UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Video Game Character learning with Artificial Intelligent Algorithms

Diploma Thesis

Georgios Fragkias

Supervisor: Hariklia Tsalapata

Volos 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Video Game Character learning with Artificial Intelligent Algorithms

Diploma Thesis

Georgios Fragkias

Supervisor: Hariklia Tsalapata

Volos 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εκπαίδευση χαρακτήρα ψηφιακού παιχνιδιού μέσω αλγορίθμων Τεχνητής Νοημοσύνης

Διπλωματική Εργασία

Γεώργιος Φραγκιάς

Επιβλέπων/πουσα: Χαρίκλεια Τσαλαπάτα

Βόλος 2023

Approved by the Examination Committee:

Supervisor **Hariklia Tsalapata**
E.DI.P, Department of Electrical and Computer Engineering, University of Thessaly

Member **Aspassia Daskalopulu**
Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Stamoulis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 26-3-2021

Acknowledgements

I would like to express my sincere gratitude to my family for their support, as well as, my professors and friends for their encouragement and guidance through all these years at the University.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Georgios Fragkias
17-2-2023

Abstract

Many of the technological advancements that occur everyday, especially with Artificial Intelligence has paved the path for rapid evolution in the game industry as well. Different Artificial intelligence algorithms are being used in each game nowadays adding more to the engaging, challenging, and personalized aspect of a video game as well as the realism of serious games. The game developed in this thesis, using the MLAgents toolkit of Unity's engine, offers a valuable perspective on the implementation of Artificial Intelligence algorithms through the successful training of an agent using deep reinforcement learning methods. The agent is acquiring the ability to interact with its surroundings in a firefighting setting, honing its decision-making skills and movements according to the positions of the fires and obstacles, placing him later to compete against a human player. The successful training of the agent demonstrates the potential for it to operate with accuracy in real-world scenarios, highlighting the significance of utilizing Artificial Intelligence algorithms.

Περίληψη

Πολλές από τις τεχνολογικές προόδους που συμβαίνουν καθημερινά, ειδικά στον τομέα της Τεχνητής Νοημοσύνης, έχουν ανοίξει το δρόμο για την ταχεία εξέλιξη στον τομέα των ψηφιακών παιχνιδιών. Σχεδόν σε κάθε παιχνίδι στις μέρες μας χρησιμοποιούνται διαφορετικοί αλγόριθμοι Τεχνητής Νοημοσύνης, προσθέτοντας περισσότερα στην ενεργητική, προκαλεστική και προσαρμοσμένη πλευρά του βιντεοπαιχνιδιού, καθώς και στην ρεαλιστικότητα των σοβαρών παιχνιδιών. Το παιχνίδι που αναπτύχθηκε σε αυτή τη διπλωματική, χρησιμοποιώντας την εργαλειοθήκη “MLAgents” της μηχανής της “Unity”, παρέχει μια πολύτιμη προοπτική για την εφαρμογή αλγορίθμων Τεχνητής Νοημοσύνης μέσω της επιτυχούς εκπαίδευσης ενός πράκτορα χρησιμοποιώντας μεθόδους βαθιάς ενισχυτικής μάθησης. Ο πράκτορας αποκτά την ικανότητα να αλληλεπιδρά με το περιβάλλον του σε ένα σενάριο πυρόσβεσης, βελτιώνοντας τις δεξιότητες λήψης αποφάσεων και τις κινήσεις του σύμφωνα με τις θέσεις των πυρκαγιών και των εμποδίων, τοποθετώντας τον αργότερα να ανταγωνιστεί έναν ανθρώπινο παίκτη.Η επιτυχημένη εκπαίδευση του πράκτορα καταδεικνύει τη δυνατότητα να λειτουργεί με ακρίβεια σε σενάρια πραγματικού κόσμου, τονίζοντας τη σημασία της χρήσης αλγορίθμων Τεχνητής Νοημοσύνης.

Table of contents

Acknowledgements **ix**

Abstract **xi**

Περίληψη **xiii**

Table of contents **xv**

List of figures **xix**

List of tables **xxi**

Abbreviations **xxiii**

1 Introduction **1**

1.1 Aim of the thesis 2

1.1.1 Contribution 2

1.2 Structure of the thesis 3

2 Digital Games **5**

2.1 Introduction to interactive computer-based games 5

2.2 Video Game Categories 6

2.3 Serious Games 12

2.4 Serious Games Categories 13

3 Artificial Intelligence and Games **15**

3.1 Introduction to Game Artificial Intelligence 15

3.1.1 What is Intelligence 15

3.1.2 What is Game AI . . . 15
3.2 Historic View of Game AI . . . 16
3.3 Importance of Artificial Intelligence in Games . . . 18
3.3.1 Benefits of Artificial Intelligence in Games . . . 18
3.3.2 Artificial Intelligence in Serious Games . . . 20
3.4 Techniques and Algorithms in Game AI . . . 22
3.4.1 Rule-Based Model . . . 22
3.4.2 Decision-Tree Model . . . 23
3.4.3 Machine Learning AI Model . . . 24
3.5 Machine Learning Models . . . 26
3.5.1 Deep Learning . . . 30
3.6 Model of this thesis . . . 31

4 Unity **33**
4.1 Game Engines . . . 33
4.2 Introduction to Unity Engine . . . 35
4.3 Reason for picking Unity . . . 36
4.4 Unity Environment . . . 37
4.4.1 Unity Scenes . . . 37
4.4.2 Unity Interface . . . 38
4.5 Unity Scripting . . . 39
4.5.1 Conventional and Visual Scripting . . . 39
4.5.2 Key Unity Script Functions . . . 40
4.5.3 Introduction to Unity Machine Learning Toolkit . . . 41
4.5.4 Agent Class . . . 42

5 The Game **45**
5.1 Game overview . . . 45
5.1.1 Method Used . . . 46
5.1.2 Agent Component . . . 46
5.2 Behavior Parameters . . . 47
5.2.1 Decision Requester . . . 48
5.3 Environment . . . 49

5.3.1 Learning Environment 49
5.3.2 Testing the Environment 50
5.3.3 Reward System 51
5.3.4 Agent's actions 52
5.3.5 Hyperparameters 53
5.4 Training Process 55
5.4.1 Training Results 56
5.5 Game Scene 59
5.5.1 Game Avatar 59
5.5.2 Game Menus 59
5.5.3 Gameplay View 60

6 Conclusion **63**
6.1 Summary 63
6.2 Future work 64

Bibliography **67**

List of figures

3.1 Nim Game [8] . . . 16
3.2 Space Invaders . . . 16
3.3 Pong Game . . . 17
3.4 Pac Man . . . 17
3.5 Black & White . . . 17
3.6 Sample of a Decision Tree . . . 23
3.7 Machine Learning Types [21] . . . 26
3.8 Deep Learning's place in Artificial Intelligence [16] . . . 30
3.9 Example of (Deep) Reinforcement learning with a serious game as environment and an artificial neural network as the agent's action selection component [23] . . . 32

4.1 Unity Logo . . . 35
4.2 Sample of Unity supported platforms. [33] . . . 35
4.3 Unity Editor Interface . . . 37
4.4 High-level view of the ML-Agent toolkit [49] . . . 42

5.1 Behavior Parameters script on one of the agents during testing. . . . 48
5.2 Unity's Decision Requester. . . . 49
5.3 The function to collect observations from the environment . . . 50
5.4 Screenshot of the Heuristic function . . . 51
5.5 On Action Received function . . . 53
5.6 Training Mode Environment . . . 56
5.7 Early training results . . . 57
5.8 Later at the training results . . . 57
5.9 Environment / Cumulative Reward Graph . . . 58

5.10 Losses / Policy Loss Graph 58
5.11 Losses / Value Loss Graph 58
5.12 The Avatar and its components 59
5.13 Game's Starting Menu 60
5.14 Screenshot of the game's result with the Agent being the winner 61

List of tables

5.1 HyperParameters 54
5.2 Network Settings 55
5.3 Extrinsic Rewards 55

Abbreviations

AI	Artificial Intelligence
NPC	non-player character
RPG	Role-Playing Games
MORPG	Massively Multiplayer Online RolePlaying Game
TRPG	Tactical RolePlaying Games
MOBA	Mobile Online Battle Area
FPS	First Person Shooters
TPS	Third Person Shooters
FSM	finite state machines
ALE	Arcade Learning Environment
ML	Machine Learning
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
RL	Reinforcement Learning
NN	neural network
ANN	artificial neural network
AR	augmented reality
VR	virtual reality
SDK	software development kit
API	Application Programming Interface
PPO	Proximal Policy Optimization

Chapter 1

Introduction

Digital games, also known as video games, are interactive computer-based games that are played on a variety of platforms, including PC, console, and mobile devices. They are typically created for the purpose of entertainment, and they often involve a goal or objective that the player must achieve. Digital games can be further divided into subcategories, such as action games, strategy games, and role-playing games, based on their gameplay mechanics and themes.

In recent years, digital games have gained increasing popularity, and they have become a major part of the entertainment industry. The global video game market is expected to surpass $200 billion in revenue by 2023, and reach over $218 billion in 2024, with mobile games and console games being the largest segments.

Serious games, on the other hand, are games that are designed for a primary purpose other than pure entertainment. They are used for a variety of purposes, such as education, training, health promotion, and policy analysis. Serious games are often designed to be more interactive and immersive than traditional educational materials, and they can be an effective way to engage learners and help them retain information.

Artificial intelligence (AI) is becoming an increasingly important aspect of digital and serious games. AI can be used to create more realistic and engaging gameplay experiences, as well as to make games more adaptive and personalized to the player. In digital games, AI can be used for a variety of purposes, such as creating non-player characters (NPCs) that have realistic behaviors and decision-making abilities, or to create dynamic and adaptive game environments. In serious games, AI can be used to provide personalized learning experiences, to track and analyze player progress, or to provide real-time feedback to players.

AI can also be used in games to provide more sophisticated game mechanics and to create new types of games. For example, AI can be used to create intelligent opponents that can adapt to the player's style of play, or even to the environment. In addition, AI can be used to create more advanced virtual reality and augmented reality experiences, which can make games more immersive and engaging.

The integration of AI in digital and serious games has seen significant growth in recent years and has become a crucial element for enhancing the overall gaming experience. AI has the potential to be utilized as an effective tool for learning, persuasion, training, and decision-making.Despite advancements in the field, further research is required to fully comprehend the efficacy of AI implementation in these types of games and to develop strategies for achieving specific objectives.

1.1 Aim of the thesis

This thesis aims to explore the potential of digital games and serious games as learning and persuasion tools, and to identify the factors that contribute to their effectiveness. Specifically, this thesis will involve the creation of a serious game using the Unity game engine, as well as the algorithms behind the creation and training of an AI agent and will examine the impact of this game on players' learning, adaptability and behavior. Through a review of the literature and analysis of empirical data, this thesis will examine the current state of Artificial Intelligence on digital games and serious games, and will seek to identify best practices for their use and implement them in a digital game.

1.1.1 Contribution

Overall, this thesis will contribute to the growing body of knowledge on digital games and serious games, especially to the AI in the game industry and will provide insights that will be useful to researchers, educators, and practitioners working in the field. The development and analysis of the serious game created in Unity will add to the understanding of the potential of AI's use in digital games, and will provide valuable insights for future game developers in this field.

1.2 Structure of the thesis

The thesis is divided into six chapters. The first chapter is the introduction, describing the aim of the thesis. The second chapter presents the video games and serious games and their definitions. The third chapter introduces the Artificial Intelligence in games, their types, and models which can be used in game development, in addition to which method was used for the purpose of this game. The fourth chapter introduces the game engines and Unity, the platform where the development of the game was implemented. The fifth chapter presents the game and its development. Finally, the last chapter is the conclusion, summarizing the thesis and presenting the future work.

Chapter 2

Digital Games

2.1 Introduction to interactive computer-based games

Video games and serious games have become an integral part of modern entertainment and education. They are both interactive computer-based games that are played on a variety of platforms, including PC, console, and mobile devices. While video games, also known as digital games, are primarily created for the purpose of entertainment, serious games are designed with a primary purpose other than pure entertainment such as education, training, health promotion, and policy analysis. Despite the difference in purpose, both video games and serious games share similar elements such as gameplay mechanics and interactive elements, and both are enjoyed by a wide audience.

Nowadays, video games are one of the most popular entertainment activities and one of the largest entertainment industries, with the total number of users increasing every year. According to Prensky [1], video games consist of certain structural characteristics such as rules, objectives, representation, interactivity, progression, and competition that contribute to the overall gaming experience and make it engaging for the user.

1 **Rules:** set limits and force the player to use specific paths, while making it fair and causing excitement.

2 **Competition:** is a key element of the game that make it more interesting and exciting for the player.

3 **Representation:** is a fundamental aspect of every video game and it refers to a specific theme that is present, either directly or indirectly, in the game. This theme is usually

portrayed through various narrative elements that are included in the game. Additionally, representation also encompasses the use of imagination which is an important aspect in determining the identity of a game.

4 **Progression:** helps players to follow their progress and achieve their goals.

5 **Goals and objectives:** they constitute the driving force of the player and are implemented through the observance of the rules. It is an important element of the game, as as a genre we are 'programmed' to pursue them [1].

6 **Interactivity:** is achieved at two levels and concerns the relationship of the player with the computer and the relationship of the player with other players. Through this step, the social character of the games, which occupies more and more areas that aim at development is demonstrated.

2.2 Video Game Categories

To this day the universal categorization of the video games has been found to be quite challenging. As the number of games continues to grow, it becomes clear that a game can often not be placed in a single category, since there is a wide variety of games that exist, different characteristics they possess, objectives they aim to achieve, audience they cater to, and other factors. The first one who tried to classify them was Caillois back in 1958, and ended up with four primary categories: games of luck, movement, competition and simulation.Different categorizations of games have emerged from studies conducted in the field, including those by game designers and websites with similar content. However, currently there is no universally accepted classification system for the whole of video games. [1]

By the time of this thesis video games can be categorized in the following:

1 **Action Games:** are characterized by fast-paced, physically challenging gameplay where the player is at the center of the action. These types of games are often easy to start playing and continue to be some of the most popular video games. Examples include early games like Donkey Kong and Galaga.

2 **Adventure Games:** are defined by the type of gameplay rather than the story or setting. While advancements in technology have allowed for more creative storytelling,

the core mechanics of adventure games have remained relatively unchanged from their text-based origins. These games typically involve players solving puzzles and gathering clues to advance the plot or gameplay. Starting back in the eighties with Colossal Cave Adventure and then followed games such as Monkey Island, Myst, King' Quest and Zork.

3 **Fighting Games:** focus the action on combat, and in most cases, hand-to-hand combat. Most fighting games feature a stable of playable characters, each one specializing in their own unique abilities or fighting style. In most traditional fighting games, players fight their way to the top, taking on more and more difficult opponents as they progress. Some popular titles are Mortal Kombat, Tekken and Street Fighter II.

4 **Platformer games:** also known as platform games, get their name from the gameplay mechanics in which the player's character interacts with platforms throughout the game, usually by running, jumping, or falling. These games typically require the player to maneuver their character across platforms to reach a goal, while confronting enemies and avoiding obstacles along the way. These games are presented in a side view using two-dimensional movement. Platforming gameplay tends to be very dynamic and challenges a player's reflexes, timing, and dexterity with controls. A good example of such games is Super Mario Bros and The Prince of Persia.

5 **Survival Games:** challenge players to navigate and survive in harsh, inhospitable environments with limited resources. The player must strategize, gather resources, and fend off threats in order to survive. These games often blend elements of strategy, action, and role-playing. This genre has seen a rise in popularity in recent years, with games like Rust, DayZ, The Forest gaining recognition as legitimate titles.

6 **Survival Horror Games:** focus on creating a sense of vulnerability and fear for the player, often by placing them in an actively hostile environment with limited resources. These games often require the player to solve puzzles and avoid enemies, rather than engage in combat as in most typical Survival Games. The genre is known for its use of eldritch monsters and isolation as a theme. Examples of popular survival horror games include Resident Evil, Silent Hill and Alone in the Dark.

7 **Stealth Games:** focus on scheming and careful execution to complete objectives, and

while they may have elements of other genres such as first-person or third-person shooters, the core gameplay revolves around sneaking and avoiding detection. Examples of this include the popular franchise Metal Gear, Thief and Dishonored which emphasizes covert actions and maneuvers.

8 **Interactive Movies:** are similar to traditional movies in that they feature animated or live-action footage, with a focus on a main storyline. However, the player is given the opportunity to make choices and take actions that can alter the course of the story, leading to different scenes and outcomes. These choices can lead to "game over" scenarios or alternate paths in the story. Dragon's Lair is the most popular game of this genre.

9 **Role-Playing Games (RPG):** are a popular game genre in which the player takes on the role of a character in a fantastical setting, allowing them to tailor their journey through the game in unique ways. The genre is also characterized by choices that influence the final outcome of the game and often have alternate endings. With origins in tabletop gaming like Dungeons & Dragons, most RPG games feature medieval or fantasy settings, but there are also sci-fi fantasy-themed games such as Mass Effect, Fallout, and Final Fantasy. Cultural differences have also had an impact on the genre, which can be categorized as Western-influenced (WRPGs) or Japanese-influenced (JPRGs).

10 **Massively Multiplayer Online Role-Playing Game (MMORPG):** These types of games are mostly played online or on platforms that have network capabilities. They also offer different game modes where players can work together or compete against each other. Also one of the most popular game genres out there has found success in the names of: World of Warcraft(WoW), Ever Quest, Lineage, Guild Wars and Star Wars The Old Republic.

11 **Tactical Role-Playing Games (TRPG):** feature gameplay that heavily emphasizes tactics. These games are typically modeled after classic tabletop role-playing games, where players and enemies are arranged on a grid and use unique skills to navigate the environment through turn-based rolls and moves. Due to their origins in tabletop gaming, they often have similarities to traditional board games, with turn-based gameplay taking place on an isometric grid. Players must use careful strategy and manage limited resources, such as armies and weapons, to overcome battles and enemies. Games like Banner Saga and Valkyria Chronicles are all such examples.

12 **Sandbox RPGs:** are games where players are given the freedom to explore the game's environment and discover new adventures. These games are highly immersive and engaging, as they feature a vast number of characters and scenarios that allow developers to create realistic virtual worlds. Witcher, Red Dead Redemption 2 and No Man's Sky are some of the most well known.

13 **Strategy Games:** focus on the player's ability to plan, make decisions, and manage resources in order to achieve a specific goal or set of goals. This can include games that focus on military strategy, city-building, resource management, and more. These games often require players to think critically and make strategic decisions in order to succeed. Examples of strategy games include Total War, Civilization, Age of Empires, and Europa Universalis.

14 **Multiplayer Online Battle Arena (MOBA):** This genre combines elements of action, role-playing, and strategy games and typically involves players controlling a single character on one of two teams, working together to defeat the opposing team by destroying their base. The gameplay often includes the use of computer-controlled units that attack on predetermined paths to assist players in this task. Dota, Dota2 and League of Legends are the biggest names of this genre.

15 **Tower Defense Games:** are intense and challenging, as players must defend their base from waves of enemy units. Successful players must be able to adapt and make quick decisions to minimize damage to their base and maximize their efficiency in defending it. Plants vs Zombies, Kingdom Rush and Bloons TD are some examples.

16 **Educational Games:** emphasize in learning something. They try to teach the user and are usually geared towards younger players. There are numerous educational games and each one aims to improve reading, math skills, and study habits, as well as spark interest in different subjects. Most popular games of the category are Oregon Trail and Pictionary.

17 **Puzzle Games:** also known as logic games, involve solving problems or navigating challenges on a single screen or playfield in order to progress. This game genre sometimes overlaps with educational games and adventure games. These games include brain training games like Brain Age as well as more casual puzzle games such as Tetris

and Minesweeper.

Trivia Games: are often characterized by the need to answer questions within a certain time limit or before another player does. Players are tested on their knowledge of general or specific subjects in order to score points and win the game. Trivia games are gaining popularity due to their casual nature, which makes them ideal for mobile devices. Examples of popular trivia games include Trivia Crack, Trivial Pursuit, and HQ.

Board Games: provide entertaining gameplay for players of all ages have stood the test of time,. These types of games are often played in social settings and involve multiple players competing against each other or working together towards a shared objective. Games like Cards Against Humanity, Risk and Monopoly are some of the many board games

Sports Games: are a genre of video games that simulate traditional sports such as football, golf, hockey, basketball, and tennis. Some of these games focus on replicating the actual gameplay of the sport, while others focus on the strategic elements. Despite being one of the earliest genres in video games, sports games continue to be popular and competitive today. They also have various sub-genres, as team sport and combat sports, that highlight specific sports or activities with some even being considered realistic simulations. Most popular series of such games are NBA2K, FIFA, Pro evolution soccer and UFC Undisputed.

Racing Games: are a genre of video games that involve players participating in a racing competition, whether it is based on real-world racing leagues or fantastical settings, they challenge players to compete against each other or the clock, while driving cars that can be realistic, modified, or standard. Racing games have evolved from their arcade origins to become immersive and competitive simulations.Examples of such games are Gran Turismo, Need for Speed and Mario Kart.

Shooter Games: involve players utilizing weapons to participate in gameplay, with the objective typically being to eliminate opponents or enemy characters. They can further be categorized by the player perspective, which totally changes the game experience, in two groups:

- **First Person Shooters (FPS):** is a type of video game that is played from the perspective of the main character, giving the player an immersive experience as if they are physically in the game. The gameplay usually involves using weapons to engage in combat and eliminate enemies or other players. The player's movements are typically mapped to the controller and the view on screen shows what the protagonist would see in real life. FPS games often include realistic sounds such as breathing and footsteps to enhance the immersion. Counter-Strike, Call of Duty, Doom and Half-Life are good examples.
- **Third Person Shooters (TPS):** centers around shooting and allows the player to see the character on-screen from a third-person perspective. Unlike other types of shooter games, such as shoot 'em ups, the player's avatar is the main focus of the camera view. Third-person shooters provide a similar level of immersion as first-person shooters, but the camera is positioned behind and slightly above the character, rather than at their eyes.Such examples of games who fall int his category are Grand Theft Auto, Fortnite and Splatoon.

23 **Rhythm Games:** are characterized by their focus on challenging a player's sense of rhythm through gameplay that simulates the performance of music, such as dancing or playing instruments. These games typically involve players pressing buttons in a specific sequence as shown on screen, and often feature multiplayer modes for competition or cooperation in simulating a musical ensemble. Examples include Dance Dance Revolution, Guitar Hero and Rock Band.

24 **Exercise Games:** are a type of video game that use specialized controllers or peripherals to mimic physical activities and exercises. They often include features like fitness tracking and progress monitoring, as seen in games like Wii Fit. These games aim to promote healthy lifestyle choices and physical activity through interactive gameplay, and have become increasingly popular as a tool for weight loss and overall wellness. They bring a new dimension to video games by providing players with benefits beyond entertainment such as improving physical and mental well-being.

25 **Serious Games and Simulation Games:** are a new concept of video games that aims to teach real-world concepts through titles that are all about immersing the player in the action of simulating a specific activity as realistically as possible. They range from

city-building games to life simulations.A sense of realism is emphasized, giving the player the sensation of actually performing. City-building has the player construct cities from the ground up. Cities: Skylines and SimCity are great examples of this genre.Life Sim games allow players to simulate life. Namely, The Sims has the player building both a family unit and a house for them to live. They are then tasked with controlling everything from job attendance to bathroom breaks. You are there every step of the way along with the characters' lives.

2.3 Serious Games

Serious games are digital games that are used for purposes other than entertainment, such as training simulations, social-criticism games, and adver-games, which are games created to promote a product, a service or a company. The term serious games was first coined in 1970 by Clark Abt in his book "Serious Games", but the idea of using games for non-entertainment purposes has been around since the publication of the first commercial digital games in the 1980s. However, it wasn't until the early 2000s that serious games began to gain traction within the academic community. [2]

The concept of using games for educational purposes has been around for decades, but it has gained more traction in recent years as research has shown the potential positive impact of digital games on learning. While serious games were originally used mainly for specialized training in fields such as medicine or the military, they are now being used in a wide range of industries and disciplines, such as language learning and engineering. [3]

The historical development of serious games can be traced back to the early 2000s, when advancements in processing power and memory formats allowed for the creation of realistic 3D renderings and immersive gaming experiences. The increase in realism and interactivity enabled groups such as the United States Army to release America's Army in 2002 for recruitment and marketing purposes. This led to the coining of the term "Computerized Serious Games" by Zyda in 2005. The release of America's Army and the introduction of the Serious Games Initiative by the Woodrow Wilson Center sparked interest in the use of games for non-entertainment purposes. This marked the beginning of the use of serious games in various fields including education and military research. [4]

As the concept of serious games has matured, the term has become more specific, refer-

ring to games designed to run on personal computers or video game consoles. Assessment models have been developed for integration into serious games, increasing their usability as classroom tools. Overall, serious games are designed to achieve specific learning outcomes through content, cognitive change, and skill-based growth, rather than just isolated skills alone. [3]

Serious games have come a long way from their origins in specialized domains such as medicine or military to being used in a variety of fields including language learning, engineering, and the workplace. While the use of video games as learning tools is becoming more prevalent, it is important to understand how video games affect the learning process. With this deeper understanding, it's possible to categorize serious games in various ways, such as by their purpose, their market or the audience.

2.4 Serious Games Categories

It is clear that Serious games can be categorized based on their purpose and market, but these classification systems have their limitations. They do not provide relevant information about the structure of the games, and none of them classify "Serious Games" as "games". So a new classification system was created and it's called G/P/S (genre/purpose/scope) for identifying and categorizing serious games. The system is based on three criteria: genre, which refers to the type of game (e.g. game-based, simulation-based); purpose, which refers to the intended outcome or objective of the game (e.g. educational, informative); and scope, which refers to the market or audience for the game (e.g. education, healthcare). [5]

More specifically we can classify serious games into the following categories:

1 **Simulations or Simulation Games:** are used to acquire or exercise skills.

2 **Edutainment:** games that combine education and entertainment.

3 **Advergames:** games with the purpose of advertising.

4 **Games-Based Learning or "Game Learning":** is an approach to teaching that utilizes the engaging elements of games, such as excitement, instant feedback, and competition, to promote and facilitate learning.

5 **Edumarket Games:** a combination of Advergames and Edutainment games.

6 **Newsgames:** news-related games, referring to recent events or writing of editorial comments.

7 **Games for Health:** are games that are used for mental or physical therapy, or for the prevention of diseases.

8 **Activism or Art Games:** are used for the expression of artistic ideas.

9 **Persuasive Games:** games that try to change the attitude or behavior of users through persuasion or other social influences.

10 **Organizational-dynamic games:** they aim at personal development of players and building their character and mainly refer to dealing with complex organizational situations.

Chapter 3

Artificial Intelligence and Games

If you have ever played a video game, you have interacted with artificial intelligence (AI). Regardless of whether you prefer race-car games like Need for Speed, strategy games like Civilization, or shooting games like Counter Strike, you will always find elements controlled by AI. AIs are often behind the characters you typically don't pay much attention to, such as enemy creeps, neutral merchants, or even animals. [6]

3.1 Introduction to Game Artificial Intelligence

3.1.1 What is Intelligence

The word intelligence is a nebulous term with multiple interpretations. The dictionary defines it as the capacity to acquire and apply knowledge, but this is a broad definition that can be interpreted in many ways. Some interpret it literally and suggest that even a thermostat can be considered intelligent, as it acquires knowledge and applies it. Others suggest that intelligence encompasses the faculty of thought and reason, but this raises more questions about the meaning of these terms. The true definition of intelligence is the subject of ongoing debate and is not fully understood. However, when it comes to game systems, intelligence is defined as the ability to acquire knowledge about the world and act on it in an effective way, through building complex plans and making decisions based on game balance and design. [7]

3.1.2 What is Game AI

Game AI refers to the code in a game that controls the computer-controlled elements and allows them to make decisions that appear smart. Instead of attempting to comprehend how

the system arrived at its judgments, game AI focuses on developing efficient and practical behaviors. Game AI is results-oriented and largely focused on how the system behaves, rather than how it thinks, in contrast to the broader subject of AI. As long as the game is enjoyable and captivating, players are often not interested about the specific techniques utilized by the AI. [7]

However, game developers also use the term AI in other ways. Some people refer to the behavioral mechanics of the game as AI, even if it uses the same mechanism as the human players. Animation selection, which determines how the decision will be performed visually, is also sometimes considered AI. Additionally, the algorithms that govern movement and collision can sometimes fall under this label, especially in games with simple AI requirements. [7]

It's important to note that, while some people may think of animation selection or movement mechanics as AI, they are actually lower level decision making and not the intelligence we are talking about. The proof of AI in games is in the pudding as far as game AI goes, how the system acts, not how it thinks. [7]

3.2 Historic View of Game AI

The history of artificial intelligence in video games dates back to the mid sixties in a game named Nim, which is a mathematical game of strategy in which two players take turns removing objects from distinct heaps or piles. Each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap or pile. Depending on the version being played, the goal of the game is either to avoid taking the last object or to take the last object. [8]

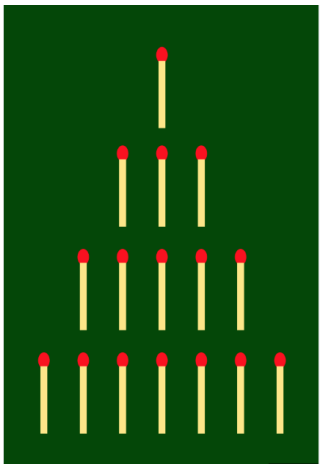

Figure 3.1: Nim Game [8]

Before that, games were either two-player, meaning that there was no computer opponent, or any non-human objects were hard-coded. An example of hard-coded video game objects are the little aliens in Space Invaders that swoop down at the human player. [9]

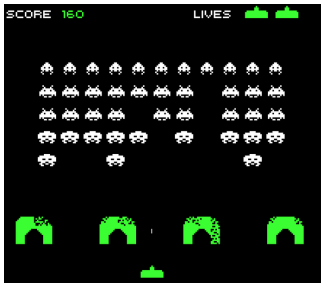

Figure 3.2: Space Invaders

One of the first examples of real AI in gaming was the computer opponent in a game named Pong. The computer paddle would do its best to block the ball from scoring by hitting it back at the player. [9]

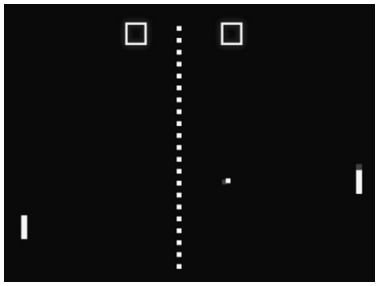

Figure 3.3: Pong Game

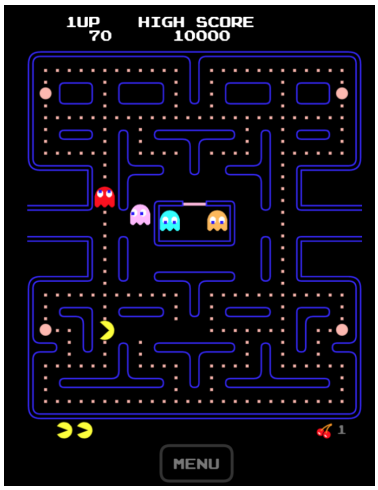

Figure 3.4: Pac Man

Figure 3.5: Black & White

After that point, games that use sophisticated AI techniques like genetic algorithms and neural networks started to prosper. In the 1980s, games like Pac-Man introduced AI patterns to its maze and fighting games started using AI technology. In the 1990s, formal AI tools like finite state machines (FSM) were used for new video game genres like real-time strategy games for path-finding problems, making real-time decisions and economic planning. As AI became more sophisticated, interactive multiple-way dialogues in games like Façade and bottom-up AI methods in games like Creatures and Black & White were used . The use of AI in sports games also became more prevalent in the 1980s and 1990s, enabling players to set variables in the AI to create player-defined coaching or managerial strategies. [10]

Video games have a long history of utilizing AI. AI researchers initially concentrated on developing computing systems that could resemble human intelligence in some ways and attain decision-making or problem-solving capacities comparable to those of humans. Early AI experiments were frequently focused on games because they provided a formal, highly constrained, yet complicated decision-making environment. In such early works, NPC behavior in games was frequently controlled by rule-based or decision-tree AI. [11]

AI research and game AI implementation both advance over time. AI is employed in a wide variety of gaming genres nowadays, including strategy and role-playing games as well as racing and shooting games. These AI-controlled NPCs are becoming more and more intelligent, with the capacity to anticipate player actions, respond in real-time, and even pick

up on player habits. More intelligent and adaptable NPCs have resulted from the fusion of AI with video games, but the user experience has also been enhanced. Games have also been significant in furthering AI research since they offer a natural training environment for assessing the effectiveness of various AI systems.

The integration of AI to video games has improved player immersion and engagement while also creating new opportunities for game design and player interactions. Deep learning and reinforcement learning, two recent developments in AI, have further pushed the limits of what is achievable in video games. Deep Q-learning is currently being used by simulation platforms like ViZDoom and the Arcade Learning Environment (ALE) to train AI agents to superhuman levels using only visual data from the screen. This has made it easier to create and test various Machine Learning (ML) algorithms in settings with observable characteristics and unique rule sets. [12]

Video games offer a natural training environment for assessing the effectiveness of various ML algorithms because to their specified rules of interaction, NPCs, and explicitly stated goals and objectives. As a result, the incorporation of AI into video games has improved the gaming experience while simultaneously serving as a crucial tool for AI research and development. [12]

3.3 Importance of Artificial Intelligence in Games

In the past few years, the gaming industry has made incredible strides, with the incorporation of AI playing a crucial role in this development. As we've already covered, the use of AI to games has improved a variety of elements by making them more engaging, responsive, and adaptive. In this chapter, we'll go more deeply into the significance of AI in gaming and examine how it might provide a wealth of additional advantages and creative fixes that might catapult the gaming sector to new heights.

3.3.1 Benefits of Artificial Intelligence in Games

Creating smarter and more adaptive NPCs is one of the areas where AI can be quite useful. Neural network technology enables NPCs to customize experiences for each player in order to enhance their enjoyment, challenge, and participation in the game. The creation of AI-enabled NPCs that are taught by emulating professional gamers is already a primary

priority for many gaming businesses. [13, 14]

AI also offers a significant deal of promise to improve the simulation performance in online games, improve the aesthetics, and make the games appear and feel more authentic and genuine. Artificial intelligence (AI) brings up even more intriguing methods to increase the immersion and interactivity of video games with the incorporation of virtual and augmented reality technology. For instance, AI up-scaling is a useful feature to enhance the graphics of online games and create representations of real-world objects. [13, 14]

Another area where AI is being used in gaming is in the detection and prevention of cheating in multiplayer games. Many video game companies use AI to analyze the patterns of player movements and keys to detect if a user is cheating or not, while cheaters use AI to cheat in a realistic way similar to humans to avoid getting detected. [13, 14]

AI can also be very important in the areas of player behavior analysis and game customisation. Providing appropriate assistance to game creators generate more individualized gaming experiences for every player by evaluating data on player preferences, gaming behaviors, and in-game actions. This can entail adjusting game difficulty settings, making unique game recommendations, or even making in-game adverts relevant to the player's preferences. [13, 14]

Finally, AI can be applied to player behavior analysis, which can assist game producers in better understanding how players engage with their products and pinpoint areas for improvement. Analyzing player engagement levels, detecting locations where players frequently lose interest, and spotting player behavior patterns that can be leveraged to enhance the overall gaming experience are some examples of this. [13, 14]

In conclusion, the integration of AI in gaming has already begun to revolutionize the industry, providing new and innovative solutions that can enhance the gaming experience for players and help game developers create more immersive and engaging games. As AI technology continues to evolve, we can expect to see even more exciting developments in the future that will take gaming to new heights.

3.3.2 Artificial Intelligence in Serious Games

Serious games are becoming an increasingly popular method of education and training, as they offer an engaging and interactive way to acquire knowledge and skills. Artificial Intelligence plays a crucial role in serious gaming, as it allows for the creation of more realistic and personalized experiences for players. In this chapter, we will explore the various ways in which AI is used in serious games, and why it is important for the field of serious gaming.

- **Emotion Recognition**

  Since emotions have an impact on memory and behavior, they have a considerable influence on the learning process. Any instructor would consider the students' emotional moods while teaching. But, because it was difficult, if not impossible to detect, the student's mood has been routinely ignored as a learner model variable in computer-based learning. With the development of emotion detection technology, learner models in serious games can now incorporate the learners' emotions, enhancing the quality of individualized instruction and feedback. Also, in games for communication training, conflict management, or acting training, the participants' emotions can be incorporated into the learning material or game situation. Finally, during play-testing, emotion data can be gathered using emotion recognition. [15]

- **Game Balancing**

  The practice of game balance is largely responsible for games' ability to be engrossing. A well-balanced serious game increases and maintains learner motivation, which is a key factor of learning, by preventing boredom and frustration and providing manageable obstacles. According to Vygotsky's theory of the Zone of Proximal Development, published in 1978, a real-time adaption of the game's complexity facilitates a more seamless learning process. The AI program regulates difficulty so that the player is pushed to develop new skills or get new knowledge without encountering activities that are unduly challenging or beyond their level of expertise. Because it continuously changes task difficulty to the proper degree and iteratively reassesses the player's skill proficiency, it results in an optimized learning curve.As a result, learning becomes extremely efficient: advancement is maximized, and no time is lost on activities that don't advance learning. The player or the teacher can utilize ongoing evaluation of the player's performance as a type of formative learning analytics to track learning progress

and spot potential learning obstacles. The study and optimization of the game's educational material are also made possible by the game difficulty assessment. [15]

- **Stealth Assessment**

  Games are expressly suited for the acquisition of highly contextualized, tacit knowledge and action-bound skills, which are notably hard to capture in formal tests and exams. Cases in point would be social skills, communication skills, group moderation skills, but also competencies such as persistence, creativity, self-efficacy, teamwork and the wider collection of twenty-first century skills, all of which are deemed essential for successful future careers and presupposing a strong link with concrete action. Given this tacit knowledge dimension, the assessments should not be administered solely as separate oral or written assignments, but instead should be directly based on the activities displayed. Stealth assessment provides an attractive alternative to the existing de-contextualized assessment methods by linking the assessment directly to the practical use of knowledge and skills in relevant situations. Moreover, these situations should entail scenarios that require the application of various competencies at the same time. This is exactly what serious games are capable of providing. [15]

- **Sentiment Analysis**

  In the context of serious games, sentiment analysis can be used, for instance, in dialogues, commonly available either in multi-player communication or in discussions with a virtual character. The arising insights about how people feel and interact during these interactions can then be fed back to the game for further usage in the game scenario, or can be provided as feedback for the game development team. Alternatively, sentiments can be extracted from written free text or spoken assignments in the game, such as reports, pitches or answers to open-ended questions. Sentiment analysis can be a powerful tool in serious games as it allows for a deeper understanding of players' emotions and interactions during gameplay, which can be used to improve the game's design, learning content and personalization. [15]

In conclusion, AI plays a crucial role in serious gaming, as it allows for the creation of more realistic and personalized experiences for players. By including the learners' emotions as a variable in the game, game balancing, stealth assessment and sentiment analysis, serious games can provide more accurate and effective formative assessment and analytics and

optimize the learning experience. It allows for the inclusion of a wider range of skills and knowledge, including social, communication and teamwork skills, and it allows for a more efficient and effective way of assessing learning progress. With the continuous advancement in AI technology, the potential for serious games to provide even more realistic and effective learning experiences is limitless.

3.4 Techniques and Algorithms in Game AI

In the previous section, we discussed the value of AI not only in gaming but in serious games as well. In this part, we will take a closer look at the techniques and algorithms used to implement AI in games.

AI techniques can be broadly categorized into three main types: rule-based, decision-tree, and machine learning-based AI.

3.4.1 Rule-Based Model

A rule-based AI system is a model that is solely based on predetermined rules. It is composed of a set of human-coded rules that result in predefined outcomes. These systems operate on the simple yet effective "cause and effect" methodology. They are deterministic by nature, meaning they can only perform the tasks and functions they have been programmed for and nothing else. They require very basic data and information in order to operate successfully. [16]

In the field of gaming, rule-based AI models are commonly used to control the behavior of NPCs and provide a basic level of intelligence to the game. These models operate on a set of fixed rules and use simple "if-then" statements to determine the actions of NPCs based on the player's actions and the game's environment. They can be used in a wide range of game genres, from racing and shooting games to strategy and RPGs. [16]

We can also consider Utility-theory-based models and game-theory-based models as Subcategories of the ruled-based model, as they also use predetermined rules and decision-making processes.

- **Utility-theory-based models** are a subcategory of rule-based models that evaluate the feasibility of NPC actions based on the utility values of those actions. For example, a utility-based model may assign a higher utility value to an NPC taking an action that is

considered safer or more beneficial to the player, and a lower utility value to an action that is considered more risky or detrimental. [17]

- **Game-theory-based models**, on the other hand, take into account the decisions of multiple NPCs in a competitive situation. These models incorporate the actions of NPCs in the game as if they are players in a strategic game, with each NPC's decisions depending on the actions of the other NPCs. This type of model can better capture the dynamic interactions between NPCs in the game, and often have stronger interpretability and universality compared to other AI models. [17]

In addition, rule-based AI can also be used in educational games, such as math games, to provide in-context rules and dynamics to attain specific learning goals. These serious rule-based games support and encourage meaningful learning by sharpening critical basic skills such as arithmetic. [18]

Overall, rule-based AI models are deterministic by nature, meaning they can only perform the tasks and functions they have been programmed for and require very basic data and information to operate successfully. They can be quickly improved by adding, deleting, or updating rules based on domain expert information or recency.

3.4.2 Decision-Tree Model

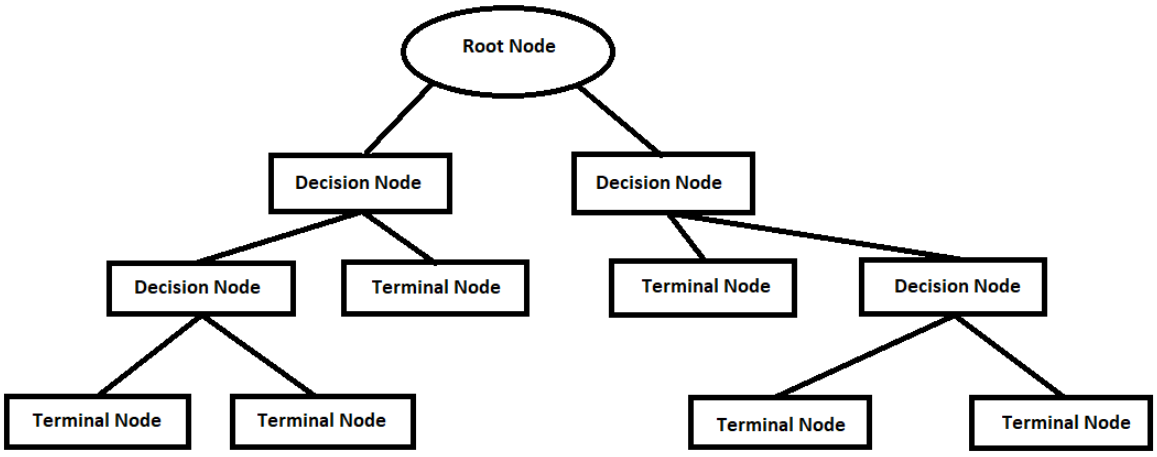

Figure 3.6: Sample of a Decision Tree

Decision tree AI models are commonly used in video games and serious games to provide players with a range of choices and predict the outcome of those choices. These games,

usually referred to as training or instructional games, are made to teach players certain skills or give them a learning experience. A branching structure called a decision tree enables players to make decisions that result in distinct and distinct results. By selecting the options they believe to be the finest, players can so design their own narrative or result. [19]

Decision trees are frequently employed in narrative-based games, allowing players to design their own plot or resolution by selecting the possibilities they believe to be most appropriate. As in the video game Star Wars Jedi: Fallen Order, where the player is given hints about the protagonist's past and future based on specific circumstances, these choice trees might be present in the game but the player may not be aware of them. Decision tree models are employed in games like The Witcher 3: Wild Hunt to create various endings based on the player's choices. [20]

In serious games, these models are used to analyze process data from simulation and game-based assessments to classify examinees' behavior patterns so as to determine their problem-solving strategies in order to support and encourage meaningful learning. For example, a serious game developed as a math game can use decision trees to help students learn about arithmetic and sharpen critical basic skills such as adding, subtracting, multiplying, and dividing. [19]

Even when players are not aware of decision tree models, they are still making decisions based on them. For example, when playing a video game, players may subconsciously make decisions that they believe are beneficial, such as attacking the enemy. This is an example of following a decision tree, as players weigh the pros and cons of certain actions without even thinking. [20]

Overall, decision tree AI models are an unavoidable aspect of video games and serious games, as they provide players with a range of choices and predict the outcome of those choices. As technology advances, decision tree models are becoming increasingly popular and practical, and they will likely be incorporated even further into video games and other industries in the future.

3.4.3 Machine Learning AI Model

Machine learning-based AI, on the other hand, uses algorithms to learn from the player's actions and adapt to their behavior. This technique is the most advanced and powerful of the three, but it also requires more resources and computation power.

Machine Learning (ML) is a subset of Artificial Intelligence analyzes data or a state and produces a taught response by using different training methods. The initiative of OpenAI, made it possible to encourage research across academia and the industry to exchange ideas and research on AI and ML, has contributed to the growing popularity of ML in gaming. As a result, new concepts, techniques, and study topics have grown exponentially. Developers may now create agents that can learn from their surroundings and even outperform their human creators. [21, 22]

The capability of ML to learn and adapt is one of its main features. ML algorithms may learn directly from data, in contrast to conventional AI techniques that rely on predetermined rules and models. This enables them to enhance their performance as more data become accessible over time. For instance, deep learning is a kind of machine learning that makes use of neural networks to mimic human learning. [21]

ML-based models are being used in the gaming business mostly for the purpose to make game characters and landscapes more realistic and adaptable. For instance, game designers can employ ML algorithms to train non-player characters (NPCs) to respond to player actions in a more convincing and realistic manner. This may result in more dynamic and interesting gaming experiences. Also, by improving resource utilization and lowering load times, ML can be utilized to enhance the functionality of game engines. [22]

Another area where ML has been making a big impact is in the realm of game development tools. A variety of ML-based tools have been developed to help game developers create better and more efficient games. For example, ML-based level generation tools can automatically create new levels that are both challenging and fun to play. Similarly, ML-based animation tools can help developers create more realistic and believable character animations. [22]

ML can be utilized in the development of realistic and adaptable game characters and environments for serious games. Providing multiple benefits, as its capacity for learning and adaptation, which enables it to enhance performance over time as new data become available. Also, a player's problem-solving techniques or the success of the game can be established by evaluating and categorizing player behavior. This can help to enhance the design of the game and to comprehend how players learn. A better and more individualized gaming experience can be achieved by using ML to create adaptive game scenarios that can adapt to the player's skill level. It can also be used to solve the problem of creating NPC behavior in complex scenarios and to facilitate the creation and adaptation of AI in serious games. [23]

Machine Learning is so aptly named because it uses various forms of training to analyze data or state and provide that trained response. These types are worth mentioning and we will focus on one particular method of learning that is being implemented in my application. Before we get to that though, let's breakdown the types of training we frequently see in ML.

3.5 Machine Learning Models

Many cutting-edge applications of artificial intelligence, such natural language processing and image identification, are built upon machine learning models. These models are developed by the training of machine learning algorithms on massive data sets, allowing the program to identify patterns and draw conclusions from data that had not previously been considered. Model training is the process of teaching a machine learning algorithm, and a machine learning model is the resulting program with specified rules and data structures. Depending on the training technique utilized, machine learning algorithms can be divided into four broad types, each having its own benefits and drawbacks. [24]

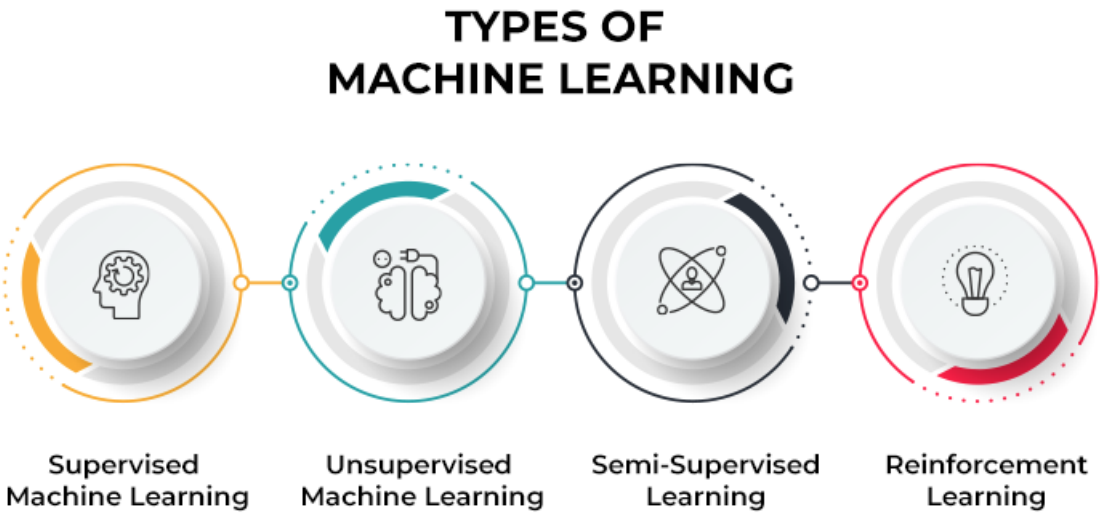

Figure 3.7: Machine Learning Types [21]

1 Supervised learning is a typical training method of machine learning. The goal of supervised learning is to predict the output based on the input data, using a set of labeled data to train the algorithm. This method of training is widely used in data science ML methods and is typically used for prediction or classification tasks. The algorithm is given a small dataset to learn from, which gives it a basic understanding of the problem and the

data points it needs to work with. This training dataset is similar in characteristics to the final dataset and provides the algorithm with the necessary labeled parameters for the problem. The algorithm then discovers the relationships between the parameters, establishing a cause and effect relationship among the variables in the dataset. After training, the algorithm understands how the data works and the relationship between the input and output. The solution is then applied to the final dataset, where it continues to improve and learn from new data. [25]

For problems like binary classification, multi-class classification, regression modeling, and ensembling, supervised learning approach works efficiently. They can be used to categorize data into two groups, select among many options, forecast continuous values, and aggregate the forecasts of various machine learning models to come up with an accurate prediction, in that order. [26]

Popular techniques for supervised learning include Navies Bayes, K-nearest neighbors, Ensemble learning, Random Forest, Linear regression, Support vector regression, etc. These techniques can be used to solve various supervised learning tasks, depending on the nature of the given data in a particular problem domain. [16]

For instance, supervised machine learning is widely deployed in image recognition, an example of this is the Google Inception, which is an image classification ML model that is trained by millions of images into various classifications. Supervised machine learning is also used in predicting demographics such as population growth or health metrics, utilizing a technique called regression. [22]

2 Unsupervised learning is a type of machine learning that does not rely on labeled data. Instead, the algorithm is trained using an unlabeled dataset and is enabled to predict the output without any supervision. The goal of unsupervised learning is to group the unsorted dataset based on the input's similarities, differences, and patterns. [25]

For example, an unsupervised learning algorithm could be used to group images of a fruit-filled container into categories based on the objects' color, shape, or other differences seen in the input images. The machine learning model would then be able to predict the output when tested with a test dataset. [21]

Unsupervised machine learning is further classified into two types: Clustering and Association. Clustering refers to grouping objects into clusters based on parameters such

as similarities or differences between objects. Examples of clustering algorithms include the K-Means Clustering Algorithm, Mean-Shift Algorithm, Principal Component Analysis, and Independent Component Analysis. Association learning refers to identifying typical relations between the variables of a large dataset. Popular algorithms for association rules include the Apriori Algorithm, Eclat Algorithm, and FP-Growth Algorithm. [16]

Unsupervised learning is often referred to as a "data-driven method" where the primary goal is to uncover patterns, structures, or knowledge from unlabeled data. Clustering, visualization, dimensionality reduction, finding association rules, and anomaly detection are some of the most common. [26]

3 Semi-supervised learning is a machine learning approach that can be regarded as a hybridization of both supervised and unsupervised learning, as it uses both labeled and unlabeled data to train a model. In contrast to supervised learning, which demands that all data be labeled, and unsupervised learning, which only uses unlabeled data, semi-supervised learning reaches a middle ground by combining the advantages of both approaches. Semi-supervised learning's objective is to create algorithms that take use of the interaction between labeled and unlabeled data to understand how this combination may alter learning behavior. [21, 16]

One of the key benefits of semi-supervised learning is that it can supplement supervised learning tasks when labeled data is hard to come by or prohibitively expensive. Additionally, given that the majority of the input is unlabeled, semi-supervised learning has the potential to be a quantitative tool for understanding human category learning. Self-training, mixture models, co-training, multiview learning, graph-based techniques, and semi-supervised support vector machines are a few of the more well-known semi-supervised learning models. The success of semi-supervised learning heavily depends on the underlying assumptions of each model, each of which has its own mathematical formulation and set of presuppositions. [27, 28]

Semi-supervised learning can be used in various applications such as machine translation, fraud detection, and data labeling. In machine translation, for example, semi-supervised learning can be used to teach algorithms to translate language based on less than a full dictionary of words. In fraud detection, it can be used to identify cases of

fraud when there are only a few positive examples. And in data labeling, algorithms can train on labeled data to automatically label new data without human interaction. [26]

4 Reinforcement learning is a machine learning technique that is inspired by how humans learn from their experiences. It involves a self-improving algorithm that uses trial and error to learn from new circumstances. The algorithm in this process is rewarded for positive outcomes and penalized for negative outcomes.

Reinforcement learning operates by putting the algorithm in a setting with an interpreter and a reward system, which is based on the psychological idea of conditioning. The output result from each iteration is delivered to the interpreter, who determines whether the result was favorable or not. The interpreter promotes the solution by rewarding the algorithm when the program gets the right answer. If the result is unfavorable, the algorithm must repeat the process until a better conclusion is obtained. [25]

Common applications of reinforcement learning include resource management, video games, and robotics. The AI component evaluates its environment, takes action, gains knowledge from its mistakes, and enhances performance through a feedback-based approach. An environment-driven approach, in which decisions are made using a reward function and the environment is often modeled as a Markov decision process, enables a reinforcement learning agent to see and comprehend its environment, take actions, and learn through trial and error. This is what Google DeepMind accomplished with its well-known AlphaGo program, which defeated the greatest Go player in history and achieved a feat at the time that was thought to be impossible. [22, 26]

Unlike supervised learning, reinforcement learning does not require labeled data and agents learn only through experiences. Some common reinforcement learning algorithms include Monte Carlo learning, Q-learning, and Deep Q Networks. Applications of reinforcement learning include trajectory optimization, motion planning, dynamic pathing, and scenario-based learning policies for highways. [16]

3.5.1 Deep Learning

As we have seen above, the four main ML models each have their own unique characteristics and applications. However, with the advancements in technology and the availability of large amounts of data, we are now able to delve even deeper into the realm of machine learning with the introduction of deep learning. With the ability to process large amounts of data and perform abstract mathematical calculations, deep learning has the potential to revolutionize the way we interact with technology and solve problems that were previously thought impossible.

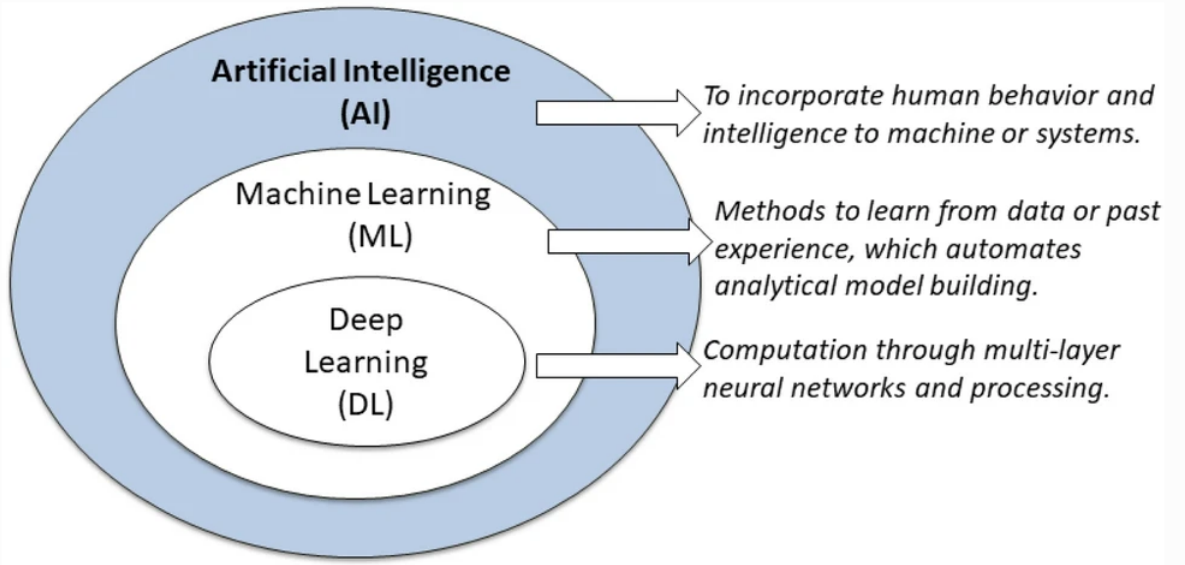

Figure 3.8: Deep Learning's place in Artificial Intelligence [16]

Deep learning is a subset of machine learning that is based on artificial neural networks, designed to imitate how humans think and learn. These networks, comprising many layers, drive deep learning and are known as Deep Neural Networks (DNNs). DNNs are able to perform complex operations such as representation and abstraction that make sense of images, sound, and text. [29]

Deep learning has recently found use in the gaming sector as well, where it is being used to produce more immersive and realistic game experiences. In the field of game AI, where it is utilized to build more perceptive and adaptable NPCs, deep learning is one of the most significant uses in games. Deep learning is being used by game developers to teach NPCs to respond to player events and make decisions more realistically. As a result, gaming can be more dynamic and interesting. [30]

Moreover, deep learning is being used to improve the graphics and visual effects in games,

making them more lifelike and immersive. With the use of deep learning, game developers can create more realistic lighting and shading effects, as well as more detailed and accurate textures. This improves the overall gaming experience and makes it more engaging for players.

In addition to traditional video games, deep learning is also being applied in the field of serious games. These games can be used for training, education, or even therapy. For example, deep learning has been used to develop a game that helps children with autism improve their social skills. [30]

3.6 Model of this thesis

In this thesis, a combination of both Reinforcement Learning, and Deep Learning, which is known as Deep Reinforcement Learning(DRL), will be utilized to train an AI agent to cope with a serious game scenario. By using DRL, the AI agent will be able to adapt to different scenarios within the game and learn from its mistakes, making it a powerful tool for training purposes.

An enhanced kind of reinforcement learning called deep reinforcement learning makes use of deep neural networks to enhance an AI agent's capacity for decision-making. DRL models educate AI agents to carry out complicated tasks in dynamic situations using a combination of deep neural networks and reinforcement learning approaches. These models are able to learn from unprocessed sensory data and can instantly adjust to changing environmental conditions. [23]

DRL has been used to train AI agents to play a variety of games, including classic arcade games, board games, and even modern-day video games. The reason for this is that DRL models can learn to play games by trial and error, similar to how humans learn. Additionally, DRL models can learn to optimize their performance by receiving feedback in the form of rewards or punishments.

Deep machine learning is used in DRL to overcome the dimensionality curse in RL, and it is influenced by research on the structure and information processing of the neocortex. In conventional RL, all state-action pairings and learnt rewards are stored in a table. Nevertheless, in DRL, a deep artificial neural network takes the place of this table (ANN). This strategy captures the hierarchical characteristics of the issue and is more condensed. It also

generalizes better to unknown conditions. [23]

Moreover, DRL can function in settings with sparse rewards, which is a prevalent issue in serious games. Also, DRL may learn to strike a balance between exploitation and exploration, which is crucial in serious games where the agent must pick up new skills while completing objectives. Examples of DRL's usefulness include the several Atari games that use it to success and computer Go. When used in a serious game, the environment is provided by the game, and the agent is given a visual depiction of the game's current state as well as a reward value. The agent can then choose the right action based on the current state representation and learnt behavior by using a deep ANN as its decision-making component. [23]

In conclusion, DRL is an ideal model to use in this thesis due to its ability to learn from trial and error, adapt to changing conditions, and handle high-dimensional and continuous state spaces to achieve the desired learning outcomes.

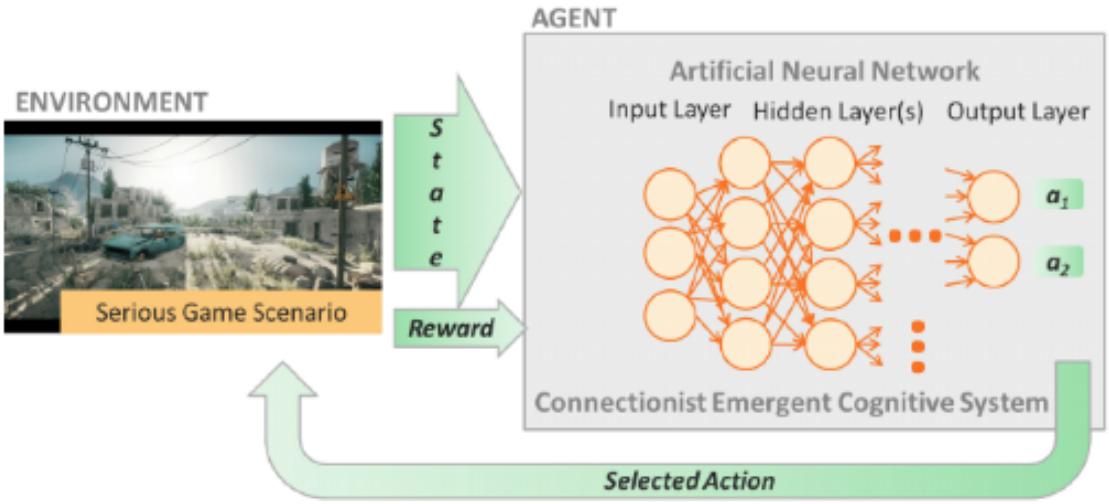

Figure 3.9: Example of (Deep) Reinforcement learning with a serious game as environment and an artificial neural network as the agent's action selection component [23]

Chapter 4

Unity

4.1 Game Engines

Game engine is a piece of software that enables the creation of video games, which is also referred to as a “game architecture” or “game framework,” with settings and configurations that optimize and simplify the development of video games across a variety of programming languages. Typically, a game engine includes the bellow features: [31].

1. A rendering engine that can handle several import formats for 2D or 3D visuals.
2. Handling of input from a keyboard, mouse, touch screen, or other hardware.
3. Game loop. (internal process that updates game events each frame)
4. Physics engine that simulates real-world activities such as collision detection and response.
5. Sound engine that controls sound effects.
6. A scene graph, which controls the relationships between graphic objects on the screen.
7. Animation for 2D sprites or 3D models.
8. Process threading enables numerous parallel processes.

Other features could involve:

1. Scripting.
2. Artificial intelligence (AI) that automatically responds to the player's actions.
3. Networking.
4. Streaming.
5. Localization support.
6. Multi-platform publishing.

The main reason for the creation of the game engines lies on the early video games themselves, which were developed with their own rendering engines, each specifically designed for one game. Over time, game engines evolved from proprietary, in-house engines to commercially developed engines that are widely available today. Game developers, can simplify and speed up the game development process by using commercially developed game engines to produce new games or to extend existing games to additional platforms.

In fact the exact features and workflow will vary from one system to another. As we have seen, some industry-level game engines have grown to be so powerful and flexible that any kind of game can be developed therein. Other are purposely kept simpler and with a narrower scope, either in their target user group or in their supported game mechanics. A few of them do not even require prior programming knowledge. Targeting full fledged game developers, or even amateurs. Some of the game engines are the following:

1. Unity
2. Unreal Engine
3. Godot
4. PHASER
5. Armory
6. CryEngine
7. Defold

4.2 Introduction to Unity Engine

Unity, which is cross-platform game engine, was first introduced at Apple's Worldwide Developers Conference in 2005, and since then, it has caused significant changes in the video game industry. The engine has now been expanded and made available on more than 25 platforms. This engine can be used to create simulations, games, and other training materials. Several industries than video games, including as film, architecture, automotive, construction, and engineering, have also embraced it. [32]

Figure 4.1: Unity Logo

Figure 4.2: Sample of Unity supported platforms. [33]

Unity is a highly popular cross-platform game engine, with a community of over 2.5 million registered developers. In addition to 3D games, Unity also offers tools and features for creating 2D games since 2013, making it a versatile platform for creating games in a variety of genres and styles. When it comes to mobile gaming, Unity is the leading choice for over half of the market. Its cross-platform capabilities and robust set of tools for optimizing performance on mobile devices make it a popular choice. Unity is also a popular choice for augmented reality(AR) and virtual reality(VR) development, with more than 60% of the content being developed using the engine, making it a top choice for creating immersive and interactive experiences. [34]

The fact that Unity has a free version available to everyone with an interest in developing games is one of its key attractions. Because of this, small, independent developers can follow their creative passions and have access to a variety of tools and methods to make their concepts a reality. Unity also offers a Pro Business subscription for $150 per month, which includes priority customer assistance, for people who need extra functionality.

Because to its highly flexible rendering technique and user-friendly tools, Unity is also

renowned for its superior graphics and cutting-edge visual effects. Developers of 3D games have the ability to make objects move smoothly and naturally, and there are numerous tutorials available to assist them. Unity supports C#, JavaScript, and Boo, so even those with little to no coding experience or understanding can use it. But, it is also possible to develop an entire game without writing any code. Those who do have coding abilities can take full advantage of the game engine with the aid of instructional resources and documentation.

One of Unity's key features is the Unity Asset Store, a digital marketplace that offers a wide range of assets for game development. These assets include sounds, 3D structures, patterns, textures, and animations, as well as complete game templates and other tools. The Asset Store is a great resource for developers looking to save time and effort in their game development process, as it allows them to access high-quality assets that they can use in their projects. Additionally, it offers both free and paid assets, giving developers the flexibility to choose the resources that best fit their needs and budget. The Asset Store also provides an opportunity for designers to sell their own assets and earn a percentage of the revenue.

The Unity community is large and supportive, making it easy for developers to ask questions and find solutions to their issues. The annual Unity conference, where developers can share experiences, exchange ideas, and connect with each other, further strengthens the sense of community. Overall, Unity is a friendly and supportive environment that is accessible to both beginner and experienced developers. [35]

4.3 Reason for picking Unity

The Unity game engine was used for the implementation of this thesis project. The free availability of the game engine and the ease of use are the main reasons that led me to choose this particular platform. Furthermore, Unity is accompanied with a prodigious Community support system that can provide solutions to many problems, combining that with many tutorials and ample support materials that are available online . In addition, the wide availability of the free assets and models in the Asset Store which were used in this application were pivotal. Finally, through the ML-Agents package that Unity provides it was possible for me to implement the AI model and make this application complete.

4.4 Unity Environment

The Unity editor environment is a graphical user interface (GUI) that provides tools and features to create, design, and develop Unity projects. It is highly customizable, allowing users to rearrange and resize windows, as well as create custom editor layouts to suit their needs.

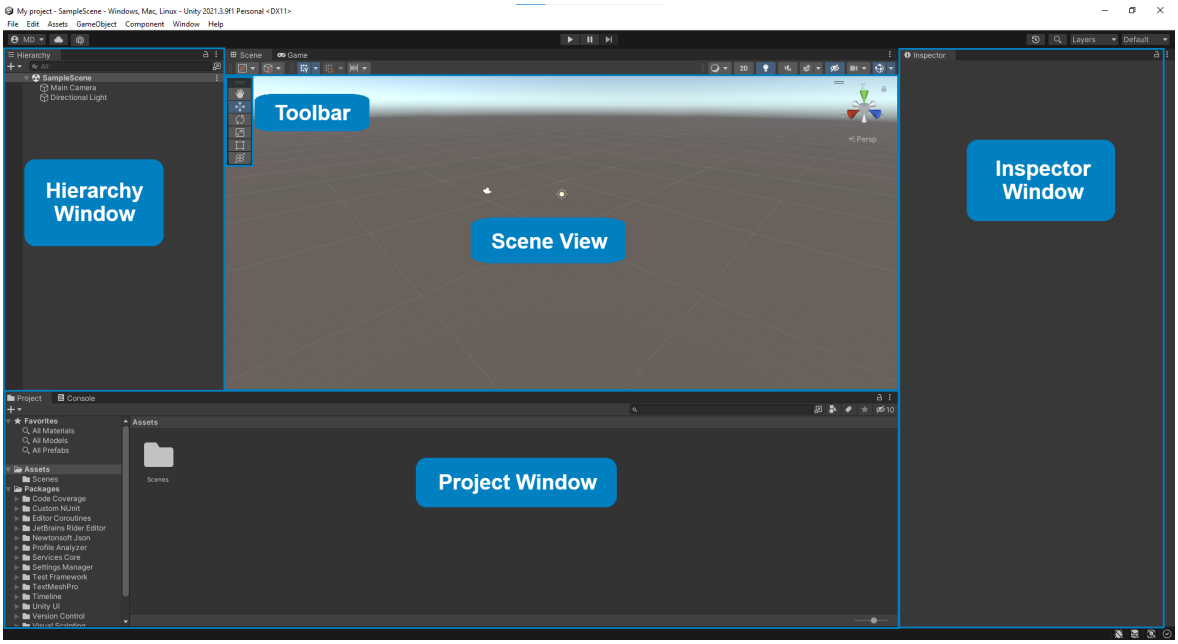

Figure 4.3: Unity Editor Interface

4.4.1 Unity Scenes

Projects in Unity Editor are organized into scenes, which are containers for everything in the experience you are creating. A scene is a self-contained environment that represents a specific location or moment in your game or application. One way to think about scenes is as discrete experiences. For example, each level in a game could be a separate scene, and the game's main menu could be another scene. A Unity project can have one scene or more than a hundred, depending on its scope and complexity. There aren't strict rules about exactly how you should organize a Unity project into scenes, except that a project must have at least one scene. You can create and manage scenes in the Unity editor using the Scene view, the Hierarchy window, and the Inspector window. You can use the Scene view to design and arrange the objects in your scene, the Hierarchy window to organize the objects into a hierarchy, and the Inspector window to configure the properties and components of the

objects. By working with scenes in the Unity editor, you can create and refine the different locations and experiences in your game or application.

4.4.2 Unity Interface

There are five key areas of the basic Unity Editor interface as it's portrayed in **Figure 4.3**, the Scene view and Game view, the Hierarchy window, the Project window, the Inspector window, and the Toolbar.

- **The Scene View** and **Game view** are two of the main viewports in the Unity editor that allow you to view and manipulate the objects in your scene and test and debug your game. The Scene view is an interactive 3D window for designing and arranging objects, while the Game view is a live preview of how the game will play. The Scene view is located in the center of the default Unity layout and is an interactive window that allows you the view and manipulation of objects in 3D space. The Game view is also located in the center of the layout and is a live preview of the game that updates in real-time as you make changes to the scene. You can use the Scene view and Game view to design, test, and debug your game. [36, 37]

- **The Hierarchy** window is a panel that displays a list of all the GameObjects in the current scene and allows you to organize and manage them. You can use the Hierarchy window to create, rename, and delete GameObjects, as well as arrange them in a hierarchy with parent-child relationships. The Hierarchy window also provides a number of other features and options for filtering, searching, and accessing various settings and options for the objects in your scene. [38]

- **The Inspector** window is a panel that displays detailed information about the selected GameObject or asset in your project. It shows the components that make up the object, as well as other properties such as its name, tag, and layer. You can use the Inspector window to view and edit the properties of a component, add or remove components from the GameObject, and customize various other properties of the object. The Inspector window is a useful tool for fine-tuning the appearance and behavior of GameObjects in your scene. [39]

- **The Project** window is a panel that displays the assets available for use in your project, organized in folders. It allows you to access and manage the assets in your project,

whether you use them in the current scene or not. You can use the Project window to browse and locate the assets you need, as well as import new assets into your project. To use an asset in your scene, you can drag it from the Project window into the Scene view or the Inspector window. The Project window is different from the Hierarchy window, which only shows the assets being used in the current scene. The Project window shows all the assets available to your entire project. [40]

- **The Toolbar** is located at the top of the Unity interface and contains buttons for changing your point of view in the scene and starting and stopping Play Mode. It also includes scene navigation functions that allow you to move, rotate, and scale your selected GameObjects. [41]

4.5 Unity Scripting

4.5.1 Conventional and Visual Scripting

In most Unity applications, scripts play an integral role in enabling responses to player input, triggering gameplay events, and generating a wide range of effects and behaviors. Although only C# is utilized in this thesis, Unity supports a variety of programming languages, some of them are Boo and UnityScript. By leveraging Unity scripting, it is possible to develop custom scripts that can be linked to both GameObjects and components in your scene, providing greater control over diverse aspects of your game. For instance, Unity scripting can be used to devise personalized controls for a player character, like movement and jumping. Additionally, scripting can be employed to create visual effects like particle systems and shaders, as well as to manage the physical behavior of objects, including collisions and rigidbody dynamics. Furthermore, scripting can be harnessed to create AI systems tailored to individual characters in your game, for instance, enemy behavior or NPC dialogue. All in all, Unity scripting is a powerful tool that empowers you to design custom behavior, enriching your projects with interactivity and depth. [42]

Unity also features Visual Scripting, which is a valuable tool for creating a variety of behaviors, interactive systems, and gameplay mechanics in Unity applications without writing conventional code. Instead of writing code, Visual Scripting employs visual nodes and connections to devise logic and define game behavior. It is particularly useful for design-

ers, artists, and other non-programmers who seek to create custom behavior in Unity. Visual Scripting presents a visual representation of your game's logic and flow, which can be easier to comprehend and modify than traditional code. With Unity Visual Scripting, it is possible to develop a diverse range of behaviors, including character movement, animation, AI, audio, and particle effects. Furthermore, it can be used to construct interactive systems like dialogue trees, inventory systems, and level progression. Ultimately, Unity Visual Scripting is a powerful tool that enables users to generate custom behavior in Unity, even without coding expertise. It makes it easier for a wider range of users to develop logic and define game flow by offering a visual interface. [43]

4.5.2 Key Unity Script Functions

In Unity, scripts communicate with the engine by implementing a class that is a descendant of the default MonoBehaviour class. The construction of a new Component type that may be affixed to GameObjects is facilitated by this class. A new instance of the class-defined object is created when a script component is attached to a game object. The name of the file in which the script is stored serves as the basis for the class name. The class name and file name must coincide for the script component to be properly assigned to a GameObject.Two of the most basic functions that are defined inside the class are Start and Update. [44]

In addition, FixedUpdate and Awake will also be used in this application since they provide additional utility to the above two functions deficiencies.

- **MonoBehaviour.Start():** When a script is activated, the MonoBehaviour class in Unity automatically runs the Start function right before any Update calls are performed. This makes it the perfect location for carrying out any initialization or setup that must be completed before the script can start running, such as setting up references to other objects in the scene, initializing variables, or establishing links with other components or systems. [45]

- **MonoBehaviour.Awake():** When a script is activated, immediately after the object is created, the Awake method is invoked. The Awake method is the best place to carry out any initialization or setup that needs to happen as soon as the object is formed, in contrast to the Start function, which is called before the script. The Start function only executes on scripts that are enabled, whereas the Awake function executes on

all scripts, whether they are enabled or disabled. There is another distinction between the Start and Awake functions. Because of this, if you have a script that is by default disabled, the Awake function will still be called when the object is created but the Start function won't be until the script is enabled. [46]

- **MonoBehaviour.Update():** The Update function is called every frame during gameplay. It is an ideal place to put code that handles frame-by-frame updates, such as movement, triggering actions, and responding to user input.

- **MonoBehaviour.FixedUpdate():** The FixedUpdate on the other hand, is called a fixed number of times per second, regardless of the frame rate. It is an ideal place to put code that handles physics updates, such as applying forces or calculating collisions. The FixedUpdate function is often used in conjunction with the Update function, with the Update function handling frame-by-frame updates and the FixedUpdate function handling physics updates. [47]

4.5.3 Introduction to Unity Machine Learning Toolkit

The Unity Machine Learning Toolkit (ML-Agent Toolkit) is a powerful tool designed for the creation of intelligent agents in simulated environments. The toolkit provides a Python API that interacts with the Unity Editor and a software development kit (SDK) that contains core functionality for creating training environments within Unity. The ML-Agents SDK consists of three key entities: Sensors, Agents, and the Academy. [48]

Agents are GameObjects in Unity that indicate the presence of agent components in the scene. These agents collect observations from various sensors, such as rendered images, ray-casts, or arbitrary length vectors. The agents then take actions based on policies, which can reference different decision-making mechanisms such as player inputs or neural network models. Finally, the agents receive rewards through a reward function, which serves as a learning signal to facilitate desired behavior. [48]

The Academy is responsible for managing agents and keeping track of the number of steps taken during simulation. It also holds environmental parameters, which enables the application of training methods like Curriculum Learning. The toolkit contains a number of high-level components, including the Learning Environment, which contains the scene and all GameObjects connected to the learning process, the Python API, which is responsible for

interacting and manipulating the Learning Environment through a low-level interface, the Communicator, which connects the low-level Python API with the Learning Environment, and the Python Trainers, which hold machine learning algorithms that enable the training of agents. [12]

In conclusion, the Unity ML-Agent Toolkit is a comprehensive toolkit for creating intelligent agents in simulated environments. It provides a powerful Python API, a robust SDK, and a range of high-level components that make it easy for developers to create intelligent agents that can learn and adapt in simulated environments. The toolkit offers a flexible and scalable solution for developing and training intelligent agents, making it an important tool for researchers and developers in the field of machine learning and artificial intelligence.

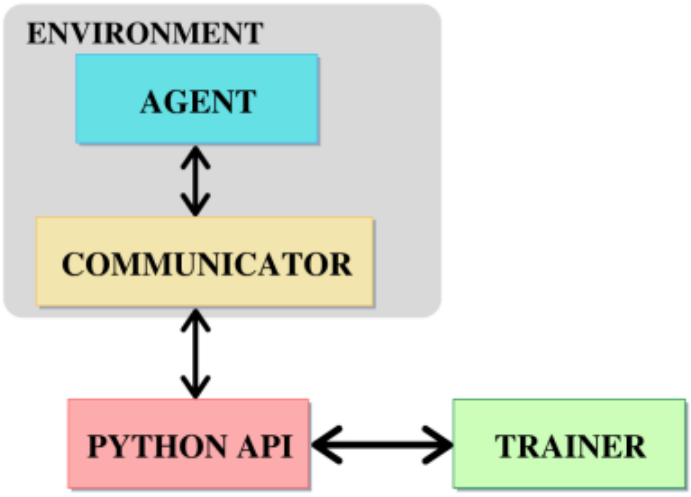

Figure 4.4: High-level view of the ML-Agent toolkit [49]

4.5.4 Agent Class

The Agent class is an important component of the Unity Machine Learning Agents (ML-Agents) toolkit, which is used to build intelligent agents for training in simulated environments. Derived from the MonoBehaviour class, the Agent class can be attached to a GameObject in a Unity scene and can receive updates in the game loop. This makes it a powerful tool for creating agents that can learn and adapt in simulated environments.

One of the key features of the Agent class is its ability to access and modify the state of the agent. The Agent class provides a number of properties that you can use to get information about the agent's current reward, available actions, and other important parameters, we will

discuss more about this topic in the later chapter. This allows you to build agents that can make decisions based on their current state and the environment they are in.

In addition to its state-related properties, the Agent class also provides a number of methods that you can use to control the behavior of the agent. For example, you can use the Agent class to send actions to the agent, request decisions from the agent, or reset the agent's state. This makes it easy to build agents that can learn and adapt to changing conditions in a simulated environment.

Overall, the Agent class is a powerful component of the ML-Agents toolkit that allows you to build intelligent agents that can learn and adapt in simulated environments. Whether you are creating a game AI, a robot simulation, or some other kind of intelligent agent, the Agent class provides the tools you need to build agents that can learn and adapt to their environment.

Chapter 5

The Game

5.1 Game overview

This chapter focuses on the development of a firefighter agent using deep reinforcement learning utilizing Unity's ML-Agent Toolkit. I will create a virtual environment in Unity to simulate a firefighter's task of putting out fires. The environment will consist of different fire sources spread at different places of the scene and the firefighter agent. The goal of the agent will be to reach the fire source and put it out, while avoiding obstacles along the way. The agent will receive rewards for putting out the fire and penalties for colliding with obstacles.

The Unity simulation environment provides a realistic firefighting scenario for the agent to interact with and learn from.The training of the firefighter agent involved using deep reinforcement learning techniques to iteratively adjust its behavior based on the outcomes of its actions and the rewards it received. The agent interacts with its environment and receives a reward signal for each action it takes. The training process was based on trial and error, allowing the agent to gradually improve its performance as it gained more experience. The agent's behavior is shaped by a deep neural network that processes high-dimensional observations and outputs decisions. The algorithm takes in the state of the environment and outputs an action for the agent to take. The agent then performs the action, the environment updates, and the process repeats. Over time, the agent will learn the optimal policy for putting out fires by maximizing the reward received. At the end of the training process, the trained firefighter agent will be tested against a human player to assess its effectiveness.

5.1.1 Method Used

The ML-Agent Toolkit as mentioned in the previous chapter is comprehensive and contains many features of interest, but due to time and scope limitations, it is not feasible to study the entire toolkit. Instead, the project will limit its evaluation to only one of the reinforcement learning techniques supplied by the toolkit, Proximal Policy Optimization (PPO). PPO was chosen because it is a more general purpose and stable algorithm compared to the other options, and has been widely used in many projects. [50]

The approach of the project is to train agents using PPO with different configurations of the reward function, observable traits, and hyperparameters. Each produced model is re-evaluated by testing it against the training data, and promising models are further tested with new test data. The training is conducted by running 8 learning agents simultaneously in a training environment. There is potential to run even more but the performance of my machine will drop significantly and terminate the training. The training process will continue until a high enough mean reward is calculated, or until ten million steps have been reached. If a training configuration fails to complete before the maximum number of steps, it will be considered insufficient.

5.1.2 Agent Component

The Agent class that was mentioned earlier is a component that enables game objects to participate in reinforcement learning. The component contains the definition of several key learning methods that are used to train the agent, including:

1 **Initialize:** A method which is called when the Agent object is first created, and its purpose is to set the initial conditions for the Agent's learning process. During the Initialize method, the Agent's initial state, or starting point, can be defined, along with any other relevant parameters that will be used throughout the learning process. The Initialize method provides a way to configure the Agent and set it up for training or inference. It is an essential step in preparing the Agent for its learning task, and it is typically called before the learning process begins.

2 **OnEpisodeBegin:** This method is called at the start of each episode of training. It is used to reset the state of the agent and set it up for the next episode. The method can be

used to initialize any variables or perform any setup that is required to begin the next episode.

3 **CollectObservations:** This method is used to collect data from the environment that is used to make decisions. The method typically defines what information the agent should observe and provide to the reinforcement learning algorithm. This information is used by the algorithm to determine the next action to take.

4 **OnActionReceived:** This method is called when the agent has received an action from the reinforcement learning algorithm. The method is responsible for taking the action in the environment and observing the results. The results of the action are used to update the agent's state and provide feedback to the reinforcement learning algorithm.

5 **Heuristic function:** is a function that can be used to specify how the agent should behave when it is not being trained. The function can be used to define a set of rules or conditions that determine the agent's behavior in certain situations. The function can be used to allow the agent to make decisions based on the state of the environment, even when it is not being trained.

5.2 Behavior Parameters

An essential part of determining agent behavior in a scene is the Behavior Parameters script, commonly known as the brain. It acts as a communication channel between the agent and the behavior that will be employed during training and is associated to each agent's GameObject in the scene.

The Behavior Parameters script includes several important parameters that determine the behavior of an agent. These include:

- **Behavior Name:** This refers to the name of a set of hyperparameters used by a training algorithm.

- **Vector Observation:** The Space Size parameter determines the number of variables observed from the training environment, while Stacked Vectors refers to the number of sampled vectors used before calculation.

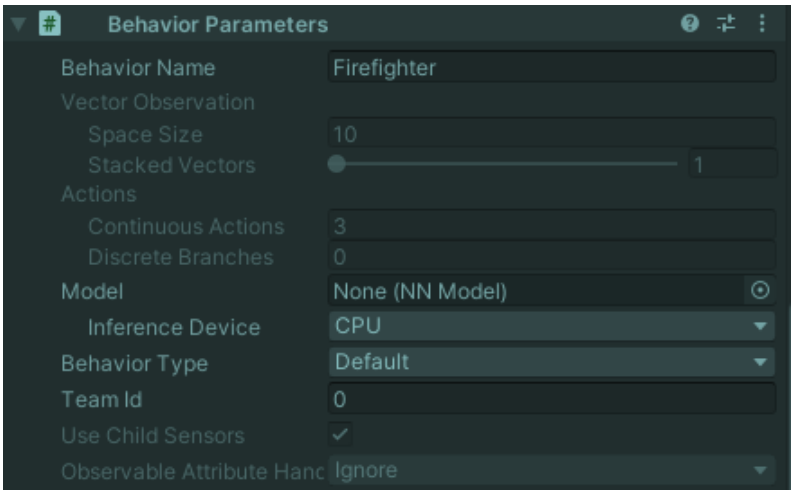

Figure 5.1: Behavior Parameters script on one of the agents during testing.

- **Actions:** The type and number of available actions that the agent can take are defined by this parameter. The type of action can be either Continuous, which represents a floating-point value between 0.0 and 1.0, or Discrete, which translates to either 1 or 0. In my case, Continuous actions were chosen as it's smoothing the movements of the agent. And the three actions represent the moving forward-backward, right-left and rotating right and left.

- **Model:** This parameter references a produced Neural Network (NN) model that will be produced after successfully training the agent and will be used to determine actions during inference. You also have the option to decide between the CPU and the GPU to perform the training.

- **Behavior Type:** This parameter determines how the brain chooses to interpret action selection. There are three types of behaviors: Inference, where selected actions are decided by an existing NN model, Heuristic, where selected actions are manually determined through code logic inside the heuristic function and Default, where actions are selected by a NN model during training.

5.2.1 Decision Requester

Another element of the ML-Agents toolset is the Decision Requester script, which is used to ask the trained AI agents for decisions. For the agent to do training in Unity, the Decision Requester script is essential since it enables the AI agents to make decisions based on the

observations they gather from their surroundings.

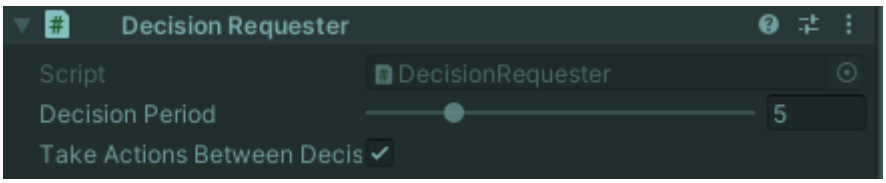

Figure 5.2: Unity's Decision Requester.

The "Take Actions Between Decisions" flag indicates whether the agent should be taking actions in between decision requests or not. If it's enabled, the agent will continuously take actions at the specified frequency. The "Decision Period" parameter sets the frequency of the decision requests. In this case, checking the "Take Actions Between Decisions" flag and having a "Decision Period" of 5 means that the agent will take actions at a frequency of once every 5 seconds.

5.3 Environment

The agent environment consists of a firefighter actor in a backyard different obstacles, and the fires he is supposed to extinguish. The agent's goal is to navigate through the backyard in order to reach the fireplaces by using a defined set of actions in as short a time as possible, while avoiding different obstacles along the way.

5.3.1 Learning Environment

For the agent to effectively learn and respond to its surroundings, it needs to have input to observe during its training process. At the start of each step, the sensor collects observations and passes them to the trainer for interpretation. Based on the observations, the trainer generates a decision response, which is then sent back to the brain to execute the appropriate action. This cycle of observation and action is crucial in learning the optimal policy for a specific problem. It is important that the data collected through observation is comprehensive enough for the agent to arrive at a viable solution.

In this case, the information our Agent collects includes the agent's local rotation (4 observations), a normalized vector pointing to the nearest fire (3 observations), a dot product that indicates whether the gun tip is in front of the fire (1 observation), a dot product that indicates whether the gun is pointing towards the fire (1 observation), and the relative distance

from the gun tip to the fire (1 observation). This results in a total of 10 observations. This helps the Agent learn to control its speed as well as his position in the environment.

Following is the code used to collect the observations needed for the agent's training:

```
/// <summary>
/// Collect vector observations from the environment
/// </summary>
/// <param name="sensor"> The vector sensors</param>
0 references
public override void CollectObservations(VectorSensor sensor)
{
    //If nearest fire is null, observe an empty array and return early
    if (nearestFire == null)
    {
        sensor.AddObservation(new float[10]);
        return;
    }

    //Observe the agent's local rotation (4 observations)
    sensor.AddObservation(transform.localRotation.normalized);

    //Get a vector from the gun tip to the nearest fire
    Vector3 toFire = nearestFire.FireCenterPosition - gunTip.position;

    //Observe a normalized vector pointing to the nearest fire (3 observations)
    sensor.AddObservation(toFire.normalized);

    //Observe a dot product that indicates whether the gun tip is infront of the fire (1 observasion)
    //(+1 means that the gun tip is directly in front of the fire, -1 means directly behind)
    sensor.AddObservation(Vector3.Dot(toFire.normalized, -nearestFire.FireUpVector.normalized));

    //Observe a dot product that indicates whether the gun is pointing towards the fire (1 observation)
    //(+1 means that the gun is pointing directly at the fire, -1 means directly away)
    sensor.AddObservation(Vector3.Dot(gunTip.forward.normalized, -nearestFire.FireUpVector.normalized));

    //Observe the relative distance from the gun tip to the fire (1 observation)
    sensor.AddObservation(toFire.magnitude / FireArea.AreaDiameter);

    //10 total observations
}
```

Figure 5.3: The function to collect observations from the environment

5.3.2 Testing the Environment

Before moving to the process of training the agent, firstly a check to ensure the right interaction with the environment was made. It can be done by extending the "Heuristic()" method in the Firefighter class. This method maps the values of the Horizontal and Vertical input axis to corresponding actions for the agent. To use the heuristic, you need to set the Behavior Type to "Heuristic Only" in the Behavior Parameters of the Firefighter agent in Unity's Inspector window. Then, by pressing the play button, you can run the scene and use the fixed keys to move the agent around the platform. Ensuring this way that there are no errors displayed in the Unity Editor Console window and that the avatar interacts successfully with the fires.

```
/// When Behavior Type is set to "Heuristic only" on the agent's Behavior Parameters,
/// this function will be called. Its return values will be fed into
/// <see cref="OnActionReceived(ActionBuffers)"/> instead of using the neural network
/// </summary>
/// <param name="actionsOut"> An output action array</param>
0 references
public override void Heuristic(in ActionBuffers actionsOut)
{
    //Create placeholders for all movement and turning
    Vector3 forward = Vector3.zero;
    Vector3 right = Vector3.zero;
    float yaw = 0f;

    //Convert keyboard inputs for movement and turning
    //All values should be between -1 and +1

    //Forward and backward
    if (Input.GetKey(KeyCode.W)) forward = transform.forward;
    else if (Input.GetKey(KeyCode.S)) forward = -transform.forward;

    // Right and left
    if (Input.GetKey(KeyCode.D)) right = transform.right;
    else if (Input.GetKey(KeyCode.A)) right = -transform.right;

    //Turn left and right
    if (Input.GetKey(KeyCode.LeftArrow)) yaw = -1f;
    else if (Input.GetKey(KeyCode.RightArrow)) yaw = 1f;

    //Combine the movement vectors and normalize
    Vector3 combined = (forward + right).normalized;

    //Add the 2 movement values and yaw to the actionsOut array
    var continuousActionsOut = actionsOut.ContinuousActions;
    continuousActionsOut[0] = combined.x;
    continuousActionsOut[1] = combined.z;
    continuousActionsOut[2] = yaw;

}
```

Figure 5.4: Screenshot of the Heuristic function

5.3.3 Reward System

The reward system is a crucial component of reinforcement learning. It provides the agent with feedback on its actions and helps the learning algorithm determine the optimal behavior policy. Rewards can be positive or negative, serving to either reinforce or penalize certain actions. Throughout the training process, rewards are accumulated with each step through either increments or decrements. The final reward is given at the end of an episode, which can end with the agent reaching its goal, crashing, or not moving far enough within a specified time frame. By providing appropriate rewards, the agent is guided towards the desired behavior and can learn to complete the assigned task efficiently.

The calculation of the positive reward (5.1)involves two parts: a constant bonus reward of 0.02 and the additional reward that is based on the angle between the agent's forward direction (f) and the direction towards the nearest fire (u). The bonus reward is calculated as 0.02 times the result of the dot product between the normalized forward direction of the agent and the normalized direction towards the nearest fire. Rewarding the agent as long as he interacts successfully with the fire and later giving him a bonus for putting it out.

As for the negative reward (5.2)it can be based on a flat number of 0.5 or 0.05 when the agent is either colliding with the boundaries of the environment or with the fire, teaching him on one side to avoid unnecessary interaction with the boundaries and different obstacles, and on the other side, punish him touching the fire with high enough points to prevent him collide all the way with the fire to get the reward from it, but at the same time not to overwhelm the reward of putting it out.

$$reward = a + b \cdot \min(1, \frac{f \cdot -u}{|f| \cdot |u|}) \tag{5.1}$$

$$reward = \begin{cases} -0.5 \\ -0.05 \end{cases} \tag{5.2}$$

5.3.4 Agent's actions

Now that observations and reward logic is completed, we can proceed with the OnActionReceived function which is used to receive and process action requests from the decision-making agent. The OnActionReceived function is called whenever the agent receives a new set of actions from the behavior parameters script.

The action received by this function can be used to control the agent's movements and animations. The function is responsible for taking the action received from the behavior parameters script, interpreting it, and applying it to the agent.

In this case, the actions received is a set of continuous values representing the movements and rotation of the agent, updating the agent's position based on the received values. This can be done by using a physics engine to simulate his movement, applying the necessary force to rotate and and move in the environment.

A screenshot of OnActionReceived function is following bellow:

```
/// <summary>
/// Called when action is received from either the player input or the neural network
/// actions.ContinuousActions represents:
/// Index 0: move vector x (+1 =  right, -1 = left)
/// Index 1: move vector z (+1 =  forward, -1 = backward)
/// Index 2: yaw angle (+1 =  turn right, -1 = turn left)
/// </summary>
/// <param name="actions"> The actions to take </param>
1 reference
public override void OnActionReceived(ActionBuffers actions)
{
    //Don't take actions if frozen
    if (frozen) return;

    //Calculate movement vector
    Vector3 move = new Vector3(actions.ContinuousActions[0], 0, actions.ContinuousActions[1]);

    // Add force in the direction of the move vector
    rigidbody.AddForce(move * moveForce);

    // Get the current rotation
    Vector3 rotationVector = transform.rotation.eulerAngles;

    //Calculate yaw rotation
    float yawChange = actions.ContinuousActions[2];

    // Calculate smooth rotation changes
    smoothYawChange = Mathf.MoveTowards(smoothYawChange, yawChange, 2f * Time.fixedDeltaTime);

    //Calculate new yaw based on smoothed values
    float yaw = rotationVector.y + smoothYawChange * Time.fixedDeltaTime * yawSpeed;

    // Apply the new rotation
    transform.rotation = Quaternion.Euler(0f, yaw, 0f);
}
```

Figure 5.5: On Action Received function

5.3.5 Hyperparameters

The toolkit's documentation states that changing the Hyper Parameters can have a big impact on how well the agent trains. Several setup options from example projects were assessed to find a good place to start. Many of these settings in this game scenario proved insufficient for the agent to learn a useful policy. [51]

It was observed that, in contrast to the other parameters, raising the batch size, buffer size, hidden units, time horizon, number of layers, and maximum steps appeared to significantly alter and improve training outcomes. As a result, the majority of parameters were left alone, while the latter could be looked into more thoroughly.

- **time horizon:**The number of experience steps collected per agent before adding to the experience buffer determines the expected reward estimate from the agent's current state.

- **batch size:** Is the number of experiences used in each iteration of gradient descent in

reinforcement learning,

- **buffer size:** Is the number of experiences collected before updating the policy model and should always be multiple times bigger than the batch size.

- **hidden units:**The number of units in each hidden layer of the neural network, representing how complex the interaction between observation variables is in determining the action.

- **number of layers:**The count of layers concealed within the neural network, indicating the quantity of layers that exist following the input observations.

- **max steps:** Is the maximum number of steps that will be performed before the training stops

Table 5.1: HyperParameters

Trainer Type	PPO
Batch Size	2048
Buffer Size	20480
Learning Rate	0.0003
Beta	0.005
Epsilon	0.2
Lambd	0.95
Num Epoch	3
Learning Rate Schedule	Linear
Keep Checkpoints	5
Checkpoint Interval	500000
Max Steps	10000000
Time Horizon	128
Summary Freq	10000
Threaded	false

Table 5.2: Network Settings

Normalize	false
Hidden Units	256
Num Layers	2
Vis encode type	Simple
Goal conditioning type	Hyper

Table 5.3: Extrinsic Rewards

Gamma	0.99
Strength	1.0
Normalize	False
Hidden Units	128
Num Layers	2
Vis encode type	Simple
Goal conditioning type	hyper

5.4 Training Process

The training process is composed of separate episodes that involve the agent taking steps in each frame, which are tracked by the Academy. During these episodes, the following events take place:

- At every step of the episode, each agent selects one action from its set of available actions.
- The training algorithm uses the information gathered from the state observations, selected actions, and obtained rewards to optimize its policy and pick the best sequence of actions to maximize the rewards received from the environment.
- The information gathered during training is saved for later analysis, with each step being recorded frame by frame.
- The state variables are updated based on the effects of the actions performed by the agents.

The episode begins when the Academy invokes the OnEpisodeBegin function, which restarts all the multiple environments to their starting position, by placing the agents at their predetermined positions, resetting every fire that the agent ended up extinguishing in that episode and setting the environment variables and specific agent values. Throughout the episode, the agents make their selections and take actions in a frame-by-frame fashion until the episode concludes.

The snapshot below showcases the training environment of 8 different agents trying to extinguish fires, during a specific episode. It's showcasing an episode where each agents has succesfully put out a number of fires. The agents are trained simultaneously, providing a rich and diverse learning experience. This approach can help the agents generalize better to a wider range of situations and improve the overall efficiency of the training process.

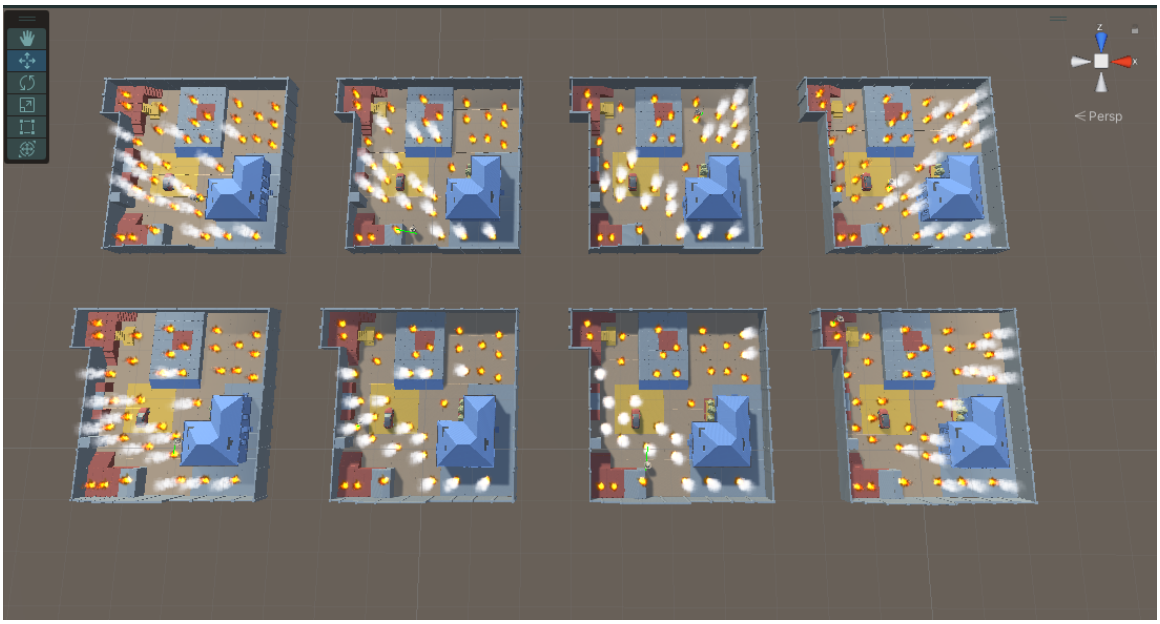

Figure 5.6: Training Mode Environment

Bellow we can also see that at the beginning of the training the agents are not yet familiar with the world, so the actions they do accumulate negative rewards since they either collide with the boundaries or hitting the fires with their bodies. But after enough episodes passed they started collecting a decent amount of positive rewards.

5.4.1 Training Results

The ML-Agents Toolkit provides a useful TensorFlow utility to monitor the training progress of the agents, named TensorBoard that allows to view statistics from your train-

```
[INFO] Firefighter. Step: 30000. Time Elapsed: 37.832 s. Mean Reward: -7.835. S
[INFO] Firefighter. Step: 40000. Time Elapsed: 46.942 s. Mean Reward: -10.824.
[INFO] Firefighter. Step: 50000. Time Elapsed: 60.020 s. Mean Reward: -10.984.
[INFO] Firefighter. Step: 60000. Time Elapsed: 69.214 s. Mean Reward: -15.390.
[INFO] Firefighter. Step: 70000. Time Elapsed: 82.210 s. Mean Reward: -13.416.
[INFO] Firefighter. Step: 80000. Time Elapsed: 91.274 s. Mean Reward: -6.655. S
[INFO] Firefighter. Step: 90000. Time Elapsed: 104.296 s. Mean Reward: -7.661.
[INFO] Firefighter. Step: 100000. Time Elapsed: 113.556 s. Mean Reward: -8.256.
[INFO] Firefighter. Step: 110000. Time Elapsed: 126.574 s. Mean Reward: -6.174.
[INFO] Firefighter. Step: 120000. Time Elapsed: 135.570 s. Mean Reward: -8.000.
[INFO] Firefighter. Step: 130000. Time Elapsed: 148.664 s. Mean Reward: -4.690.
[INFO] Firefighter. Step: 140000. Time Elapsed: 158.064 s. Mean Reward: -9.829.
[INFO] Firefighter. Step: 150000. Time Elapsed: 171.084 s. Mean Reward: -9.460.
[INFO] Firefighter. Step: 160000. Time Elapsed: 180.115 s. Mean Reward: -8.813.
[INFO] Firefighter. Step: 170000. Time Elapsed: 193.150 s. Mean Reward: -9.567.
[INFO] Firefighter. Step: 180000. Time Elapsed: 202.429 s. Mean Reward: -9.799.
[INFO] Firefighter. Step: 190000. Time Elapsed: 215.527 s. Mean Reward: -10.675
[INFO] Firefighter. Step: 200000. Time Elapsed: 224.599 s. Mean Reward: -5.290.
[INFO] Firefighter. Step: 210000. Time Elapsed: 233.910 s. Mean Reward: -6.213.
```

Figure 5.7: Early training results

```
[INFO] Firefighter. Step: 2010000. Time Elapsed: 2202.317 s. Mean Reward: 22.042.
[INFO] Firefighter. Step: 2020000. Time Elapsed: 2215.161 s. Mean Reward: 25.623.
[INFO] Firefighter. Step: 2030000. Time Elapsed: 2224.174 s. Mean Reward: 22.371.
[INFO] Firefighter. Step: 2040000. Time Elapsed: 2236.792 s. Mean Reward: 27.029.
[INFO] Firefighter. Step: 2050000. Time Elapsed: 2245.810 s. Mean Reward: 27.977.
[INFO] Firefighter. Step: 2060000. Time Elapsed: 2258.637 s. Mean Reward: 30.616.
[INFO] Firefighter. Step: 2070000. Time Elapsed: 2267.615 s. Mean Reward: 21.895.
[INFO] Firefighter. Step: 2080000. Time Elapsed: 2280.243 s. Mean Reward: 28.570.
[INFO] Firefighter. Step: 2090000. Time Elapsed: 2289.290 s. Mean Reward: 29.513.
[INFO] Firefighter. Step: 2100000. Time Elapsed: 2298.325 s. Mean Reward: 34.083.
[INFO] Firefighter. Step: 2110000. Time Elapsed: 2311.074 s. Mean Reward: 33.686.
[INFO] Firefighter. Step: 2120000. Time Elapsed: 2319.865 s. Mean Reward: 29.208.
[INFO] Firefighter. Step: 2130000. Time Elapsed: 2332.681 s. Mean Reward: 28.457.
[INFO] Firefighter. Step: 2140000. Time Elapsed: 2341.638 s. Mean Reward: 29.074.
[INFO] Firefighter. Step: 2150000. Time Elapsed: 2354.395 s. Mean Reward: 33.296.
[INFO] Firefighter. Step: 2160000. Time Elapsed: 2363.218 s. Mean Reward: 23.740.
[INFO] Firefighter. Step: 2170000. Time Elapsed: 2376.435 s. Mean Reward: 26.426.
[INFO] Firefighter. Step: 2180000. Time Elapsed: 2385.652 s. Mean Reward: 24.571.
[INFO] Firefighter. Step: 2190000. Time Elapsed: 2398.583 s. Mean Reward: 34.290.
[INFO] Firefighter. Step: 2200000. Time Elapsed: 2407.492 s. Mean Reward: 26.405.
```

Figure 5.8: Later at the training results

ing sessions. Bellow are the results of the agents training after five million and ten million steps, showing the importance of the steps that were required to achieve a convincing result.

Presenting the three key metrics, Environment/Cumulative Reward, Losses/Policy Loss and Losses/Value Loss. The Environment/Cumulative Reward displays the mean cumulative episode reward over all agents, and should increase during a successful training session. As in this case showing a continuous growth till the eighth million step and after that starting to stabilize. The Losses/Policy Loss indicates the mean magnitude of the policy loss function, which shows how much the policy is changing. It should decrease during a successful training session, as the agents has been accustomed to the actions that will add up the negative reward and will start avoid them. The Losses/Value Loss shows the mean loss of the value function update, which is an indication of how well the model predicts the value of each state. It should increase while the agent is learning and decrease once the reward stabilizes.

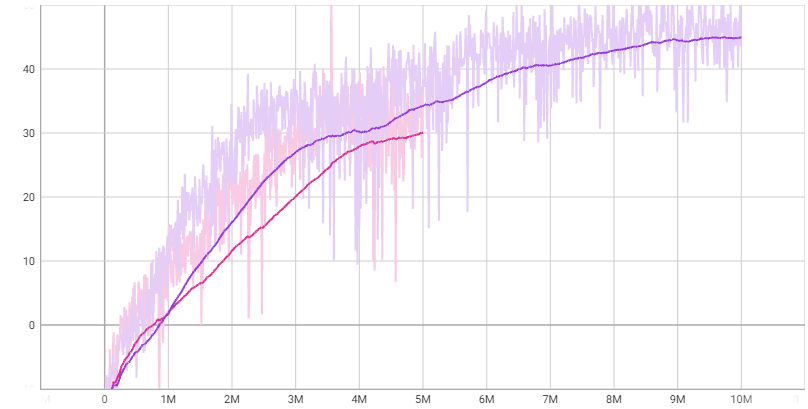

Figure 5.9: Environment / Cumulative Reward Graph

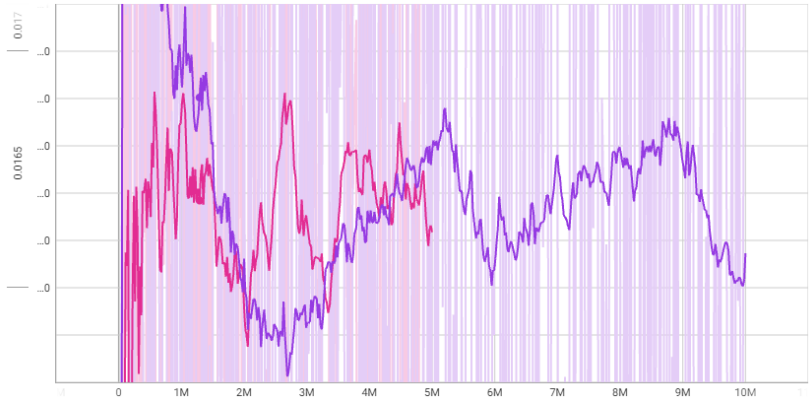

Figure 5.10: Losses / Policy Loss Graph

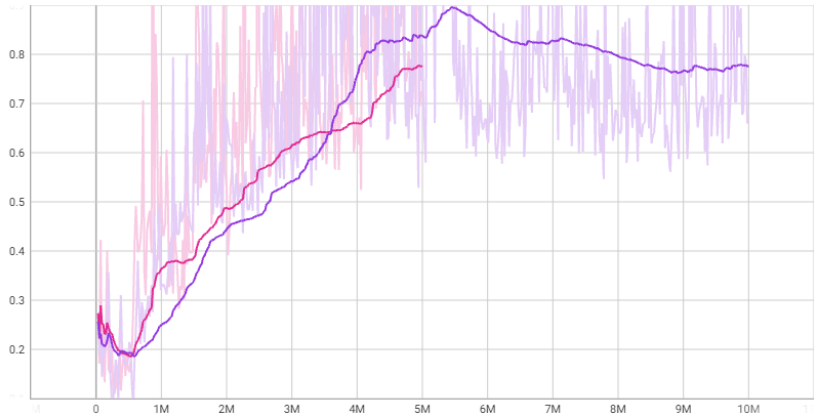

Figure 5.11: Losses / Value Loss Graph

The conclusion of the training process results in a saved ".onnx" file of the neural network. This NN can then be integrated into the Unity project by placing it in the "Model" box in the Behavior Parameters script. To test the performance of the NN, all but one environment is deleted and the agent is tested in real time. This is the result of the best performing NN from all training runs, and thus is selected to proceed for the completion of the game.

5.5 Game Scene

5.5.1 Game Avatar

In the bellow figure the avatar used in this game for both the agent and the human player can be noticed, as well as all his components. The avatar includes a “Rigidbody”, which is a solid body that assists the movements in the environment, two colliders, one for his body and one for the hose he wields, that are essential parts for the interaction with the flames and the obstacles. He also contains Ray Perception Sensors which are used to detect obstacles, measure distances, or detect specific objects of interest in the game environment in this case the fires, by casting rays in specific directions. Finally it contains the necessary scripts for the agent to move, observe, and decide the actions to take in the environment, whose major functionalities were discussed above.

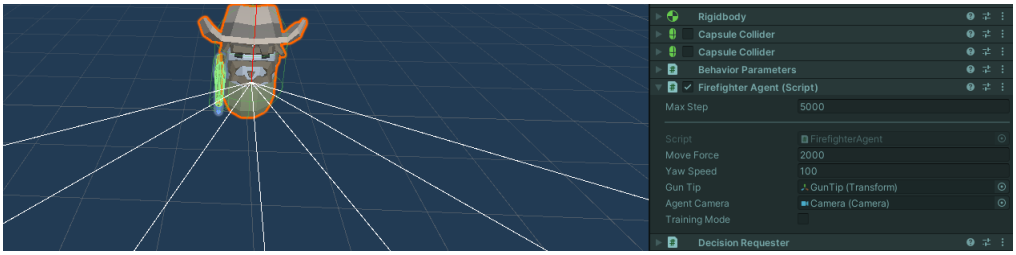

Figure 5.12: The Avatar and its components

5.5.2 Game Menus

The main menu of the game is the point of contact for the user when they start playing. There is background music playing while the user is in this scene, and also throughout the entire game. In this scene, there is a background of the game environment, the Start and Exit buttons. The world consists of simple graphics and objects, to balance the training experience of the agent with the player’s gaming experience. The camera model provides a first-person view of the game to the player giving a more realistic feeling and increase the sense of presence in the game world.

Figure 5.13: Game's Starting Menu

After the Start button is pressed the game ready to begin. The player is placed next to a fire ready to compete against the trained agent. The game provides a simple timer of "3, 2, 1, Go!" giving a short time for the player to prepare before he begins to extinguishing fires. Fires will begin to spawn at different places of the environment which both of them have to extinguish.

5.5.3 Gameplay View

During the game time at the top of the screen there is a bar that indicates how many fires each one has put out. The movement of the player's avatar is with the "W, A, S, D" buttons plus the left and right arrow keys to help the avatar turn either left or right, The player has to be flexible, fast and observant as he has to decide the best possible paths he is going to take for each fire, since the agent is also trained to get to the fire's place in an optimal way. Each time a fire is extinguished a smoke signal will begin to appear marking the end of the individual flame, collecting a point and reminds you to continue with the next remaining fires.

Figure 5.14: Screenshot of the game's result with the Agent being the winner

The game concludes with a message of "Agent wins!" or "You Win!" based on who ended up extinguishing the most fires, the number of fires is set to an odd number, to prevent the draw scenarios.

Chapter 6

Conclusion

This section of the thesis provides an overview of the research that has been conducted, summarizing the key findings and conclusions reached. This section is also devoted to discussing potential future directions and applications of DRL within the Unity engine. The aim is to provide insight into the potential future developments and advancements that can be made in this field, deepening our understanding and capabilities within the realm of DRL.

6.1 Summary

Artificial Intelligence is one of the most rapidly growing fields of technology that has revolutionized many industries, including the gaming industry. AI algorithms are increasingly being used in video games to make them more engaging, challenging, personalized, and realistic. The use of AI algorithms in games has improved the overall gaming experience, making games more interactive and engaging for players.

Deep Reinforcement Learning has become a crucial part of the gaming industry due to its ability to provide realistic and interactive gaming experiences. In video games, the AI agents can now interact with their environment in real-time and make decisions based on the positions of objects and obstacles. This has improved the overall gaming experience and has made the games much more challenging and entertaining for players. Using the Unity engine and its MLAgents toolkit, this thesis showcases the potential of DRL by successfully training an AI agent to interact with its surroundings in a firefighting setting and make decisions based on the positions of fires and obstacles.The successful training of an AI agent using DRL in the game developed in this thesis, as the agent is capable of competing at even grounds with a hu-

man player, or even outperforming him, highlights the significance of utilizing AI algorithms and the potential for these algorithms to operate with accuracy in real-world scenarios.

On a final note this thesis tries to show the capabilities of a trained AI agents in real-life applications beyond the game environment. For example, in a firefighting scenario, as in this case, a trained AI agent can be deployed to make quick and effective decisions, as well as move efficiently in order to put out the fires. Similarly, in an earthquake scenario, a trained AI agent can be used to navigate through the rubble and make decisions on the most effective way to rescue trapped individuals. In these real-world situations, the decision-making abilities and efficiency of a trained AI agent can greatly improve response times and potentially save lives. The significance of successfully training an AI agent in as realistic simulated environments as possible, further highlights the potential for using these algorithms in real-world scenarios, making it a significant development in the field of Artificial Intelligence.

Here is the link, where the code implemented during the present work is uploaded to GitHub. (**`https://github.com/gfragkias/Thesis`**)

6.2 Future work

The current game application provides a strong base for additional research in the field of AI. The success of training an AI agent in a realistic simulated environment highlights the potential for these algorithms to be applied in real-world scenarios. However there are still some aspects of this application to be improved.

One key area where it possibly will provide with even better results, is a more in depth study of the hyper-parameters the agent is using, to make an even stronger NN for the agent, thus enhancing even more the training results.

One of the priorities is to expand the environment to include even more realistic and complex scenarios, to create an even more decisive and up to real-world trained agent. An example of this could be incorporating a spreading time for the fires, determined by the amount of time it takes for the AI agent to extinguish each one. Or even give bigger interaction space the agent has to put out each fire, as depending on the fire scale it will be more difficult to approach it.

Finally, without missing the game aspect of this application, in order to compete in the crowded market the current game can be further developed and improved with a focus on

both the visual graphics and gameplay mechanics. Making it more engaging for someone to play the game.

Bibliography

[1] M Prensky. *Μάθηση Βασισμένη στο Ψηφιακό Παιχνίδι. Επιστημονική επιμέλεια: Μεϊμάρης Μ.* Μεταίχμιο, Αθήνα, 1 edition, 2009.

[2] D. Djaouti, J. Alvarez, JP. Jessel, and O. Rampnoux. *Origins of Serious Games. In Serious Games and Edutainment Applications*. Springer, London, London, 1 edition, 2011. https://doi.org/10.1007/978-1-4471-2161-9_3.

[3] Manuel Freire, Ángel Serrano-Laguna, Borja Manero, Ivan Martinez-Ortiz, Pablo Moreno Ger, and Baltasar Fernández-Manjón. Game learning analytics: Learning analytics for serious games. pages 1–29, 04 2016. doi:10.1007/978-3-319-17727-4_21-1.

[4] Richard L. Lamb, Leonard Annetta, Jonah Firestone, and Elisabeth Etopio. A meta-analysis with examination of moderators of student cognition, affect, and learning outcomes while using serious educational games, serious games, and simulations. *Computers in Human Behavior*, 80:158–167, 03 2018.

[5] Damien Djaouti, Julian Alvarez, and Jean-Pierre Jessel. Classifying serious games: the g/p/s model. *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*, 01 2011.

[6] Ai in video games: Toward a more intelligent game. 2017. `https://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/`. accessed: 16 Jan 2023.

[7] Schwab B. *AI Game Engine Programming*. Course Technology, Boston, MA, 2 edition, 2009.

[8] Wikipedia contributors. Nim — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Nim&oldid=1126742498, 2022. accessed: 16 Jan 2023.

[9] J. Wexler. Artificial intelligence in games. https://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf, May 2002.

[10] History of ai use in video game design. https://bigdataanalyticsnews.com/history-of-artificial-intelligence-in-video-games/, 2021. accessed: 16 Jan 2023.

[11] Julian Togelius Georgios N. Yannakakis. *Artificial Intelligence and Games*. Springer, Cham, 1 edition, 2018. http://gameaibook.org.

[12] Pontus Andersson. Future-proofing video game agents with reinforced learning and unity ml-agents. Degree Project, Luleå University of Technology, 2021. https://www.diva-portal.org/smash/get/diva2:1605238/FULLTEXT01.pdf.

[13] Darbinyan R. How artificial intelligence can empower the future of the gaming industry. Forbes Magazine, https://www.forbes.com/sites/forbestechcouncil/2022/07/13/how-artificial-intelligence-can-empower-the-future-of-the-gaming-industry/, 2022. accessed: 16 Jan 2023.

[14] E. Eliaçık. Ai in gaming: A complete guide. https://dataconomy.com/2022/04/artificial-intelligence-games/, 2022. accessed: 16 Jan 2023.

[15] Wim Westera, Rui Prada, Samuel Mascarenhas, Pedro Santos, João Dias, Manuel Guimarães, Konstantinos Georgiadis, Enkhbold Nyamsuren, Kiavash Bahreini, Zerrin Yumak, Chris Christyowidiasmoro, Mihai Dascalu, Gabriel Gutu-Robu, and Stefan Ruseti. Artificial intelligence moving serious gaming: Presenting reusable game ai components. *Education and Information Technologies*, 25, Jan. 2020.

[16] Iqbal H. Sarker. Ai-based modeling: Techniques, applications and research issues towards automation, intelligent and smart systems. *SN Computer Science*, 3, Feb. 2022.

[17] G. Chen, W.and Ren, Q. Cao, J. Song, Y. Liu, and C. Dong. A game-theory-based approach to modeling lane-changing interactions on highway on-ramps: Considering the bounded rationality of drivers. *Mathematics*, 11, Jan. 2023.

[18] Marín-Vega, Humberto Giner Alor-Hernández, Luis Omar Colombo-Mendoza, Cuauhtémoc Sánchez-Ramírez, Jorge Luis García-Alcaraz, and Liliana Avelar-Sosa. Zeus – a tool for generating rule□based serious games with gamification techniques. *IET Software*, 14:88–97, Apr. 2020.

[19] Jianbin Fu, Diego Zapata, and Elia Mavronikolas. Statistical methods for assessments in simulations and serious games. *ETS Research Report Series*, 2014(2):1–17, Jul. 2014.

[20] Brandon Antonette. Decision trees in video games. `https://medium.com/@antoneb/decision-trees-in-video-games-3ea3f251f96e`, 2019. accessed: 16 Jan 2023.

[21] Vijay Kanade. What is machine learning? definition, types, applications, and trends for 2022. `https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/`, 2022. accessed: 16 Jan 2023.

[22] Michael Lanham. *Learn unity ML-agents - fundamentals of unity machine learning: Incorporate New Powerful ML algorithms such as deep reinforcement learning for games*. Packt Publishing Ltd., Birmingham, UK, 1 edition, 2018.

[23] Aline Dobrovsky, Uwe M. Borghoff, and Marko Hofmann. Applying and augmenting deep reinforcement learning in serious games through interaction. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(2):198–208, Mar. 2017.

[24] What are machine learning models? `https://www.databricks.com/glossary/machine-learning-models`, 2022. accessed: 16 Jan 2023.

[25] What is machine learning: Definition, types, applications and examples. `https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/`, 2019. accessed: 16 Jan 2023.

[26] What is machine learning and why is it important? https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML, 2021. accessed: 16 Jan 2023.

[27] Yinglin Duan, Tianyang Shi, Zhengxia Zou, Jia Qin, Yifei Zhao, Yi Yuan, Jie Hou, Xiang Wen, and Changjie Fan. Semi-supervised learning for in-game expert-level music-to-dance translation. *CoRR*, Sep 2020. https://arxiv.org/abs/2009.12763.

[28] Xiaojin Zhu and Andrew B. Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.

[29] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[30] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing, 2017.

[31] A. Andrade. Game engines: a survey. *EAI Endorsed Transactions on Game-Based Learning*, 2:150615, Nov. 2015.

[32] Afzal Hussain, Haad Shakeel, Faizan Hussain, Nasir Uddin, and Turab Ghouri. Unity game development engine: A technical survey. *University of Sindh Journal of Information and Communication Technology*, 4, Oct. 2020.

[33] Unity compatible platforms. retrieved from. https://unity.com/. accessed: 22 Dec 2022.

[34] M. Foxman. United we stand: Platforms, tools and innovation with the unity game engine. *Social Media + Society*, 5, Nov. 2019.

[35] What makes unity so popular in game development? https://www.arnia.com/what-makes-unity-so-popular-in-game-development/. accessed: 22 Dec 2022.

[36] The scene view. https://docs.unity3d.com/Manual/UsingTheSceneView.html. accessed: 22 Dec 2022.

[37] The game view. https://docs.unity3d.com/Manual/GameView.html. accessed: 22 Dec 2022.

[38] The hierarchy window. https://docs.unity3d.com/Manual/Hierarchy.html. accessed: 22 Dec 2022.

[39] The inspector window. https://docs.unity3d.com/Manual/UsingTheInspector.html. accessed: 22 Dec 2022.

[40] The project window. https://docs.unity3d.com/Manual/ProjectView.html. accessed: 22 Dec 2022.

[41] The toolbar window. https://docs.unity3d.com/Manual/Toolbar.html. accessed: 22 Dec 2022.

[42] Unity scripting. https://docs.unity3d.com/Manual/ScriptingSection.html. accessed: 22 Dec 2022.

[43] Unity visual scripting. https://docs.unity3d.com/Manual/com.unity.visualscripting.html. accessed: 22 Dec 2022.

[44] Script creation and controll. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. accessed: 22 Dec 2022.

[45] Monobehaviour.start() function. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html. accessed: 22 Dec 2022.

[46] Monobehaviour.awake() function. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html. accessed: 22 Dec 2022.

[47] Monobehaviour.fixedupdate() function. https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html. accessed: 22 Dec 2022.

[48] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange.

Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[49] Kaan Baris Jari Hanski, Biçak. An evaluation of the unity machine learning agents toolkit in dense and sparse reward video game environments. Bachelor Thesis, Uppsala University Department of Game Design, 2021. https://www.diva-portal.org/smash/get/diva2:1563588/FULLTEXT01.pdf.

[50] Lucas A.E Pineda Metz. An evaluation of unity ml-agents toolkit for learning boss strategies. Master's thesis, Master's Degree Project, Reykjavík University, 2020. http://hdl.handle.net/1946/37111.

[51] Unity-technologies. training ml-agents. https://github.com/Unity-Technologies/ml-agents/blob/release_20_branch/docs/Training-ML-Agents.md#training-with-mlagents-learn, 2022. accessed: 28 Jan 2023.