HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Designing and creating a 3D RPG Action game using Unity Engine

NGUYỄN HOÀNG TÙNG
tung.nh187282@sis.hust.edu.vn

Major: INFORMATION TECHNOLOGY

Supervisor:	Ph.D. Trịnh Anh Phúc	_____________
		Signature
Department:	Computer Science	
School:	Information and Communications Technology	

HANOI, 08/2023

ACKNOWLEDGMENTS

I would like to publicly thank everyone who helped finish my thesis and to express my sincere gratitude and appreciation. First and foremost, I want to express my deep gratitude to my supervisor, Dr. Trịnh Anh Phúc, lecturer at School of Information and Communication Technology at Hanoi University of Science and Technology, for all of his support, encouragement and insighful advices for me throughout this project. His knowledge and commitment have been crucial in determining the focus and caliber of this thesis. A special thank to all my friends and colleagues for sharing ideas and experiences with me along with their steadfast support. Our camaraderie has been a great motivation for me to conclude this work. And last but not least, I extend my appriciation to my family, espcially my parents, who have showed unwavering love and encouragement for me during this challenge. Without them, I would have not been able to make it this far. Without the assistance and participation of everyone mentioned above, this thesis would not have been feasible. This journey has been made rewarding and fulfilling thanks to their help. I am sincerely grateful for their presence in my life and the role they played in this academic achievement.

ABSTRACT

As the society grows and the need of entertainment keeps elvolving from time to time, gaming has appeared as one of media in this era of technology, especially to teenagers. Captured the oppurtunity, since 2010s, Vietnam has facilitated the gaming industry, allowed thousands of companies to make their ways to create several of Vietnamese-branded games. In fact, Vietnam is among the top in the world in terms of developing and publishing mobile games in the international market. However, while mobile games are thriving, other platforms such as PC, Playstation or Nintendo Switch aren't really getting enough attention. Thus, developers for this market have not been successful as those in mobiles. Understanding this problem, I want to contribute to make a change to this situation by making quality games to help people have to more insighful look into games on other platforms, specifically PC in this case. Inspired by classics like Diablo or War of Warcraft I have created this game "Knights and Wizards", a medival RPG single-player game where players play the role of a knight on his journey to fight enemies from the neighbour kingdoms. An unique experience with different setting of medival environments supported by visually appealing graphics along with a difficult challenge created by dangerous kinds of enemies are what the game hope to achieve. Powered by Unity engine, a efficient tool in creating games, the product ensures it can provide the best performance to players, improve the overall quality with fluent and cohesive gaming experience. In the end, despite of not being something that groudbreaking in terms of videogames, I hope this project can deliver a change of pace to people, and help them get a better look at the future of PC games.

Student implement

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION **1**

1.1 Motivation 1

1.2 Objectives and scope of the graduation thesis 3

1.3 Tentative solution 4

1.4 Thesis organization 5

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS **6**

2.1 Status survey 6

2.2 Functional Overview 7

2.2.1 General use case diagram 7

2.2.2 Detailed use case Player Controller 8

2.2.3 Detailed use case AI Controller 9

2.2.4 Detailed use case Health 9

2.2.5 Detailed use case Base Stats 10

2.2.6 Detailed use case Save Data 10

2.2.7 Profession process 10

2.3 Functional description 12

2.3.1 Use case Player Moving description 12

2.3.2 Use case Player Saving description 12

2.3.3 Use case Player Equipping Weapon description 13

2.3.4 Use case Player Attacking description 13

2.3.5 Use case Enemy Patrolling description 14

2.3.6 Use case Enemy Attacking Player description 15

2.4 Non-functional requirement 15

CHAPTER 3. METHODOLOGY 16

3.1 Unity Engine 16

3.1.1 Scenes 17

3.1.2 Terrain 17

3.1.3 Rigibody 17

3.1.4 Navigation Mesh 18

3.1.5 Scriptable Object 18

3.1.6 Events 19

3.2 Lazy Evaluation 20

CHAPTER 4. EXPERIMENT AND EVALUATION 21

4.1 Architecture design 21

4.1.1 Software architecture selection 21

4.1.2 Overall design 23

4.1.3 Detailed package design 24

4.2 Detailed design 28

4.2.1 User interface design 28

4.2.2 Layer design 30

4.2.3 Database design 34

4.3 Application Building 38

4.3.1 Libraries and Tools 38

4.3.2 Achievement 38

4.3.3 Illustration of main functions 39

4.4 Testing 42

4.5 Deployment 44

CHAPTER 5. SOLUTION AND CONTRIBUTION 45

5.1 Navigation Mesh 45

5.2 AI Controller.. 47

5.3 Lazy Evaluation 49

5.4 Unity Optimization 50

CHAPTER 6. CONCLUSION AND FUTURE WORK 53

6.1 Conclusion.. 53

6.2 Future work.. 54

REFERENCE 56

LIST OF FIGURES

Figure 2.1 General use case diagram 7
Figure 2.2 Player Controller use case 8
Figure 2.3 AI Controller use case 9
Figure 2.4 Health use case 9
Figure 2.5 Base Stats use case 10
Figure 2.6 Save Data use case 10
Figure 2.7 Profession process of the game 11

Figure 4.1 Game Engine package 24
Figure 4.2 Levels package 25
Figure 4.3 Character package 26
Figure 4.4 Items package 27
Figure 4.5 Saving System package 28
Figure 4.6 Illustration for the font used 28
Figure 4.7 Game menu illustration 29
Figure 4.8 Gameview illustration 30
Figure 4.9 Base Stats class 31
Figure 4.10 Health class 31
Figure 4.11 Mover class 32
Figure 4.12 Fighter class 32
Figure 4.13 Saveable Entity class 33
Figure 4.14 Player Controller class 33
Figure 4.15 Exeperience class 34
Figure 4.16 AI Controller class 34
Figure 4.17 Databse diagram 35
Figure 4.18 Scriptable object Progression in Editor 37
Figure 4.19 Main Menu of the game 39
Figure 4.20 Gameplay display 40
Figure 4.21 Different cursors being shown for different types of action . 40
Figure 4.22 Players can choose different weapons to equip 41
Figure 4.23 Attacking enemy 41
Figure 4.24 Teleporting through portal 42
Figure 4.25 Distant attack exception 43

Figure 5.1 Scene after being baked by NavMesh 46
Figure 5.2 CanMoveTo() method 46

Figure 5.3 GetLength() method 47
Figure 5.4 PatrolBehaviour() method 48
Figure 5.5 IsAggrevated() Method 48
Figure 5.6 How to store and intialize lazy value 50

LIST OF TABLES

Table 2.1 Player Moving use case 12
Table 2.2 Player Saving use case 12
Table 2.3 Player Equipping Weapon use case 13
Table 2.4 Player Attacking use case 13
Table 2.5 Enemy Patrolling use case 14
Table 2.6 Enemy Attacking use case 15

Table 4.1 Lists of used libraries and tools 38
Table 4.2 Info of product 39

LIST OF ABBREVIATIONS

Abriviation	**Full Expression**
AI	Artifactual Intteligent
API	Application Programming Interface
CPU	Central Processing Unit
ECS	Entity - Component - System
FPS	First Person Shooter
FPS	Navigation Mesh
Indie	Indepedent
OS	Operating System
PC	Personal Computer
RPG	Role Playing Game
UI	User Interface

CHAPTER 1. INTRODUCTION

1.1 Motivation

Different sorts of entertainment have arisen to fulfill the growing desire for human enjoyment in the modern period. Traditional forms of entertainment like music, film, and art have evolved to meet society's needs. Gaming, on the other hand, have also appeared alongside them and are predicted to usher in a new revolution in the digital age. In the sphere of technology, game development is now a hot topic, and it has grown in significance over time.

The first electronic games in the world were born in the mid-20th century [1], bringing a new wave to the entertainment industry, particularly targeting the younger generation. Since then, millions of different games have been released, with countless companies ranging from giant corporations to indie developers working together to develop a diverse and rich range of electronic game products. Game genres have become incredibly diverse and varied, existing in different forms on platforms ranging from arcade machines to early-generation consoles and computers. The demands of players for game quality have become increasingly stringent over time, requiring game companies to push themselves to innovate and surpass the limitations in terms of graphics, sound, gameplay, and more. They never cease to challenge developers to become even more creative and unique.

The game industry revolution truly exploded in the early 21st century [2], and at the same time, it began to make its way into Vietnam. The first games to enter the Vietnamese market were mainly online games, targeting primarily young teenagers. Timeless classics such as MU Online[3], Audition[4], and Gunbound[5] can be mentioned. Technology companies saw the potential of electronic games in Vietnam and started acquiring them, exemplified by FPT [6] and VNG [7]. By the mid-2010s, the rise of hundreds of mobile game development companies in our country, primarily focusing on casual and arcade games, truly ignited the game industry. Society's perspective on electronic games has been increasingly positive, with previously held negative biases gradually fading away and making room for encouragement, recognition, and deserved support. With an open-minded and modern outlook and the support of investors, more and more game developers in Vietnam are venturing into entrepreneurship to pursue their dreams.

The relentless increase of both large and small game companies in Vietnam in recent years is truly encouraging. However, a common goal can be observed among most of these companies: they are all targeting the mobile market with casual games. As the matter of fact, when it comes to creating and releasing mobile games on the global market, Vietnam is among the top countries [8]. This is a safe direction, as casual games are simple, do not require much time or effort from players, and are suitable for entertainment purposes after long hours of work or study. Moreover, they appeal to all age groups, captivating players with their simplicity and attractiveness. Releasing games on mobile platforms is also more accessible. Most of these games are free, accompanied by advertisements or in-app purchases. Perhaps in the future, mobile casual games will continue to maintain a certain level of popularity and receive attention from the majority of players. However, that simplicity rarely leaves a profound impression on players, and most of them will eventually feel bored, especially the young audience of the new generation. They demand more from a game, seeking pure recreational entertainment and desiring to experience new emotional challenges they have never encountered before. It is for these needs that many new game genres have emerged, running on larger platforms such as computers, to meet the operational requirements of the games. While the game market on PC and console platforms is booming and vibrant in foreign countries, in Vietnam, it has not yet become widespread. The majority of computer games with a large player base are imported online games, while offline games only attract a small portion of passionate and genuinely interested players. There do exist offline game development companies on the computer platform, with titles such as Hoa (platformer) [9], 7554 (FPS) [10], or recently Thần Trùng (horror) [11], which have been heavily invested in and hold high expectations. Although they have garnered some attention and recognition within the domestic and international communities, they still haven't received much deserved attention and support due to the limited gaming culture in our country. However, they are pioneers paving the way for a new direction in the game industry in Vietnam, hoping that in the near future, Vietnamese audiences will have a fresh and modern perspective on electronic games, recognizing them not only as a simple digital entertainment tool but also as a form of art in the new era, deserving of respect, to provide opportunities for many game developers in our country to truly unleash their creativity and bring a fresh breeze to the Vietnamese electronic game scene.

Understanding this issue, I want to contribute to a part of the future of the game industry in Vietnam and help people have a new and modern perspective on it. For that reason, instead of aiming to create an Android/IOS platform game, I have chosen the PC platform for my game to provide more complex features. With the help of Unity, one of the most famous game design engines today, I have designed and developed an Action RPG game set in a medieval backdrop, with a top-down perspective and mouse input, incorporating various features. While this game may not be intricate or perfect, I hope it can bring players a new and exciting experience.

1.2 Objectives and scope of the graduation thesis

As mentioned above, currently, the majority of the game market in Vietnam consists of casual mobile games and online esports games. The frequent use of mobile devices and the interest in participating in or watching tournaments contribute to this trend. The presence of single-player games on computers is still limited, as the attention for these games has not been properly placed, despite the efforts of developers to explore the potential of these genres. Two notable single-player computer games in Vietnam are "Hoa" by PM Studios and "7554" by Hiker Games. With "Hoa," PM Studios drew inspiration from popular platformers like Ori and The Blind Forest [12] to create a dreamy world with eye-catching graphics, soothing soundtracks, and a gentle and adorable storytelling approach. "7554" by Hiker Games, released in 2011, is an FPS game inspired by the glorious victory of the Vietnamese people in the battle of Điện Biên Phủ, delivering an intense and heroic war experience. Both games have made a strong impression not only on Vietnamese audiences but also garnered attention from the international gaming community. In recent years, we have also seen the release of various first-person horror games from indie companies such as Thần Trùng and upcomings like Đồng Cỏ Lau or Tai Ương [13]. They all follow and learn from the success of foreign games, aiming to provide Vietnamese players with new perspectives and unique experiences compared to the majority of games in the current market. However, there have also been unavoidable failures. After the success of "7554," Hiker Games aimed to create another FPS game inspired by the resistance war against the American Empire called "300475" on a much larger scale. However, their fundraising efforts failed, and the project had to be canceled, causing disappointment among Vietnamese gamers [14].

Although single-player computer games in our country have not boomed like mobile games at the moment, I believe that in the future, as the entertainment needs of gamers become more demanding and Western cultures continue to influence us, there will be a change in perception and more support for these genres. Therefore, the game I have designed is not just a form of entertainment on a computer but also a step towards the future game market in our country.

"Knights and Wizards" is a 3D action role-playing game with a top-down third-person perspective inspired by the famous game "Diablo" [15]. Set in the medieval period, players will take on the role of a knight fighting against enemies from the neighboring kingdom. The game primarily uses mouse input, where the mouse is responsible for most of the player's actions, from movement to attacks and other actions. Players will have access to a diverse weapon system stored in personal inventories. The player's strength will increase gradually through experience gained during gameplay. The game also features a convenient save-load system that allows players to save their progress. The graphics are simple yet appealing, creating multiple levels with constantly changing environmental terrain to ensure players don't get bored. Additionally, there are various animations for different actions and enjoyable sound effects. The game's difficulty increases over time, with a variety of enemies with their own AI for movement and attacks, providing real challenges for players.

Despite not being a breatkthrough in term of videogames nor standing out among new games, but comparing to the majority of current casual games, Knights and Wizards hopes it can bring a execting experience, a change of pace in their library of games.

1.3 Tentative solution

To serve the most efficient design and development of the game, I use the Unity engine. Unity [16] is a game design tool that was created in 2005 and caters to both 2D and 3D on various platforms, with the main programming language being C#. With its user-friendly and accessible interface, especially for beginners, Unity has become the most popular game engine. With countless built-in features, APIs, and rich libraries, Unity provides developers with a range of powerful tools to build games.

With the help of the Unity engine and C# language, we will address the game's feature challenges. This includes building a character system in the game, including both players and enemies, sharing fundamental features such as movement, attacking, and a set of statistics to track progress. The enemies will have an AI system to autonomously perform behaviors like patrolling or attacking the player when in proximity. For players, the control system will be mouse-based, distinguishing actions such as movement and attacking. We will design a diverse weapon system that can be used by both players and enemies, with the player having a separate inventory system to store weapons. We will implement a save-load feature when transitioning between game levels and allow users to manually save their game progress. Graphics, animations, sound effects, and some APIs for the design process will be sourced from free resources available online, primarily from the Unity Asset Store [17].

Ultimately, the desired outcome is a complete game with all the mentioned features. While the game may not be overly complex, it should provide entertainment and a new experience for users. The captivating and immersive aspects of the game will be discovered by the players themselves during their journey alongside the main character.

1.4 Thesis organization

The remaining part of the thesis is presented as follows:

Chapter 2 discusses the practicality of the game, provides an overview and description of the features that will appear in the game, and illustrates the business process used in the design using various types of charts.

The technology and knowledge employed in the product will be introduced in Chapter 3. This chapter explores Unity's features as well as the custom technologies created for the game's development. Understanding and making the most of Unity's features is key because they will be used extensively in the creation of the game. The rationale for selecting the best techniques for the design will be highlighted, and the scripting methods employed will also be discussed.

Chapter 4 delves into the detailed system design for the game. The cornerstone of the development process, the construction process, and the most fundamental details about the product are all covered in this chapter. This chapter will show the architectural model and the designs for the game's packages and classes. Despite being less important, the game's database will nonetheless be discussed and shown in this chapter. We will then continue with the project's standout features and test them. Finally, some more information will be given, such as product specifications or details on the game's release and distribution. The thesis' longest chapter is this one.

The thesis' most significant contributions are highlighted in Chapter 5. These contributions are the most noteworthy successes that I am most proud of throughout the thesis' implementation. They signify a greater comprehension of the programming methods used to make the product function as efficiently as possible.

The conclusion chapter will offer remarks on the outcomes and comparisons to related products on the market. I will be able to see where I need to make improvements and determine how I should proceed with my future development as a result of this.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

2.1 Status survey

Games have been developed to fulfill the growing demand for amusement in recent decades, as was discussed in the preceding chapter. The Vietnamese market has evolved in recent years, showing a more accepting attitude toward gaming, which has given many local game developers the opportunity and inspiration to distribute their works. Arcade and casual mobile games continue to command the majority of media attention in Vietnam. It is true that games on platforms other than mobile haven't really stood out or drawn much notice among those created by Vietnamese developers. Online games that have been around for a while and built a solid reputation make up the majority of the player bases for PC games. This pattern is not specific to Vietnam but applies to Southeast Asia in general.

However, there are opportunities for other platforms as well. There is a younger generation that shows special interest in PC games such as RPGs, open-world, FPS, and various console games. As mentioned above, more and more game developers are emerging in Vietnam for PC. Although their success has not been resounding, it will pave the way and encourage others to continue exploring the untapped potential in the Vietnamese market.

Most of the games developed on the PC platform in Vietnam belong to indie companies. The most famous Vietnamese games like "Hoa" and "7554" use popular and widely-used engines like Unity and Vision [18]. The graphics and art direction of these games are influenced by their sources of inspiration. "Hoa" showcases a dreamy and sparkling style inspired by "Ori and The Blind Forest," while "7554" portrays the rustic, authentic, and brutal nature of war like "Call of Duty" [19]. Additionally, there are many other PC games, mainly in the arcade or puzzle genre, developed by young students who are bringing a breath of fresh air to the gaming industry in our country.

For all games, the most important aspect to evaluate their success undoubtedly lies in gameplay. A game's gameplay doesn't necessarily have to be overly complex, but it must ensure engagement, attractiveness, and provide clear objectives or challenges tailored to the players' needs. Other elements such as sound, graphics, and direction also significantly contribute to the completeness of a game. A good game may not need to be perfect in every aspect, and breakthroughs don't always yield effective results, but it is certain that the fundamental element it must guarantee is gameplay that appeals to players. This is not an easy task for inexperienced game developers who lack significant experience.

2.2 Functional Overview

2.2.1 General use case diagram

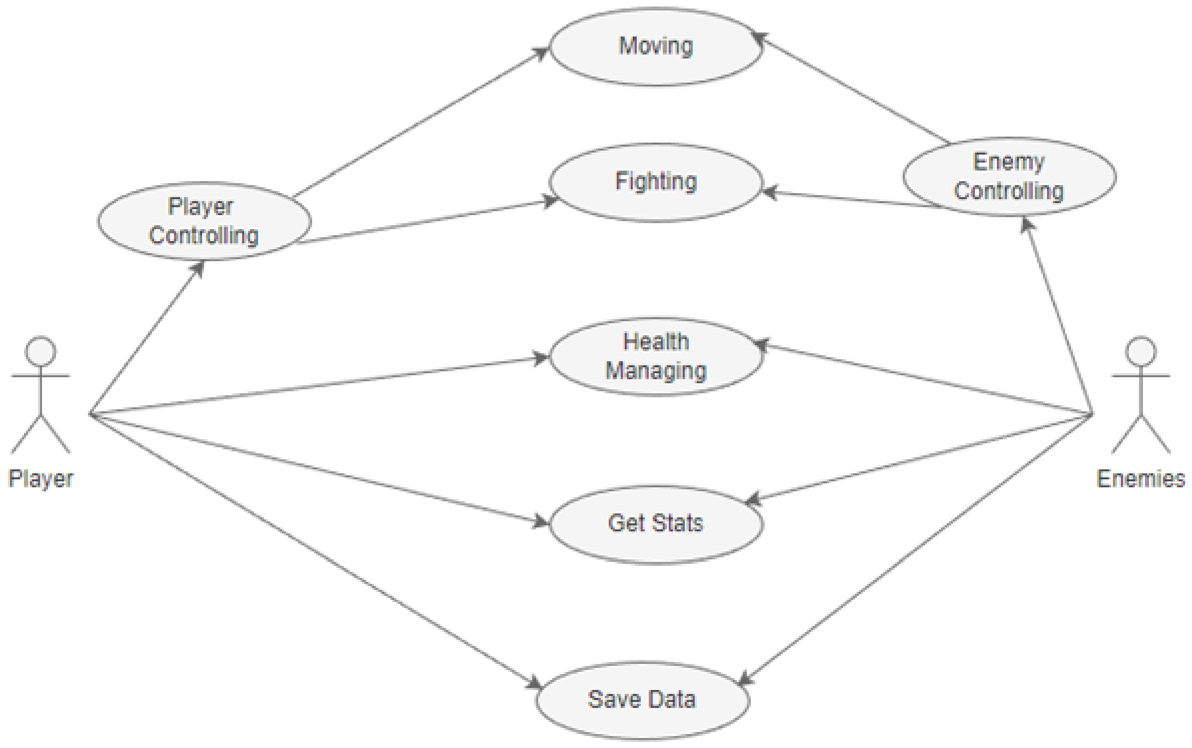


Figure 2.1: General use case diagram

Throughout the entire gameplay process, there will be two main entities involved: the player and enemies. The player represents the primary user, while enemies will represent various types of adversaries.

The Player Controlling and Enemy Controlling are the two main use cases used for controlling the player and enemies, respectively. These use cases play a crucial role in controlling the actions of the entities during gameplay. The specific actions include the Moving (responsible for movement on the map) and the Fighting (responsible for the entire combat system).

The Health Managing use case is responsible for managing the character's health in the game, primarily determining when the character will die.

Get Stats is a use case that allows characters to access the game's stat storage system. This system provides a set of detailed statistics for each character type in the game.

The Save Data stores information about the character when transitioning between levels or playthroughs.

2.2.2 Detailed use case Player Controller

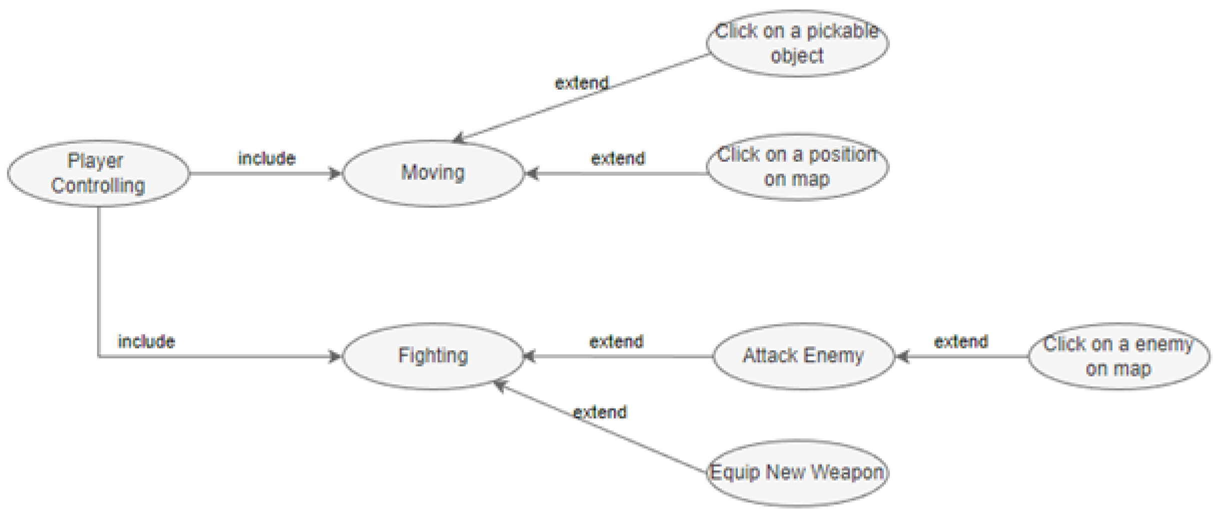


Figure 2.2: Player Controller use case

The player will perform their actions through the Player Controlling use case. The primary input used will be the left mouse button. Depending on the attribute pointed to by the mouse cursor, different actions will be triggered.

The Player Controlling consists of two specific use cases: Mover and Fighter. The Moving helps the player move by clicking the mouse on a position on the map, and the player will move to that position. If the player clicks on an enemy, the Fighting use case will be executed, and the player will move to the enemy's position and attack them. Additionally, the Fighter is responsible for equipping weapons for the player.

2.2.3 Detailed use case AI Controller

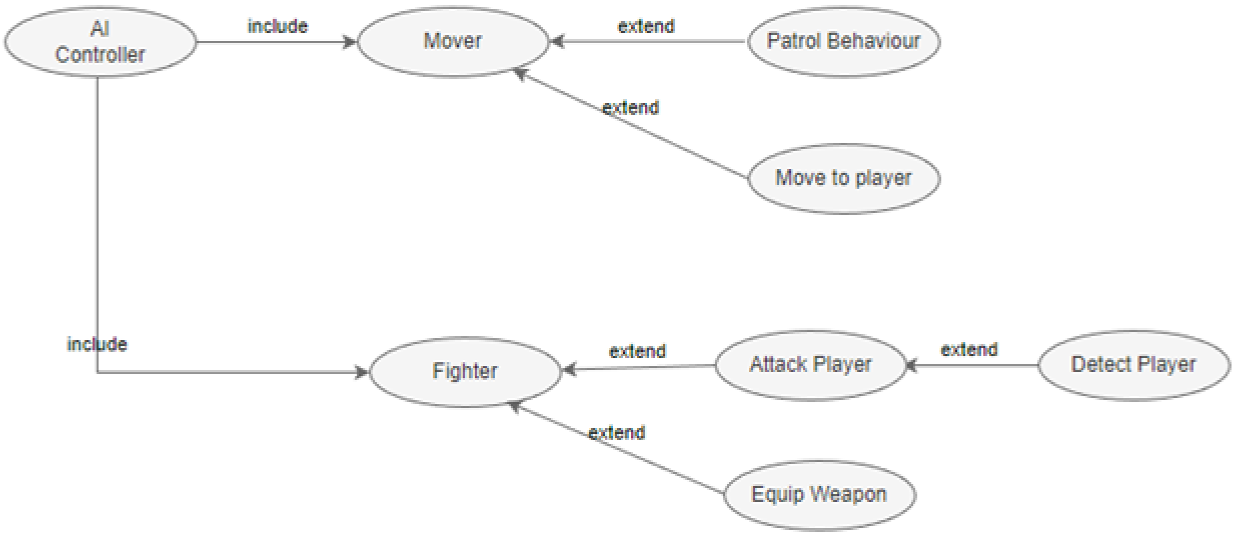


Figure 2.3: AI Controller use case

Responsible for controlling all actions of the enemies. Similar to the previous use case, the Enemy Controlling also commands the two main use cases for the corresponding entities: Mover and Fighter. For the Moving, enemies will have a patrolling path to move along. If they detect the player within the allowed range, they will move towards the player. The Fighting is in charge of enemy attacks on the player.

2.2.4 Detailed use case Health

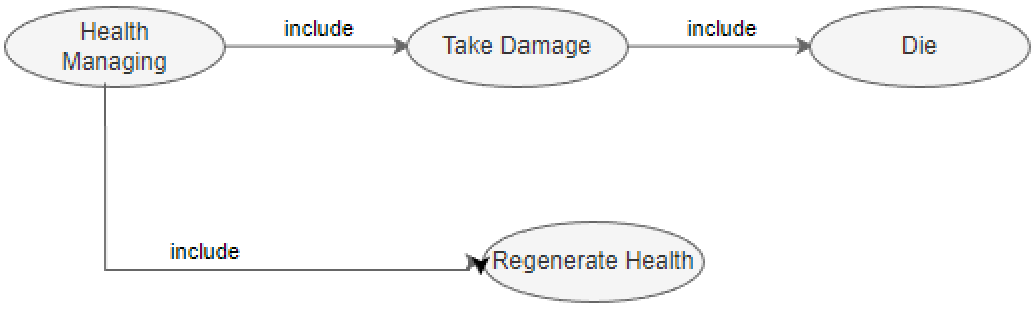


Figure 2.4: Health use case

Health Managing describes the changes in the character's health during gameplay. When the character's health reaches zero, they die. In the case of the player character, the game will end. Additionally, the player will have a health regeneration capability.

2.2.5 Detailed use case Base Stats

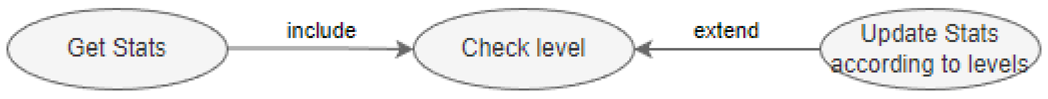


Figure 2.5: Base Stats use case

Get Stats allow entities in the program to access game data to retrieve their own statistics. Each character will have attribute stats depending on two factors: the specific character type and the current level of the character.

2.2.6 Detailed use case Save Data

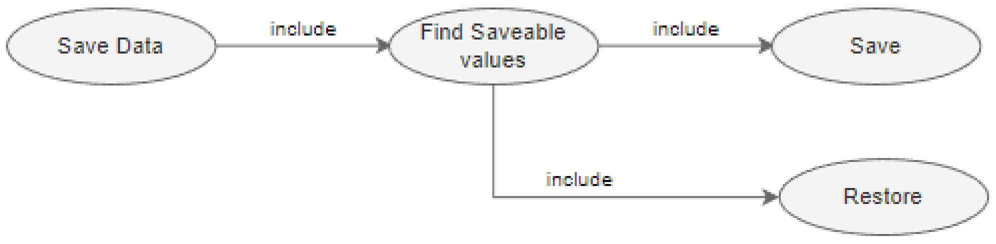


Figure 2.6: Save Data use case

Last is the Save Data use case. It will check all check all the values belong the character that required to be saved and carried over. These information will be stored locally and can be called by the user.

2.2.7 Profession process

The player will start the game. At the main menu screen, the player has two options: they can either start a completely new game with a new save file or continue the game if a previous save file exists. If the player chooses to start a new playthrough, they will begin at the first level.

In each level, the player will use the left mouse button to control their actions. They click to move to reachable positions on the screen. Clicking on an enemy will attack them. The player can also move around the map to find weapons or beneficial items and use them. Defeating an enemy will reward the player with corresponding experience points. These experience points accumulate to level up the player. When reaching certain thresholds, the player will level up. Stats such as health and damage will gradually increase with the player's level. The player will progress through each level to reach the final stage, defeat the boss, and win the game. If the player is attacked and defeated, the game will end. At that point, they have two options: they can continue from the nearest save point or start a completely new playthrough.

During the process, the saving system will work automatically or when invoked by the player. All attributes in the level will be saved in real-time. The saving process occurs when the player selects the save button or when the player transitions between levels, in which case the system will automatically save the progress.

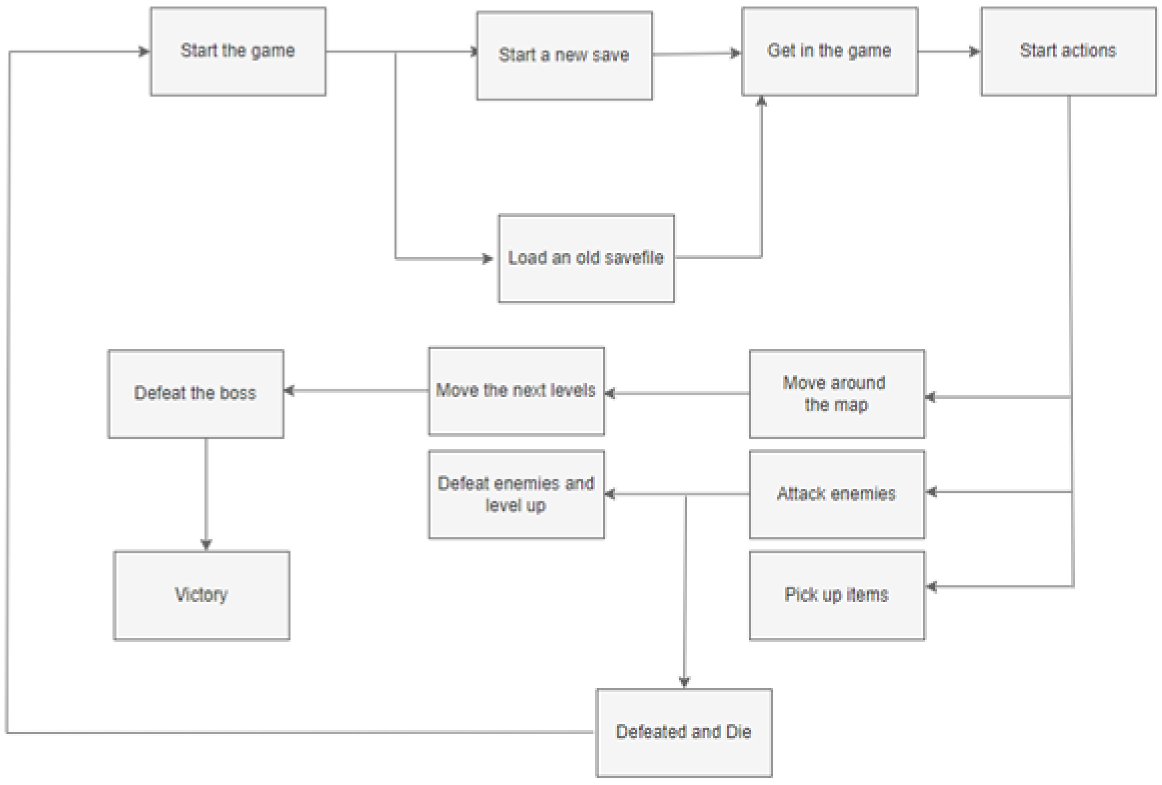


Figure 2.7: Profession process of the game

2.3 Functional description

2.3.1 Use case Player Moving description

Usecase	Details
Name	Player Moving
Description	Move player to a certain position
Actor	Player
Pre-condition	Player is in gameplay Player is alive The destination is movable
Post-condition	
Flow of events	Player move the cursor to valid position Click the left mouse button Player move to that position

Table 2.1: Player Moving use case

2.3.2 Use case Player Saving description

Usecase	Details
Name	Player Saving
Description	Player saves the progression
Actor	Player
Pre-condition	Player is alive
Post-condition	A new savefile is created/ Overdrive an old savefile
Flow of events	Player press the S button on keyboard Or player goes through a portal

Table 2.2: Player Saving use case

2.3.3 Use case Player Equipping Weapon description

Usecase	Details
Name	Player Equiping Weapon
Description	Equip the player with a new weapon
Actor	Player
Pre-condition	Player is in gameplay Player is alive There is a pickable weapon appearing on the map
Post-condition	Player is equipped with that weapon
Flow of events	Player move the cursor on the pickable weapon Player move towards the weapon Player equip the new weapon
Exceptional path	At stage 3, if the player already has a weapon, the current will be destroyed before equipping new weapon

Table 2.3: Player Equipping Weapon use case

2.3.4 Use case Player Attacking description

Usecase	Details
Name	Player Attacking
Description	Make player fight a certain target
Actor	Player
Pre-condition	Player is in gameplay Player is alive The destination is movable The target must be in the valid range of attacking
Post-condition	
Flow of events	Player move the cursor on the enemy Click the left mouse button Player move towards the selected enemy Player attack the enemy

Table 2.4: Player Attacking use case

2.3.5 Use case Enemy Patrolling description

Usecase	Details
Name	Enemy Patrolling
Description	Enemies automatically moving along their paths to look for player
Actor	Enemy
Pre-condition	Player is in gameplay Player is alive Enemy is alive Enemy has a patrol path
Post-condition	Player is equipped with that weapon
Flow of events	Enemy moves to the closet point of their patrol path Enemy stands there for a moment Enemy moves to the next point
Exceptional path	At any stage, if player is spotted in the range of enemy, the enemy will aggrevate nearby allies and moves toward the player to attack If the player moves out of the chasing range after a certain amount of time, enemy will return to its last position before chasing and continue the patrolling

Table 2.5: Enemy Patrolling use case

2.3.6 Use case Enemy Attacking Player description

Usecase	Details
Name	Enemy Attacking
Description	Enemy chasing and fighting the player
Actor	Enemy
Pre-condition	Enemy is alive Player is alive Player moves into the chasing range of the enemy (or enemy is aggrevated by nearby enemies)
Post-condition	
Flow of events	Enemy chases towards the player Enemy reaches range of its attacking weapon Enemy starts attacking player
Exceptional path	At stage 3, during attacking, if player moves out of attacking range, enemy will stop attacking, chase again till reaching within range At stage 3, if player gets out of chasing range after a certain amount of time, enemy will stop chasing, stand there for a moment before returning to its previous location and continue patrolling

Table 2.6: Enemy Attacking use case

2.4 Non-functional requirement

One non-functional requirement ís essential to the game is the performance. In term of optimization, Unity doesn't provide the best tools and options for the developers to enhancing the quality of the game. Still, there are things to do that can help with the performance to significantly increase the overall experience for the player. The details of the optimiztion for will be carefully accessed in chapter 5, Solution Contribution.

CHAPTER 3. METHODOLOGY

3.1 Unity Engine

The major tool used to design and create this game was the Unity engine. Unity Engine is a powerful, cross-platform game development and application tool developed by Unity Technologies. Currently, Unity is one of the most popular tools in the game and application industry. Its availability on various platforms such as Windows, macOS, Linux, along with support for 2D graphics, 3D graphics, virtual reality (VR), augmented reality (AR), and its user-friendly interface for beginners, as well as numerous powerful tools for application development, have made Unity the most famous engine in the present era. Unity reduces dependence on programming through its drag-and-drop features when working on its Editor. Many powerful features and tools in Unity are simplified by allowing users to directly interact with the screen instead of dealing with complex programming. This key factor has made Unity a highly popular game and application development tool. It provides a comprehensive development environment with an intuitive and user-friendly interface. Users can build their games and applications by dragging and dropping components, creating logical systems and controls, and implementing high-quality graphics and sound effects.

Unity provides a large and dynamic community with millions of members worldwide, including developers, artists, and designers. This community provides rich documentation, resources, suggestions, and support for Unity users. Learning materials, online courses, forums, and events are also abundant and help users enhance their skills and knowledge. This makes learning and using Unity easier and provides many opportunities for collaboration and knowledge sharing.

Unity uses C# as its main programming language, supporting high-quality 3D, 2D, AR, and VR graphics with graphics technologies such as DirectX, OpenGL, Vulkan. It provides powerful tools for creating and managing assets, setting up game logic using mathematics and physics, rendering high-performance images and animations, creating special effects, and handling events in the game. In addition to games, Unity is also a powerful tool for simulating real-world problems. By using mathematical logic, Unity can simulate real-life situations, making it easy for users to compare and evaluate these events as they occur in reality.

Unity supports a diverse range of platforms, allowing game and application development on operating systems such as Windows, macOS, Linux, Android, iOS, PlayStation, Xbox, Nintendo Switch, and more. This flexibility enables Unity to easily reach a vast number of players worldwide. From independent companies to leading global companies, it has contributed to the creation of many famous and successful games and applications such as Pokémon GO, Cuphead, Hollow Knight, Ori and the Blind Forest, and many more.

The game "Knights and Wizards" is built entirely on Unity and maximizes its features for design. Some prominent Unity technologies used in the development process are:

3.1.1 Scenes

Scenes are the core part of Unity. They represent levels, screens, or other elements in your game or application that the player or user can interact with.

All components of the game are contained within scenes: 3D or 2D objects, lighting, sound, UI (user interface), scripts, special effects, and more.

The game is built with multiple levels, where each level is a scene. Switching between scenes is done through player-triggered events supported by Unity. Scenes are linked together not only through movement but also by connecting data and states to create a seamless representation of the player's progress throughout the game.

3.1.2 Terrain

In the game, players control their characters to move on various types of terrains. The terrain of the game is built using Unity's terrain tool.

Terrain in Unity is a powerful feature for developing land and terrain environments in games and applications. It allows you to create diverse areas such as mountains, meadows, deserts, forests, and more, using built-in tools and functions in Unity. These tools enable you to adjust height, slope, and terrain by dragging, aligning, and using brushes.

Terrains use textures and surfaces to paint on the terrain. Depending on the design needs, using multiple textures and surfaces on the same terrain enhances diversity and realism in the environment. Additionally, the terrain tool provides a useful tree painting feature. Players create a desired tree library, then place them on the terrain and adjust density, size, and height as desired.

3.1.3 Rigibody

Characters in the game and many other objects are subject to the constraints of physics (gravity, collisions). Rigidbody handles this.

Rigidbody is a component in Unity used to simulate physics and handle collisions for objects in the game or application. Rigidbody is attached to objects to control their movement and interaction with physical forces. Rigidbody allows manipulation of movement, collisions, velocities, and other physical constraints in the game. It simulates realistic interactive situations or desired behaviors programmed by developers..

3.1.4 Navigation Mesh

Navigation Mesh is a powerful AI tool integrated into Unity. Navigation Mesh (NavMesh) in Unity is a feature that helps handle and navigate the movement of objects in the game. NavMesh is a 3D object created from a terrain model or a collection of objects, and it contains information about walkable and non-walkable areas in the environment. NavMesh is used to determine the path for characters, AI, or any moving objects in the game.

NavMesh creates a grid that spans the entire space of the game. This grid serves multiple purposes, with the primary task being the ability to move and find paths. Speed and distance traveled will be automatically calculated by NavMesh and applied to agents in the game.

Using NavMesh, we can select walkable areas or obstacles that hinder movement. All game objects processed with NavMesh are called agents. For dynamic agents, they will rely on the map grid built according to the player's desires to move on it. You can add, delete, or modify walkable and non-walkable areas on the NavMesh to create a suitable movement environment for your game. Unity provides functions and methods to determine and compute paths on NavMesh, including routing algorithms.

NavMesh in Unity provides a convenient and efficient way to navigate objects in the game, ensuring that they move smoothly and avoid collisions with obstacles. It is an important feature in building open-world, combat, adventure, and various other game genres. The game uses NavMesh attached to all characters to support movement on the terrain. Whether the player can move to a specific area or not, or whether AI is designed to allow enemies to move automatically and navigate, NavMesh handles it.

3.1.5 Scriptable Object

Scriptable Objects are an important feature in Unity that allows you to create and customize objects that can store game data and logic without needing a specific instance. It provides a flexible approach to managing and using data in the game. Scriptable Objects in this game are used primarily to store data for character classes and weapon types. Scriptable Objects act as a database for the game and can be accessed at any time.

Scriptable Object is a base class in Unity that allows you to create objects that can store data and functions like resources, configurations, states, and behaviors. They can be inherited to create custom objects for specific purposes. Scriptable Objects allow you to define and store data in basic data types such as integers, floats, strings, as well as custom data types you define. This enables you to store information and configurations for objects in the game. Scriptable Objects can also be created and used during game development or stored in data files. This helps you manage and reuse data easily.

One of the main benefits of Scriptable Objects is the ability to edit and customize data without changing the source code or rebuilding the game. You can change the values of Scriptable Object properties in the Inspector and save the changes without rewriting the code. Scriptable Objects are a powerful tool in Unity that allows you to manage game data and logic flexibly and easily. With Scriptable Objects, you can create flexible configurations, organize game data and logic, and increase reuse and reusability during development.

The game uses scriptable objects as a database to store data and parameters for all character classes in the game. Depending on different class types of different characters, variations of data can be easily changed and customized due to the flexibility of scriptable objects. Along with that, the system of weapon types and other pickable items in the game is also initialized from scriptable objects, depending on their purpose, they will be designed differently.

3.1.6 Events

Unity Event is a tool in Unity that allows you to create and manage events and functions related to those events. It helps create interactions and communication between objects and components in the game easily and flexibly.

During gameplay, different events will be called based on real-time interactions. For example, the attack event of characters will trigger functions such as weapon swing or projectile shooting, causing damage or receiving damage.

The significant point of using Unity Event is that it saves a considerable amount of system resources during game processing. Instead of having functions wait in continuous updates until the conditions are met, we can convert them into events, wait for the events to be triggered, and let Unity process the functions sequentially as required.

In addition, the game also utilizes other outstanding technologies of Unity such as Animator, Particle systems, Lightning, etc.

3.2 Lazy Evaluation

Lazy evaluation is a programming technique that allows you to defer computation or execution of a task until the result is actually required. In the context of game programming, lazy evaluation can be applied to optimize system resource usage and improve overall game performance.

When developing a game, there are many situations where computations or loading of resources are not needed immediately. Instead, lazy evaluation can be used to calculate or load resources only when they are actually needed. This helps minimize memory usage, CPU resource usage, and enhance game performance.

Lazy Initialization is the simplest way to implement lazy evaluation. Instead of initializing an object right from the start, we wait until it is accessed for the first time before creating it. This helps save resources and improve performance.

In the specific situation of programming this game, lazy evaluation plays an important role in handling data before and after the save/load process of the game. When a player saves their progress in the game, the necessary data will be stored. In the next session, if the player chooses to load the saved data, lazy evaluation will delay the computation of old data, waiting for the game scene to be fully loaded, and then restore and overwrite the remembered data from the previous session. Lazy evaluation ensures accuracy and avoids data hazards. Data hazard (also known as data dependency hazard) is an issue related to data conflicts during the execution of instructions. Data hazard occurs when there is data dependency between instructions, and the order of their execution causes conflicts or delays in accessing data. The specific handling of this issue will be presented in Chapter 5.

CHAPTER 4. EXPERIMENT AND EVALUATION

4.1 Architecture design

4.1.1 Software architecture selection

Unlike the majority of website and general application designs, game design involves certain differences in data management and processing. Therefore, the choice of architectural design for games will differ from models like MVC, MVP, and others. In game development, especially with games that use the Unity engine, the Entity-Component-System (ECS) model is the most optimal and effective choice [20]. Unlike the traditional object-oriented OOP architecture, ECS separates entities, components, and systems to allow programmers to easily manage and organize them in a game. Its advantages in improving performance, scalability, and maintainability, especially in complex and data-intensive games, make ECS the most outstanding software design model in game development. As its name suggests, ECS consists of three main components:

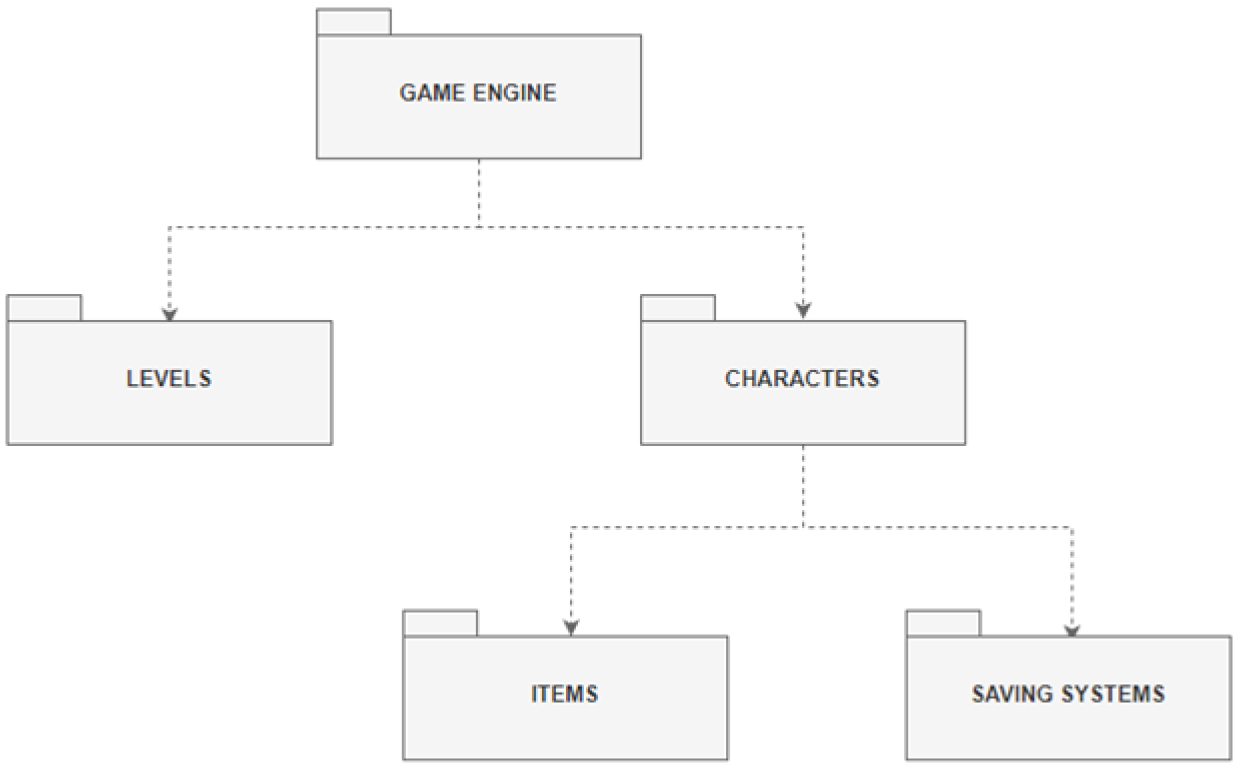
- Entity: An entity is an object or an object in your game. It serves as a container for components and does not contain any logic code. The entity is an abstract entity with no functions or methods, only a list of components attached to it.
- Component: A component is pure data and contains only data fields. It does not contain behavior or methods. Components represent specific attributes of an entity. For example, a Position component may contain information about the x, y, z coordinates of an object in 3D space.
- System: A system performs processing and logic for components. Each system performs a series of operations on specific components. The system does not store any state or information about the entity it processes. Instead, it directly accesses the necessary components from entities to perform calculations.

In game development, the ECS model offers several advantages: (i)Performance: Since components do not contain behavior, ECS minimizes unnecessary data access and optimizes memory layout. This leads to higher performance, especially in scenarios with many entities and components. (ii) Scalability: ECS makes it easier to expand and reuse code. You can easily add or remove new components without affecting existing system code. (iii) Parallel Processing: With systems directly operating on components of entities, parallel processing becomes easier. This helps increase processing speed and the game's ability to scale.

Overall, the Entity-Component-System model is a powerful and efficient architecture for game development that has become increasingly popular due to its ability to handle the complexities of modern games and improve performance on various platforms. In Unity, the ECS model is implemented using the Data-Oriented Technology Stack (DOTS) and Unity.Entities framework, enabling developers to build high-performance and scalable games. The game "Knights and Wizards" uses the ECS model in Unity to design its architecture. Each part of the model is presented as follows:

- Entity: In Unity, entities are represented as GameObjects, acting as containers for components. GameObjects represent all elements appearing in the scene, and they can be individual objects or prefabs with inherited children. Each GameObject contains specific components relevant to it. For example, objects in the scene like trees and houses are GameObjects that usually have only a few components, like Transform. More important GameObjects, like Characters, contain a series of components representing various attributes of characters in the game, further inherited by two prefabs - Player and Enemy - with their characteristic attributes.
- Component: Components inside entities represent specific characteristics of the entity. All components in Unity inherit from the Component class. The scripts (classes) we write to use on GameObjects are also components inherited from MonoBehaviour. Component values or attributes can be directly changed in the Editor. For instance, the Player GameObject will have numerous components serving different purposes, such as Unity components like Transform, MeshRenderer, NavMeshAgent, as well as custom-written scripts like Health, Mover, Fighter, and Player Controller.
- System: This part is hidden within Unity and doesn’t require our direct intervention. Unity has various independent systems operating in the background to serve different purposes in building and running the game. Notable Unity systems include the RenderSystem that processes Renderer components (including mesh, sprite, particle, etc.) and the PhysicsSystem that handles Rigidbody components.

4.1.2 Overall design



The diagram above represents the component packages of the game "Knights and Wizards" and their dependencies. First is the Game Engine package. The Game Engine is the broadest package, encompassing the entire game and responsible for handling the application, information, and data. The Game Engine acts as the brain of the game, where sound, images, rendering processes, animations, and other processing logic take place. All other component packages depend on the Game Engine.

The Levels package contains different levels of the game. Levels manage different game stages, ensuring a seamless progression for the player without interruptions.

The Characters package handles all information related to character types in the game. Depending on the referenced character class, the corresponding classes and data types will be accessed. Both Characters and Levels are component packages dependent on the Game Engine.

Next, there are two packages dependent on Characters. The Items package contains various usable items in the game. There are two types of items designed: combat weapons and consumable items. Depending on the item type, different handling methods will be implemented.

Finally, there is the Saving Systems package. This system saves the necessary information between gameplay sessions. The Saving Systems store the states of each character at the called time and replay those states when called again.

4.1.3 Detailed package design

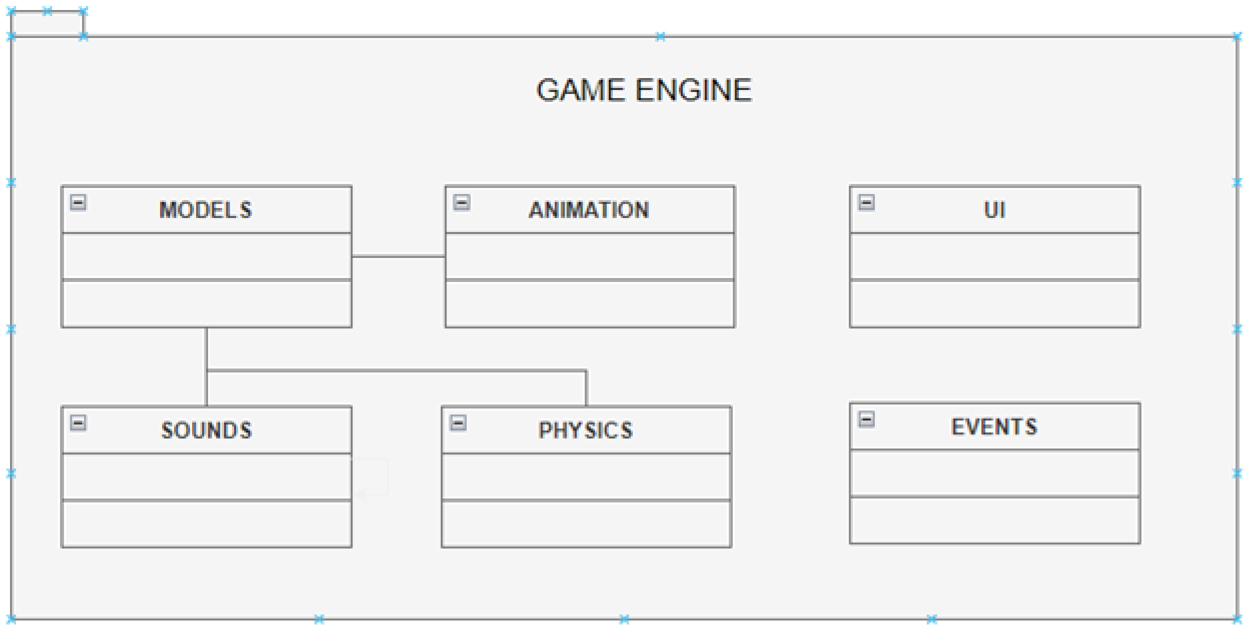


Figure 4.1: Game Engine package

Package Game Engine is the encompassing package of the entire game. The Game Engine consists of Unity components used in the construction process. The fundamental and core elements of the game's operation are handled by the Game Engine. The models (free assets obtained from the Unity Store) are processed by Unity. These models will be combined with other processing components of Unity, such as Animation, Sounds, or Physics, to make the models function most effectively. Events are called based on the situations encountered by the player during gameplay. Along with that, the UI contains interfaces that interact with the user, providing necessary information during gameplay, as well as buttons for starting a new game or continuing from a previous session.

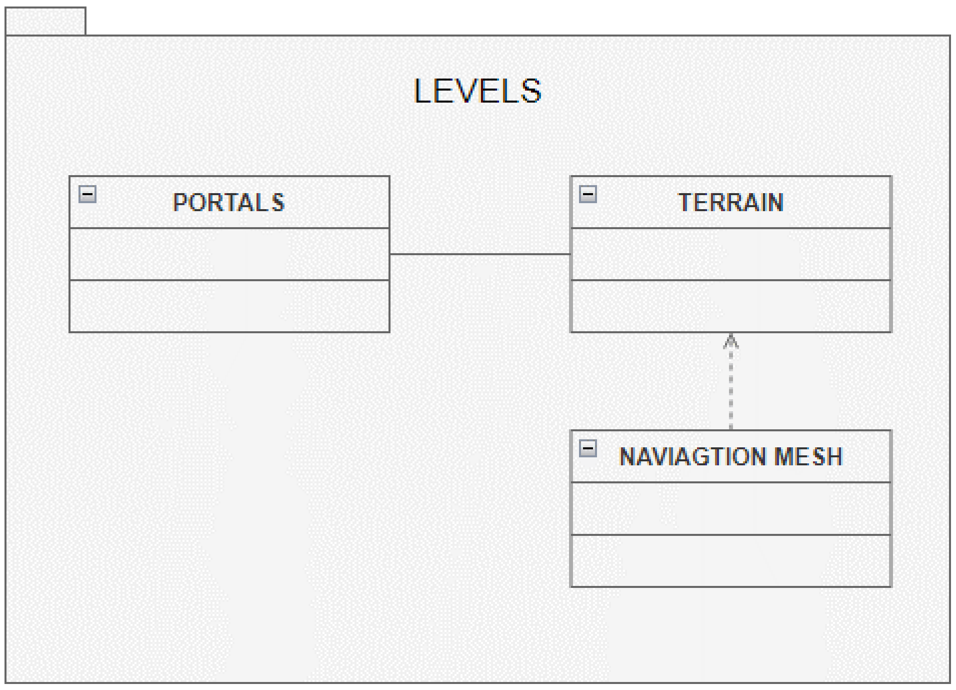


Figure 4.2: Levels package

The Levels package contains all the gameplay levels throughout the game. Each level is represented by a scene, and its terrain is designed using Terrain. A scene may use a primary terrain and have neighboring terrains to create environments. Based on the terrain design and the placement of obstacles on it, the Navigation Mesh will build a grid evenly distributed over the entire terrain to calculate the movement capabilities of the characters. Terrains are connected through Portals, and a level can contain multiple portals, leading to different levels.

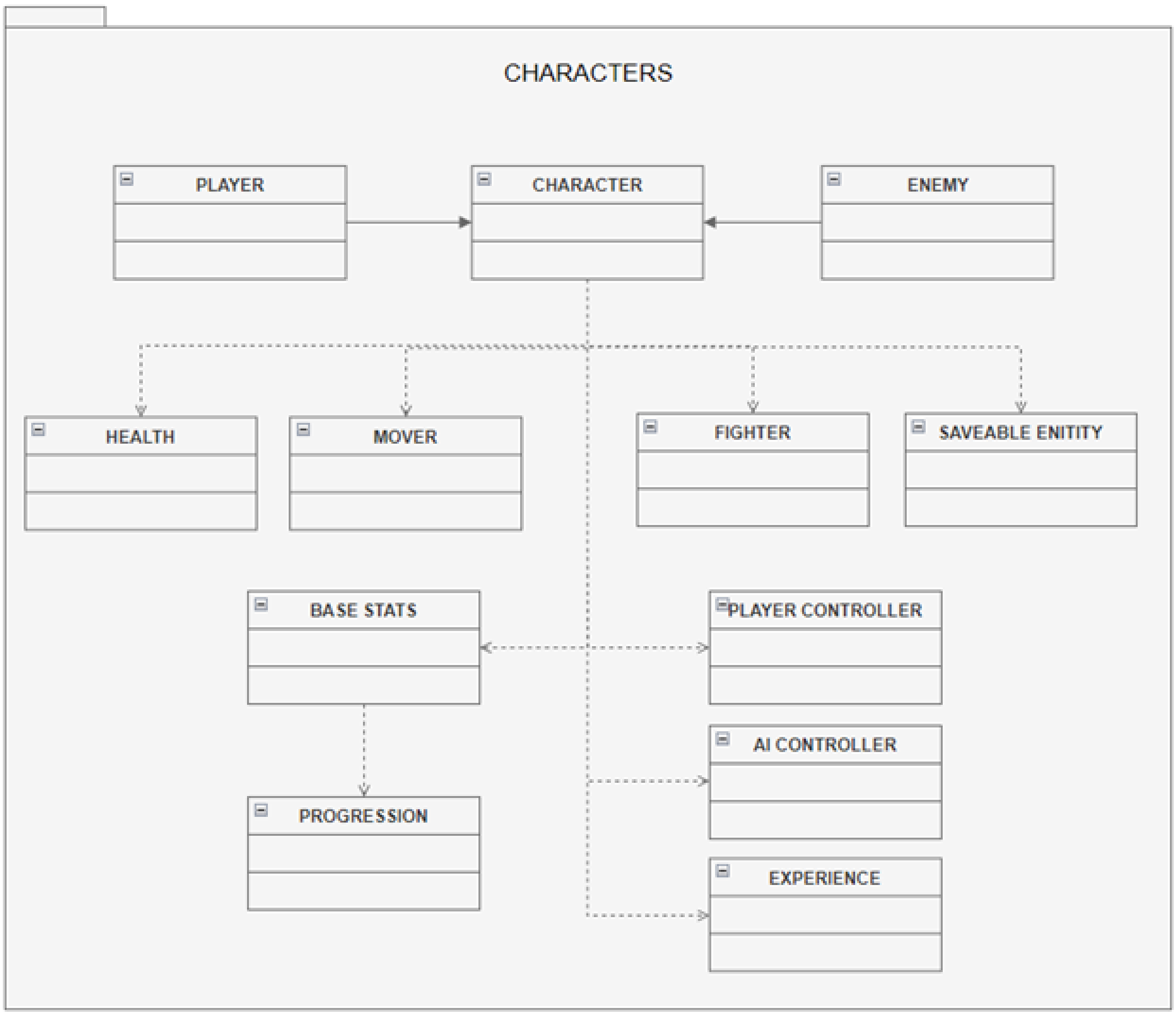


Figure 4.3: Character package

The Characters package is the most important package in the game. It represents the two main types of characters that appear: Player and Enemy, both inheriting from the common Character class. The Character class is determined by various factors, depending on the component classes: Health (determines health amount, handles life and death, receives damage, heals the character), Mover (determines movement speed, handles character movement capabilities), Fighter (determines damage and handles the character's attack abilities against other objects), Base Stats (retrieves data and parameters from the Progression database, contains information about the referenced character class, uses this data to pass on to other classes within Character), and Saveable Entity (determines unique IDs for characters, captures information that needs to be stored). Additionally, there are several classes specific to character types.

For the Player, the Player Controller class is responsible for creating an input-based control system for the player to control their character, handling logic when interacting with various objects during gameplay. The Experience class determines the current experience level of the character, helping the player level up when reaching a certain threshold.

For Enemies, they have their own AI Controller class, enabling them to operate autonomously. The AI Controller helps enemies automatically move, guard, patrol along predefined routes, detect if the Player enters their range, and initiate pursuit and attack.

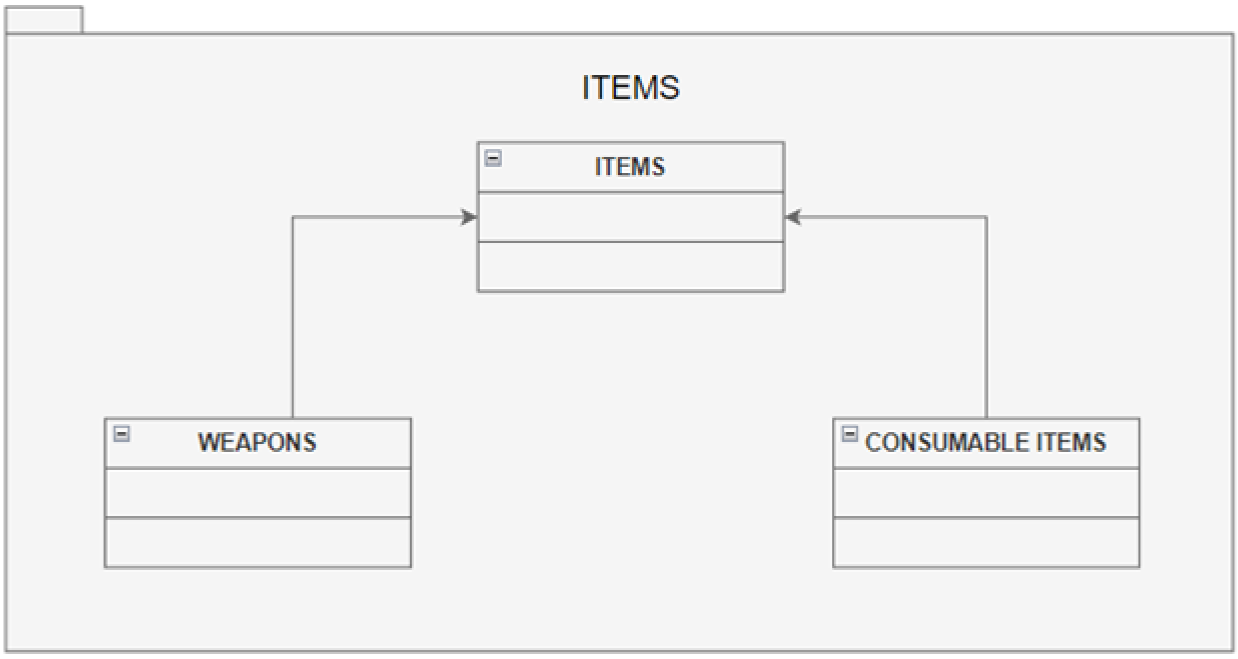


Figure 4.4: Items package

The Items package contains various usable equipment in the game. The Items class defines the name and methods of each equipment. Depending on the item's purpose, there are two subclasses inheriting from Items: Weapons and Consumable Items. For Weapons, they specify the weapon type, such as damage, attack range, attack speed, or usage style, which are retrieved from the Weapon Resources (database for weapons). When the player picks up a weapon item, it will automatically equip the new weapon (replacing the existing one if any). Consumable items are items that can be consumed immediately upon acquisition. Depending on their specific function, they can have different effects. The item may help the Player recover or act as a beneficial enchantment, increasing movement speed or attack speed.

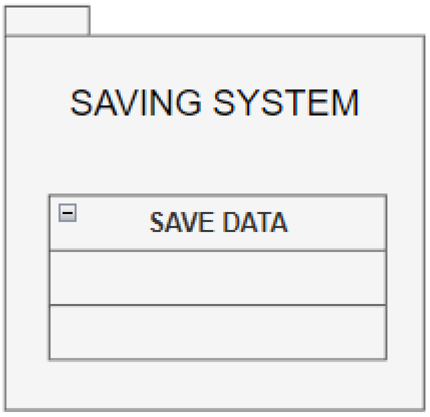


Figure 4.5: Saving System package

The Saving System package is used to store information about the characters and restore them. The Saveable Entity class is used to gather the necessary information to be stored for the characters, and the Save Data stores this data in a .sav file generated by the saving system. This file is stored in the persistent data (Appdata/LocalLow/DefaultCompany) on the user's computer. Players can use the Save command to create a new save file or overwrite the existing one, Load to load data from the file, restoring the state of all characters to the last saved state, or Delete to delete the save file entirely.

4.2 Detailed design

4.2.1 User interface design

The game is designed for PC interface with a resolution of 1920 x 1080. The game's UI is designed to be simple, user-friendly, and visually appealing, ensuring that users understand the displayed content on the screen and the steps required to perform specific tasks. The UI design will use two main color tones, yellow and brown, to match the medieval setting of the game. The game utilizes the Fette National Fraktur font, which adds a stylish and classic touch to the overall atmosphere of the game.

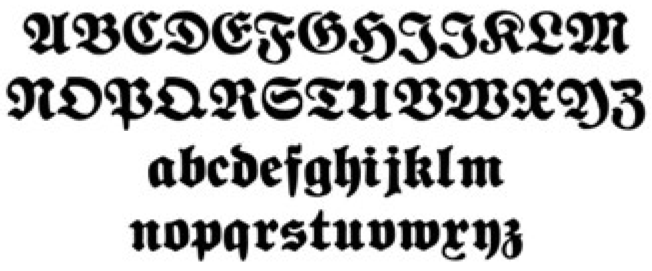


Figure 4.6: Illustration for the font used

The game will have two main interface screens. Firstly, there is the main Menu screen. The main Menu screen appears every time the player starts the game or exits the game from within gameplay. At the main Menu screen, the top section will display the game's logo along with a prominently displayed title. Below that, there will be buttons that perform basic game tasks, arranged in a single vertical line, positioned in the middle of the screen. The buttons will have yellow text on a brown background. The order of the buttons will be Start, Load, Instruction, and Exit. The Instruction button will lead the player to a screen providing instructions on the actions performed in the game. Behind the buttons and logo, there will be a scene with a simple design consisting of the environment, structures, and enemies, helping new players visualize the game's graphics and setting.

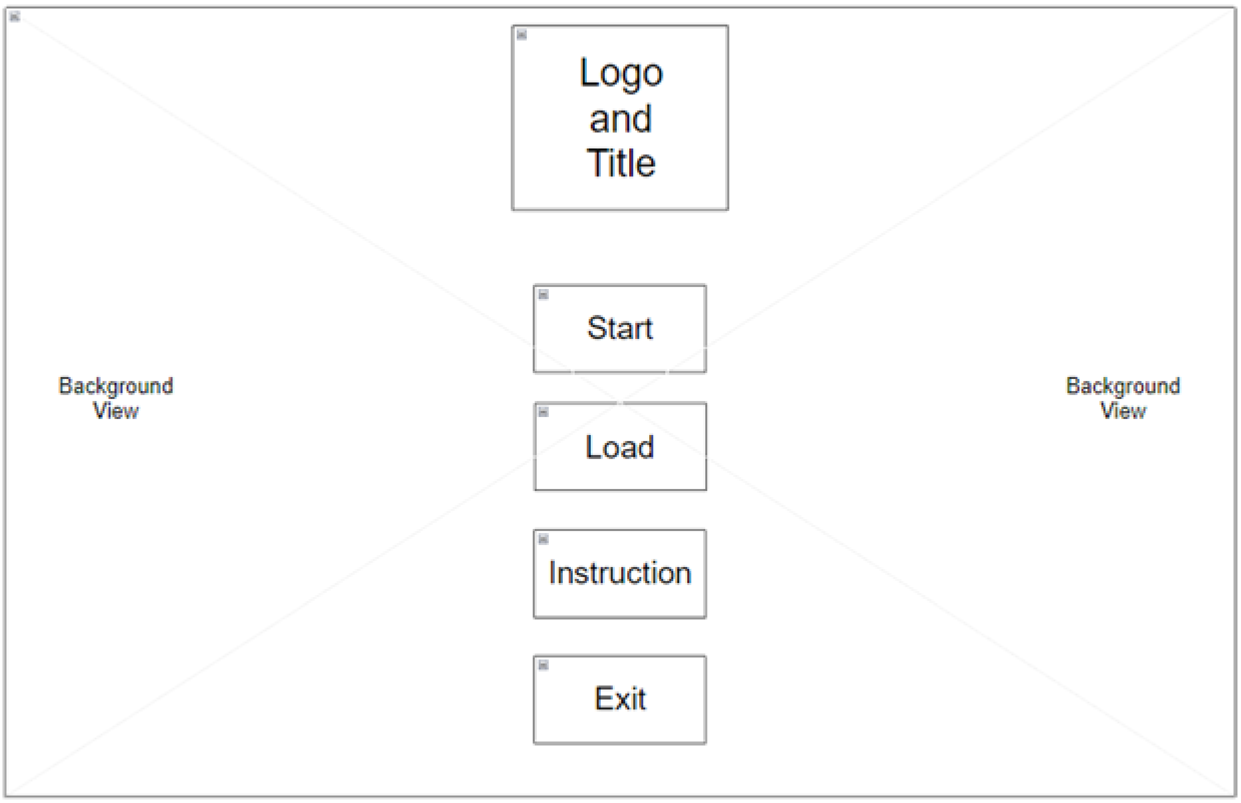


Figure 4.7: Game menu illustration

The other main interface of the game is the main gameplay screen. The gameplay screen ensures that the UI elements appear tidy and easily understandable to provide players with necessary information while avoiding distractions and hindrances caused by interactive components on the screen. In the top-right corner, important information that players need to pay attention to will be displayed. A health bar in green color indicates the player's current health. Below the health bar, there will be the player's experience points earned during gameplay and the current level of the player. This allows the user to track their progress at the current moment.

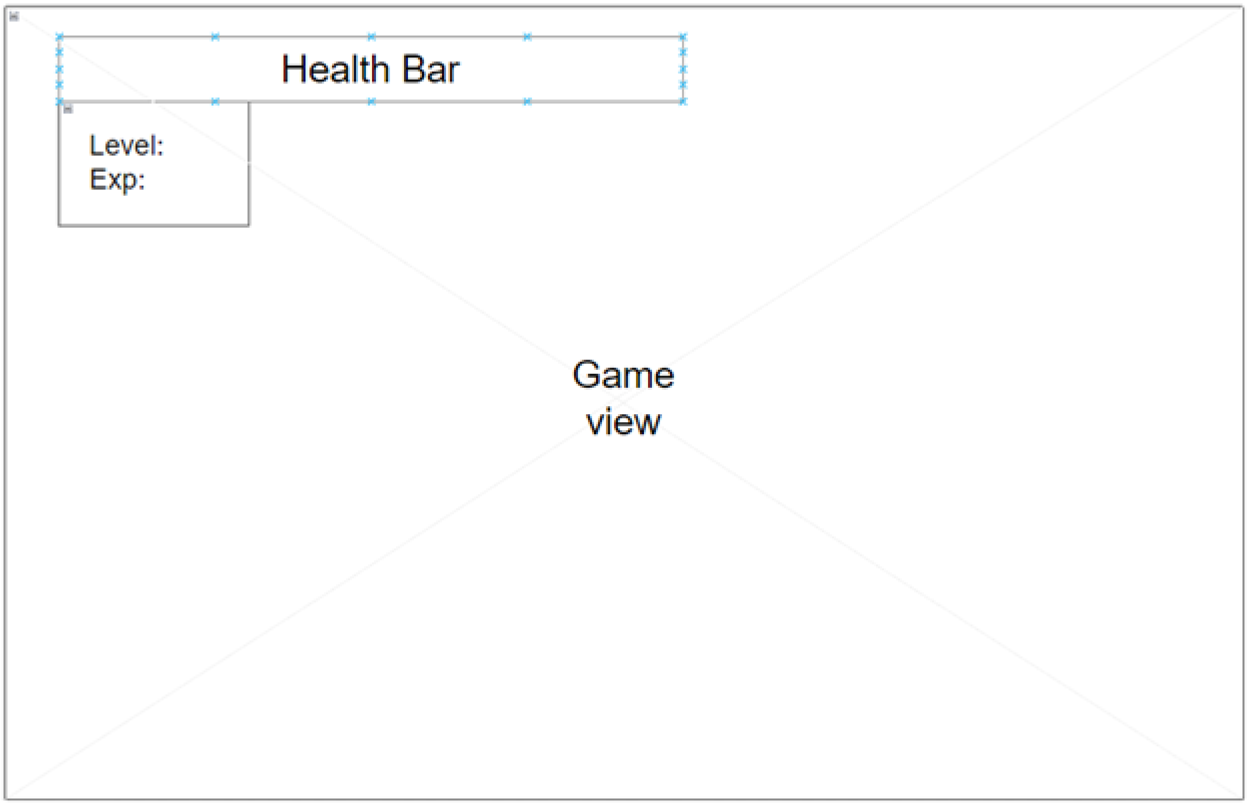


Figure 4.8: Gameview illustration

4.2.2 Layer design

Characters can be considered the most important and complex component of the game since in an RPG game, as all events revolve around characters. The Character class is a major class that depends on several underlying classes to shape it. The two main types of characters in the game are Player and Enemy, both inheriting the characteristics of the Character class. Additionally, they have their own specific classes to operate according to their respective nature.

First, let's review the shared classes used by both Player and Enemy within the Character class:

- Base Stats: This class accesses the game's Progression data. Based on the character class being used and the character's level, Base Stats retrieves important information specific to that character. This information includes health, experience points for leveling up (for players), experience points awarded when defeated (for enemies), and damage. These stats increase gradually as the character's level progresses (players level up as they progress in the game, while enemies become stronger as they appear later in the game). The data retrieved depends on whether the character is a player or an enemy. If it's an enemy, it needs to specify the enemy type. Additionally, Base Stats is responsible for accessing the Experience class if the character is a player, tracking the player's experience points and leveling up when reaching specific thresholds, as well as updating and refreshing the stats based on the database.

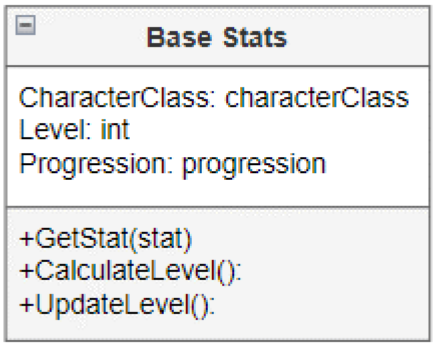


Figure 4.9: Base Stats class

- Health: This class represents the character's health. It uses the Base Stats class to access the corresponding health information for the character. Health is responsible for the character's survival in the game. The character may lose health when attacked and regain health when using a healing item or leveling up. If a character's health reaches zero, the character dies. If the character is the player, the game session ends, or if the character is the final boss, the player wins, and the game concludes.

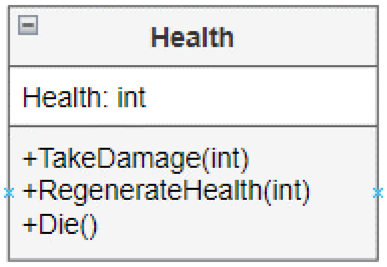


Figure 4.10: Health class

- Mover: This class handles the character's movement capabilities. Mover contains attributes for the character's maximum movement speed and maximum distance the player can move. It enables the character to move to specific positions on the map using the Navigation Mesh. Mover checks if the character's target position is reachable and, if so, initiates the movement behavior. If the character is a player, Mover uses the NavMesh to calculate the path to the cursor position. Only if the NavMesh determines the distance within an allowable range, the player can move to the new position.

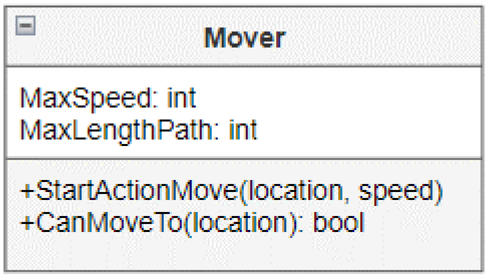


Figure 4.11: Mover class

- Fighter: This class manages the character's attack actions. Fighter determines the two hands (game objects) of the character, allowing the character to equip weapons and identify the type of weapon they will use. The damage inflicted is determined by the combination of two factors: the base damage from Base Stats and the specific damage of the equipped weapon. Additionally, the equipped weapon provides information on whether it is a left or right-handed weapon for proper handling, and whether it is a melee or ranged weapon. Depending on the usage, Fighter invokes the corresponding attack method, either direct melee attacks or ranged projectile attacks. Fighter also assists the character in equipping or unequipping weapons. Before initiating an attack, it checks if the target is attackable. If it is, it further checks if the target is within the attack range. If the target is not in range, Fighter calls the Mover class to help the character move within the attack range and initiate the attack process.

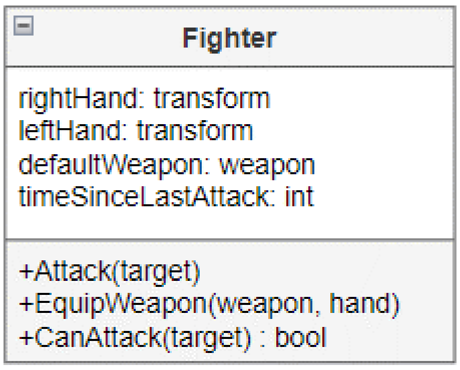


Figure 4.12: Fighter class

- Saveable Entity: This class allows characters to store information between gameplay sessions. It assigns a unique ID to each character for each gameplay session. This ID remains unchanged throughout a single gameplay session, only reset when the user deletes the save file. It helps identify and accurately store the necessary information for characters. Saveable Entity retains the required attributes of the character in each respective class and returns them in the next game load.

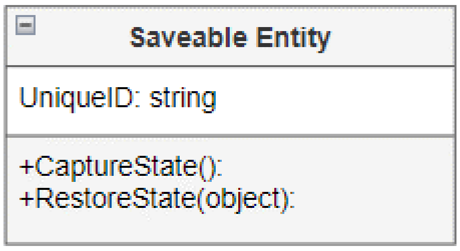


Figure 4.13: Saveable Entity class

In addition to the shared classes mentioned above, there are two specific classes for each character:

- Player Controller: This class controls the player's character. Player Controller uses input, mainly the left mouse button, allowing the player to control all actions of their character during gameplay. Depending on the type of action the player is aiming for, this class returns a corresponding cursor to visually indicate to the player the nature of that action. For example, when interacting with the map terrain, the cursor will appear normal to indicate a movable area, or change to a red cross if that position is not accessible. Similarly, when the player hovers over other elements such as enemies or pickable items, different cursors will appear accordingly. When the player clicks the left mouse button to confirm an action, if successful, the player will move to the designated cursor position and perform the action. All these interactions are supported and processed by Unity's NavMesh and Raycast. The Raycast handles the information about the cursor's target on the map and categorizes them, while the NavMesh calculates information about the pre-drawn navigation paths.

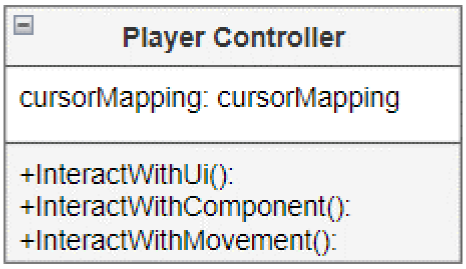


Figure 4.14: Player Controller class

- Experience: This class represents the player's specific experience. The player will have an experience system that gradually increases based on their gameplay progress. Each time the player defeats an enemy, they will be rewarded with a corresponding amount of experience points. The experience system accumulates these points, and the information will be transferred to Base Stats.

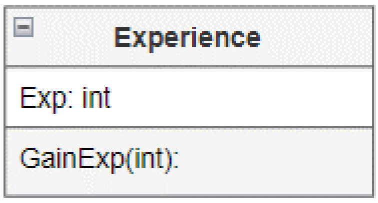


Figure 4.15: Exeperience class

- AI Controller: This class controls the enemy characters. It acts as the brain of the AI-controlled bots, enabling them to operate autonomously. AI Controller has several attributes to serve its functions, such as chase radius, suspicion time, patrol distance, engagement range, etc. Firstly, if an enemy has a patrol route, they will move sequentially along the designated points on that patrol route. If there is no patrol route, the enemy will stand guard at a fixed position. During the patrol time, if the player moves into the enemy's alert area, they will perform the following events. The enemy will detect the player and initiate an alert for all nearby allies within the engagement range. All alerted enemies will turn and pursue the player. If the enemy enters the attack range and is able to attack, they will perform an attack and inflict damage on the player. If the player escapes from the enemies, they will continue pursuing for a certain period of time until they cannot reach the player, then they will stop, enter a suspicion state for a certain duration, and finally return to their original position before initiating the pursuit and resuming their tasks.

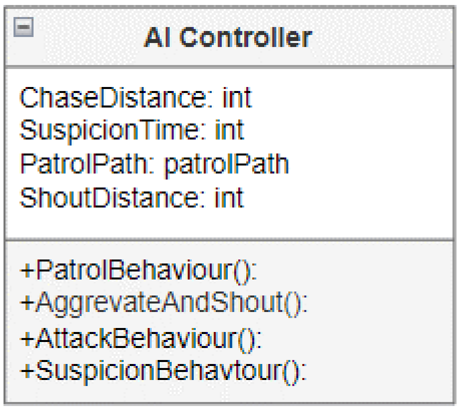


Figure 4.16: AI Controller class

4.2.3 Database design

Contrary to the majority of applications, Unity games, especially smaller ones,a separate database management system is not required.. The reason is that most data related to entities in the game is directly written onto them through components. Unity accesses the attributes and values of these components directly to process data as required by the user. However, even with entities distributed across the game, there are still some pieces of information that need to be stored centrally for easy management. In this game, a database is designed solely for accessing stats related to characters.

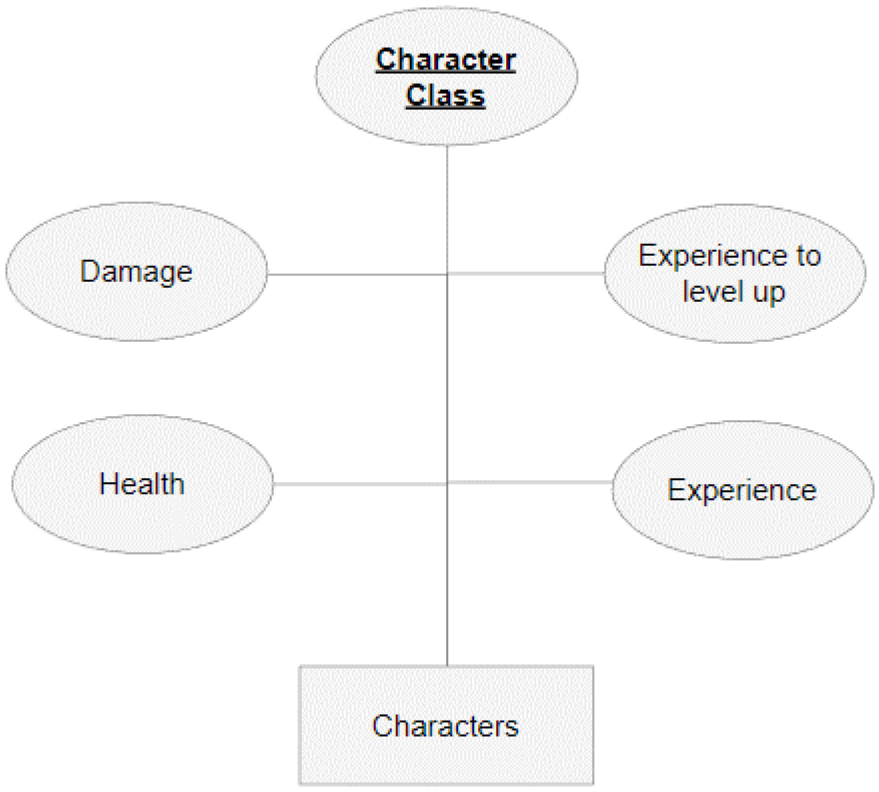


Figure 4.17: Databse diagram

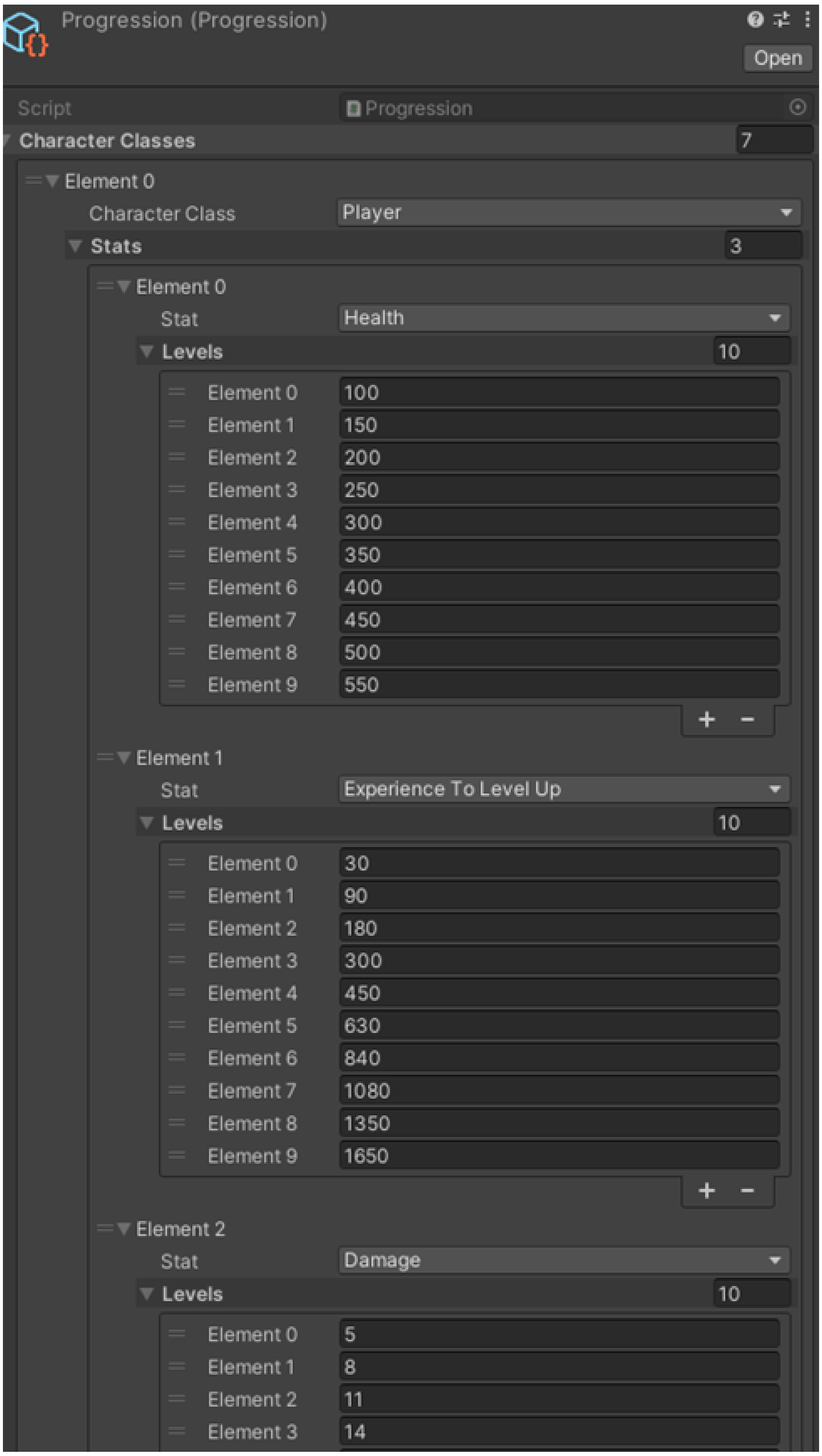
The model above diagram used is used to represent the design of the database used for the game. The Character entitiy, its attributes, and the relationships are depicted to describe how data in the game is stored. The game's database is not too large and primarily used to describe the main components. Progression is the name of this database, as its job is to keep track of all characters' progression during the gameplay.

First is the entity. An entity is an object or concept with its own meaning in the system. There is only one main entity that are crucial to the game: Characters (which encompasses all the characters in the game, including both Player and Enemy). Since Character is the only entity that matters in the database, no relationship to other entity was formed.

The Characters entity holds attributes that represent the properties of the characters owned in the game. The Character Class is the primary key, distinguishing the classes of different characters, and the remaining attributes vary depending on the character's class. The player will have a unique class called Player, while all other classes represent different types of Enemies. The other attributes in Progressiong include Damage (the character's specific damage), Health (maximum health of the character), Experience (specific attribute for Enemies, the amount of experience gained when defeated), and Experience to Level Up (specific attribute for the Player, the amount of experience needed to level up). Each of these attributes will have a list of level data, which change accordingly to the level of the character.

As described above, the database is not overly complex and extensive. Therefore, instead of using large-scale database management systems like SQL, NoSQL, or Firebase, we will use Scriptable Objects, a built-in feature of Unity, to store and manage game data such as data structure, images, sounds, etc. Scriptable Objects don't have full-fledged functionality like advanced database management systems, lacking features such as querying, simultaneous editing, logging, data security, and transaction management. However, in this specific game scenario where the database is designed to be relatively simple and uncomplicated, Scriptable Objects can be a convenient alternative for designing, storing, and editing game data. All the data is stored offline directly on the user's computer system (LocalLow) without the need for third-party cloud services for storage.

To use and store data with Scriptable Objects, we will apply the model described above for the design. The name of the Scriptable Object, or this database, as said before, is Progression, representing the tracking of character progress. There is system of serializable arrays contains the details of all stats. The title of each array, or the primary key, is the Character class. Inside these, there are two types of value, first being the accessable stats belong to the class, then an ascending list of numbers to represent the values of that stat on different levels. Since all of these are serializable, we can directly edit, add, or remove data on the Editor according to our preferences for the database. Scriptable Objects cannot be used directly as an object in the scene. Instead, we will reference them through the BaseStat class on the characters to access this database.



```
Progression (Progression)                                   Open
Script                          Progression
Character Classes                                            7
  Element 0
    Character Class             Player
    Stats                                                    3
      Element 0
        Stat                    Health
        Levels                                              10
          Element 0             100
          Element 1             150
          Element 2             200
          Element 3             250
          Element 4             300
          Element 5             350
          Element 6             400
          Element 7             450
          Element 8             500
          Element 9             550
                                                          +  −
      Element 1
        Stat                    Experience To Level Up
        Levels                                              10
          Element 0             30
          Element 1             90
          Element 2             180
          Element 3             300
          Element 4             450
          Element 5             630
          Element 6             840
          Element 7             1080
          Element 8             1350
          Element 9             1650
                                                          +  −
      Element 2
        Stat                    Damage
        Levels                                              10
          Element 0             5
          Element 1             8
          Element 2             11
          Element 3             14
```

Figure 4.18: Scriptable object Progression in Editor

To access a specific stat of of a character class, 3 informations would be requested: the Character class being mentioned, the Stat being asked, and the current level of the character. The Progression script will build a lookupTable based on the array that we created above. lookupTable is a nested dictionary looks like this:

Dictionary<CharacterClass, Dictionary<Stat, float[]» lookupTable

The outer Dictionary key is CharacterClass, representing the character type, which is also the primary key of the database. Each key has its own value, which is an inner Dictionary. The inner Dictionary has a key called Stat, with a value of a float array. Stat represents the attributes owned by the characters as described in the diagram above, and each Stat has a float array to represent its values at each level. Using this dictionary, the script will look for requested stat of the Character class, then return the exact value of the stat at the current level of the character.

4.3 Application Building

4.3.1 Libraries and Tools

Purpose	**Tool**	**Version**	**URL**
IDE	Visual Studio Code	1.80	code.visualstudio .com
Programming Language	C#		
Game Engine	Unity Editor	2022.1.0b2	unity.com
Lazy Evaluation	Lazy Value Package by GameDevTV [21]		gamedev.tv
Saving System	Saving System Package by GameDevTV		gamedev.tv

Table 4.1: Lists of used libraries and tools

4.3.2 Achievement

The project's outcome, as introduced in the beginning of this document, is a Unity RPG game set in a medieval context, using mouse input as the primary control. In the game, players assume the role of a knight, using collected weapons to fight against enemy knights and wizards from neighboring kingdoms, progressing towards defeating the powerful final boss, a mighty sorcerer. The game is designed in a linear fashion, consisting of 4 levels with different terrains and environments, gradually increasing in difficulty. The save-load feature allows players to save their progress and continue at a different point in time.

Info	Details
Application size	520MB
Number of C# scripts	46
Total lines of code	1914
Number of assets packages	26

Table 4.2: Info of product

4.3.3 Illustration of main functions

As we open the game, the main menu of the game will appear with different options to choose. Players can either start a brand new game by choose the Play button, or resume their last saved progress by clicking the Load button. Instruction will display details of how the game works.

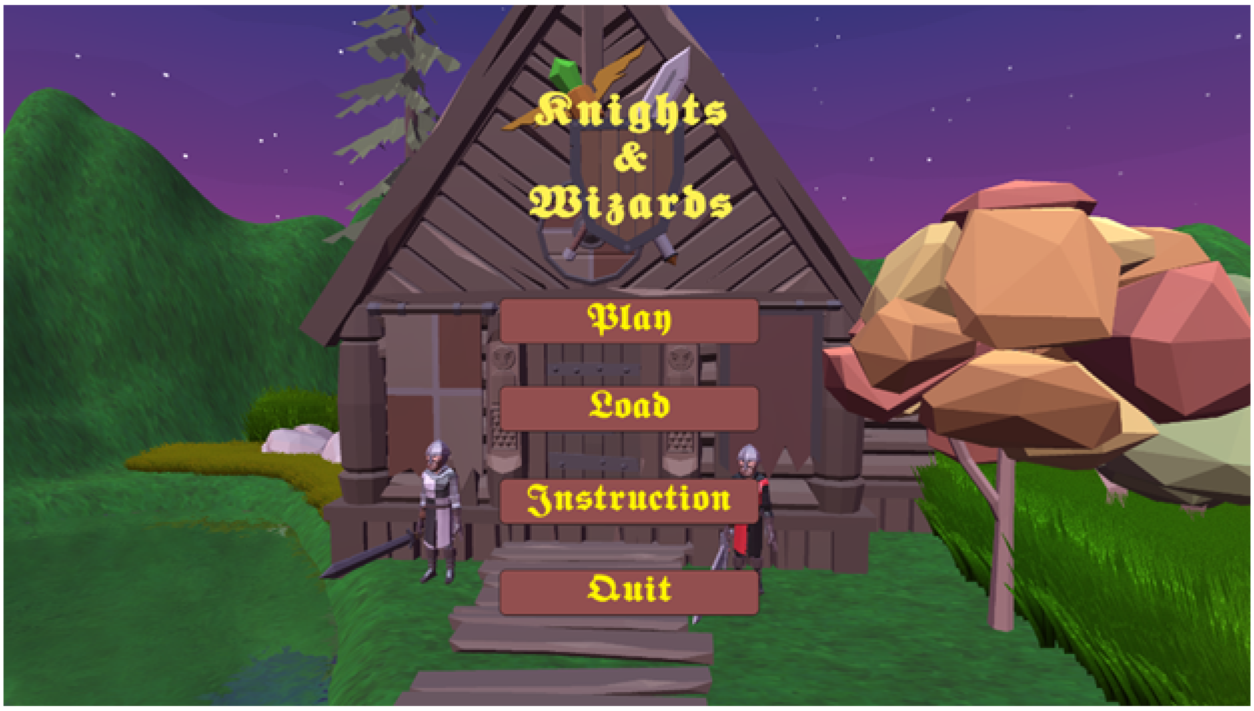


Figure 4.19: Main Menu of the game

Below is the main display during the gameplay. The player will moving in a map to perform actions. Inside the map, there will be diffrent kind of foes to fight against and pickable objects to use. Player can act however they want, but in the end, the main objective is moving to the portal to go to another level.

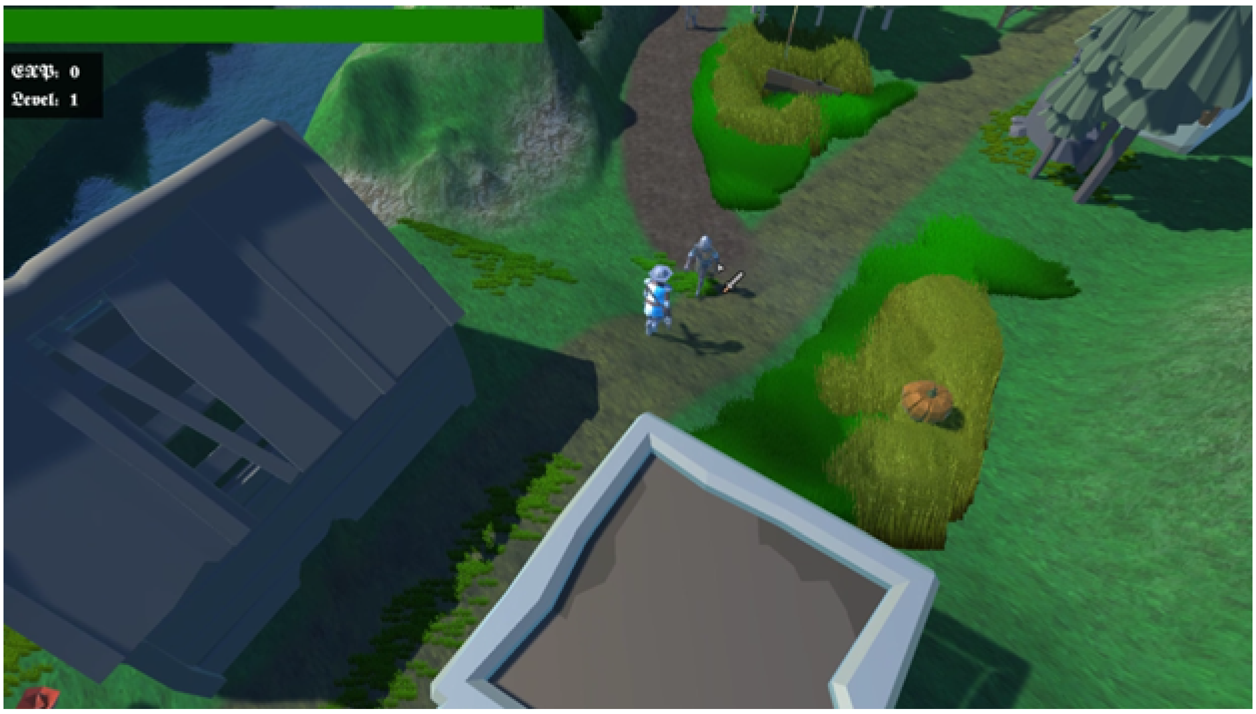


Figure 4.20: Gameplay display

To control their character in the game, users primarily interact with the right mouse button. In the main gameplay screen, players click on positions on the map to make the character perform corresponding actions. For example, if a player wants to move to a nearby location, they simply click the mouse cursor on that position, and if it's a valid location, the character can move there. The same applies to other actions such as attacking enemies or picking up weapons. The mouse cursor will change its shape accordingly to indicate the type of action the player will perform upon clicking. There will be different cursors for movement, item pickup, and attack actions, as well as additional cursors to display information such as indicating that a position is inaccessible or representing the game's UI.

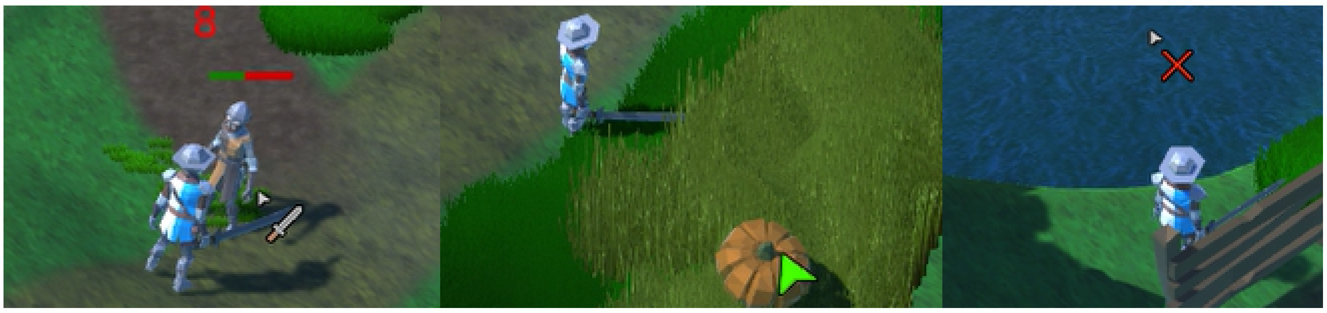


Figure 4.21: Different cursors being shown for different types of action

There multiple kinds of different weapons, from long-ranged to close-ranged throughout each map. Player can choose whatever weapon they desire to use, by simply click on weapon location. The character will get closer that weapon to equip it. Be remind that if you're already owning a different weapon before, the previous one will be discarded to make place for the new weapon.

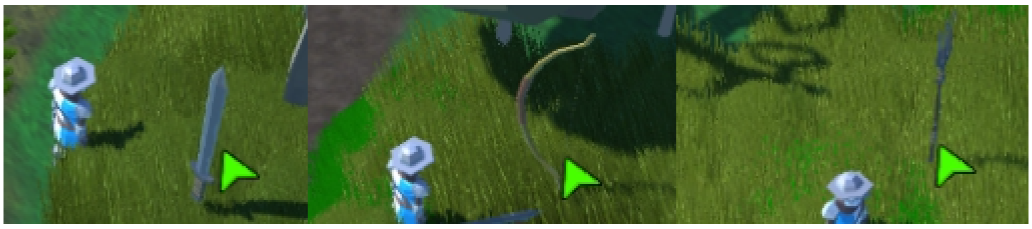


Figure 4.22: Players can choose different weapons to equip

Player can attack one enemy at a time. When attacking, depends on the weapon being used, the player will strike differently to damage the enemy. If the enemy dies, player will be rewarded with experience to level up and increase self stats.

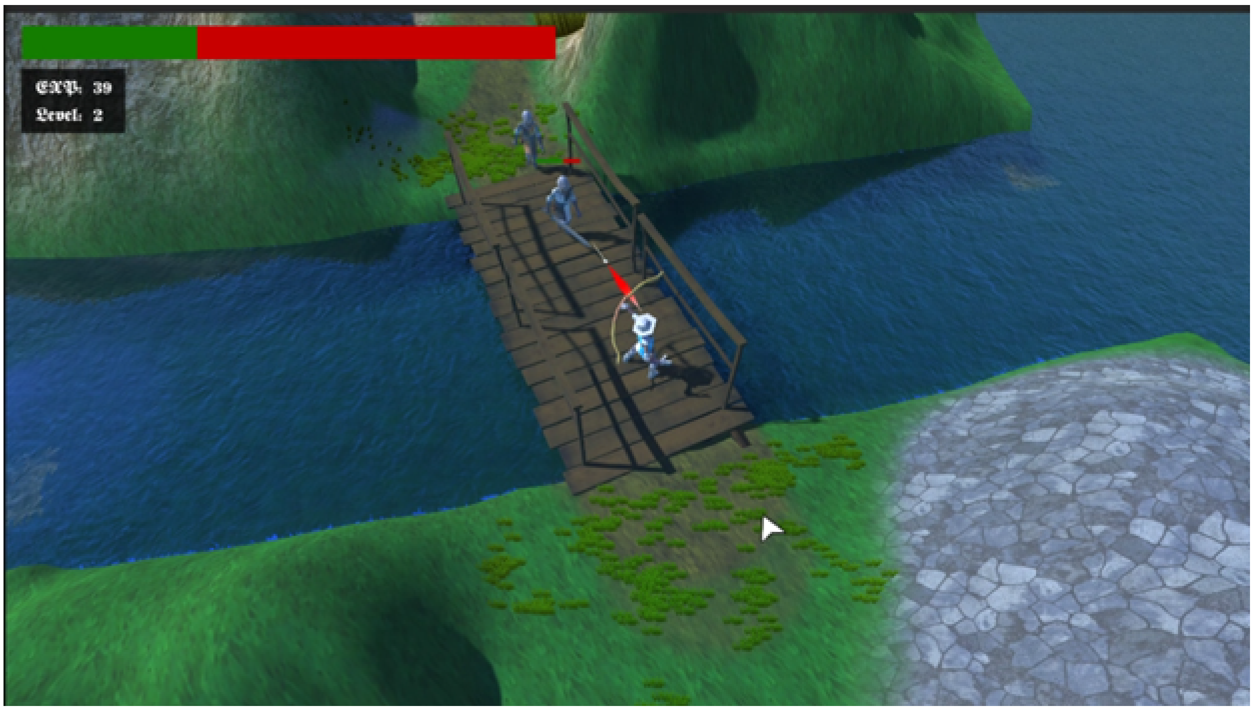


Figure 4.23: Attacking enemy

On the end of the map there is at least one portal, sometimes two, marked with a white circle. These portals will take the player to another level. Each portal is assigned to one specific level, and can go back to travel to the previous one. When the player steps into the circle, there should be a transition loading screen, then they will be teleported to the next respective destination.

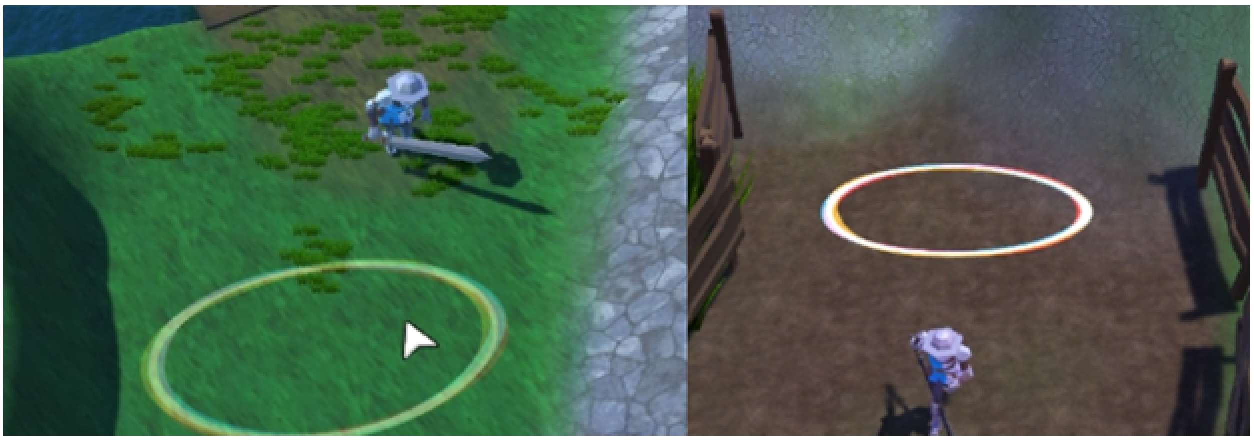


Figure 4.24: Teleporting through portal

Players can also utilize the save game feature to preserve their game progress. This feature is activated when the player presses the "S" key on the keyboard. At that moment, a savegame file is created and stored within the user's computer system. Players can press the "L" key to load this savegame file or select the "Load Game" button in the main menu screen. The load-save feature is also automatically triggered whenever players transition between different levels.

4.4 Testing

To facilitate the testing phase of the game, I will select two prominent features to represent their tasks and ensure that their processes are carried out as expected.

Firstly, we will test the enemy's behavior when they interact with the player. Here, we will examine how the actions are performed, while the logic and programming aspects will be discussed in detail in Chapter 5.

Enemies, whether they are in a stationary guarding state or patrolling along their designated path, will have a chaseDistance. This circle represents their "field of view" or vigilance. If the player moves within this range while the enemy is active, they will initiate a series of attacking actions.

The first thing they will do is turn towards the player and alert all nearby enemies within their range. Then, depending on the nature and situation, all the detected enemies will either move to the player's position or attack immediately. If the player is out of the weapon range possessed by the enemy, they will move closer to the player until it is within range and then initiate the attack. If the player runs away from the enemy, they will continue to chase and attack. During this chase, if the player goes beyond the chaseDistance of the enemy after a certain period of time, the enemy will pause and check if the player returns. If the player returns, the pursuit and attack will continue; otherwise, the enemy will return to its original position.

In a special case, an enemy may be using a long-range weapon (like a bow) while standing at a position where NavMesh cannot build a direct path between the player and that enemy. In such a situation, if the player moves within the weapon's range, the enemy will still initiate the attack. However, if the player goes beyond the attack range, the enemy will be forced to stop its actions and cannot continue the pursuit. It will wait for a short period of time before returning to its guarding position.

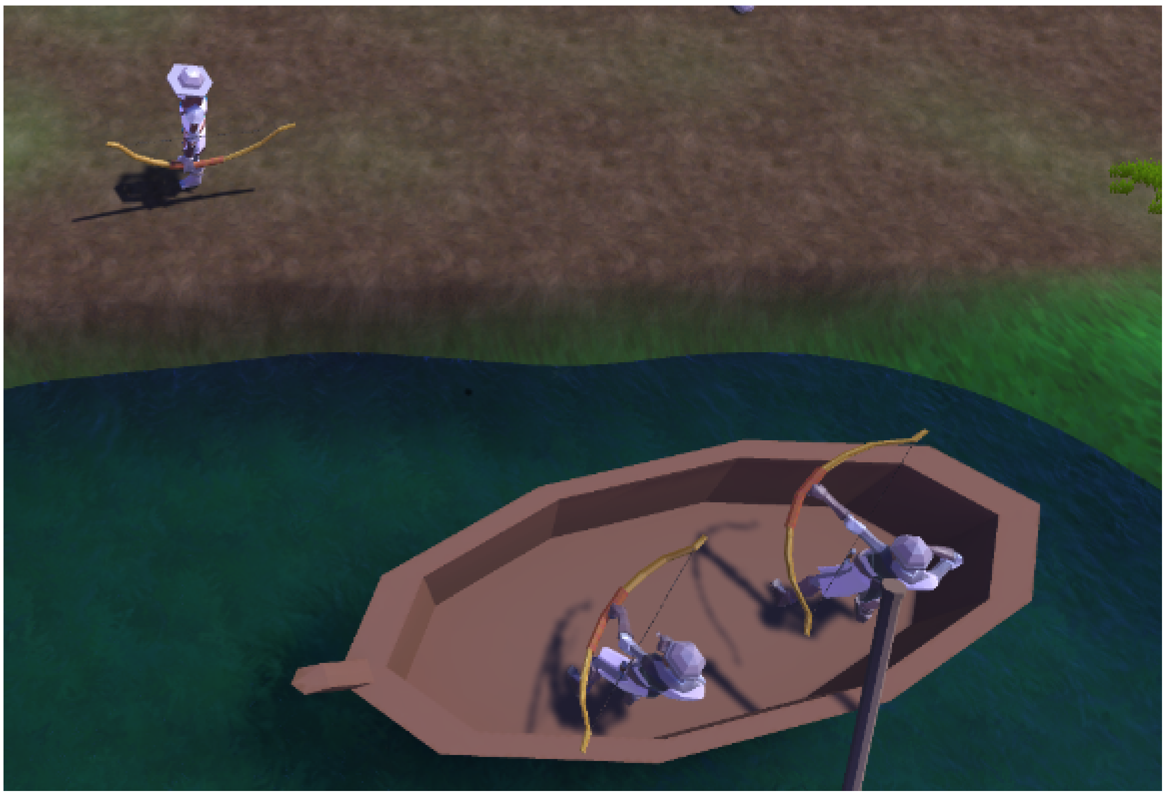


Figure 4.25: Distant attack exception

The second feature that needs to be tested for accurate functionality is the game's save-load feature. This feature utilizes a saving system package designed and provided by GameDevTV. The saving system stores the necessary data specified by the programmer for initialization and retrieval when loading.

The save game can be triggered when the user presses the "S" key on the keyboard or it will be automatically used when transitioning between levels. The purpose of this is not only to allow players to save their progress but also to serve as a tool for seamless information transfer when moving between scenes. When loading a new scene, all game objects in the scene will be recreated, and their values will return to default. Immediately after that, when everything in the scene has finished initializing, the load savefile process will take place. This ensures that the values are accurately overridden. For example, with a defeated enemy (health reduced to 0) at any position, when saving and loading, that enemy will be reinitialized. Its health will be restored to 100 percent, and it will appear at its initial designated position. However, immediately after that, the Load function will restore the values from the savefile, causing the enemy to move to its last position at the time of saving, health reduced to 0, and the enemy will die again immediately. To hide the unsatisfactory nature of this system, a blurred screen layer will be applied whenever the Load function is called to allow Unity to perform the necessary preparations for the level.

Furthermore, there is a point to note regarding saving positions. When restoring the saved position of the player, NavMesh may interfere with the position at the time of scene initialization, resulting in conflicts in assigning the character's position. Therefore, for safety, each time a savefile is loaded, the NavMeshAgent on the player will be temporarily disabled, the position will be manually updated according to the savefile, and then the NavMeshAgent will be enabled again.

4.5 Deployment

This game is designed exclusively to Windows OS computer. As for the majority of Unity games, there won't be an option for changing ingame graphics quality. However, the game was optimized as much as possible so the system requirement for running it wouldn't be much of a problem. Basic computer is capable of handling the game and running it smoothly.

Knights and Wizards will be uploaded on itch.io [22], a website was made for game developers to publish their works and control how content is sold. My game is free to play. This isn't a WebGL game therefore players will have to download the game from the site and run it on their device.

Link download: https://hidenearbush.itch.io/knights-n-wizards

Click the download button, extract the .rar file to get the game folder. To run the game, in the folder, simply open K&W.exe. You will be directed to the main menu of the game.

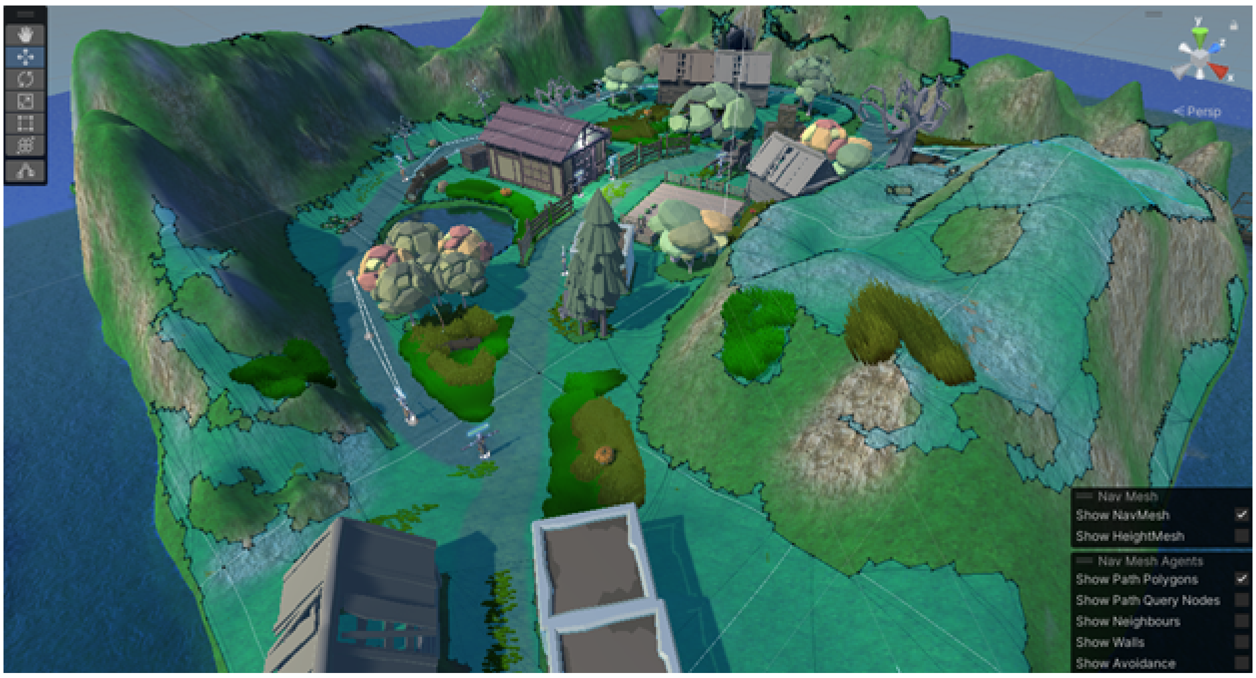
CHAPTER 5. SOLUTION AND CONTRIBUTION

This chapter will present some important and notable content during the design and development process of the Knights and Wizards game. The complexity and intricacy of game programming have been greatly simplified with the help of Unity. However, it is also thanks to Unity that programmers are able to maximize its features to develop new content and more creative methods.

5.1 Navigation Mesh

Firstly, as mentioned in chapter 3, Navigation Mesh is one of the built-in features of Unity that is utilized in this project. NavMesh is a tool in Unity's AI library used to determine and calculate paths for objects in a 3D environment. A mesh with complex shapes based on the terrain of the environment is constructed by NavMesh to allow objects to move intelligently, automatically calculating and tracking safe and efficient paths between points in the environment. Compared to simple movement based solely on vector momentum, rigid body, and collider, the use of NavMesh coordination allows objects to perform more complex and accurate movements, which are simpler and more realistic.

To build the NavMesh, it needs to determine the entire environment of the game and the agents present in it. In this case, the main environment of the game is built on terrains. The NavMesh grid will cover all these terrains, and depending on the programmer's settings for radius, height, slope, or step, the NavMesh will determine valid corresponding areas on those terrains. Then, we identify the components appearing in the scene. These environmental components include all obstacles appearing on the terrain that can affect character movement, such as trees, houses, or other objects used for scene decoration. We will tag these objects as static to indicate that they are static objects. For characters moving in the scene (player and enemies), they need to be equipped with the NavMesh Agent component. NavMesh Agent will determine the movement components on the NavMesh grid and use some parameters to assist the movement, such as speed or angular speed. Once all the components in the scene have been identified and integrated into the NavMesh, we can start baking the NavMesh to create the grid on the terrain.

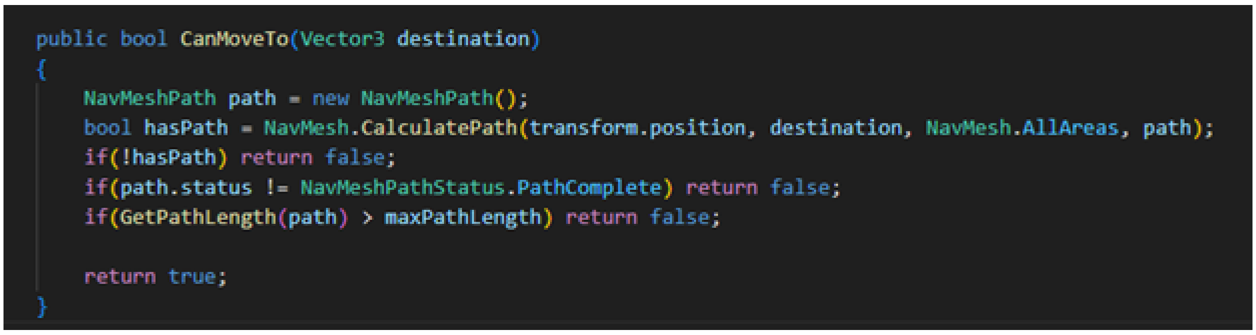


Nav Mesh
- Show NavMesh ✓
- Show HeightMesh

Nav Mesh Agents
- Show Path Polygons ✓
- Show Path Query Nodes
- Show Neighbours
- Show Walls
- Show Avoidance

Figure 5.1: Scene after being baked by NavMesh

Next, to utilize the NavMesh features for smooth and automated movement, we will implement these methods in the Mover class of the characters. The methods in Mover will utilize NavMesh to interpret the logic and create movement for the characters. In this game, players will control their character's movement by clicking the mouse cursor on a position on the map, and for enemies, they will run towards the player's position. Therefore, the first step is to check if the targeted position is reachable. The CanMoveTo() function is used to solve this issue.

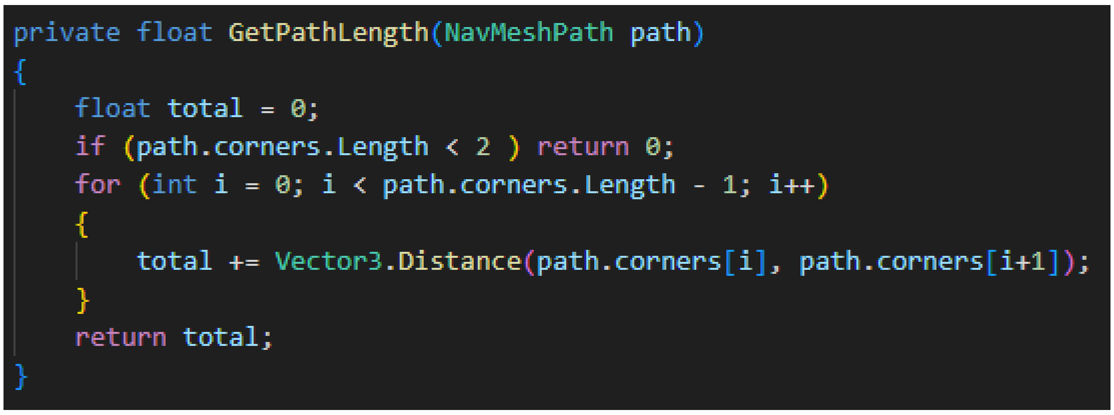


```
public bool CanMoveTo(Vector3 destination)
{
    NavMeshPath path = new NavMeshPath();
    bool hasPath = NavMesh.CalculatePath(transform.position, destination, NavMesh.AllAreas, path);
    if(!hasPath) return false;
    if(path.status != NavMeshPathStatus.PathComplete) return false;
    if(GetPathLength(path) > maxPathLength) return false;

    return true;
}
```

Figure 5.2: CanMoveTo() method

CanMoveTo() calculates the path from the character's current position to the desired position. If these two points cannot create a complete and smooth path that ends at the destination, the function will return false. Additionally, if the distance between the two points exceeds the allowed limit (predefined), it will also return false. If all the above conditions are not met, the function will return true, confirming that the mouse cursor position is reachable.



```
private float GetPathLength(NavMeshPath path)
{
    float total = 0;
    if (path.corners.Length < 2 ) return 0;
    for (int i = 0; i < path.corners.Length - 1; i++)
    {
        total += Vector3.Distance(path.corners[i], path.corners[i+1]);
    }
    return total;
}
```

Figure 5.3: GetLength() method

To calculate the path length, we sum up the distances of each small segment. NavMeshPath is created by several small segments connected together. If the path consists of more than 2 points (corners), we calculate the distance between each continuous pair of points and add them all together. Calculating the path distance is to compare it with the maximum distance the character can autonomously navigate, ensuring that players are not forced to move in small segments according to their own logic to overcome terrain obstacles. Instead, players can click on a distant position and let the NavMesh automatically find a long path. This applies not only to movement but also to aiming at and attacking enemies or picking up items from a distance.

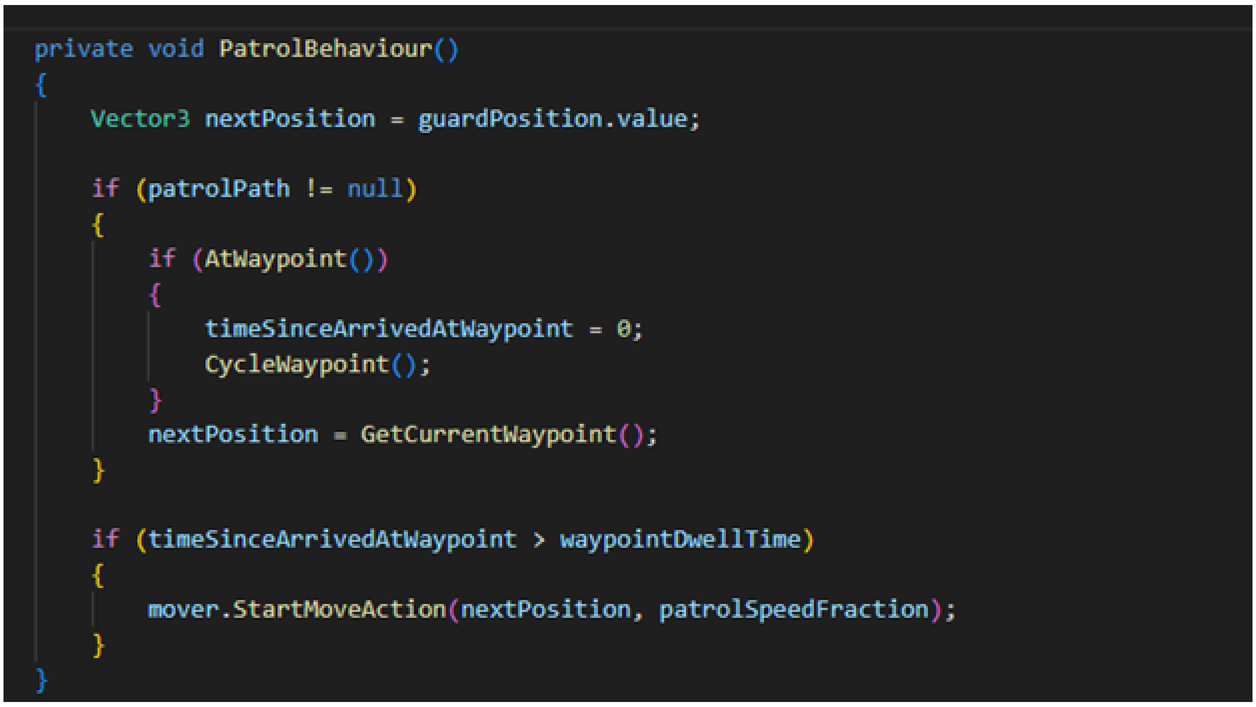
If the CanMoveTo() function returns true, allowing the character to move successfully, the next step is simply to call the MoveTo() function of NavMesh to move the character to the desired position.

NavMesh is also used to handle specific situations in player control. In the PlayerController class, we have the RayCastNavMesh() function used to check if the mouse cursor's position exists within the active NavMesh before allowing the player to move to that position. Using ScreenPointToRay to obtain the corresponding cursor position on the gameplay screen, then checking the NavMesh with the SamplePosition() function. We also have the RaycastsSorted() function, which returns an array of RaycastHit containing all objects intersected by the ray and sorts them from nearest to farthest. The purpose of this function is to ensure that the mouse cursor accurately identifies the closest object in case there are multiple pickable items nearby.

5.2 AI Controller

In addition to relying on NavMesh for movement on the map, enemies also need their own AI to have autonomous capabilities. That's why the AI Controller class is built. This class will create logical actions to help the bots in the game operate efficiently. The functions in the class are used to generate different actions for enemies.

By default, enemies will stand still and guard a fixed position. However, for enemies with a Patrol Path, they will perform patrolling. The PatrolBehaviour() function controls the enemy to move sequentially through each waypoint on its patrol path. At each waypoint, it will stop for a while to observe before moving to the next waypoint. The waypoints will form a closed circle.



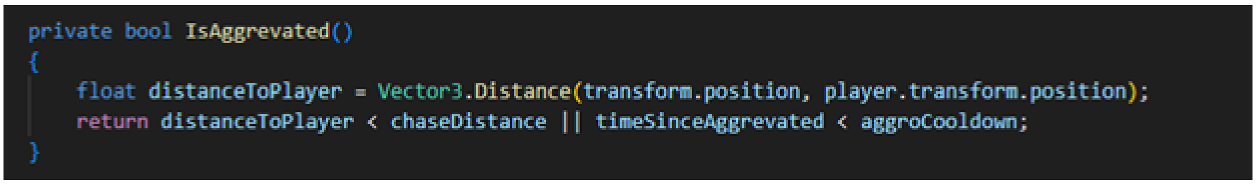
```
private void PatrolBehaviour()
{
    Vector3 nextPosition = guardPosition.value;

    if (patrolPath != null)
    {
        if (AtWaypoint())
        {
            timeSinceArrivedAtWaypoint = 0;
            CycleWaypoint();
        }
        nextPosition = GetCurrentWaypoint();
    }

    if (timeSinceArrivedAtWaypoint > waypointDwellTime)
    {
        mover.StartMoveAction(nextPosition, patrolSpeedFraction);
    }
}
```

Figure 5.4: PatrolBehaviour() method

Next, there will be a series of actions describing events if the enemy detects the player. Each enemy will have a radius called chaseDistance. If the player moves into this range, a series of events will be triggered. The AggrevateAndShout function will find all its allies within the shoutDistance range and reset the timeSinceAggrevated of all enemies to 0. As long as timeSinceAggrevated is less than aggroCooldown or the player is still within the chaseDistance range, the IsAggrevated() function will still return true.



```
private bool IsAggrevated()
{
    float distanceToPlayer = Vector3.Distance(transform.position, player.transform.position);
    return distanceToPlayer < chaseDistance || timeSinceAggrevated < aggroCooldown;
}
```

Figure 5.5: IsAggrevated() Method

At that point, if the enemy can validly attack the player, the AttackBehaviour() will initiate the attack action. The enemy will move to the appropriate range of its weapon's reach and start attacking the player. If the player moves out of range, the enemy will continue to pursue.

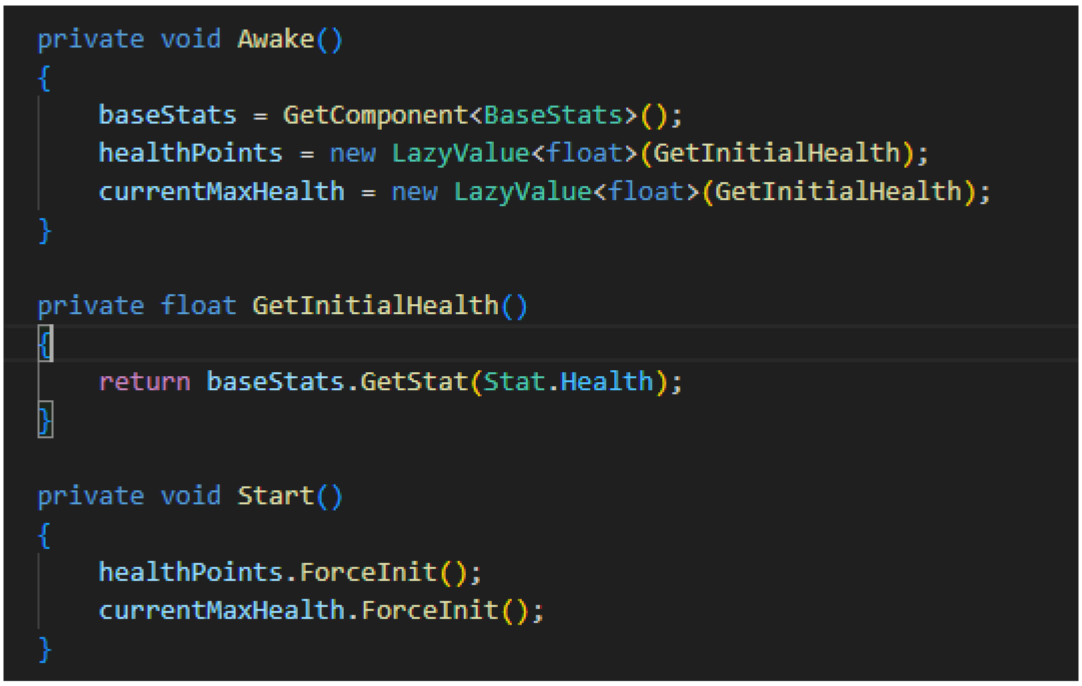
After a period of pursuit, if the player has moved out of the enemy's pursuit range and timeSinceAggrevated has exceeded the aggroCooldown threshold, the enemy will stop (cancel its current action) and stand still for a certain suspicionTime before returning to its previous task before detecting the player.

5.3 Lazy Evaluation

In Chapter 3, Lazy Evaluation is also mentioned as one of the prominent technologies used in this project. In summary, Lazy Evaluation is a programming method for deferring a task until the result is actually required. It can be applied to optimize the performance of the game and achieve on-demand results.

One of the important benefits of lazy evaluation is that it helps prevent data hazards in programming. Data dependency hazards occur when there is interdependence between data and their execution order. There are three types of data hazards: Read after Write, Write after Write, and Write after Read. They are also known as race conditions. The most common case is Read after Write. This situation occurs when a command returns a result that depends on a data that has not been computed or accessed yet. In game programming with Unity, when we initialize multiple data that depend on each other in the Start() or Awake() commands of different classes, it is difficult to fully control which Start() or Awake() command will finish first, resulting in unexpected output data. To mitigate this, we want the Awake function to be called first when the class is initialized, so it can set up all the data that will be used for computation in this class and possibly in other public classes, ensuring that all states are safely accessible when Start is called. However, at Awake, the functions of MonoBehaviour and Scriptable Object classes are not allowed to be called, as these classes themselves also have Awake functions, leading to another race condition.

To ensure safety and accuracy in initializing and accessing data, Lazy Initialization, an implementation of lazy evaluation, will help us initialize these states before the Start() time. In this game, we use a package provided by GameDevTv called LazyValue to implement Lazy Initialization.



```
private void Awake()
{
    baseStats = GetComponent<BaseStats>();
    healthPoints = new LazyValue<float>(GetInitialHealth);
    currentMaxHealth = new LazyValue<float>(GetInitialHealth);
}

private float GetInitialHealth()
{
    return baseStats.GetStat(Stat.Health);
}

private void Start()
{
    healthPoints.ForceInit();
    currentMaxHealth.ForceInit();
}
```

Figure 5.6: How to store and intialize lazy value

For example, in the Heath class, instead of calling

healthPoints = baseStats.GetStat(Stat.Health)

in Start, which could lead to a race condition, we use LazyValue to initialize it safely from Awake. This healthPoints attribute is stored in a container and initialized by LazyValue at Awake. Note that now healthPoints will belong to the type LazyValue<float> instead of a simple float, so to access it, we need to use healthPoints.value. ForceInit is a function to ensure that this state is initialized when Start is called, assuming that in some cases it may not be called or used.

5.4 Unity Optimization

In addition to using advanced programming techniques to ensure the effective operation of the game, we also need to pay attention to the game's performance. Performance plays a crucial role in the quality of the game and is entirely dependent on how the programmer optimizes their product. Since Unity does not have the same amount of performance optimization as more advanced game engines, this is especially crucial for Unity games. As a result of my understanding of this problem, I have used several Unity capabilities as well as game programming tricks to improve the game's performance. First, the project will make use of the following programming techniques:

- Create cache for frequently used variables: such as GetComponent. If used multiple times, it is best to initialize the cache at Awake to avoid frequent calls in Update, as lookup functions often consume a lot of performance and time.

- Replace Update with Coroutines: unlike the Update function (or FixedUpdate), which is called every frame, Coroutines can pause execution at any time and resume later. For functions that do not need to be executed on every frame, prioritizing the use of Coroutines significantly reduces performance overhead.
- Prioritize identifying objects by Component or Class: comparing string properties also affects program performance. If tags must be used, CompareTag() will consume fewer resources compared to using tag == ".".
- Limit the use of FindWithTag and FindObjectOfType: both of these functions consume a significant amount of resources. Similarly, other popular Unity APIs such as SendMessage or Find.
- Only run code when necessary: place conditional constraints in Update to avoid continuous code execution, for example, running the program only when there are changes in data.
- Use TextMeshPro instead of Text: Unity's old Text component has become outdated and does not achieve maximum performance efficiency. Therefore, switching to TextMeshPro, which is better supported in newer Unity versions, is recommended.

Other than code improvement techniques, some other tasks can be performed in the Editor to enhance the game's performance:

- Prioritize primitive colliders: Use primitive colliders (sphere, capsule, box) over mesh colliders, which are less resource-intensive when there is a large number of colliders. The efficient order of colliders when used from highest to lowest performance impact: sphere > capsule > box > collider. Additionally, for mesh colliders, Convex Mesh Collider is lighter than Non-Convex Mesh Collider, although it may generate unnecessary colliders based on the actual shape of the object.
- Occlusion Culling: This is a feature of Unity that disables the rendering of objects obscured by objects in front of them. When the game is running, all objects within the camera view are rendered simultaneously, including those that would have been obscured from the camera's perspective (frustum culling). Reducing the number of renderers at the same time to avoid wasting resources significantly improves game performance. To set up occlusion culling, tags are assigned to objects. Occluder Static for objects that can hide other objects

from view, and Occludee Static for objects that can be hidden by other objects, but cannot hide other objects themselves (usually small, translucent, or blurry objects). Then, go to the Occlusion Culling window and bake the scene.

- Increase the number of Canvases to reduce CPU usage: when a UI on a canvas changes, all properties within that canvas need to be redrawn. Avoid putting all UI elements together in one canvas to avoid this. This can be applied to the main menu of the game; however, it may not improve much since the menu is not a critical scene that requires optimization.
- Baked Lighting: Real-time lighting is calculated on each frame in real-time based on the camera's perspective for rendering. This can be quite resource-intensive. Instead, we can use Baked Lighting, which calculates all lighting in the scene in advance and exports them to 2D texture lightmaps applied to the scene. Baked Lighting is particularly effective for complex 3D scenes, significantly improving performance.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

Knights and Wizards is a PC game designed in the RPG style set in a medieval context. The game possesses features that contribute to the characteristic of RPG games, with simple yet engaging and challenging gameplay for players.

Through my learning and usage of Unity, as well as experiencing various other games to gain a multidimensional perspective, I have built a game that utilizes many prominent features of Unity to support and implement advanced programming techniques in the game design process. The assets used in the game are also diverse and rich in color and style, effectively portraying the desired setting. This has resulted in an engaging and interesting game with complex and advanced features. For a student with limited experience in game programming, this can be considered a complete and decent product. However, compared to other RPG products on the market, the game still lacks many aspects.

A good RPG must consist of elements such as a compelling and well-developed storyline, an expansive and well-crafted world-building with deep, meaningful choices and consequences. Proper character progression and challenging gameplay are also needed to improve the overall quality. In terms of these factors, Knights and Wizards still have much room for improvement. It needs to enhance various aspects to truly capture the attention of customers in the market. Additionally, although some advanced programming techniques have been integrated into the game development process, their utilization has been limited. More complex game features in the future will require even higher expertise. Lastly, the game's UI system is somewhat monotonous and lacks creativity in interface design, although it serves the purpose of displaying information, it can be dull and would benefit from more distinctive features or even breakthroughs.

Although there are areas that need improvement, the process of working on this project and developing the game has provided me with valuable experience. Being exposed to many useful built-in tools of Unity has opened up new opportunities in game design and development, such as designing movement with NavMesh or creating a database with Scriptable Objects. My programming skills have also significantly improved by utilizing new methods in both C# and Unity, as well as learning how to optimize game performance from programming to editor design with features like occlusion culling. Additionally, my soft skills, such as task and time management, workflow creation, and scriptwriting for presentations, have also progressed.

6.2 Future work

The game presented in the project still has many areas that need improvement before being released to the market. Specific goals need to be set for each part of the game to make it more polished.

Currently, the game consists of only four interconnected levels with various assets arranged to create distinct environments for each level. The number of levels is too few for a fully-fledged product, and there should be more new levels arranged to form a longer journey, allowing players to have more choices for progress. The assets used also need to be more diverse, visually appealing, and varied. The main obstacle to this is that all the assets used in the game are free assets, and in the future, with financial support, the quality of these assets will undoubtedly be much better.

The weapons in the current game are still limited and need more variety. In addition to weapons, more items can be designed for players, such as armor, shoes, hats, or consumable items. With more items like this, it is essential to build an inventory system for the player. This inventory will contain all the items that the player collects during the game, rather than immediately using them like in the current version. With it, players can possess multiple different items of the same type and choose which one to equip on their character.

The enemies in the game also need to be upgraded in terms of quality, especially the bosses. Currently, there are only knights and wizards, as the name suggests, but we should design more types of enemies to make the combat system more diverse and engaging. The most important aspect is the bosses. The game currently has only one boss in the final level, but there should be multiple sub-bosses in the intermediate levels to increase the challenge of the game. The moveset and weapons used by these bosses should also become more creative and groundbreaking, as boss fights in games are one of the most crucial elements.

REFERENCE

[1] A. Kaufman, *What was the first video game? detailing the first at-home video game, who invented it and more*. [Online]. Available: https://www.usatoday.com/story/tech/2022/09/15/what-was-the-first-oldest-video-game/8021599001/.

[2] R. Chikhani, *The history of gaming: An evolving community*. [Online]. Available: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/.

[3] *Mu online*. [Online]. Available: https://muonline.webzen.com/.

[4] *Audition*. [Online]. Available: https://au.vtc.vn/.

[5] *Audition*. [Online]. Available: http://web.gbviet.vn/.

[6] *Fpt*. [Online]. Available: https://fpt.com.vn/.

[7] *Vnggames*. [Online]. Available: https://vnggames.com/.

[8] L. Mỹ, *Việt nam nằm trong top đầu thế giới về sản xuất và phát hành game di động trên thị trường quốc tế*. [Online]. Available: https://vietnamnet.vn/viet-nam-nam-trong-top-dau-the-gioi-ve-san-xuat-va-phat-hanh-game-di-dong-tren-thi-truong-quoc-te-i416826.html.

[9] P. Studios, *Hoa*. [Online]. Available: https://www.pm-studios.com/Projects/Hoa.

[10] H. Games, *7554*. [Online]. Available: http://www.hikergames.com/vi/game/pc/7554-69.html.

[11] D. Studios, *Thần trùng*. [Online]. Available: https://store.steampowered.com/app/1726400/The_Death__Thn_Trng/.

[12] M. Studios, *Ori and the blind forest*. [Online]. Available: https://www.orithegame.com/blind-forest/.

[13] K. Nâu, *Chơi thử đồng cỏ lau demo: Hấp dẫn và đầy hứa hẹn*. [Online]. Available: https://motgame.vn/dong-co-lau-demo-hap-dan-va-day-hua-hen-c4123.game.

[14] V. Long, *Tựa game đề tài sử việt 300475 bị hủy bỏ, game thủ việt tiếc nuối cho một dự án đầy tiềm năng*. [Online]. Available: https://www.oneesports.vn/industry-news/hiker-games-dung-du-an-300475/.

[15] Blizzards, *Diablo immortals*. [Online]. Available: https://diabloimmortal.blizzard.com/en-us/.

[16] *Unity*. [Online]. Available: https://unity.com/.

[17] *Unity assest store*. [Online]. Available: `https://assetstore.unity.com/`.

[18] *Havok vision*. [Online]. Available: `https://www.havok.com/`.

[19] Activisions, *Call of duty*. [Online]. Available: `https://assetstore.unity.com/`.

[20] T. H. M. B, *Entity-component system là gì ?* [Online]. Available: `https://viblo.asia/p/entity-component-system-la-gi-oOVlYLPBZ8W`.

[21] *Gamedevtv*. [Online]. Available: `https://www.gamedev.tv/`.

[22] *Itch.io*. [Online]. Available: `https://itch.io/`.