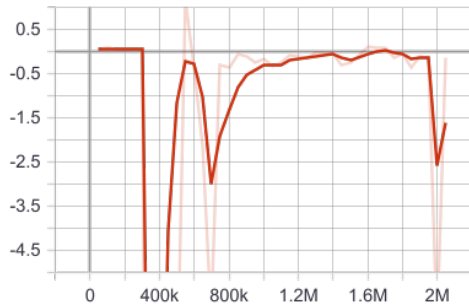
Results:

- The agent trained for about 10M steps.
- The final trained agent was able to complete the first difficulty but could not pass the second difficulty.
- The algorithm results (figure 5.31) show that the policy loss and the entropy are unstable.

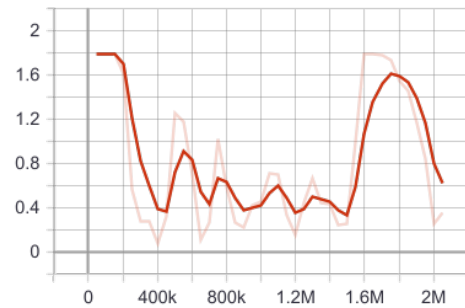
### 5.1.1.31 Player simulation experiment 31

This experiment tried to use a different system to compensate for the problems found when using the SAC algorithm. However, the agent could not be trained since the system was not enough to train the algorithm.

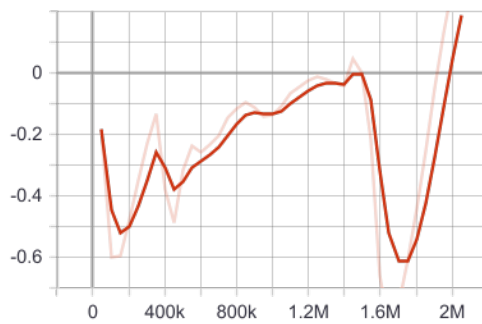
Figure 5.30: Player simulation experiment 29 results



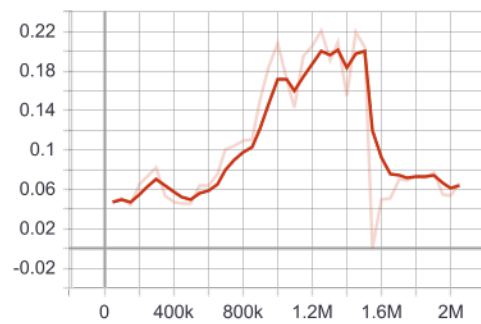
(a) Mean cumulative episode reward over the training steps



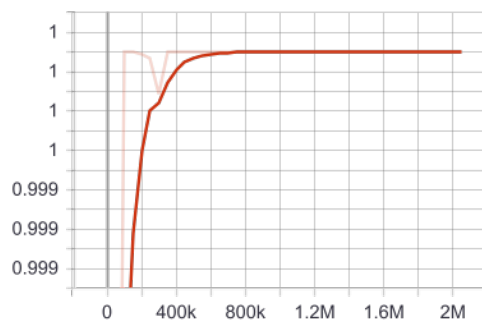
(b) Policy entropy over the training steps



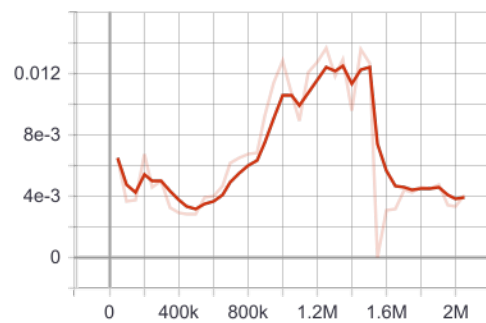
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps

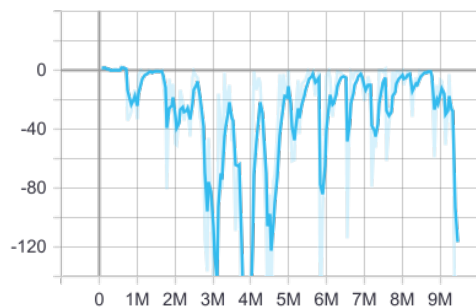


(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps

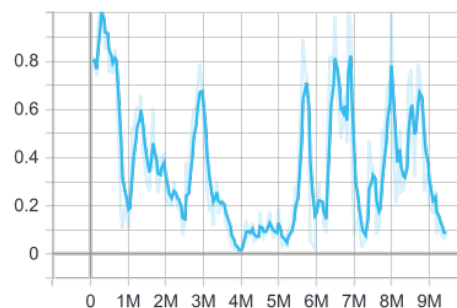


(f) GAIL discriminator's estimate for the policy generated, over the training steps

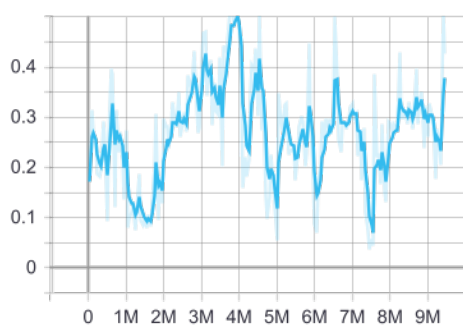
Figure 5.31: Player simulation experiment 30 results



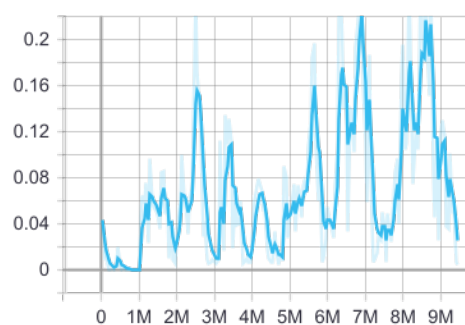
(a) Mean cumulative episode reward over the training steps



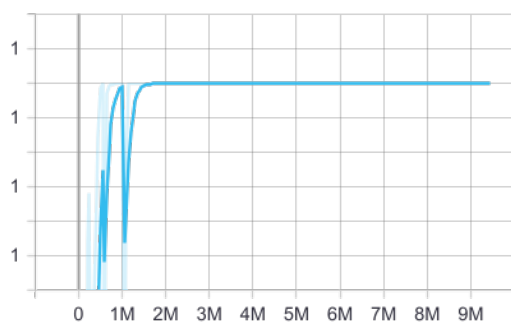
(b) Policy entropy over the training steps



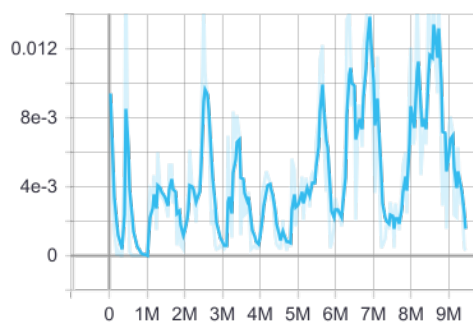
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.31: Player simulation experiment 32 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, <b>hidden_units = 512</b> , num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 1.0</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0</b> , learning_rate = 0.0003, use_actions = true, use_vail = false
GAIL network settings	normalize = false, hidden_units = 64, num_layers = 1, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 1.0</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 20
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

### 5.1.1.32 Player simulation experiment 32

In this experiment, the PPO algorithm was used in conjunction with curriculum learning and the use\_actions of GAIL. The table 5.30 shows the configuration used for this experiment.

Results:

- The agent trained for about 2M steps.
- The final trained agent was not able to complete the first difficulty.
- The algorithm results (figure 5.32) show that the policy loss was constant, and the entropy was decreasing during the training session

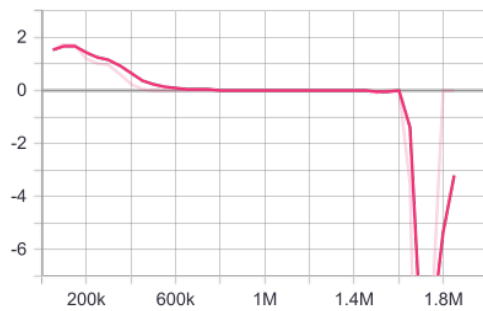
### 5.1.1.33 Player simulation experiment 33

In this experiment, the algorithm SAC was used without the GAIL or behavioural cloning algorithms. This experiment allows using the SAC algorithm without memory restrictions. The table 5.32 shows the configuration used for this experiment.

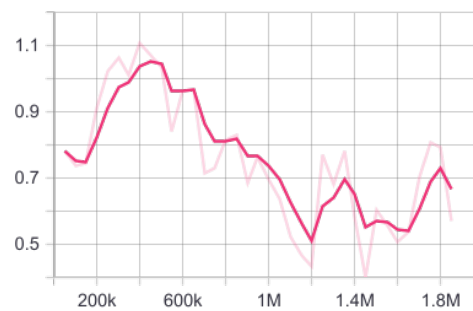
Results:

- The agent trained for about 7M steps.

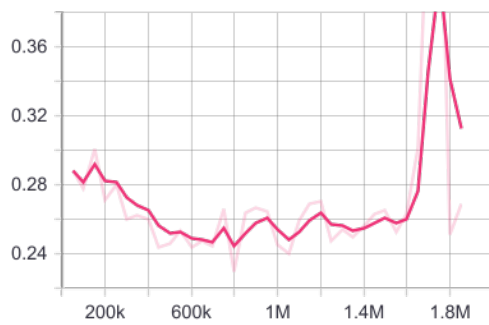
Figure 5.32: Player simulation experiment 32 results



(a) Mean cumulative episode reward over the training steps



(b) Policy entropy over the training steps



(c) Mean magnitude of the policy loss function over the training steps



Table 5.32: Player simulation experiment 33 configuration

Settings	Description
Sac hyperparameters	batch_size=128, buffer_size=500000, learning_rate = 0.0003, learning_rate_schedule = constant, buffer_init_steps = 10000, tau = 0.005, steps_per_update = 10.0, save_replay_buffer = false, init_entcoef = 0.05, reward_signal_steps_per_update = 10.0
Network settings	normalize = false, <b>hidden_units = 512</b> , num_layers = 3, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 1</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, lose = -1
GAIL	<b>None</b>
GAIL network settings	<b>None</b>
Behavioral cloning	<b>None</b>
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 20
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

- The final trained agent was not able to complete the second difficulty of the curriculum learning.
- The algorithm results (figure 5.33) show that the policy loss increases, and the entropy is unstable during the training session.

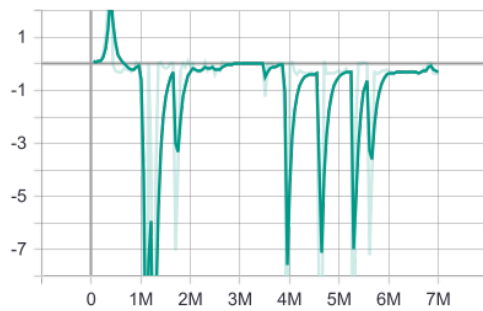
#### 5.1.1.34 Player simulation experiment 34

In this experiment, the PPO network was further increased in size, as well as the GAIL network. The table 5.33 shows the configuration used for this experiment.

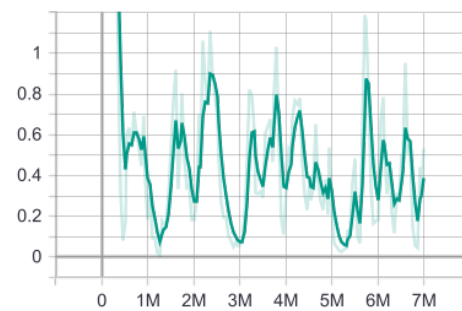
Results:

- The agent trained for about 4.5M steps.
- The final trained agent could not complete the second difficulty since the problem of jumping stairs is still present.
- The algorithm results (figure 5.34) show that the reward decreased and remained constant for the rest of the training. Furthermore, the policy loss and entropy increased, and the pretraining loss remained relatively constant.

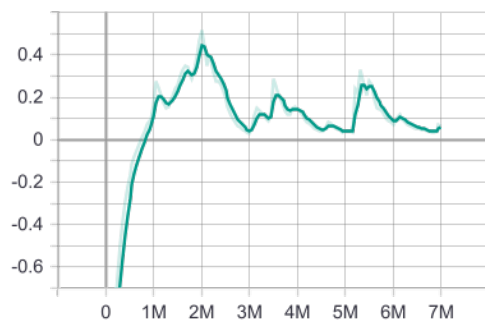
Figure 5.33: Player simulation experiment 33 results



(a) Mean cumulative episode reward over the training steps



(b) Policy entropy over the training steps

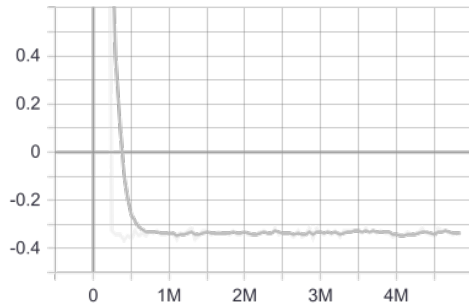


(c) Mean magnitude of the policy loss function over the training steps

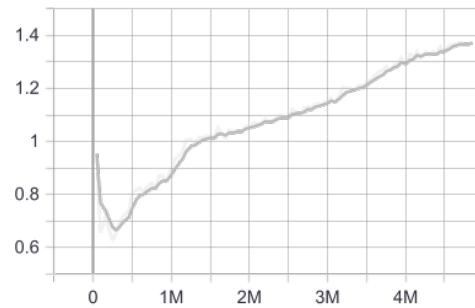
Table 5.33: Player simulation experiment 34 configuration

Settings	Description
PPO hyperparameters	batch_size=128, buffer_size=1024, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambda = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, <b>hidden_units = 2048</b> , <b>num_layers = 12</b> , vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, <b>strength = 0.25</b>
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -0.05 per frame, lose = -1
GAIL	gamma = 0.99, <b>strength = 0.75</b> , learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, <b>hidden_units = 1024</b> , <b>num_layers = 6</b> , vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, <b>strength = 0.8</b> , samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 20
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

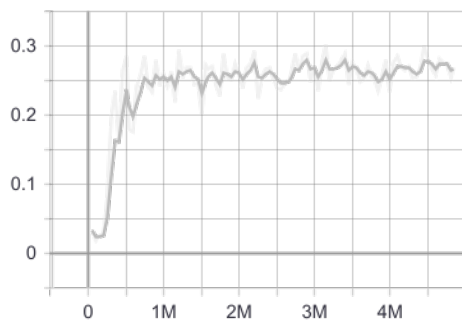
Figure 5.34: Player simulation experiment 34 results



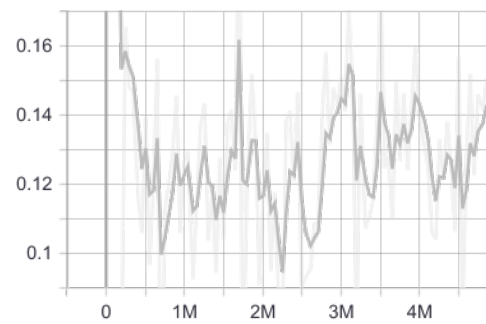
(a) Mean cumulative episode reward over the training steps



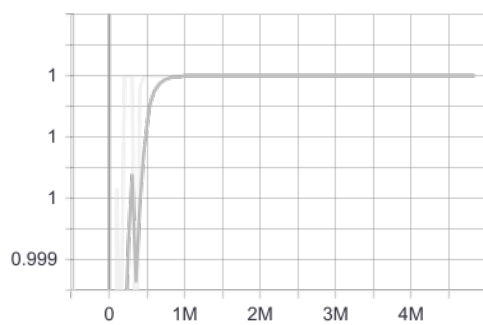
(b) Policy entropy over the training steps



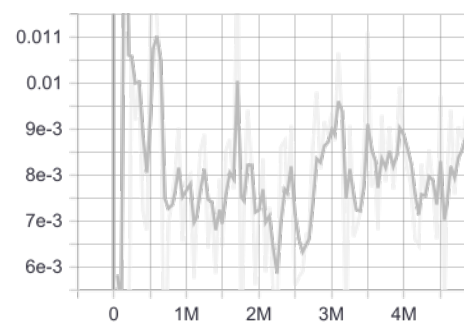
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

Table 5.34: Player simulation experiment 35 configuration

Settings	Description
PPO hyperparameters	batch_size=128, <b>buffer_size=128000</b> , learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 2048, num_layers = 12, vis_encode_type = simple
Memory	sequence_length = 64, memory_size = 256
Reward extrinsic	gamma = 0.99, strength = 0.25
Reward details	coins = 0.1, moving forward = dx/250, completing section = 0.3, standing still >7 seconds = -.05 per frame, lose = -1
GAIL	gamma = 0.99, strength = 0.75, learning_rate = 0.0003, use_actions = false, use_vail = false
GAIL network settings	normalize = false, hidden_units = 1024, num_layers = 6, vis_encode_type = simple, memory = none
Behavioral cloning	steps = 0, strength = 0.8, samples_per_update = 1024, num_epoch = None, batch_size = None
Demonstrations	Recording of ten games, which corresponds to about 45 minutes of gameplay
Observations	Image grayscale + ray perception sensor
Training settings	Number of environments = 8, timescale for game = 20
Simpler game	1 minute and 15 seconds, 1 life
Curriculum learning	Each section with a minimum 2 reward

### 5.1.1.35 Player simulation experiment 35

In this experiment, the PPO buffer\_size was increased. The table 5.34 shows the configuration used for this experiment.

Results:

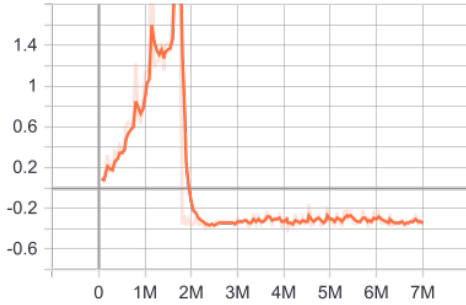
- The agent trained for about 7M steps.
- The final trained agent could not complete the second difficulty since the problem of jumping stairs is still present.
- The algorithm results (figure 5.35) show that the reward decreased and remained constant for the rest of the training. Furthermore, the policy loss and entropy increased.

## 5.1.2 Experiments overall results and analysis

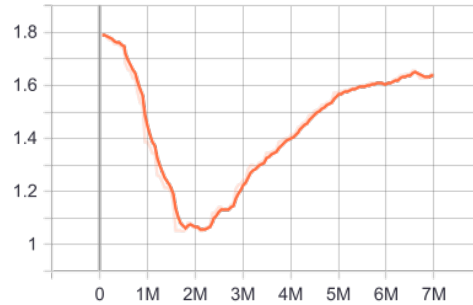
Although the overall objective of the player simulation was not achieved, some experiments showed some potential to accomplish the desired results. The following list describes the overall results and analysis for the experiments performed:

- The use of several environments during training improves the overall amount of steps done and improves the training time. Furthermore, the timescale value also improves the training

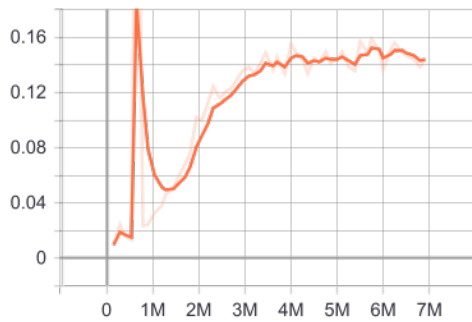
Figure 5.35: Player simulation experiment 35 results



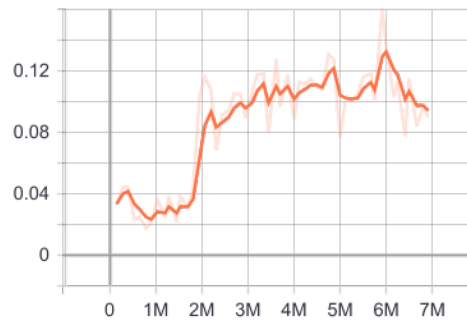
(a) Mean cumulative episode reward over the training steps



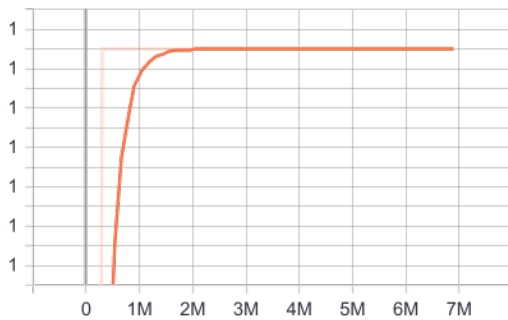
(b) Policy entropy over the training steps



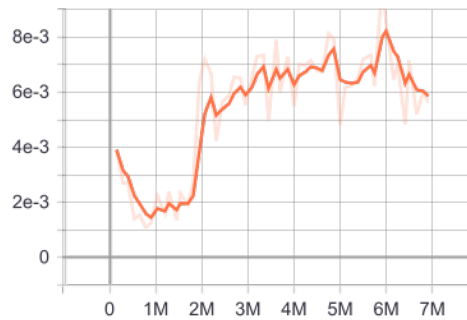
(c) Mean magnitude of the policy loss function over the training steps



(d) Mean magnitude of the GAIL discriminator loss over the training steps



(e) GAIL discriminator's estimate for the expert demonstrations, over the training steps



(f) GAIL discriminator's estimate for the policy generated, over the training steps

times, but if the value is too large, the game can skip frames, making the character take an extended movement that the algorithm was not anticipating, affecting the final result.

- The setting `use_actions` seems to give a better result when set to false. This setting allows GAIL to discriminate based on the action taken. However, sometimes the same state can have different actions depending on the past information and to simplify the problem faced, not discriminating based on actions is the better approach.
- The behavioral cloning `samples_per_update` value was changed based on the ml-agents toolkit recommendation for it to be the same value as `buffer_size`.
- The use of recordings from one hour to thirty minutes does not significantly affect the final result. The expert demonstration should at least complete each section two times and even go through several different situations and states in that section. However, the recordings were made using random faced sections, which can affect the final expert demonstration and in this case, using an extended recording is better since it is more probable for the user to complete all sections.
- The use of memory in the agent network affects the agent's decision-making, especially in section 15. In this section, the agent would usually lose, making actions different from the recorded expert demonstrations. With the memory, the agent follows the demonstration strategy of waiting and then moving forward. Thus, the use of memory helps the agent replicate some of the user's strategies. Furthermore, a large memory seems to impair the agent's performance, while a small memory does not allow the agent to remember enough information to complete some part of the sections.
- In terms of observations, the use of scalar observations confused the decision making of the agents. This problem happens because the expert observations do not hold enough data points for every situation in the game, such as when the user has different amounts of lives or coins or time remaining. The use of images and the ray perception sensor gave overall better results. Furthermore, using only images in the experiments was slightly better, giving better agent's performance. However, this dissertation did not explore the use of images, which should be investigated in future work.
- All the settings values used for PPO and SAC algorithms were changed based on the recommendations of the ml-agents toolkit documentation.
- The network size for the PPO and GAIL algorithm should only be bigger if the game is more complex, requiring a more detailed understanding of the observations gathered. The size of 256 for `hidden_units` and 3 for `num_layers` for the PPO algorithm, and 128 `hidden_units` and 2 `num_layers` for the GAIL algorithm gave the best result for the agent. Small-sized networks can also be seen in player simulation works (section 2.2.5) and other reinforcement learning problems.

- The use of the setting *use\_vail* does not seem to affect the agent's performance, and its usage in this problem was not evident by the results. However, the use of *vail* increases the training time, as referenced by the toolkit's documentation.
- The use of an extrinsic reward helps to orientate the agent when presented with a situation that the expert demonstration had a low amount of examples or even none. However, creating this reward function was not a trivial task, as every change made to the final agent had a different performance.
- The curiosity reward gave an overall worse result than when the agent was not using it. This reward is more suited to situations where the objective is to create a ground-up agent, and the environment rewards are very sparse. Furthermore, this reward will allow the agent to visit states where he has not been before, which in the case of this game, can help create an agent that knows how to backtrack to find rewards.
- The strength value given to each reward affects the final result. The objective of this player simulation was to create an agent that can replicate the expert demonstrations. Therefore, the GAIL and behavioral cloning algorithm should be given the majority of importance, and the extrinsic reward should have a supporting role in the training to assist with states where there are not many examples.
- The use of behavioral cloning has a significant impact on the agent's final results. This algorithm helps the agent replicate the user's actions in certain states. It also helps during the training since it rapidly starts up the agent to perform the correct actions, reducing the overall training time. Additionally, the ml-agents toolkit also allows this algorithm to be trained for several steps and then suspend it for the remainder of the training. In this work, this algorithm was always used to help train the agent to replicate the user's actions in each section and state.
- The use of curriculum learning in the experiments did not prove beneficial since the agent did not train over all the sections, being stuck on one difficulty for most of the training session. However, curriculum learning helped to identify the primary problems that the agent had with each section. To further improve the usage of this method, for each difficulty, the agent should have a corresponding expert demonstration instead of receiving demonstrations for all sections at the same time.
- In the experiments with the best results, the agents could complete some sections almost perfectly, while they had more trouble in other sections. These problems included jumping over the stairs, the input lag caused by the movement system and the lack of demonstrations for some specific states and situations, which affected the final agents' performance and can be resolved in future work.
- The use of a simple game and the extrinsic reward restricted the ability of the agent to copy the user's strategy since these methods create a general approach to playing the game.



Table 5.35: Adaptivity agent PPO configuration

Settings	Description
PPO hyperparameters	batch_size=256, buffer_size=102400, learning_rate = 0.0003, beta = 0.005, epsilon = 0.2, lambd = 0.95, num_epoch = 3, learning_rate_schedule = linear
Network settings	normalize = false, hidden_units = 512, num_layers = 4, vis_encode_type = simple
Memory	sequence_length = 10, memory_size = 32
Reward extrinsic	gamma = 0.99, strength = 1

However, these methods helped with the training of the agent. In future work, the creation of more distinct agents to perform different strategies, as demonstrated by the user, should be explored.

All the experiment results are essential for future exploration of this topic. The results can serve as a starting position to explore, improve and overcome the challenges faced when creating these player simulation agents.

In a future iteration, the results should also include other statistics about the performance of each agent during training and during a testing phase. These can include, for example, the number of times the agent completed each section and its movement metrics. This information should allow for a complete comprehension of the agent's performance and allow the creation of a better final agent.

## 5.2 Adaptivity system

The adaptivity system results consist of two parts: the adaptivity agent and adapted game user results. The adaptivity agent was also trained using the Unity ml-agents toolkit, and the results obtained are similar to those described in section 5.1. The user results are based on the user response from the questionnaire and the *csv* files collected from the gaming sessions. The following two sections will describe the results and analyze them.

### 5.2.1 Adaptivity agent results

The following subsections will present the results of the training session of the adaptivity agent. The agent was trained using the PPO and SAC algorithms available in the ml-agents toolkit. The results from these algorithms are displayed using the Tensorboard graphs, and each experiment was done sequentially with slight changes made to each one. Furthermore, the PPO and SAC algorithms configurations did not suffer significant changes over the multiple experiments. The following table 5.35 describes the configuration used for PPO, and the table 5.36 describes the configuration for SAC.

Table 5.36: Adaptivity agent SAC configuration

Settings	Description
Sac hyperparameters	batch_size=256, buffer_size=1000000, learning_rate = 0.0003, learning_rate_schedule = constant, buffer_init_steps = 0, tau = 0.005, steps_per_update = 10.0, save_replay_buffer = false, init_entcoef = 0.05, reward_signal_steps_per_update = 10.0
Network settings	normalize = false, hidden_units = 512, num_layers = 4, vis_encode_type = simple
Memory	sequence_length = 10, memory_size = 32
Reward extrinsic	gamma = 0.99, strength = 1
Training settings	Number of environments = 8, timescale for game = 20

### 5.2.1.1 Adaptivity agent experiment 1

This experiment is the baseline algorithm used for the adaptivity agent and was improved in the following experiments. From the figure 5.36, the following results can be observed:

- The agent trained for about 25M steps.
- The cumulative reward started to converge to a value.
- The policy loss declined and then remained relatively constant.
- The entropy also declined and remained relatively constant.
- The agent selects the easiest section the majority of the time. This is majorly due to the punishment for repeating sections not being high enough.

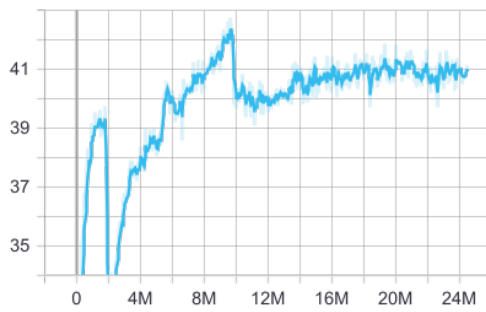
### 5.2.1.2 Adaptivity agent experiment 2

In this experiment, the SAC algorithm was tested for the adaptivity agent. From the figure 5.37, the following results can be observed:

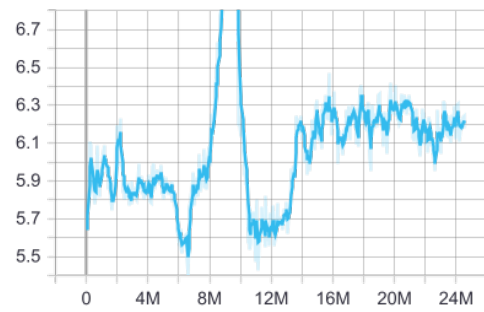
- The agent trained for about 11M steps.
- The cumulative reward and episode length was slightly unstable.
- The policy loss and the entropy also had unstable values.

Overall the use of SAC seems to give more inconsistent results than using the PPO algorithm. However, the SAC algorithm can still be explored and modified in future work to give better results.

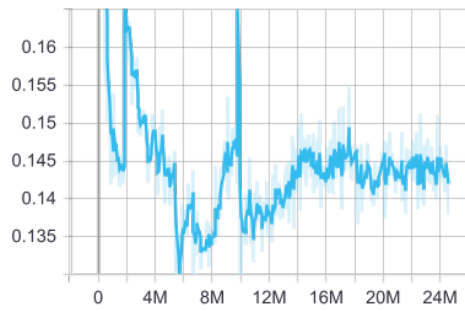
Figure 5.36: Adaptivity agent experiment 1 results



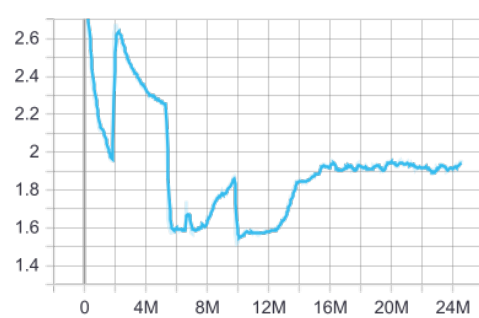
(a) Mean cumulative episode reward over the training steps



(b) Mean length of each episode over the training steps

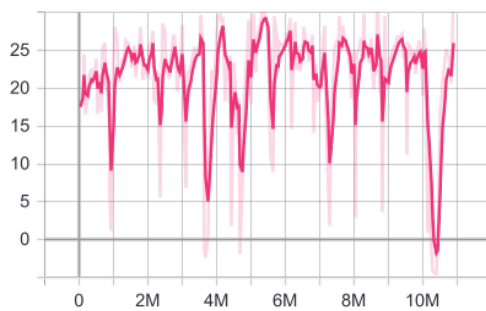


(c) Mean magnitude of the policy loss function over the training steps

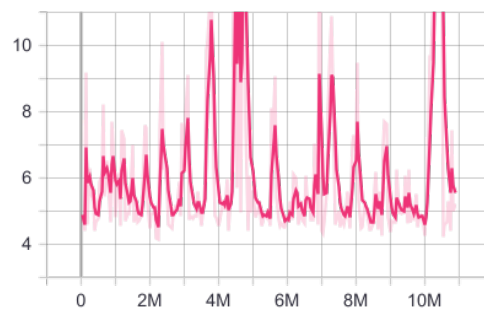


(d) Policy entropy over the training steps

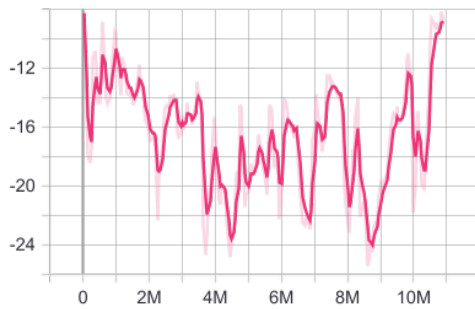
Figure 5.37: Adaptivity agent experiment 2 results



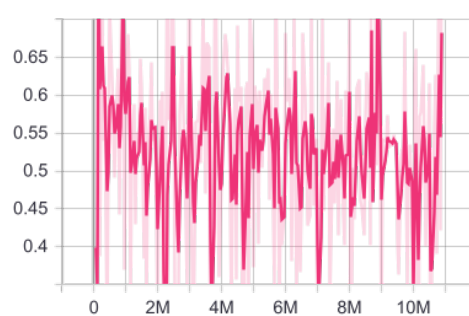
(a) Mean cumulative episode reward over the training steps



(b) Mean length of each episode over the training steps



(c) Mean magnitude of the policy loss function over the training steps



(d) Policy entropy over the training steps

### 5.2.1.3 Adaptivity agent experiment 3

In this experiment, the reward for repeating sections was increased to prevent the agent from selecting too many repeated sections, as seen in previous experiments. From the figure 5.38, the following results can be observed:

- The agent trained for about 13M steps.
- The cumulative reward started to converge to a value.
- The policy loss declined during the training.
- The entropy also declined during the training.
- The agent still sometimes shows repeated sections to the player.

### 5.2.1.4 Adaptivity agent experiment 4

In this experiment, to try to solve the repeated section's problem, the LSTM was increased. The sequence length was increased to 64 and the memory size to 128. From the figure 5.39, the following results can be observed:

- The agent trained for about 1.8M steps.
- The cumulative reward and the episode length declined during the training.
- The policy loss increased, and the entropy decreased during the training.

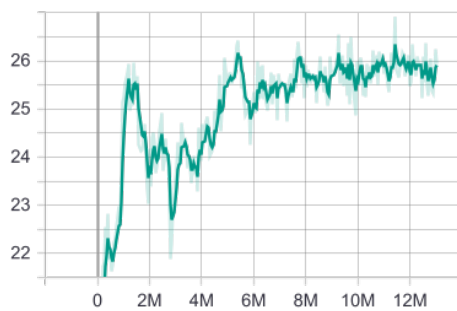
This experiment was not successful, and a smaller LSTM should be used, and the memory size should be similar to the size of the section tolerance of the personalities.

### 5.2.1.5 Adaptivity agent experiment 5

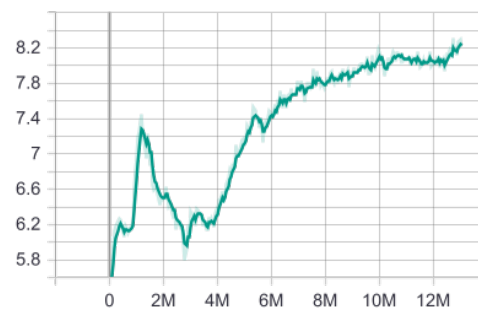
In this experiment, the reward was change to multiply the number of jumps and backtrack by the lives lost. This new reward should reflect better real players. From the figure 5.40, the following results can be observed:

- The agent trained for about 21M steps.
- The cumulative reward and the episode length started to converge to a value.
- The policy loss slightly declined, and then its value fluctuated around 1.82.
- The entropy declined over the training session.
- The agent has a good performance and selects sections depending on the personality playing.

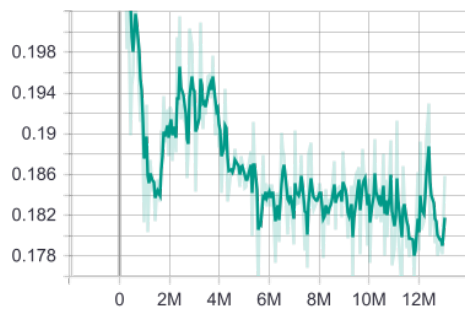
Figure 5.38: Adaptivity agent experiment 3 results



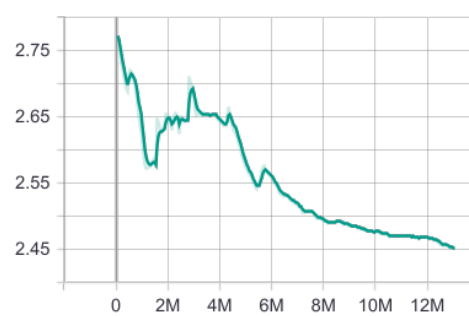
(a) Mean cumulative episode reward over the training steps



(b) Mean length of each episode over the training steps

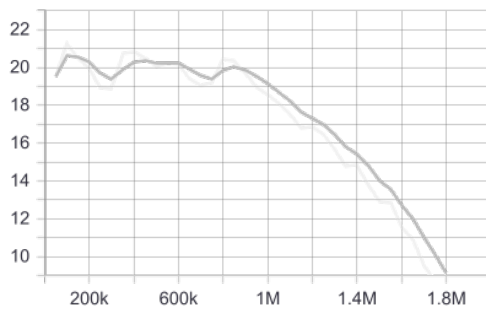


(c) Mean magnitude of the policy loss function over the training steps

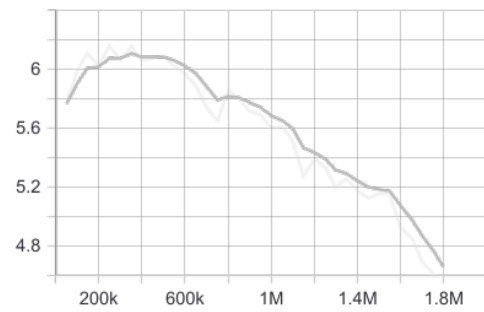


(d) Policy entropy over the training steps

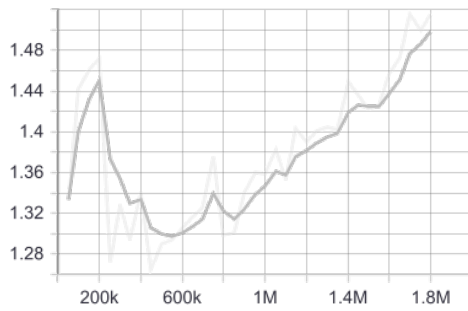
Figure 5.39: Adaptivity agent experiment 4 results



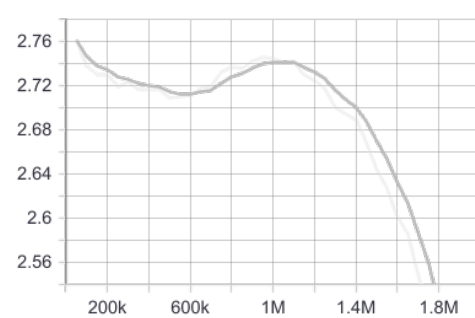
(a) Mean cumulative episode reward over the training steps



(b) Mean length of each episode over the training steps

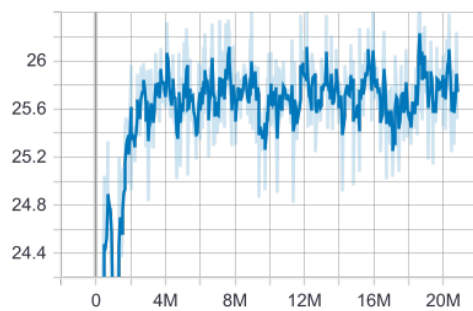


(c) Mean magnitude of the policy loss function over the training steps

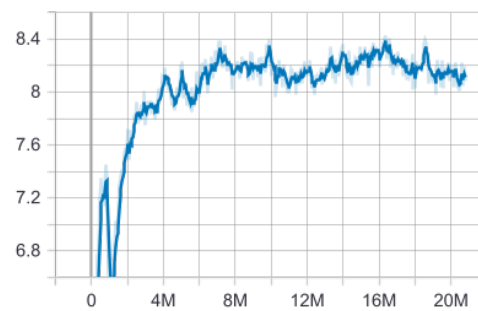


(d) Policy entropy over the training steps

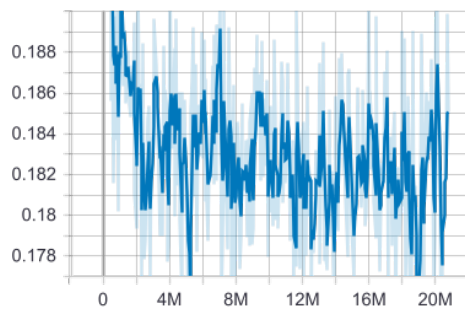
Figure 5.40: Adaptivity agent experiment 5 results



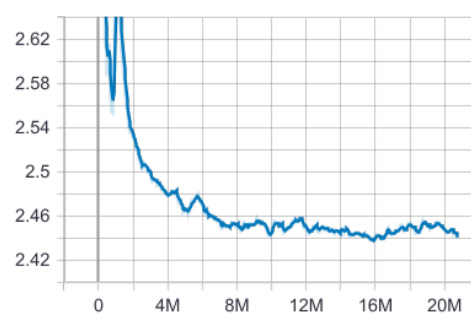
(a) Mean cumulative episode reward over the training steps



(b) Mean length of each episode over the training steps



(c) Mean magnitude of the policy loss function over the training steps



(d) Policy entropy over the training steps



### 5.2.2 Adaptivity agent overall results and analysis

The final agent used for the adapted game shows that he can choose the sections depending on the observations collected by the user playing the game. The following list summarises the results observed and their respective analysis:

- Modifying the reward values for each personality changes the overall result of the adaptivity agent. Therefore, for creating a successful agent, the reward function needs to represent each personality and their objectives almost perfectly, which is not a trivial task. For example, in the first experiments, the agent would only show section 3 since no personality would lose lives and go their preferred speed. In the following experiments, the rewards were modified, and the punishment for repeating sections was also increased to solve this problem.
- Although the final agent shows that he selects the sections based on the personality playing, the agent can still be improved since the policy loss did not regularly decrease during training (it stayed around 1.82). To improve the algorithm, the hyperparameters can be tweaked, and the network architecture can also be modified and tested to enhance the adaptivity agent's performance.
- The final agent still shows repeated sections sometimes to the players. Additionally, the agent also shows the same sections to the same players at the start of each game, creating a repeated experience between games.

### 5.2.3 Adapted game results

After developing the adaptivity agent, the adaptivity system was incorporated into the game and tested with real users. After the user played the round of the game, they responded to a questionnaire consisting of questions from the Game Experience Questionnaire (GEQ) [31] and other relevant questions (the complete questionnaire is present in the appendix B). The questions used from the GEQ are from the core module and are the following:

- 1 - I felt content
- 2 - I felt skilful
- 3 - I was interested in the game's story
- 4 - I thought it was fun
- 5 - I was fully occupied with the game
- 6 - I felt happy
- 7 - It gave me a bad mood
- 8 - I thought about other things

- 9 - I found it tiresome
- 10 - I felt competent
- 11 - I thought it was hard
- 12 - It was aesthetically pleasing
- 13 - I forgot everything around me
- 14 - I felt good
- 15 - I was good at it
- 16 - I felt bored
- 17 - I felt successful
- 18 - I felt imaginative
- 19 - I felt that I could explore things
- 20 - I enjoyed it
- 21 - I was fast at reaching the game's targets
- 22 - I felt annoyed
- 23 - I felt pressured
- 24 - I felt irritable
- 25 - I lost track of time
- 26 - I felt challenged
- 27 - I found it impressive
- 28 - I was deeply concentrated in the game
- 29 - I felt frustrated
- 30 - It felt like a rich experience
- 31 - I lost connection with the outside world
- 32 - I felt time pressure
- 33 - I had to put a lot of effort into it

Each of these questions can be answered on a scale from 1 to 5 (in the GEQ, they are from 0 to 4, but it is the same as using 1 to 5), from “not at all” to “extremely”. Furthermore, these questions can be associated with a category (from the GEQ) and scored by the average response of each question in that division. The categories include:

- **Competence**, with questions 2, 10, 15, 17 and 21.
- **Sensory and Imaginative Immersion** with question 3, 12, 18, 19, 27, 30.
- **Flow** with questions 5, 13, 25, 28 and 31.
- **Tension/Annoyance** with questions 22, 24 and 29.
- **Challenge** with questions 11, 23, 26, 32 and 33.
- **Negative affect** with questions 7, 8, 9 and 16.
- **Positive affect** with question 1, 4, 6, 14 and 20.

Additionally, the questionnaire also requires the user to submit their csv files containing gameplay information.

In total, 16 players answered the questionnaire and played 22 adapted games and 21 non-adapted games. Thus, there were 174 adapted and 107 non-adapted sections completed by the users. From the gameplay session information and the questionnaire, the following results and analysis were made from the comparison of the adapted versus non-adapted version of the game:

- The player completes more sections on the adapted game from the total sections played.
- From figure 5.41, the only sections shown in the adapted game to the players are 1, 2, 4, 6, 11 and 12. These sections shown are dependent on the reward function, and the personality values created. Also, the sections are shown depending on the user playing the game, and if more players with a large variety of strategies played the game, all sections would probably be shown at least one time (the adaptivity agent shows all sections at least one time with the personalities playing).
- In the adapted game, the players made fewer backtracks (figures 5.42a and 5.42b). This can mean that the players do fewer backtracks than expected, or the sections played do not require going back (from the figure 5.41, sections 9 and 14 are the most feasible of a backtrack to happen).
- From figure 5.43, the players spent less time playing each section (figure 5.44), which is also caused by the sections selected by the agent being faster to complete. Also, the players perform fewer jumps (figure 5.45). Additionally, the players’ average velocity increased (figure 5.46), caused by the more accessible selection of sections from the agent.

### Adapted vs non-adapted sections shown

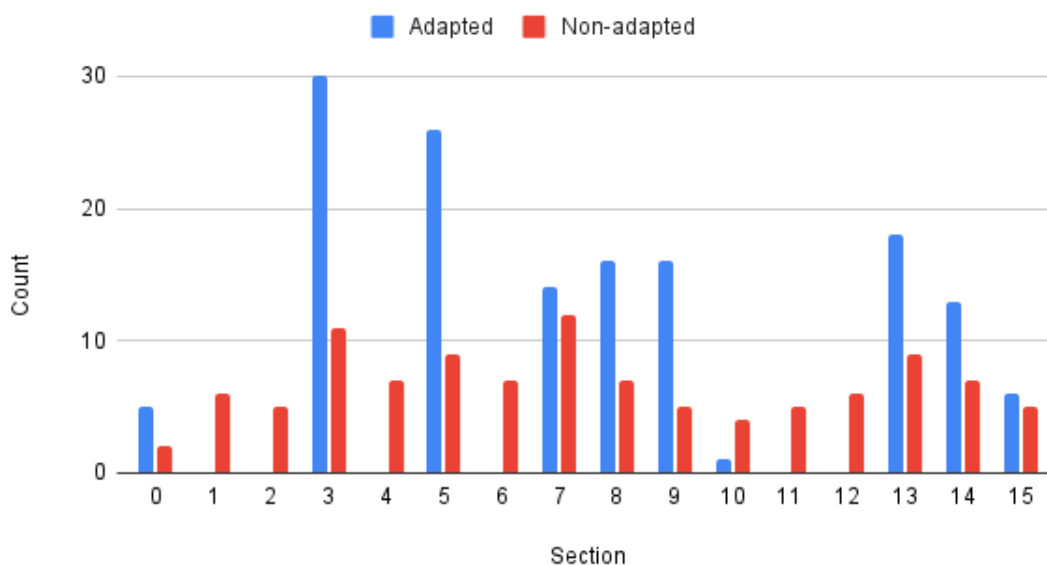


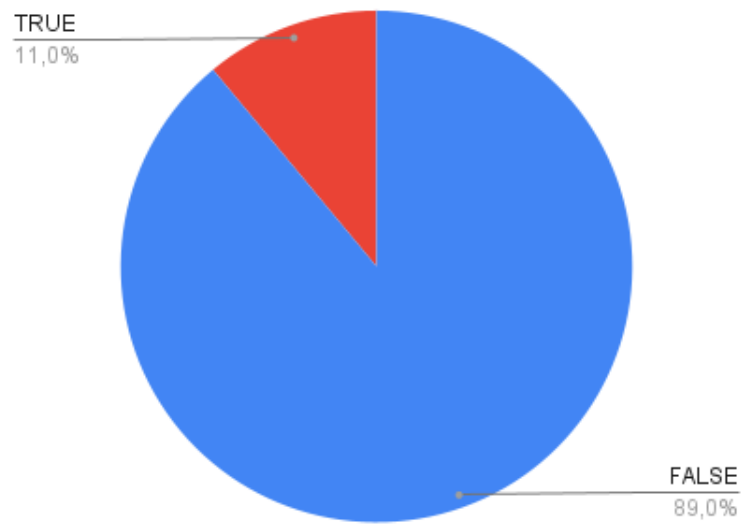
Figure 5.41: Adapted vs non-adapted section shown

- In general (figure 5.43), the players lose fewer lives in the game. Additionally, the players collected about the same amount of coins. This is caused because not all sections contain hidden or optional coins to collect, and most coins are required to complete the section.
- From figure 5.47, at the end of each game, the players collect more coins (figure 5.48), spend slightly more time playing the game (figure 5.49), have a better end score (figure 5.50) and have constant higher average speeds (figure 5.51). This is also caused because the difficulty of gameplay is lower in the adapted game.
- From figure 5.52, 62,5% of players realised what round the adaptivity was implemented.
- From the responses to the questionnaire, each answer component yielded almost the same results (figure 5.53), except for the negative affect (figure 5.54), which gave the players a slightly more negative experience, and the positive affect (figure 5.55), which gave the players a slightly less positive experience. These categories are associated with the gaming experience and affected by the game difficulty, which was lower and caused some boredom.
- Although the categories did not yield the most desirable result, the player felt that the game was more effortless in the specific question results (figure 5.56). However, the user also responded that the adapted game was more interesting and provided a richer experience.

Additionally, from observing the section shown to the player, one player had section 9 repeated three times consecutively in the first game. For instance, section 9 is usually shown first by the agent to new players until he understands more about the user's strategy.

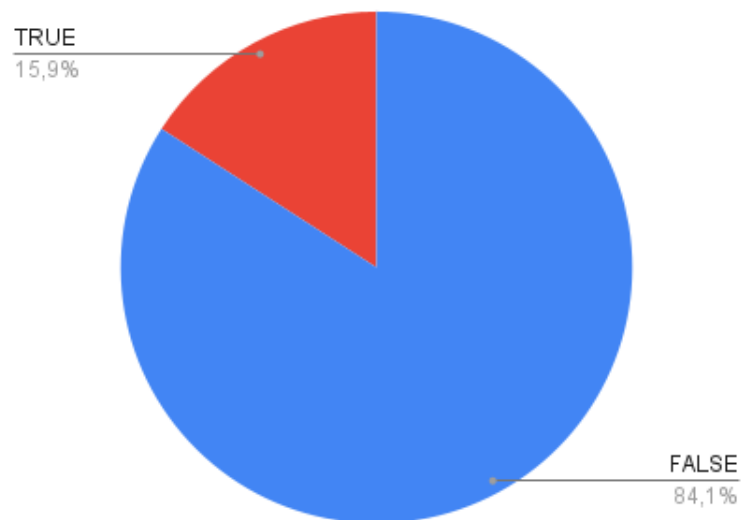
Figure 5.42: Adaptivity agent experiment 5 results

### Number of backtracks in adapted version in 174 sections



(a) Adapted version number of backtracks

### Number of backtracks in non-adapted version in 107 sections



(b) Non-adapted version number of backtracks

### Difference between non-adapted and adapted version for each section

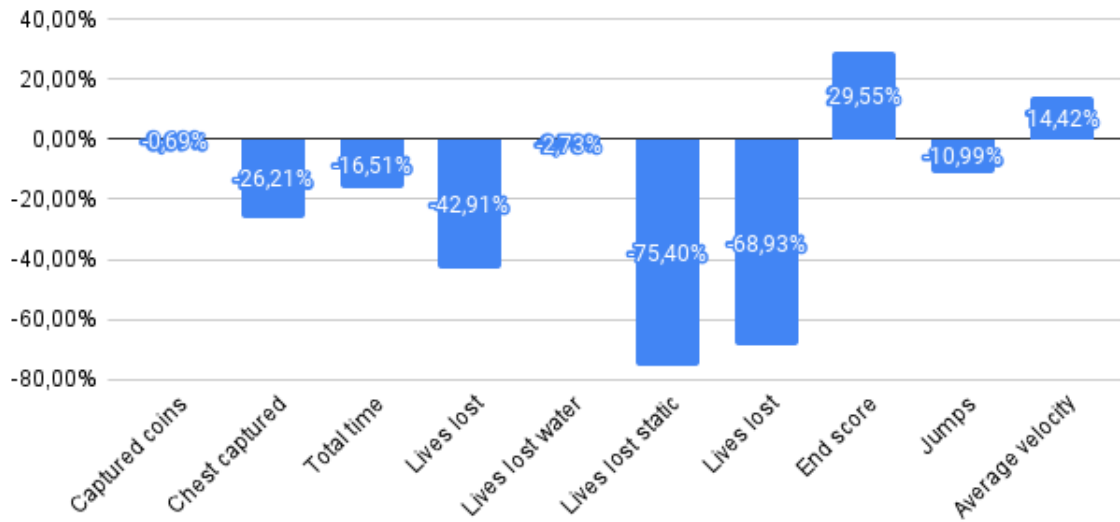


Figure 5.43: Difference between non-adapted and adapted version for each section

### Total time spent in each section

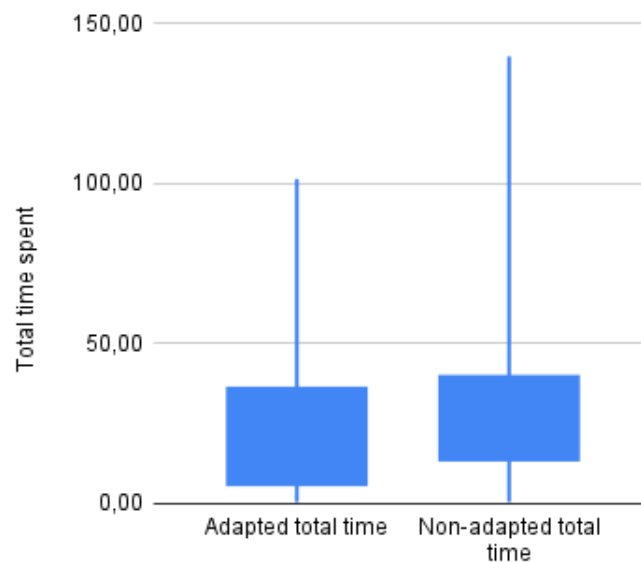


Figure 5.44: Total time spent in each section

Finally, some users also wrote an additional note. These notes included that the game was too easy and sometimes caused boredom. However, it was also stated that the game offered a more exciting experience than the standard gameplay. Furthermore, one user wrote that the game could use a leaderboard and more animations to open chests related to competitive and visual feedback

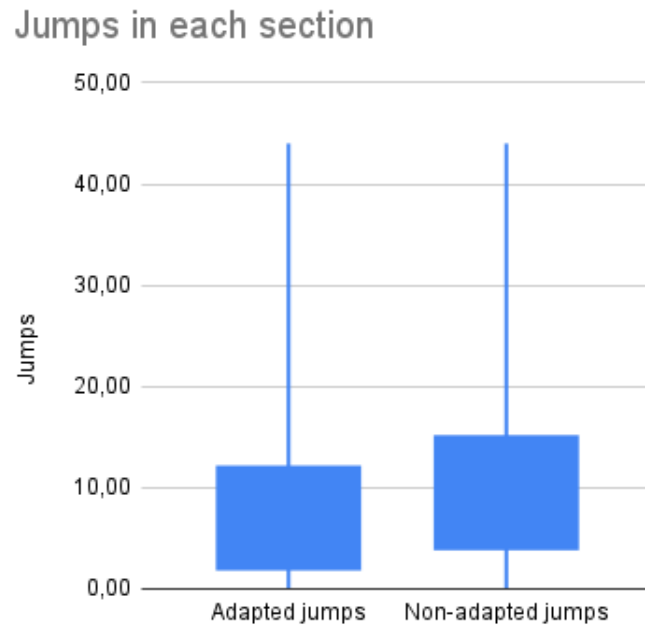


Figure 5.45: Jumps in each section

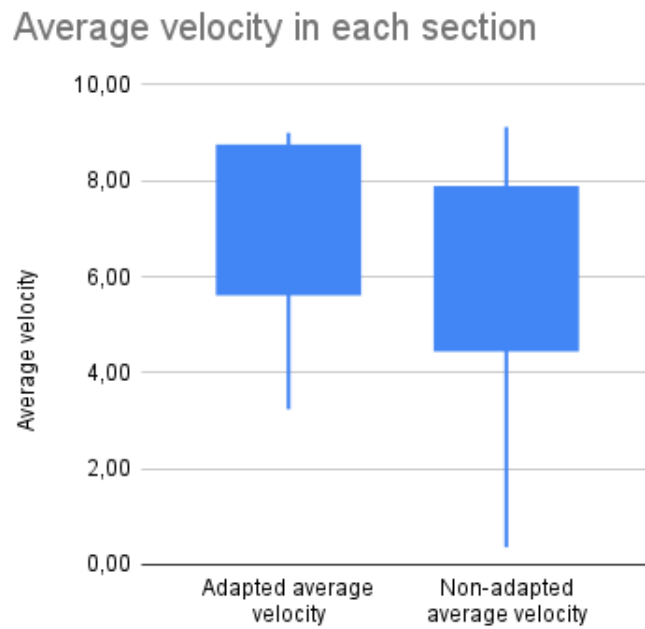


Figure 5.46: Average velocity in each section

but are related to the game design.

Difference between non-adapted and adapted version for each game

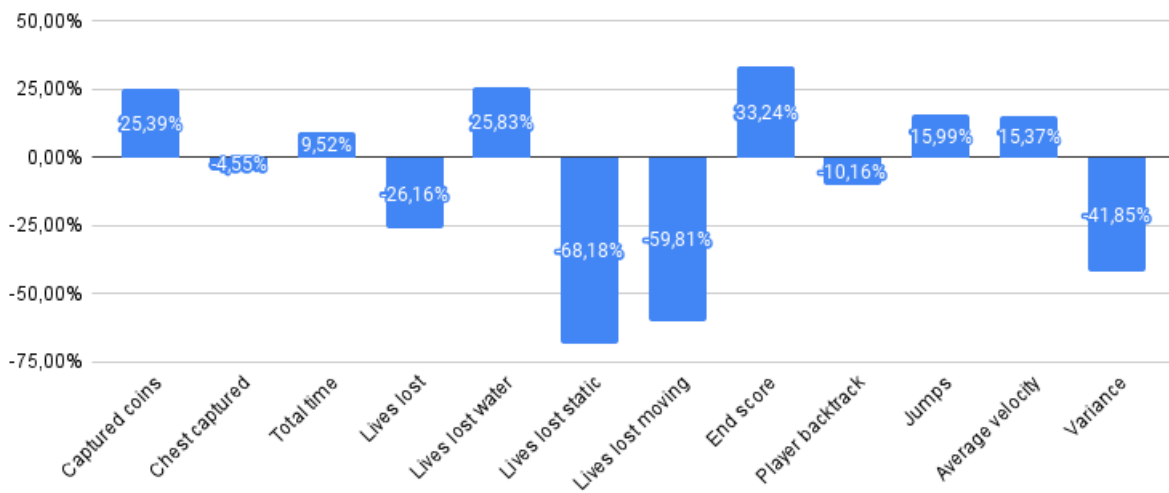


Figure 5.47: Difference between non-adapted and adapted version for each game

Captured coins at the end of a game

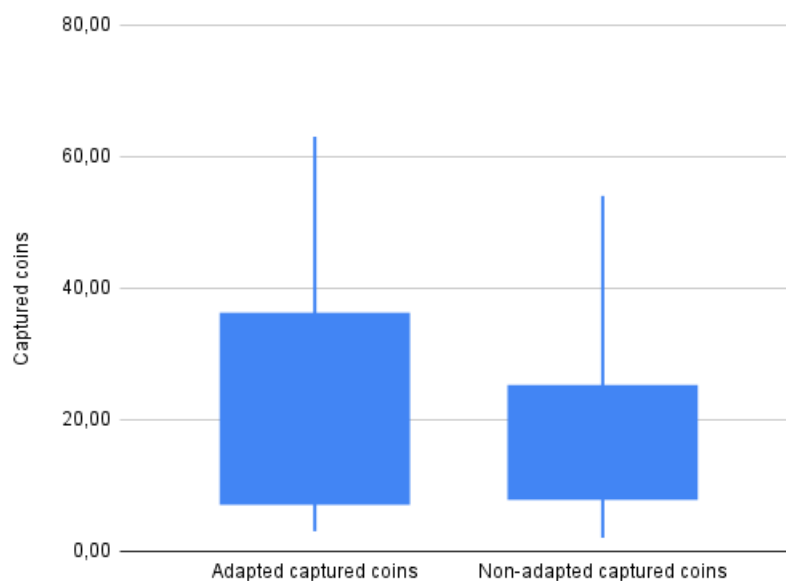


Figure 5.48: Captured coins at the end of a game

### 5.3 Summary

In this chapter, several results obtained from all the systems created were presented. Firstly, the player simulation experiments were presented in section 5.1. Although the overall system could not be used in the final adapted game, the experiments performed showed that this type of player simulation has much potential, and with some more experiments and tweaks to the agent and the



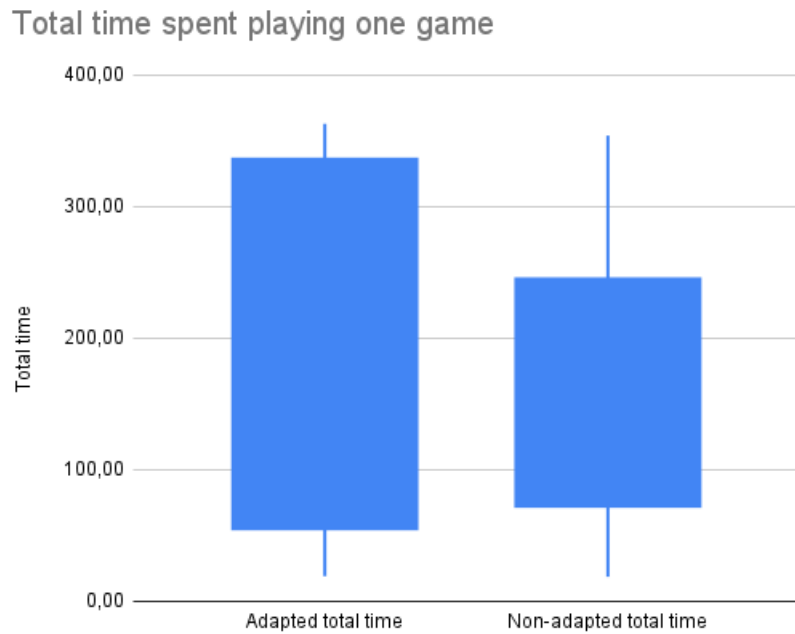


Figure 5.49: Total time spent playing one game

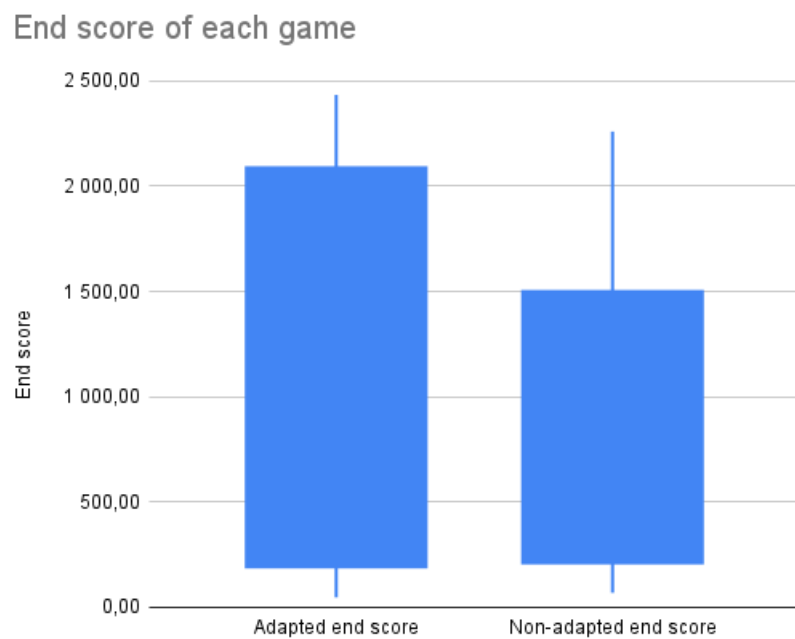


Figure 5.50: End score of each game

algorithms, the system can be improved. Then, in section 5.2.1, the adaptivity agent training results were displayed, and their analysis was discussed. Additionally, it also includes some proposed changes to the algorithm and the personality rewards, which could improve to create a more robust

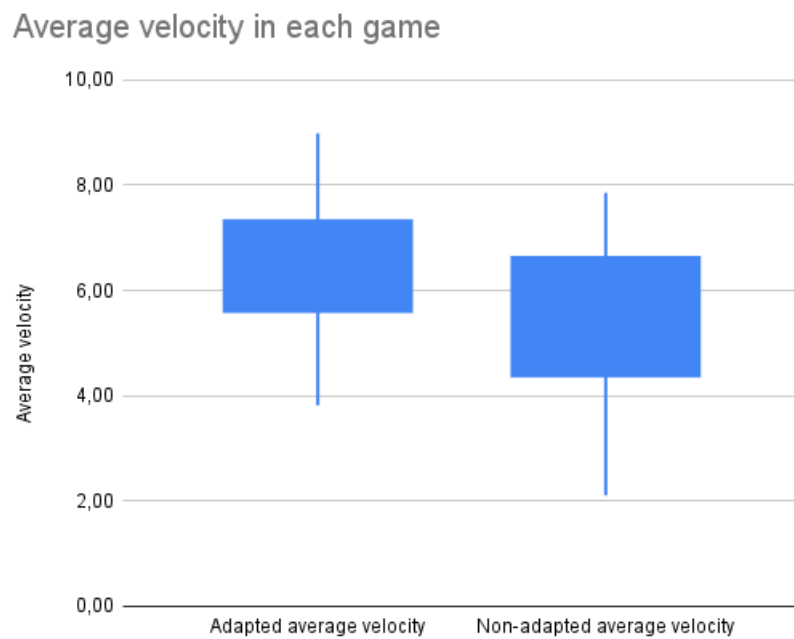


Figure 5.51: Average velocity in each game

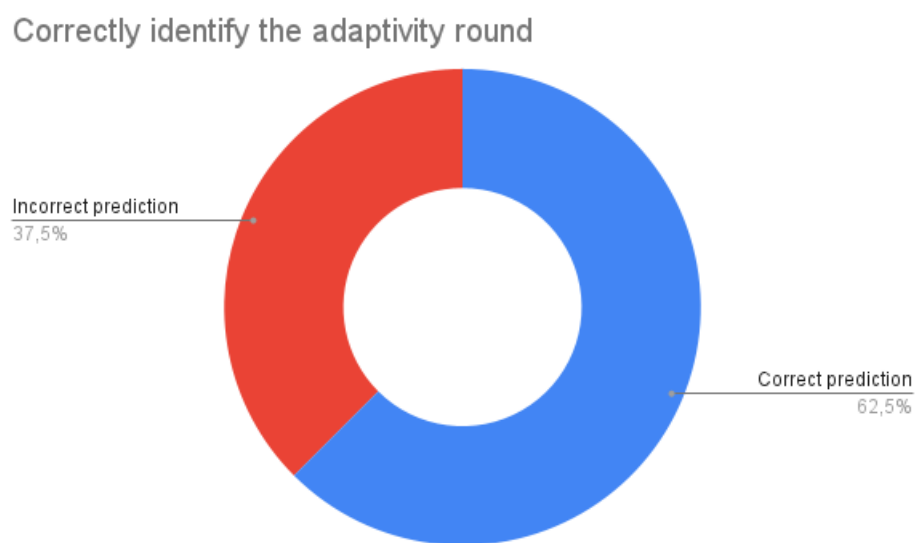


Figure 5.52: Correctly identify the adaptivity round

adaptivity agent. Finally, the results from users playing the adapted and non-adapted game were discussed and analysed in section 5.2.3. The overall adapted game created a more accessible and unique experience. However, the users also found it more boring, which means that the reward from each personality should be tweaked to more accurately describe the real users.

Average GEQ Core module component scores

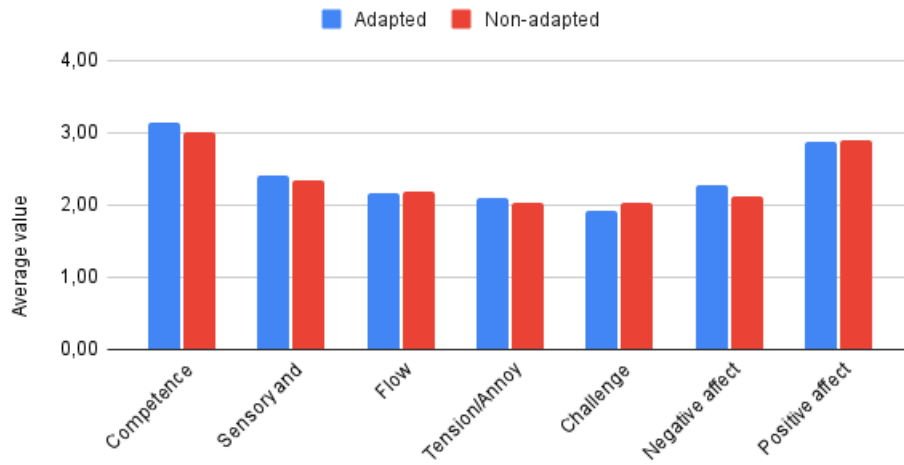


Figure 5.53: Average GEQ Core module component scores

Adapted vs non-adapted negative affect

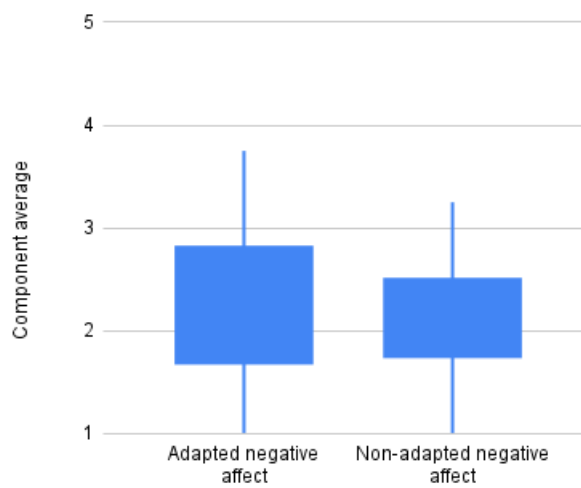


Figure 5.54: Adapted vs non-adapted negative affect

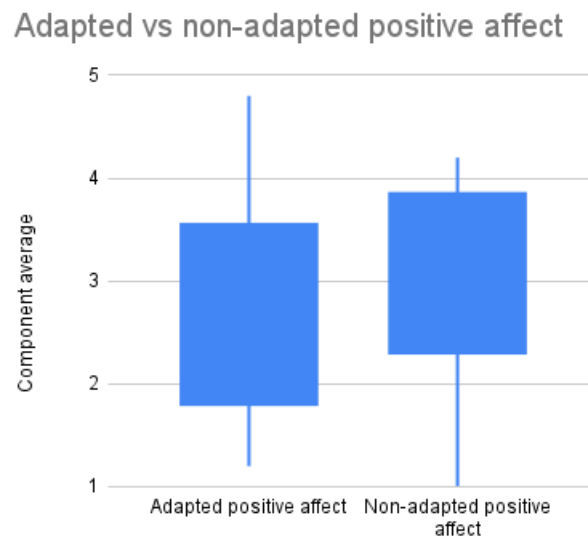


Figure 5.55: Adapted vs non-adapted positive affect

Average questionnaire response

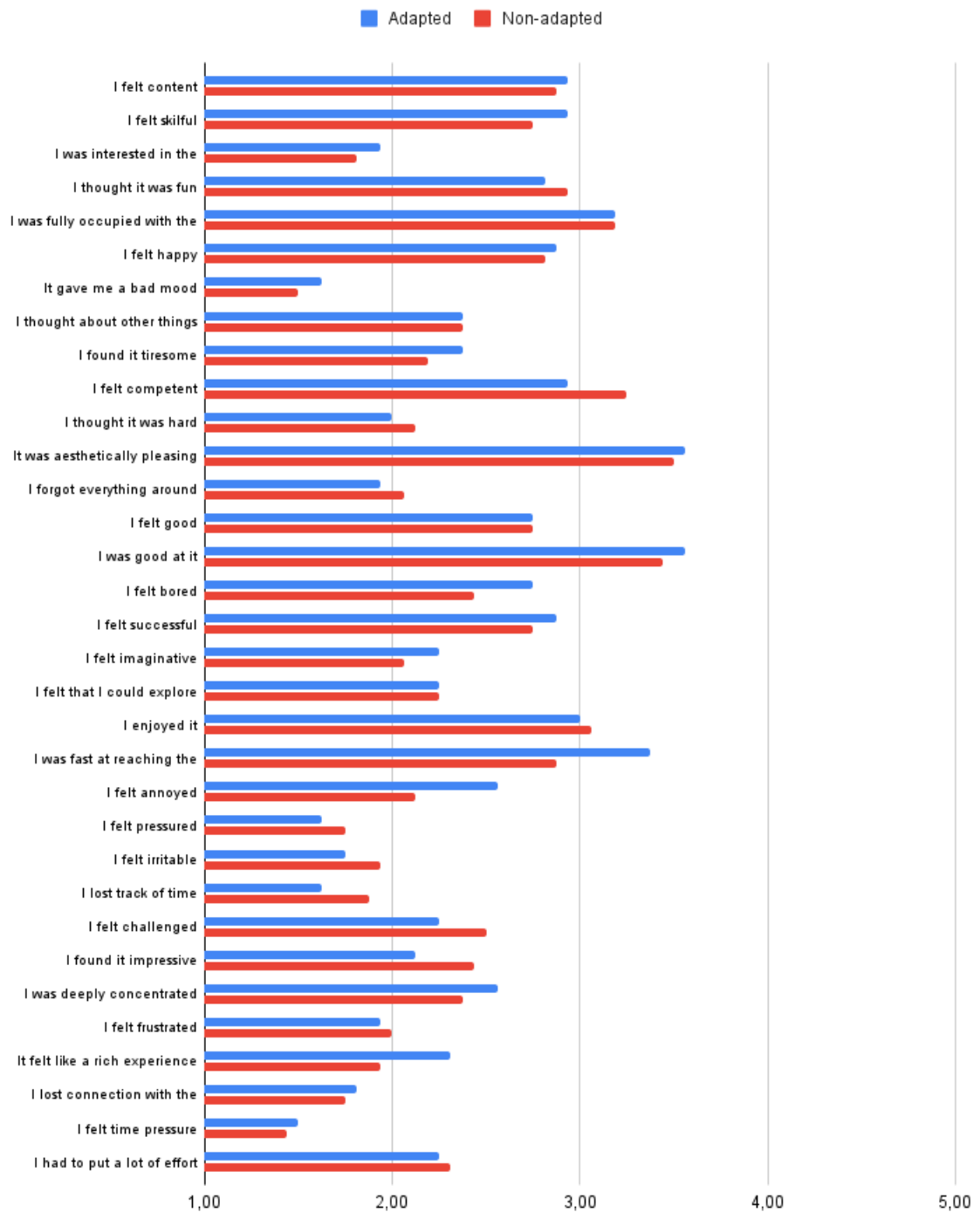


Figure 5.56: Average questionnaire response



## Chapter 6

# Conclusions

The initial idea for the adaptivity system included only the adaptivity agent training. However, a reinforcement learning algorithm needs to be trained with a large number of steps. In this case, real users would play the game, and the adaptivity system would train over it. This situation would only be possible if there were many users for the experiment and were available to test and give their feedback at the end of each section about their gaming experience. This circumstance would not be ideal for providing a realistic gaming experience since the user would have to suspend the gameplay to respond to questions. Additionally, in the context of this dissertation, there would not be a sufficient number of users to play the game to train the algorithm. This dissertation opted to implement a player simulation to generate enough data to train the adaptivity agent to solve this problem.

Although the player simulation did not reveal a success at replicating the users' behaviours, it still got good results worth exploring in future iterations. From the training of the player simulation agent with demonstrations (from myself), the agent could complete several sections and even replicate some demonstrated movements. On the other hand, on some sections, the agent would have trouble completing them. This was caused because: some sections were more extensive than others, and the agent would lose before he could get to the end of the section; the agent sometimes did not perform a jump over the stairs; the observations may be too complex; and the possible lack of complete demonstrations for each section. The GAIL and the behavioral cloning algorithms can complete this task if more testing and modifications are made to improve the simulated agent. Additionally, other results should also be collected to better understand the algorithm's performance, for example, which sections were complete, their amount, and movement statistics.

To advance with the implementation of the adaptivity system, the game and section results still need to be generated. For this, the personality system was implemented to represent the possible player that would play the game. Each personality was created based on my personal experience of the game and my perception of difficulty. The creation of the personalities was made this way since there was not enough time to collect enough user information to create clusters of personalities using unsupervised reinforcement learning. It is usual for game developers to create a difficulty level by their own perception of the game and then iterate to improve the gaming experience. In

the same way, after collecting user data on the adapted game, the problem with the personality system became apparent, and improvements to each personality could be made to represent the real users better.

The adaptivity agent was successfully created to alter the game sections to maximize the reward of each personality. Although, it could also be improved since the results showed that sections are still shown repeatedly. Additionally, the agent also seems to repeat the initial section for the same players if they show the same strategy. This would not be ideal for an optimal gaming experience and could be improved in future iterations. Also, the adaptivity agent focuses on several sections more than others, which can mean that the rewards need to be changed, or those sections had design problems that can be improved. This means that the agent can also be used to assist with the level design of the game.

From the user responses and their gameplay data, the adaptivity agent impacted the gaming experience. However, it did not improve the overall gaming experience because it became too easy and more boring. From the specific questions of the questionnaire, the users also reported that the game had a different feeling and provided a more interesting experience. Although the overall adapted game did not yield the expected improvement in the gaming experience, it could adapt its content and provide the user with a different experience, confirming that gaming adaptivity impacts the gaming experience. Furthermore, the adapted game can also be used for different objectives by modifying the reward function of the adaptivity algorithm.

## 6.1 Limitations

The time available for completing this dissertation limited and simplified some systems that could be further explored, for example, the player simulation system. Additionally, the number of users who responded to the questionnaire limited some of the analysis made about the adaptivity agent. Finally, the lack of experience with reinforcement learning algorithms limited its usage for this specific problem. Furthermore, the analysis of the results and the iteration of experiences is also constrained by the lack of advanced knowledge of these algorithms.

## 6.2 Future work

The several implemented system can still be investigated and modified to improve their performance. For the player simulation, the following list defines the proposed future modification that can be implemented:

- Using only images for the observations was not tested thoroughly and could have better performance. Furthermore, to remove more clutter in the image, the coins, water and enemies animations should also be removed.
- The problem of the jumps over pits and stairs should be further investigated since it is the main obstacle for creating the player simulation.



- The Unity ml-agents toolkit provides complete customization for the reinforcement learning algorithms, which were not fully explored in this dissertation and can improve the simulation performance.

For the adaptivity agents and personality system, the following list describes the suggested improvement and changes that can be made:

- Each personality can be created using clustering techniques on real players' gameplay data to represent better the users who will play the game. If the already created personalities are explored, their rewards should be changed to provide each personality with a correct challenge for their skill.
- For the observations of the adaptivity agent, past information about the time spent and the number of backtracks was not used in the agent's training and should be explored in future work.
- The sections already created can also be changed to all be the same size. Additionally, other section can also be created to represent different challenges and objectives for the players.
- Instead of deciding the next section presented, the agent can procedurally create the level for the player, presenting them with different amounts of coins, enemies, and jumps. Additionally, other game elements can be modified to provide a different gaming experience, as described in section [art game elements].

These suggestions can be explored in future iterations to create a better version of the adaptivity system.



## **Appendix A**

# **Personalities**

Table A.1: Personality 1 - Experienced player that wants to collect coins

Setting	Value
Tolerance for repeating sections	3
New level importance	30
Coins importance	0.75
Coins section	0-1->1, 1-3->3, 2-4->4, 3-1->1, 4-6->6, 5-3->3, 6-11->13, 7-5->5, 8-9->9, 9-12->14, 10-5->5, 11-9->9, 12-4->4, 13-5->5, 14-4->6, 15-0->0
Chest importance	0.75
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-1->1, 15-0->0
Life importance	0.5
Life lost importance	3
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->0, 7-0->0, 8-0->0, 9-0->0, 10-0->1, 11-0->1, 12-0->1, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->1;m->0, 5-w->1;s->1;m->0, 6-w->1;s->0;m->1, 7-w->0;s->1;m->0, 8-w->1;s->1;m->0, 9-w->1;s->1;m->0, 10-w->0;s->0;m->1, 11-w->0;s->0;m->1, 12-w->0;s->1;m->0, 13-w->1;s->0;m->1, 14-w->1;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-2->2.3, 1-16->16.2, 2-6->6.2, 3-4.6->4.8, 4-28->28.5, 5-13.5->13.7, 6-60->60.5, 7-21.5->21.7, 8-38.5->38.7, 9-47->47.5, 10-24->24.2, 11-62->62.5, 12-27->27.2, 13-33->33.2, 14-35->35.5, 15-12->12.2
Speed importance	1
Speed preference	9
Speed section	0-8.2->8.5, 1-4.2->4.5, 2-5.5->5.8, 3-8.2->8.5, 4-3.5->3.8, 5-7.6->7.9, 6-2.7->3, 7-8.6->8.9, 8-7->7.3, 9-6.2->6.5, 10-7->7.3, 11-3.9->4.2, 12-8.1->8.4, 13-8.6->8.9, 14-3.9->4.2, 15-8.6->8.9
Jumps section	0-1->1, 1-4->5, 2-4->4, 3-1->1, 4-9->10, 5-7->7, 6-22->23, 7-5->5, 8-18->19, 9-13->14, 10-10->11, 11-25->26, 12-8->9, 13-4->4, 14-14->15, 15-0->0
Backtrack probability	0-0%, 1-100%, 2-0%, 3-0%, 4-100%, 5-0%, 6-100%, 7-0%, 8-0%, 9-100%, 10-0%, 11-100%, 12-5%, 13-0%, 14-100%, 15-0%
New score importance	2
Concentration level preferred	8.5
Concentration importance	7
Concentration	0-2, 1-1, 2-4, 3-0, 4-7, 5-4, 6-7, 7-1, 8-10, 9-5, 10-6, 11-9, 12-1, 13-0, 14-7, 15-0
Skill level preferred	8.5
Skill importance	6
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	7.5
Challenge importance	10
Challenge	0-1, 1-0, 2-2, 3-0, 4-7, 5-1, 6-5, 7-0, 8-4, 9-3, 10-9, 11-9, 12-6, 13-0, 14-3, 15-0

Table A.2: Personality 2 - Experienced player that just wants to go fast

Setting	Value
Tolerance for repeating sections	4
New level importance	30
Coins importance	0.1
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->2, 5-3->3, 6-9->11, 7-5->5, 8-9->9, 9-8->8, 10-5->5, 11-5->6, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.1
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-1->1, 15-0->0
Life importance	0.5
Life lost importance	5
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->0, 7-0->0, 8-0->0, 9-0->0, 10-0->1, 11-0->1, 12-0->1, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->1;m->0, 5-w->1;s->1;m->0, 6-w->1;s->0;m->1, 7-w->0;s->1;m->0, 8-w->1;s->1;m->0, 9-w->1;s->1;m->0, 10-w->0;s->0;m->1, 11-w->0;s->0;m->1, 12-w->0;s->1;m->0, 13-w->1;s->0;m->1, 14-w->1;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-2->2.3, 1-7->7.2, 2-6->6.2, 3-4.6->4.8, 4-13->13.3, 5-13.5->13.7, 6-29->29.5, 7-21.5->21.7, 8-38.5->38.7, 9-38.5->39, 10-24->24.2, 11-41.5->42, 12-27->27.2, 13-33->33.2, 14-18.5->18.8, 15-12->12.2
Speed importance	5
Speed preference	8
Speed section	0-8.2->8.5, 1-8.5->8.8, 2-5.5->5.8, 3-8.2->8.5, 4-8.4->8.7, 5-7.6->7.9, 6-6.1->6.4, 7-8.6->8.9, 8-7->7.3, 9-7.7->8, 10-7->7.3, 11-6->6.3, 12-8.1->8.4, 13-8.6->8.9, 14-7.9->8.2, 15-8.6->8.9
Jumps section	0-1->1, 1-1->1, 2-4->4, 3-1->1, 4-5->7, 5-7->8, 6-12->14, 7-5->6, 8-18->19, 9-13->14, 10-10->11, 11-17->18, 12-8->9, 13-4->4, 14-14->15, 15-0->0
Backtrack probability	0-0%, 1-0%, 2-0%, 3-0%, 4-0%, 5-0%, 6-0%, 7-0%, 8-0%, 9-0%, 10-0%, 11-0%, 12-1%, 13-0%, 14-0%, 15-0%
New score importance	10
Concentration level preferred	1
Concentration importance	7
Concentration	0-3, 1-1, 2-5, 3-0, 4-2, 5-5, 6-8, 7-0, 8-9, 9-3, 10-8, 11-10, 12-2, 13-0, 14-5, 15-0
Skill level preferred	8.5
Skill importance	6
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	8.5
Challenge importance	5
Challenge	0-1, 1-0, 2-3, 3-0, 4-2, 5-2, 6-8, 7-0, 8-10, 9-4, 10-10, 11-9, 12-4, 13-0, 14-1, 15-0

Table A.3: Personality 3 - Casual player that wants to collect coins

Setting	Value
Tolerance for repeating sections	3
New level importance	24
Coins importance	0.75
Coins section	0-1->1, 1-3->3, 2-4->4, 3-1->1, 4-4->5, 5-3->3, 6-9->13, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->8, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.75
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life importance	0.5
Life lost importance	3
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->1, 6-0->1, 7-0->0, 8-0->1, 9-0->0, 10-0->2, 11-0->2, 12-0->2, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->0;m->0, 5-w->1;s->0;m->0, 6-w->1;s->0;m->3, 7-w->0;s->1;m->0, 8-w->1;s->0;m->0, 9-w->1;s->0;m->0, 10-w->1;s->0;m->3, 11-w->1;s->0;m->10, 12-w->1;s->3;m->3, 13-w->0;s->0;m->1, 14-w->0;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-3.5->4.5, 1-17->19, 2-8->9, 3-5->6, 4-29->32, 5-17->19, 6-50->60, 7-22.5->24, 8-40->42, 9-49->52, 10-28->31, 11-64->68, 12-30->33, 13-35->37, 14-37->40, 15-20->23
Speed importance	1
Speed preference	6.5
Speed section	0-4.3->4.8, 1-4.1->4.4, 2-4.2->4.5, 3-7.7->8, 4-3.4->3.7, 5-6.5->6.8, 6-3.6->4.7, 7-8->8.5, 8-6.3->6.7, 9-5.9->6.4, 10-5.5->6, 11-3.8->4.1, 12-7.6->8, 13-8.4->8.7, 14-3.8->4.1, 15-4.1->4.4
Jumps section	0-1->2, 1-4->5, 2-5->6, 3-1->2, 4-6->8, 5-8->9, 6-16->22, 7-5->6, 8-20->22, 9-13->15, 10-12->14, 11-25->27, 12-8->11, 13-4->5, 14-14->16, 15-0->0
Backtrack probability	0-0%, 1-100%, 2-0%, 3-0%, 4-10%, 5-0%, 6-40%, 7-0%, 8-0%, 9-100%, 10-0%, 11-100%, 12-60%, 13-0%, 14-60%, 15-0%
New score importance	2
Concentration level preferred	6
Concentration importance	7
Concentration	0-2, 1-1, 2-4, 3-0, 4-7, 5-4, 6-7, 7-1, 8-10, 9-5, 10-6, 11-7, 12-1, 13-0, 14-7, 15-0
Skill level preferred	6.5
Skill importance	5
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	5
Challenge importance	7
Challenge	0-2, 1-3, 2-4, 3-0, 4-5, 5-4, 6-8, 7-2, 8-10, 9-8, 10-10, 11-10, 12-6, 13-0, 14-0, 15-0

Table A.4: Personality 4 - Casual player that wants to go fast

Setting	Value
Tolerance for repeating sections	4
New level importance	26
Coins importance	0.1
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-8->10, 7-5->5, 8-9->9, 9-8->8, 10-5->5, 11-6->6, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.1
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-0->0, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life importance	0.5
Life lost importance	5
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->1, 6-0->1, 7-0->0, 8-0->1, 9-0->0, 10-0->2, 11-0->2, 12-0->2, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->0;m->0, 5-w->1;s->0;m->0, 6-w->1;s->0;m->3, 7-w->0;s->1;m->0, 8-w->1;s->0;m->0, 9-w->1;s->0;m->0, 10-w->1;s->0;m->3, 11-w->1;s->0;m->10, 12-w->1;s->3;m->3, 13-w->0;s->0;m->1, 14-w->0;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-3.5->4.5, 1-9->10, 2-8->9, 3-5->6, 4-17->18, 5-17->18, 6-36->38, 7-22.5->23.5, 8-40->41, 9-42->43, 10-28->29, 11-49->50, 12-30->31, 13-35->36, 14-23.5->24.5, 15-20->21
Speed importance	4
Speed preference	6.8
Speed section	0-4->4.4, 1-6.6->7, 2-4.1->4.5, 3-7.6->8, 4-6.1->6.5, 5-6.4->6.8, 6-5.1->5.5, 7-8.1->8.5, 8-6.3->6.7, 9-7->7.4, 10-5.6->6, 11-4.8->5.2, 12-7.6->8, 13-8.3->8.7, 14-6.3->6.7, 15-8.3->8.7
Jumps section	0-2->3, 1-4->5, 2-5->6, 3-1->2, 4-5->6, 5-8->9, 6-12->14, 7-5->6, 8-20->22, 9-13->15, 10-12->13, 11-20->22, 12-8->11, 13-4->5, 14-6->7, 15-0->0
Backtrack probability	0-0%, 1-0%, 2-0%, 3-0%, 4-0%, 5-0%, 6-0%, 7-0%, 8-0%, 9-0%, 10-0%, 11-0%, 12-5%, 13-0%, 14-0%, 15-0%
New score importance	10
Concentration level preferred	3
Concentration importance	7
Concentration	0-3, 1-1, 2-4, 3-0, 4-2, 5-6, 6-9, 7-0, 8-10, 9-5, 10-10, 11-10, 12-3, 13-0, 14-5, 15-7
Skill level preferred	6.5
Skill importance	5
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	6
Challenge importance	5
Challenge	0-3, 1-1, 2-5, 3-0, 4-2, 5-3, 6-8, 7-1, 8-10, 9-8, 10-10, 11-10, 12-3, 13-0, 14-0, 15-4

Table A.5: Personality 5 - Casual player that plays with care

Setting	Value
Tolerance for repeating sections	5
New level importance	26
Coins importance	0.1
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.1
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life importance	1
Life lost importance	10
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->0, 7-0->0, 8-0->2, 9-0->1, 10-0->2, 11-0->1, 12-1->3, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->0;m->0, 5-w->1;s->0;m->0, 6-w->1;s->0;m->1, 7-w->0;s->1;m->0, 8-w->5;s->1;m->0, 9-w->1;s->0;m->0, 10-w->1;s->0;m->6, 11-w->1;s->0;m->4, 12-w->1;s->4;m->1, 13-w->6;s->0;m->1, 14-w->0;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-3.5->4.5, 1-17->19, 2-8->9, 3-5->6, 4-29->32, 5-17->19, 6-50->60, 7-22.5->24, 8-40->42, 9-49->52, 10-28->31, 11-64->68, 12-30->33, 13-35->37, 14-37->40, 15-20->23
Speed importance	2
Speed preference	5.5
Speed section	0-4->4.4, 1-4->5.5, 2-4->4.5, 3-7.5->7.7, 4-6->6.4, 5-6->6.5, 6-4.5->5.2, 7-7.5->8.2, 8-5.5->6.2, 9-6.5->7.2, 10-5->5.7, 11-4.2->5, 12-6.5->7.5, 13-8->8.5, 14-6->6.5, 15-3.3->4.4
Jumps section	0-2->3, 1-4->4, 2-5->6, 3-1->1, 4-6->7, 5-8->9, 6-12->14, 7-5->6, 8-20->23, 9-13->15, 10-12->16, 11-20->25, 12-10->12, 13-4->5, 14-6->7, 15-0->0
Backtrack probability	0-0%, 1-0%, 2-0%, 3-0%, 4-0%, 5-0%, 6-5%, 7-0%, 8-0%, 9-100%, 10-0%, 11-0%, 12-75%, 13-0%, 14-0%, 15-0%
New score importance	1
Concentration level preferred	9
Concentration importance	7
Concentration	0-5, 1-3, 2-6, 3-0, 4-6, 5-6, 6-9, 7-2, 8-10, 9-2, 10-10, 11-10, 12-1, 13-3, 14-3, 15-8
Skill level preferred	6.5
Skill importance	5
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	7
Challenge importance	7
Challenge	0-3, 1-1, 2-5, 3-0, 4-2, 5-5, 6-8, 7-1, 8-10, 9-10, 10-10, 11-10, 12-6, 13-2, 14-1, 15-7



Table A.6: Personality 6 - Casual player that has trouble with moving enemies

Setting	Value
Tolerance for repeating sections	3
New level importance	26
Coins importance	0.2
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.2
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life importance	0.75
Life lost importance	6
Life lost section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->2, 7-0->0, 8-0->0, 9-0->0, 10-2->3, 11-1->3, 12-0->2, 13-0->0, 14-0->2, 15-2->3
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->1;s->0;m->0, 5-w->1;s->0;m->0, 6-w->1;s->0;m->8, 7-w->0;s->1;m->0, 8-w->10;s->1;m->0, 9-w->1;s->0;m->0, 10-w->1;s->0;m->15, 11-w->1;s->0;m->15, 12-w->1;s->4;m->1, 13-w->1;s->0;m->0, 14-w->0;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-3.5->4.5, 1-9->10, 2-8->9, 3-5->6, 4-17->18, 5-17->18, 6-39->43, 7-22.5->23.5, 8-40->41, 9-42->53, 10-32->36, 11-55->60, 12-34->36, 13-35->36, 14-27->29, 15-25->30
Speed importance	1
Speed preference	6.3
Speed section	0-4->4.4, 1-6.6->7, 2-4.1->4.5, 3-7.6->8, 4-6.1->6.5, 5-6.4->6.8, 6-4.2->5, 7-8.1->8.5, 8-6.3->6.7, 9-7->7.4, 10-4->4.5, 11-4->4.5, 12-7->7.5, 13-8.3->8.7, 14-5.5->5.9, 15-3.5->4.1
Jumps section	0-2->3, 1-4->5, 2-5->6, 3-1->2, 4-5->6, 5-8->9, 6-14->17, 7-5->6, 8-20->22, 9-13->15, 10-14->18, 11-22->25, 12-10->13, 13-4->5, 14-7->8, 15-0->1
Backtrack probability	0-0%, 1-0%, 2-0%, 3-0%, 4-0%, 5-0%, 6-0%, 7-0%, 8-0%, 9-100%, 10-0%, 11-0%, 12-30%, 13-0%, 14-0%, 15-0%
New score importance	3
Concentration level preferred	2
Concentration importance	7
Concentration	0-0, 1-4, 2-2, 3-0, 4-3, 5-4, 6-8, 7-0, 8-4, 9-4, 10-10, 11-10, 12-7, 13-5, 14-6, 15-10
Skill level preferred	6.5
Skill importance	5
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	3
Challenge importance	7
Challenge	0-1, 1-6, 2-1, 3-0, 4-2, 5-2, 6-9, 7-0, 8-5, 9-5, 10-10, 11-10, 12-7, 13-1, 14-8, 15-9

Table A.7: Personality 7 - Casual player that has trouble with platforming sections

Setting	Value
Tolerance for repeating sections	3
New level importance	26
Coins importance	0.2
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.2
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-1->1, 7-0->0, 8-0->0, 9-1->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life importance	0.75
Life lost importance	6
Life lost section	0-0->0, 1-0->0, 2-1->1, 3-0->0, 4-0->1, 5-1->1, 6-1->2, 7-0->0, 8-2->3, 9-0->1, 10-3->3, 11-2->3, 12-0->0, 13-0->0, 14-0->0, 15-0->0
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->10;s->1;m->0, 5-w->1;s->0;m->0, 6-w->3;s->0;m->1, 7-w->0;s->1;m->0, 8-w->1;s->0;m->0, 9-w->1;s->0;m->0, 10-w->3;s->0;m->1, 11-w->5;s->0;m->1, 12-w->1;s->0;m->1, 13-w->1;s->0;m->4, 14-w->0;s->0;m->1, 15-w->0;s->0;m->1
Time section	0-3.8->4.5, 1-9->10, 2-9->11, 3-5->5.5, 4-18.5->19.5, 5-18.5->20, 6-38->40, 7-22.5->23.5, 8-45->47, 9-45->46, 10-32->35, 11-52->54, 12-30->31, 13-35->36, 14-23.5->24.5, 15-20->21
Speed importance	1
Speed preference	6.3
Speed section	0-4->4.2, 1-6.5->7, 2-3.8->4.2, 3-7.8->8, 4-5.8->6.1, 5-6->6.5, 6-4.9->5.1, 7-8.3->8.5, 8-5.6->6.1, 9-6.6->7, 10-4.8->5.2, 11-4.2->4.5, 12-7.6->8, 13-8.4->8.7, 14-6.4->6.7, 15-4.2->4.4
Jumps section	0-2->3, 1-4->4, 2-6->7, 3-1->1, 4-6->7, 5-9->10, 6-13->15, 7-6->7, 8-23->26, 9-15->17, 10-14->18, 11-23->26, 12-8->9, 13-4->5, 14-6->7, 15-0->0
Backtrack probability	0-0%, 1-0%, 2-0%, 3-0%, 4-0%, 5-0%, 6-0%, 7-0%, 8-0%, 9-100%, 10-0%, 11-0%, 12-80%, 13-0%, 14-0%, 15-0%
New score importance	3
Concentration level preferred	2
Concentration importance	7
Concentration	0-5, 1-1, 2-6, 3-0, 4-5, 5-5, 6-6, 7-4, 8-10, 9-10, 10-10, 11-8, 12-2, 13-0, 14-2, 15-0
Skill level preferred	6.5
Skill importance	5
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	4
Challenge importance	7
Challenge	0-4, 1-1, 2-5, 3-0, 4-4, 5-5, 6-7, 7-0, 8-10, 9-10, 10-10, 11-10, 12-1, 13-1, 14-0, 15-0

Table A.8: Personality 8 - Inexperienced user playing the game for the first time

Setting	Value
Tolerance for repeating sections	6
New level importance	32
Coins importance	0.2
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.2
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->1, 7-0->0, 8-0->0, 9-0->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->1, 15-0->0
Life importance	0.5
Life lost importance	1
Life lost section	0-0->1, 1-0->0, 2-0->2, 3-0->0, 4-0->2, 5-1->2, 6-1->2, 7-0->1, 8-2->3, 9-1->2, 10-3->3, 11-2->3, 12-2->3, 13-0->1, 14-1->1, 15-1->2
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->4;s->1;m->0, 5-w->6;s->1;m->0, 6-w->1;s->0;m->1, 7-w->0;s->1;m->0, 8-w->8;s->1;m->0, 9-w->10;s->1;m->0, 10-w->1;s->0;m->4, 11-w->1;s->0;m->3, 12-w->1;s->4;m->2, 13-w->5;s->0;m->0, 14-w->1;s->0;m->4, 15-w->0;s->0;m->1
Time section	0-5.3->5.7, 1-13->15, 2-13->14, 3-8->10, 4-22->24, 5-21->23, 6-43->45, 7-27->29, 8-48->50, 9-50->52, 10-35->37, 11-58->60, 12-36->38, 13-41->44, 14-30->32, 15-27->29
Speed importance	1
Speed preference	4.5
Speed section	0-3.6->3.8, 1-3.6->3.8, 2-3.6->3.8, 3-7.1->7.3, 4-5.5->5.8, 5-5.6->5.8, 6-4->4.3, 7-7->7.3, 8-5->5.3, 9-6->6.3, 10-4.5->4.8, 11-3.7->4, 12-6->6.3, 13-7.5->7.8, 14-5.4->5.8, 15-3.3->3.6
Jumps section	0-3->4, 1-5->6, 2-6->7, 3-2->2, 4-8->9, 5-10->11, 6-13->15, 7-7->8, 8-25->27, 9-17->19, 10-15->17, 11-27->29, 12-13->14, 13-6->7, 14-9->10, 15-1->2
Backtrack probability	0-5%, 1-40%, 2-5%, 3-8%, 4-15%, 5-15%, 6-40%, 7-10%, 8-25%, 9-100%, 10-50%, 11-75%, 12-90%, 13-5%, 14-15%, 15-7%
New score importance	2
Concentration level preferred	8
Concentration importance	7
Concentration	0-5, 1-2, 2-6, 3-0, 4-6, 5-6, 6-10, 7-2, 8-10, 9-2, 10-10, 11-10, 12-1, 13-3, 14-5, 15-10
Skill level preferred	3.5
Skill importance	4
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	5
Challenge importance	7
Challenge	0-3, 1-1, 2-5, 3-0, 4-4, 5-5, 6-8, 7-1, 8-10, 9-10, 10-10, 11-10, 12-9, 13-1, 14-1, 15-6

Table A.9: Personality 9 - Inexperienced player that plays the game with care

Setting	Value
Tolerance for repeating sections	3
New level importance	24
Coins importance	0.2
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.2
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->1, 7-0->0, 8-0->0, 9-0->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->1, 15-0->0
Life importance	0.5
Life lost importance	1
Life lost section	0-0->1, 1-0->0, 2-0->2, 3-0->0, 4-0->2, 5-1->2, 6-1->3, 7-0->1, 8-2->3, 9-1->2, 10-3->3, 11-2->3, 12-2->3, 13-0->1, 14-1->1, 15-1->3
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->4;s->1;m->0, 5-w->6;s->1;m->0, 6-w->1;s->0;m->3, 7-w->0;s->1;m->0, 8-w->8;s->1;m->0, 9-w->10;s->1;m->0, 10-w->1;s->0;m->4, 11-w->1;s->0;m->3, 12-w->1;s->4;m->2, 13-w->5;s->0;m->0, 14-w->1;s->0;m->4, 15-w->0;s->0;m->1
Time section	0-4.5->5.3, 1-11.5->13.5, 2-10.5->11.5, 3-6.5->8.5, 4-19.5->21.5, 5-19.5->21.5, 6-39.5->41.5, 7-24.5->26.5, 8-45.5->47.5, 9-47.5->49.5, 10-31.5->33.5, 11-54.5->56.5, 12-33.5->35.5, 13-37.5->39.5, 14-26.5->28.5, 15-24.5->26.5
Speed importance	2
Speed preference	5
Speed section	0-3.5->3.8, 1-3.5->3.8, 2-3.5->3.8, 3-7->7.3, 4-5.5->5.8, 5-5.5->5.8, 6-4->4.3, 7-7->7.3, 8-5->5.3, 9-6->6.3, 10-4.5->4.8, 11-3.7->4, 12-6->6.3, 13-7.5->7.8, 14-5.6->5.8, 15-3.3->3.6
Jumps section	0-4->6, 1-6->8, 2-7->9, 3-3->5, 4-8->11, 5-10->13, 6-13->17, 7-7->10, 8-24->28, 9-17->21, 10-15->19, 11-26->30, 12-13->16, 13-6->9, 14-9->12, 15-1->5
Backtrack probability	0-5%, 1-40%, 2-5%, 3-20%, 4-17%, 5-17%, 6-40%, 7-15%, 8-30%, 9-100%, 10-50%, 11-70%, 12-95%, 13-5%, 14-20%, 15-7%
New score importance	2
Concentration level preferred	2
Concentration importance	7
Concentration	0-4, 1-0, 2-8, 3-0, 4-1, 5-5, 6-8, 7-2, 8-10, 9-5, 10-10, 11-10, 12-7, 13-1, 14-3, 15-6
Skill level preferred	3.5
Skill importance	4
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	3
Challenge importance	7
Challenge	0-2, 1-3, 2-6, 3-0, 4-3, 5-5, 6-9, 7-1, 8-10, 9-8, 10-8, 11-8, 12-6, 13-1, 14-0, 15-7

Table A.10: Personality 10 - Inexperienced hyperactive player

Setting	Value
Tolerance for repeating sections	4
New level importance	28
Coins importance	0.2
Coins section	0-1->1, 1-2->2, 2-4->4, 3-1->1, 4-2->4, 5-3->3, 6-9->10, 7-5->5, 8-9->9, 9-8->14, 10-5->5, 11-6->7, 12-4->4, 13-5->5, 14-0->0, 15-0->0
Chest importance	0.2
Chest section	0-0->0, 1-0->0, 2-0->0, 3-0->0, 4-0->0, 5-0->0, 6-0->1, 7-0->0, 8-0->0, 9-0->1, 10-0->0, 11-0->0, 12-0->0, 13-0->0, 14-0->1, 15-0->0
Life importance	0.5
Life lost importance	1
Life lost section	0-0->1, 1-0->0, 2-0->2, 3-0->0, 4-0->2, 5-1->2, 6-1->2, 7-0->1, 8-2->3, 9-1->2, 10-3->3, 11-2->3, 12-2->3, 13-0->1, 14-1->1, 15-1->2
Life lost type section	0-w->0;s->1;m->0, 1-w->0;s->0;m->1, 2-w->0;s->1;m->0, 3-w->0;s->0;m->0, 4-w->4;s->1;m->0, 5-w->6;s->1;m->0, 6-w->1;s->0;m->1, 7-w->0;s->1;m->0, 8-w->8;s->1;m->0, 9-w->10;s->1;m->0, 10-w->1;s->0;m->4, 11-w->1;s->0;m->3, 12-w->1;s->4;m->2, 13-w->5;s->0;m->0, 14-w->1;s->0;m->4, 15-w->0;s->0;m->1
Time section	0-5->5.5, 1-12->14, 2-11->12, 3-7->9, 4-20->22, 5-20->22, 6-40->42, 7-25->27, 8-46->48, 9-48->50, 10-32->34, 11-55->57, 12-34->36, 13-38->40, 14-27->29, 15-25->27
Speed importance	1
Speed preference	4.5
Speed section	0-3.7->4, 1-3.7->4, 2-3.7->4, 3-7.2->7.5, 4-5.7->6, 5-5.7->6, 6-4.2->4.5, 7-7.2->7.5, 8-5.2->5.5, 9-6.2->6.5, 10-4.7->5, 11-3.9->4.2, 12-6.2->6.5, 13-7.7->8, 14-5.7->6, 15-3.5->3.8
Jumps section	0-3->3, 1-5->5, 2-6->6, 3-2->2, 4-7->8, 5-9->10, 6-12->14, 7-6->7, 8-23->25, 9-16->18, 10-14->16, 11-25->27, 12-12->13, 13-5->6, 14-8->9, 15-0->2
Backtrack probability	0-0%, 1-30%, 2-0%, 3-2%, 4-10%, 5-10%, 6-30%, 7-5%, 8-20%, 9-100%, 10-40%, 11-60%, 12-80%, 13-0%, 14-10%, 15-1%
New score importance	2
Concentration level preferred	2
Concentration importance	7
Concentration	0-2, 1-1, 2-4, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-8, 10-10, 11-10, 12-8, 13-3, 14-3, 15-6
Skill level preferred	3.2
Skill importance	4
Skill	0-1, 1-1, 2-3, 3-0, 4-3, 5-4, 6-7, 7-2, 8-10, 9-6, 10-10, 11-10, 12-2, 13-1, 14-4, 15-5
Challenge level preferred	4.5
Challenge importance	4
Challenge	0-3, 1-2, 2-5, 3-0, 4-4, 5-4, 6-8, 7-4, 8-10, 9-8, 10-10, 11-10, 12-7, 13-2, 14-2, 15-5



## **Appendix B**

# **Questionnaire**

The round B questions are the same as round A questions.

Age \*

A sua resposta

Gender \*

Selecionar ▼

How good are you at platforming games? \*

1 2 3 4 5

Never played ○ ○ ○ ○ ○ Expert

Figure B.1: Questionnaire



**Round A questions**

These questions are related to the rounds A

Please indicate how you felt while playing the game in the A's rounds for each of the items \*

	Not at all	Slightly	Moderately	Fairly	Extremely
I felt content	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt skilful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was interested in the game's story	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought it was fun	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was fully occupied with the game	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt happy	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It gave me a bad mood	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure B.2: Questionnaire

I thought about other things	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found it tiresome	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt competent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought it was hard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It was aesthetically pleasing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I forgot everything around me	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt good	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was good at it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt bored	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt successful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt imaginative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt that I could explore things	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I enjoyed it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was fast at reaching the game's targets	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure B.3: Questionnaire

I felt annoyed	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt pressured	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt irritable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I lost track of time	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt challenged	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found it impressive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was deeply concentrated in the game	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt frustrated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It felt like a rich experience	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I lost connection with the outside world	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt time pressure	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I had to put a lot of effort into it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

[Anterior](#) [Seguente](#)  Página 2 de 4

Figure B.4: Questionnaire

### Final section

After playing the game, please upload the 2 files: data\_game.csv and data\_section.csv. When you exit the game, a window will open with the location of these files. If this does not happen, please go to the folder "C:\Users\  
<username>\AppData\LocalLow\Bayat Games\Red Runner" (replace <username> with your pc user name) \*

[↑ Adicionar ficheiro](#)

What round do you think had the adaptivity implemented?

Round A

Round B

Do you have anything to add?

A sua resposta

---

**Thank you for your time and your answers**

Any questions can be forward to [up201605314@fe.up.pt](mailto:up201605314@fe.up.pt)

[Anterior](#) [Submeter](#)


 Página 4 de 4

Figure B.5: Questionnaire

## **Appendix C**

# **Player simulation results**

The full results can be found at [repository folder](#)



## **Appendix D**

# **Adaptivity agent results**

The full results can be found at [repository folder](#)





## Appendix E

# Additional questionnaire results

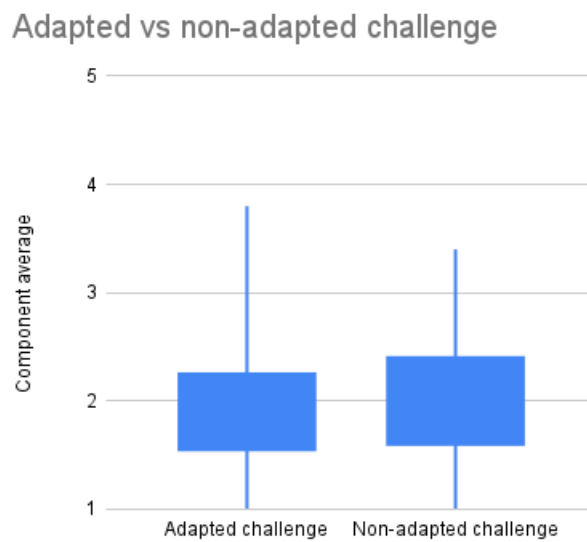


Figure E.1: Adapted vs non-adapted challenge

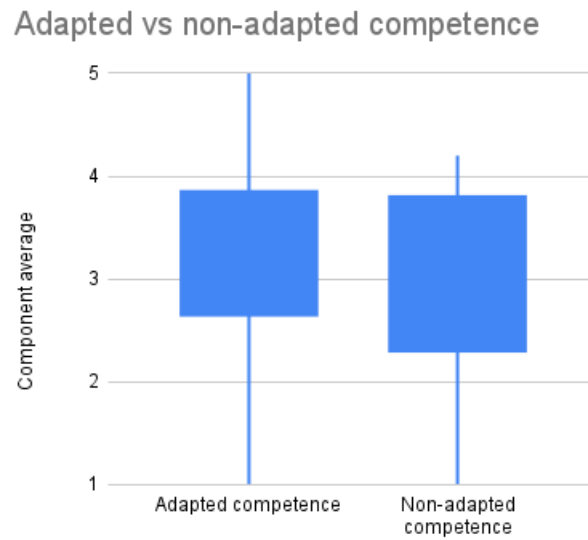


Figure E.2: Adapted vs non-adapted competence

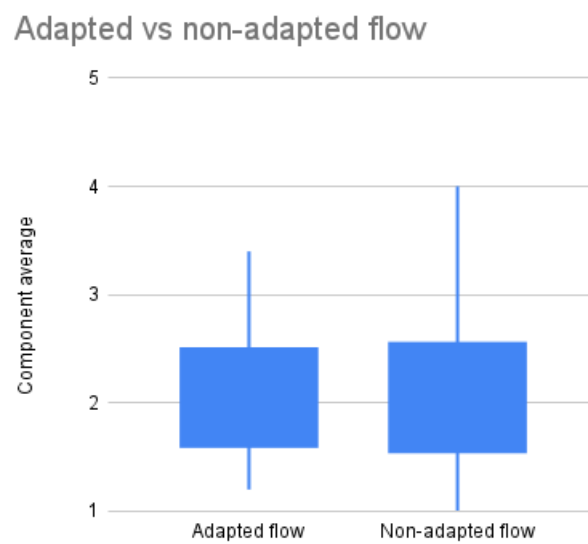


Figure E.3: Adapted vs non-adapted flow

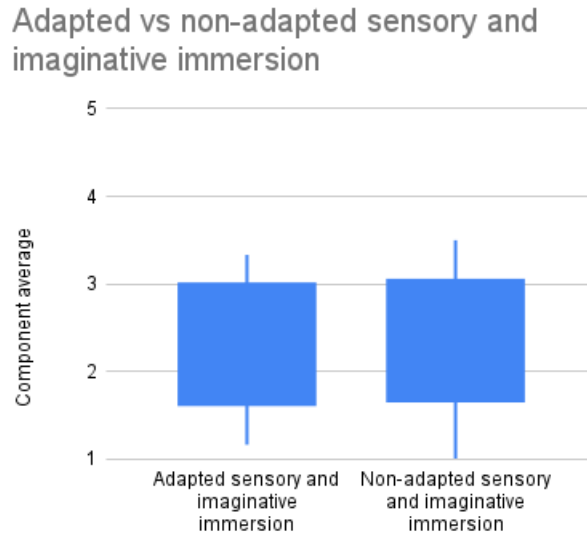


Figure E.4: Adapted vs non-adapted sensory and imaginative immersion

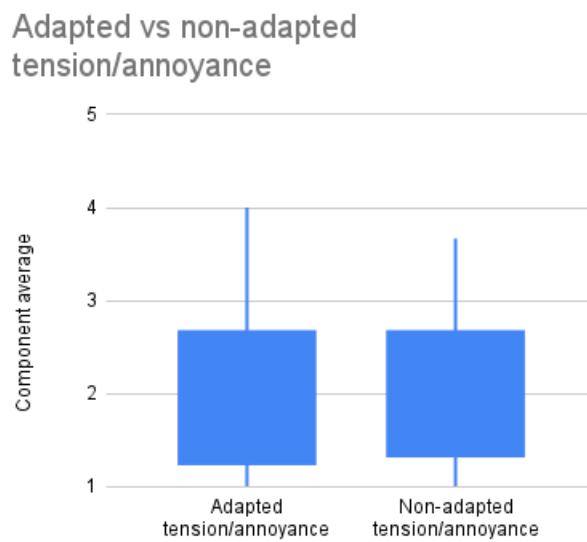


Figure E.5: Adapted vs non-adapted tension/annoyance

Adapted game questionnaire responses

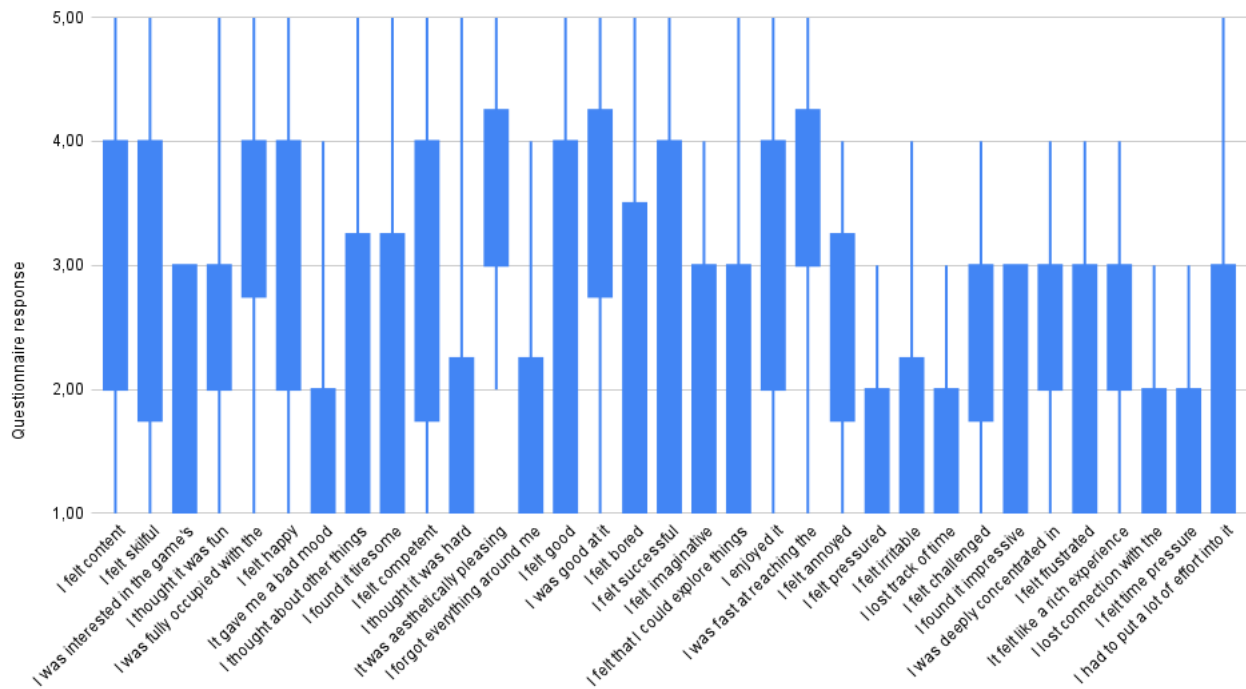


Figure E.6: Adapted game questionnaire responses

Non-adapted game questionnaire responses

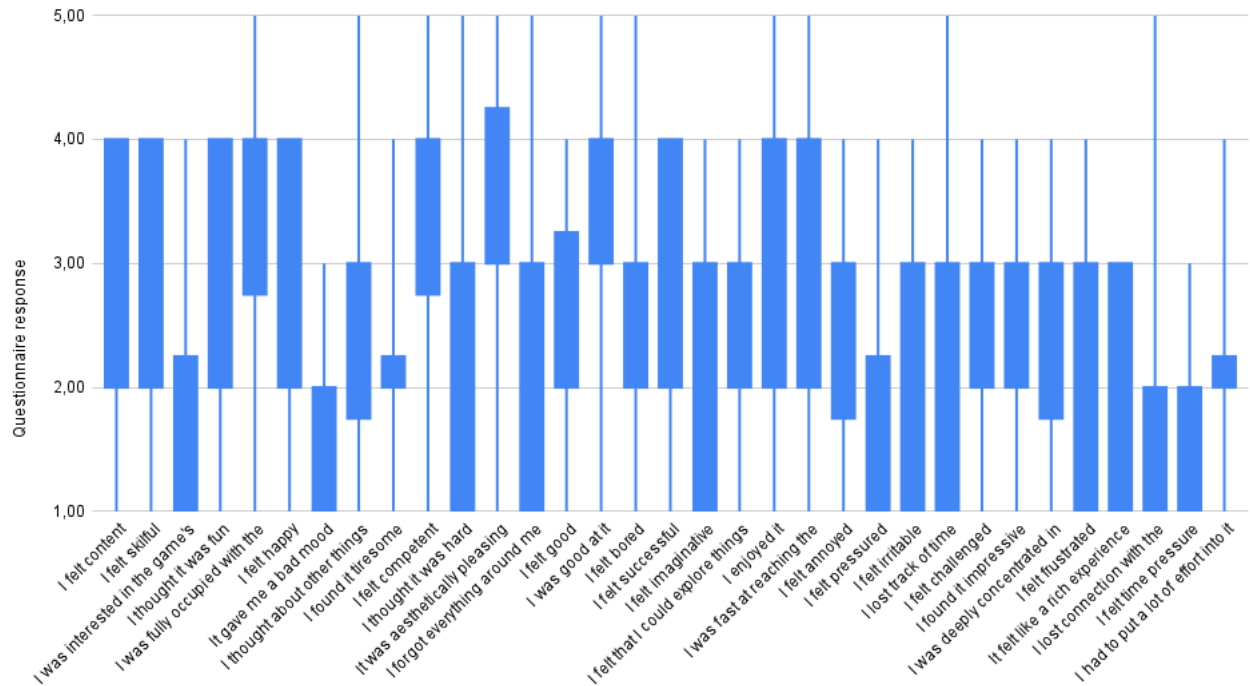


Figure E.7: Non-adapted game questionnaire responses

Gender

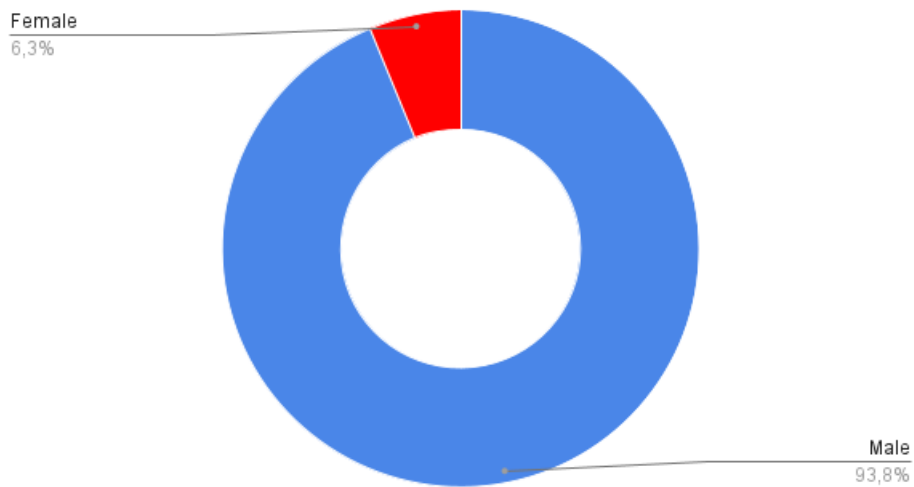


Figure E.8: Gender of the questionnaire responses



# References

- [1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning, 2018.
- [2] Ryan Macdonell Andrias and Mohd Shahrizal Sunar. User/player type in gamification. *International Journal of Advanced Trends in Computer Science and Engineering*, 8(1.6 Special Issue):89–94, 2019.
- [3] Atari. Breakout, 1976.
- [4] Yusuf Aytar, Tobias Pfaff, David Budden, Tom Le Paine, Ziyu Wang, and Nando De Freitas. Playing hard exploration games by watching YouTube. *Advances in Neural Information Processing Systems*, 2018-Decem:2930–2941, 2018.
- [5] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent Tool Use From Multi-Agent Autocurricula. sep 2019.
- [6] Sander Bakkes, Chek Tien Tan, and Yusuf Pisan. Personalised gaming: A motivation and overview of literature. In *ACM International Conference Proceeding Series*, 2012.
- [7] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent Complexity via Multi-Agent Competition. oct 2017.
- [8] Richard Bartle. Hearts, Clubs, Diamonds, Spades: Players Who Suit Muds. *Journal of MUD Research*, 1(1):19, 1996.
- [9] Chris Bateman and Richard Boon. *21st century game design (Game Development Series)*. Charles River Media, Inc., USA, 2006.
- [10] Chris Bateman, Rebecca Lowenhaupt, and Lennart E. Nacke. Player typology in theory and practice. In *Proceedings of DiGRA 2011 Conference: Think Design Play*, 2011.
- [11] Bayat Games. Red Runner, 2017.
- [12] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, pages 41–48, 2009.
- [13] Blizzard Entertainment. StarCraft, 1998.
- [14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. jun 2016.
- [15] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation. oct 2018.

- [16] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ACM International Conference Proceeding Series*, volume 148, pages 161–168, 2006.
- [17] Jenova Chen. Flow in games (and everything else), 2007.
- [18] Mihaly Csikszentmihalyi and Mihaly Csikzentmihaly. *Flow: The psychology of optimal experience*, volume 1990. Harper Row New York, 1990.
- [19] Alena Denisova and Paul Cairns. Adaptation in digital games: The effect of challenge adjustment on player performance and experience. In *CHI PLAY 2015 - Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pages 97–102, 2015.
- [20] M. Faure, P. Gaillard, B. Gaujal, and V. Perchet. Online learning and game theory. A quick overview with recent results and applications. *ESAIM: Proceedings and Surveys*, 51, oct 2015.
- [21] Julian Frommel, Fabian Fischbach, Katja Rogers, and Michael Weber. Emotion-based Dynamic Difficulty Adjustment Using Parameterized Difficulty and Self-Reports of Emotion. In *CHI PLAY 2018 - Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 173–185, 2018.
- [22] Zoubin Ghahramani. Unsupervised learning. In *Summer School on Machine Learning*, pages 72–112. Springer, 2003.
- [23] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. jun 2014.
- [24] Venkat Gudivada, Amy Apon, and Junhua Ding. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software*, 10(1):1–20, 2017.
- [25] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. jan 2018.
- [26] John A Hartigan and Manchek A Wong. A K-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):100–108, 1979.
- [27] Marti A Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.
- [28] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, pages 4572–4580, jun 2016.
- [29] Danial Hooshyar, Liina Malva, Yeongwook Yang, Margus Pedaste, Minhong Wang, and Heuseok Lim. An adaptive educational computer game: Effects on students’ knowledge and learning attitude in computational thinking. *Computers in Human Behavior*, 114, 2021.
- [30] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.
- [31] W A IJsselsteijn, Y A W de Kort, and K Poels. *The Game Experience Questionnaire*. Technische Universiteit Eindhoven, 2013.



- [32] Wijnand A IJsselsteijn, Yvonne A W de Kort, and Karolien Poels. The game experience questionnaire. *Eindhoven: Technische Universiteit Eindhoven*, 46(1), 2013.
- [33] João Jacob, Ana Lopes, Rui Nóbrega, Rui Rodrigues, and António Coelho. Towards player adaptivity in mobile exergames. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10714 LNCS, pages 278–292. Springer International Publishing, Cham, Switzerland, 2018.
- [34] Lars Jensvold. *Left 4 Dead*, 2011.
- [35] Mariana Neto João Álvaro Ferreira, João Augusto Lima, João Carlos Maduro. Exploring Multi-Output Regression and Reinforcement Learning for Game Adaptivity. 2021.
- [36] Arthur Juliani. Solving sparse-reward tasks with Curiosity, 2018.
- [37] V. Khaustov and M. Mozgovoy. Learning Believable Player Movement Patterns from Human Data in a Soccer Game. *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, 2020.
- [38] Konami. *Pro Evolution Soccer 2008*, 2007.
- [39] N. Lazzaro. Why We Play Games: Four Keys to More Emotion Without Story. *Game Developer Conference (GDC)*, pages 1–8, 2004.
- [40] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. sep 2015.
- [41] Ricardo Lopes and Rafael Bidarra. Adaptivity challenges in games and simulations: A survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):85–99, 2011.
- [42] Ricardo Lopes, Elmar Eisemann, and Rafael Bidarra. Authoring adaptive game world generation. *IEEE Transactions on Games*, 10(1):42–55, 2018.
- [43] Darina Lynkova. Video Game Statistics [Click the “Start” Button], 2020.
- [44] Matt Makes Games. *Celeste*, 2018.
- [45] Olana Missura and Thomas Gärtner. Player modeling for intelligent difficulty adjustment. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5808 LNAI, pages 197–211, 2009.
- [46] Shigeru Miyamoto and Nintendo. *Super Mario Bros.*, 1985.
- [47] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. feb 2016.
- [48] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.
- [49] Mikhail Moshkov and Beata Zielosko. Supervised Learning. In *Machine learning techniques for multimedia*, pages 113–126. Springer, 2011.

- [50] Lennart E Nacke, Chris Bateman, and Regan L Mandryk. BrainHex: preliminary results from a neurobiological gamer typology survey. In *International conference on entertainment computing*, pages 288–293. Springer, 2011.
- [51] Nintendo EAD. Mario Kart Wii, 2008.
- [52] Asheyy Noblega, Aline Paes, and Esteban Clua. Towards adaptive deep reinforcement game balancing. In *ICAART 2019 - Proceedings of the 11th International Conference on Agents and Artificial Intelligence*, volume 2, pages 693–700, 2019.
- [53] Spyros Papadimitriou and Maria Virvou. Adaptivity in scenarios in an educational adventure game. In *2017 8th International Conference on Information, Intelligence, Systems and Applications, IISA 2017*, volume 2018-Janua, pages 1–6. IEEE, Piscataway, NJ, USA, 2018.
- [54] Quantic Dream. Heavy Rain, 2010.
- [55] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable Baselines3. <https://github.com/DLR-RM/stable-baselines3>, 2019.
- [56] Sudharsan Ravichadiran. *Deep Reinforcement Learning with Python*. Packt, 2 edition, 2020.
- [57] Claudia Redaelli and Giuseppe Riva. Flow for Presence Questionnaire. In *Digital Factory for Human-oriented Production Systems*, pages 3–22. Springer, 2011.
- [58] Simão Reis, Luís Paulo Reis, and Nuno Lau. Game Adaptation by Using Reinforcement Learning Over Meta Games. *Group Decision and Negotiation*, 2020.
- [59] Rockstar Studios. Max Payne 3, 2012.
- [60] Louis Schmidt, Taichi Watanabe, and Koji Mikami. Adjusting the game difficulty by changing AI behaviors with Reinforcement Learning. In *Proceedings - NICOGRAPH International 2020, NicoInt 2020*, page 94, 2020.
- [61] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, jul 2017.
- [62] Na Shi, Xumin Liu, and Yong Guan. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010*, pages 63–67. Ieee, 2010.
- [63] Fatimah Sidi, Payam Hassany Shariat Panahy, Lilly Suriani Affendey, Marzanah A. Jabar, Hamidah Ibrahim, and Aida Mustapha. Data quality: A survey of data quality dimensions. In *Proceedings - 2012 International Conference on Information Retrieval and Knowledge Management, CAMP'12*, pages 300–304. IEEE, 2012.
- [64] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, dec 2017.
- [65] Penelope Sweetser, Daniel Johnson, Peta Wyeth, Aiman Anwar, Yan Meng, and Anne Ozdowska. GameFlow in different game genres and platforms. *Computers in Entertainment*, 15(3), 2017.

- [66] Penelope Sweetser and Peta Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.
- [67] Gustavo F. Tondello, Rina R. Wehbe, Lisa Diamond, Marc Busch, Andrzej Marczewski, and Lennart E. Nacke. The gamification user types Hexad scale. In *CHI PLAY 2016 - Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, pages 229–243, 2016.
- [68] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral Cloning from Observation. may 2018.
- [69] Unity Technologies. Unity, 2005.
- [70] Valve Corporation. Dota 2, 2013.
- [71] Giel Van Lankveld, Pieter Spronck, Jaap Van Den Herik, and Arnoud Arntz. Games as personality profiling tools. In *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pages 197–202, 2011.
- [72] Willem Waegeman, Krzysztof Dembczyński, and Eyke Hüllermeier. Multi-target prediction: a unifying view on problems and methods. *Data Mining and Knowledge Discovery*, 33(2):293–324, 2019.
- [73] Abdul Wahid. Big data and machine learning for Businesses, 2017.
- [74] Wanxiang Li, Chu-Hsuan Hsueh, and K. Ikeda. Imitating Agents in A Complex Environment by Generative Adversarial Imitation Learning. *2020 IEEE Conference on Games (CoG)*, 2020.
- [75] Nick Yee. The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. *Presence: Teleoperators and Virtual Environments*, 15(3):309–329, 2006.
- [76] Taizo Yoshikawa, Viktor Losing, and Emel Demircan. Machine learning for human movement understanding. *Advanced Robotics*, 34(13):828–844, 2020.