Pathways of Destiny

University of Mauritius

Software Engineering Year 1 (2023)

SIS1040Y - Software Design Fundamentals and Programming

Presented by:

Name	*Student ID*
Prithvi Jay Krishna Deelah	**2211015**
Shivesh Ramjeeawon	**2210971**
Lovesh Dhoounmoon	**2211220**
Nileshwar Saleegram	**2211036**

Lecturer: Mr. **Somveer Kishnah**

Abstract

This report serves as the documentation behind the process and decision-making during the development of our new game called Pathways of Destiny. All details about the challenges we encountered during the development journey have been clearly outlined, alongside plenty of useful images and figures, to help the reader better comprehend the intricacies of the game. During the development period, an array of tools was used paired with informative guides, which have all been referenced.

Table of Contents

1. Introduction

1.1 Introduction

Pathways of Destiny is an open world 3D fantasy role-playing game which takes place in a medieval style universe with breathtaking sceneries. It is built upon the Unity engine, using C#, for its viability and ever-increasing support. The style and story of Pathways of Destiny draws a profound inspiration from other video games titles such as the Dark Souls series, Elder Scrolls V: Skyrim, Genshin Impact among others. As players embark on this exhilarating journey, they will have to battle through various enemies, mini bosses and use their ingenuity to solve puzzles before engaging the final boss.

1.2 Problem Statement

Most RPGs nowadays put an overwhelming emphasis on the endgame content to the detriment of their story and casual playlists. Consequently, this approach creates a high bar for new players who are willing to venture into the RPG/MMO-RPG genre but are unable to dedicate the time necessary to conquer those goals. The endless repetitive grind in the late to end game ubiquitous to RPG style games is also a discouraging factor, dissuading potential players from even considering engaging with this genre.

1.3 Aims & Objectives

1.3.1 Aims

We aim to make a game where the player will have to use their puzzle-solving skills more to advance in the world without being too difficult to scare off casual players. We also wish to keep the "power grind" to a minimum, so as to not fall victim to the "power creep" issue plaguing other RPGS, but high enough so that the player does feel justly rewarded for their time and skill invested in the game. We wish to make an inviting open world, full of surprises for the player to find and provide them with a deep storytelling experience which extends beyond just the main quest of the story. "It's about the journey, not the destination," is the philosophy we adopted for the universe when developing Pathways of Destiny.

1.3.2 Objective

The game shall instill withing the player a sense of exploration and curiosity through the world building and story. It shall provide the player with fun and challenging puzzles to advance in the world over the typical power grind. The game will be riddled with a bunch of easter eggs which can provide rewards when found. Boss fights shall put the player's reflexes along with their macro and micro skills to the test. These will hopefully revitalize their interest in RPG games and thus prompt them to comeback during major updates to the game.

1.4 Scope

The scope of the game is as follows:

- The game is set in an open-world medieval-style universe.
- The game will be available as single player.
- The game will be played from a third-person perspective.
- Players will have the option to choose from various characters to play from.
- The game will be set in a 3D environment.
- The game will progress with increasing difficulty, but not feature a hard power requirement.
- The game will have an in-game economy which the players can interact with to purchase various in-game items.

2. Background Study

2.1 Basic Concept

Pathways of Destiny is an adventurous role-playing game. The objective of the game is to explore an open world map in order to defeat area bosses to acquire key items for the final area. Just like any other role-playing game, the user will be able to play as one of four given characters and beat the game without the character’s health reaching zero.

2.2 Existing Applications

2.2.1 Dark Souls

Released in 2011, Dark Souls is an adventure and storytelling game that takes place in a magical and medieval world. You play as a third-person character, your aim is to fight monsters throughout the game and talk to Non-Playable Characters (NPCs) in order to advance the progress of the story in the game. By defeating enemies, you accumulate points which is used to level up your stats for your character. You make use of different types of hand weapons and magic spells for your combat which results in different types of play-experience depending on your choice of weapon.

2.2.2 Genshin Impact

Genshin Impact is an action role-playing game developed by miHoYo. It is available on all gaming platforms. The game features an anime-style open-world environment and an action-based battle system using elemental magic and character-switching. You can play the game using various characters which have unique abilities. This game requires an internet connection to play.

2.2.3 The Elder Scroll V: Skyrim

The Elder Scrolls V: Skyrim is a fantasy action role-playing game, playable from either a first- or third-person perspective. The player may freely roam over the land of Skyrim, an open world environment consisting of wilderness expanses, dungeons, caves, cities, towns, fortresses, and villages. Each city and town in the game world has side activities or side quests for the player to engage in. Players may navigate the game world more quickly by riding horses or utilizing a fast-travel system that allows them to move their character immediately to a previously discovered location. The player may also hunt animals for crafting repertoires.

2.2.4 Proposed Features

After a careful analysis and evaluation of the existing games, we have decided to implement some features into our application.

From Dark Souls:

- Include the combat system.
- Include the environment style.
- Include the Third person Camera.
- Include the roll/dodge animation.

From Genshin Impact:

- Include the unique abilities for characters.
- Include the Co-Op system.
- Include the weapon classes.
- Include the bow shooting system.

From The Elder Scrolls V: Skyrim:

- Include the Fast Travel system.
- Include the crafting system.
- Include the health bar system.

2.3 Potential Tools

Tools that were considered during the development of Pathways of Destiny game.

2.3.1 Game Engines

A game engine is a software framework that provides developers with tools and functionalities to create video games. It handles tasks such as rendering graphics, physics simulation, audio management, and asset management. Game engines enable developers to focus on game design and logic without needing to build every aspect from scratch.

Unity Engine

Unity is one of the most widely used game engines, known for its versatility and ease of use. It supports multiple platforms, including PC, consoles, mobile devices, and VR/AR. Unity offers a large community and an extensive asset store, making it easier to find resources and support. It uses C# and provides visual scripting with Bolt.

Unreal Engine

Unreal Engine, developed by Epic Games, is renowned for its high-quality graphics and realistic rendering capabilities. Unreal Engine supports a wide range of platforms and provides a visual scripting system called Blueprint, making it accessible to non-programmers. It offers a robust set of tools for creating immersive 3D environments and supports advanced features like physics simulation, AI, and multiplayer networking. It uses C++ as its main programming language.

CryEngine

CryEngine is a powerful game engine known for its stunning visuals and realistic lighting effects. It offers an intuitive visual scripting system called Flowgraph, which allows designers to create gameplay mechanics without coding. CryEngine provides a wide range of tools for creating detailed environments and supports advanced features like physics-based animation and dynamic weather effect.

Godot

Godot is an open-source game engine that has gained popularity for its user-friendly interface and flexibility. Games can be created either in C++ or C# among many other programming languages. Godot supports 2D and 3D game development, provides a comprehensive set of tools, and allows for easy exporting to multiple platforms. It also has an active community and extensive documentation.

2.3.2 3D Graphic Softwares

3D graphic software enables artists to create and manipulate three-dimensional digital models, animations, and visual effects. These software tools provide artists and designers with a range of features such as modeling, texturing, rigging, animation, and rendering. They are crucial in industries like gaming, animation, visual effects, architecture, and product design, enabling the creation of lifelike and immersive 3D content.

Maya Autodesk

Maya is a comprehensive 3D modeling, animation, and rendering software widely used in the entertainment industry. It offers a wide range of tools for creating detailed 3D models, character animation, and visual effects. Maya supports various rendering engines and integrates well with other software in the Autodesk suite.

Blender

Blender is a free and open-source 3D creation suite that offers a full range of features for modeling, animation, rigging, rendering, and more. It has an active community and extensive documentation making it beginner friendly. Blender supports various file formats and is suitable for both professional and independent creators.

Adobe Dimension Dn

Adobe Dimension, is a 3D modeling and character creation tool that allows users to create and customize 3D human characters for use in various projects, including games, animations, and visualizations. It provides a user-friendly interface and integrates well with other Adobe Creative Cloud applications.

Cinema 4D

Cinema 4D is a versatile 3D software known for its user-friendly interface and powerful capabilities. It provides a wide range of tools for modeling, animation, texturing, and rendering. Cinema 4D is used in various industries, including motion graphics, visual effects, and product visualization.

2.3.3 2D Graphic Softwares

Adobe Photoshop

Adobe Photoshop is a versatile and widely-used software for creating and editing 2D images. It offers a comprehensive set of tools for photo editing, digital painting, graphic design, and more. Photoshop provides advanced features like layers, filters, and a wide range of brushes.

Affinity Designer

Affinity Designer is a professional-grade vector graphics editor known for its performance and versatility. It offers a wide range of tools for creating precise illustrations, icons, and UI designs. Affinity Designer supports both vector and raster workflows and provides features like advanced blending modes and artboards

Gimp

GIMP (GNU Image Manipulation Program) is a free and open-source software that offers powerful 2D image editing capabilities. It provides a variety of tools for photo retouching, image composition, and graphic design. GIMP supports layers, filters, and customizable brushes.

2.3.4 Integrated Development Environment

An IDE is a software application that provides a comprehensive set of tools for software development. It typically includes features such as code editing, debugging, compiling, and project management in a unified interface. IDEs enhance productivity by offering features like code completion, syntax highlighting, and project templates. They are widely used by developers to write, test, and debug code efficiently within a single development environment.

MonoDevelop

MonoDevelop is an open-source IDE that is often used for scripting in Unity game development. It offers code editing, debugging, and project management features, with a focus on C# and Unity development workflows.

Visual Studio

Visual Studio is a widely used IDE that supports various programming languages and is suitable for scripting in game development. It offers powerful code editing features, debugging tools, and integration with game engines like Unity and Unreal Engine.

Atom

Atom is a customizable and free source code editor by Github that is widely used for scripting in game development. It offers a wide range of packages and themes, and its interface is highly customizable to suit individual preferences and workflows.

Visual Studio Code

VS Code is a lightweight, free, and highly extensible code editor that supports scripting in various languages. It offers a wide range of extensions and plugins to enhance functionality for game development, including integration with game engines and popular scripting languages

2.3.5 Additional Tools

Mixamo

Mixamo is an online platform that provides a vast library of pre-made 3D character animations. It offers a user-friendly interface where one can browse, customize, rig and apply animations to their own 3D characters. Mixamo simplifies the process of adding high-quality animations to games or 3D projects.

Github

GitHub is a web-based platform for version control and collaboration, primarily used for hosting and sharing software development projects. It provides features like code repository hosting, issue tracking, pull requests, and team collaboration tools. GitHub enables developers to work together, manage project versions, and track changes efficiently.

2.4 Summary of findings

Pathways of Destiny will be an RPG game where the player chooses from four unique characters, each with their own abilities, to defeat regional bosses and face the ultimate challenge of the final boss.

Following extensive research, we decided to use Unity as game engine as it is beginner friendly and has a plethora of video tutorials. C# will be used as scripting language since it is more suitable for the development of the proposed game. Additionally, some of the proposed tools could be used to some extent at certain stages throughout the game's development.

3. Analysis

3.1 Proposed System

Pathways of destiny is a third-person role-playing, action & adventure game where the players choose between four different characters each with unique skills and ability.

Menus that are in the game:

1. Main Menu: New Game, Load Game (if player has a previously saved game), Options, Credits, Quit
2. Setting Menu: Graphic setting, Sound setting, Help
3. Pause Menu: Continue, Settings, Exit to Main Menu, Quit Game

Each of the four characters can perform basic movements such as: move around, jump, attack and block attack.

The differences in characters are their various choices in weapon.

The main objective of the player in this game is to complete different goals in which they fight local bosses and finally the main boss in the end.

Players can upgrade their respective weapon at a weaponsmith using coins they collected by completing goals and killing enemies.

The weapon smith sells different kinds of upgrades and potions suitable for the player's respective character.

3.2 Functional Requirements

1.Basic Game Requirements		
ID	Requirements	Priority
FR100	The game shall have a title screen	1
FR101	The game shall have a start menu	1
FR102	The game shall have a Option menu	1
FR103	The game shall have a character selection panel	1
FR104	The game shall have a win/end condition	1
FR105	The game shall have a player and enemies	1
FR106	The game shall have a Main Menu scene, Selection Menu scene, Village scene, Level1 scene, Level 2 scene, level 3 scene	

2.Start Menu Requirements		
ID	Requirements	Priority
FR200	The start menu shall prompt player to choose character upon selecting ‘New Game’ option.	2
FR201	The start menu shall load the existing scene, player health and position if player selects ‘Load Game’ option.	1
FR202	The start menu shall display the setting panel when user selects ‘Options’ option.	1
FR203	The game shall quit when user selects ‘Quit’ option.	1
FR204	The start menu shall display the credits panel when user selects ‘Credits’ option.	1
FR205	The selected option shall play a sound upon clicking.	2

3.Options panel Requirements		
ID	Requirements	Priority
FR300	The setting panel shall display graphic panel when user selects ‘Graphic’ option	1
FR301	The setting panel shall display Sound panel when user selects ‘Graphic’ option	1
FR302	The setting panel shall display help panel when user selects ‘Help’ option	1
FR303	The setting panel shall display Main Menu panel when user selects ‘Back’ option	1
FR304	The selected option shall play a sound upon clicking.	2

4.Pause panel Requirements		
ID	Requirements	Priority
FR400	Pause panel should be displayed and game should be paused when user presses ‘Esc’ key	1
FR401	The pause panel shall contain ‘Continue’, ’Options’, ‘ Main Menu’ and ‘Quit’ options	1
FR402	The game shall resume when ‘Resume’ option is selected	1
FR403	The ‘Options’ panel shall be displayed if it is selected	1
FR404	The game shall load the ‘Main Menu’ scene when ‘Main Menu’ is selected	1
FR405	The game shall quit when ‘Quit’ option is selected	1
FR406	The selected option shall play a sound upon clicking.	2

5.Main Story		
ID	Requirements	Priority
FR500	The system shall have 3 different levels on different scenes	1
FR501	The system shall show have ogres, a mini boss and a final boss.	1
FR502	The system shall have ogres and a mini boss on level 1.	1
FR503	The system shall have ogres level 2	1
FR504	The system shall have ogres and final boss on level 3	1

6.Player Movement Requirements		
ID	Requirements	Priority
FR600	The game shall allow basic character movement upon pressing WASD keys.	1
FR601	The game shall allow character to jump upon pressing ‘space bar’ key	1
FR602	The game shall allow character to sprint upon pressing ‘shift key	1
FR603	The game shall allow character to attack upon pressing ‘left click’ on the mouse	1
FR604	The game shall allow player to block attack upon pressing the ’Q’ key if selected character possess the ability	1
FR605	The game shall allow character to perform ‘Heavy Attack’ upon pressing ‘shift’ key and ‘left click’ on the mouse	1

FR606	The game shall allow character to perform a combination attack upon pressing ‘left’ mouse key repeatedly if selected character possess the ability	2
FR607	The game shall allow player to aim upon holding the ‘right’ mouse key is given selected character possess that ability	1
FR608	The game shall allow the given character to throw weapon or shoot arrow when left mouse key is pressed while holding the right mouse key if selected character possess the ability	1
FR610	The system shall play animation upon performing the respective movement.	1
FR611	The system shall allow player to interact with game object if player is in a given range.	1
FR612	The system shall play sound for the respective animation.	2

7.Enemy Gameplay		
ID	Requirements	Priority
FR700	The enemy shall attack the user	2
FR702	The system shall play respective enemy animation such as walking and attacking	1
FR703	The boss should have different phases	1
FR612	The system shall play sound for the respective animation.	2

8.Gameplay Chest		
ID	Requirements	Priority
FR800	The system shall play a sound when the user collects open the chest.	1
FR801	The coins shall automatically collect regeneration ability when he is within a range	1
FR802	The chest shall remain open after the player collects them.	1

9. User Interface		
ID	Requirements	Priority
FR900	The system shall display the current health of the player in a health bar	1
FR901	The system shall display the cool down time for ‘Heavy attacks’	1

11. NPC		
ID1100	Requirements	Priority
FR1101	NPC shall only interact with player within a range	1
FR1102	The pop-up panel to interact with the NPC shall be displayed within that range	1
FR1103	The system shall play the NPCs' respective animation	2

12. Selection Menu		
ID	Requirements	Priority
FR1200	The system shall have 4 different characters.	1
FR1201	The system shall display character and character descriptions.	1
FR1202	The system shall display the next character and description when the user selects '>'	1
FR1203	The system shall display the previous character and description when user selects '<'	1
FR1204	The system shall load the village scene when the user selects 'Select' option.	1
FR1205	The selected option shall play a sound upon clicking.	2

3.3 Non-Functional Requirements

ID	Requirements
NFR 1	The game must run on Windows 10/11 and Linux
NFR 2	The game should be developed in Unity Game Engine
NFR 3	The game should be developed in C#
NFR 4	The system shall not contain many bugs.
NFR 5	The game shall run at least 50 fps on an intel i5 8400 8th generation chip with an integrated graphic card.
NFR 6	The game shall be less than 1 GB

3.4 Tools Used

Unity Game Engine
Compared to other game engines, Unity is beginner friendly with large amounts of documentation and tutorials available. It also offers multi-platform and long-term support (LTS).

Visual Studio 2019
Visual Studio 2019 is also a beginner friendly IDE that supports C# and has an IntelliCode feature to speed up the coding process. Additionally, it is also compatible with unity.

Mixamo
It offers a plethora of already rigged animations and characters free of charge. It also allows user to put animations on chosen rigged character.

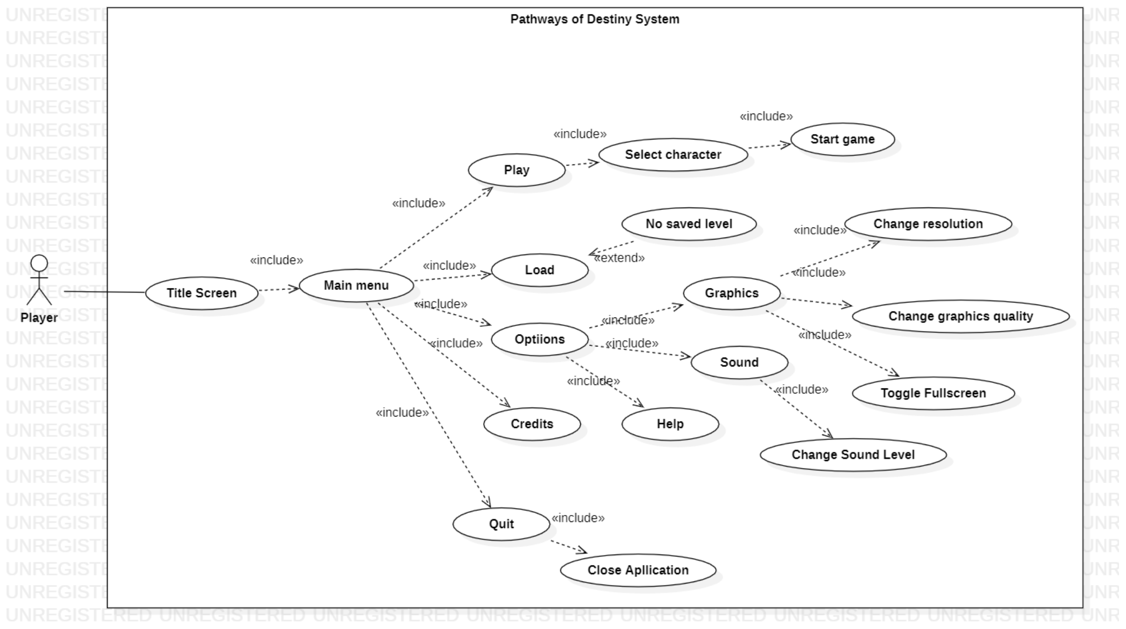
Adobe Photoshop
Adobe Photoshop allowed us to create mockup of characters and other visual elements such as the title and logo designs through its extensive range of tools and features for artistic work.

Gimp
It is a graphic design tool that allowed us to finalize the logo and title design of the game. It is a great free alternative to Adobe Illustrator.

Blender
Blender is a free 3d object design tool that is beginner friendly with a variety of documentation and tutorial available.

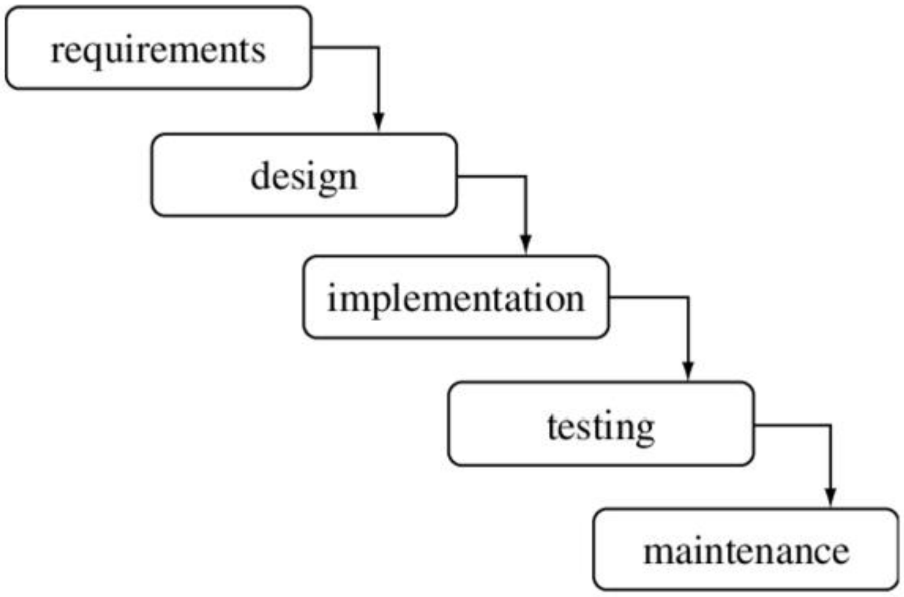
Github
We can easily collaborate on the same project using Github as it allows us to make changes without requiring to transfer large files**.**

3.5 Use Case Diagram



4 Design

4.1 Modelling Approach



The Waterfall model can be considered a suitable approach for developing a game in Unity under specific circumstances. This linear and sequential methodology involves distinct phases like requirements analysis, design, implementation, testing, and deployment. For a Unity game development project, where the scope and requirements are well-defined from the outset, the Waterfall model offers benefits such as a clear and structured process. This is particularly useful when the game's design and mechanics are less likely to undergo significant changes during development. Unity's integrated development environment provides a stable platform, and the Waterfall model's step-by-step approach aligns well with Unity's component-based architecture. However, it's crucial to acknowledge that the Waterfall model's rigid nature can be less adaptable to the iterative nature of game development, where creative experimentation and user feedback might lead to frequent design changes.

The Waterfall model could be considered suitable for developing a game under certain conditions and requirements. Here are some reasons why it might be a viable choice:

1. Well-Defined Requirements: If the game's concept, mechanics, and overall design are thoroughly documented and unlikely to change significantly during development, the Waterfall model's structured approach can help ensure that the project stays on track.

2. Stability in Design: Waterfall is conducive to projects where the design is stable from the beginning. Game projects that have a clear vision and are less likely to undergo major design changes can benefit from the model's sequential nature.

3. Predictable Timeline: Waterfall's linear progression allows for relatively accurate estimation of timelines and resource allocation. This can be advantageous for planning the development cycle of a game, ensuring that milestones are met and resources are managed efficiently.

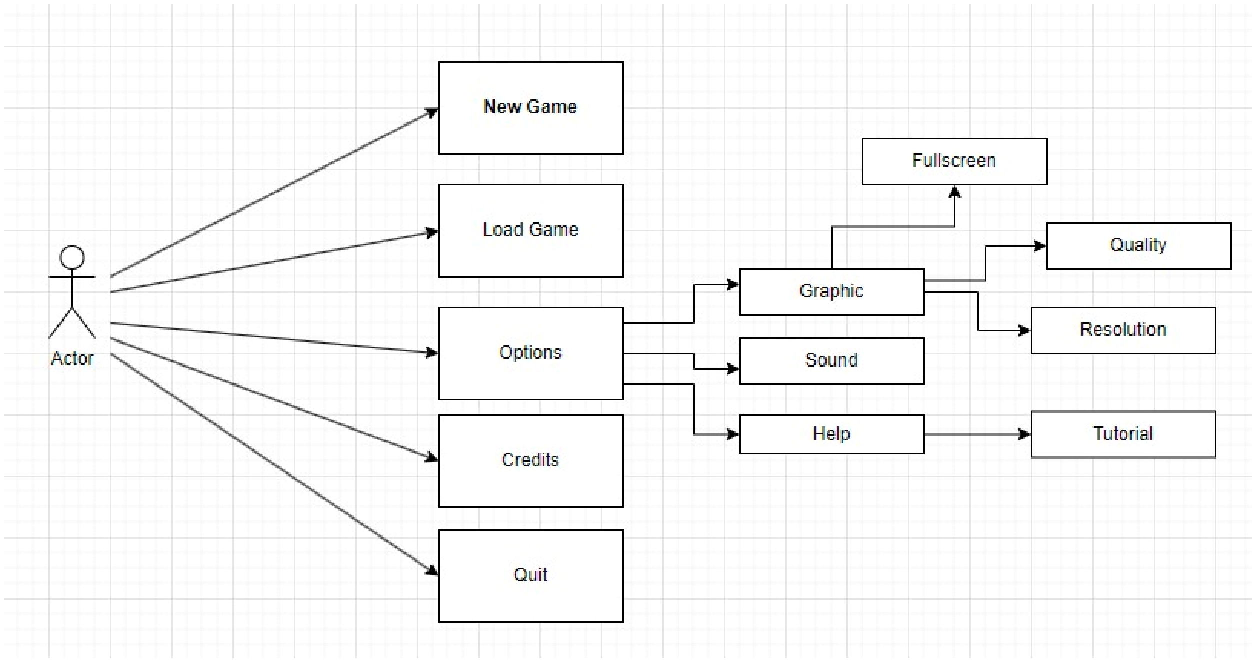
4. Clear Phases: The distinct phases in the Waterfall model, such as design, implementation, and testing, facilitate a focused and organized development process. This can be particularly useful when managing a large development team.

5. Unity's Integrated Environment: Unity provides a comprehensive and stable environment for game development. Since Unity games often follow a component-based architecture, the structured phases of Waterfall can align well with Unity's development approach.

6. Resource Allocation: Waterfall's upfront planning can help in allocating resources like budget, staff, and technology appropriately at the beginning of the project.

4.2 Architectural Design

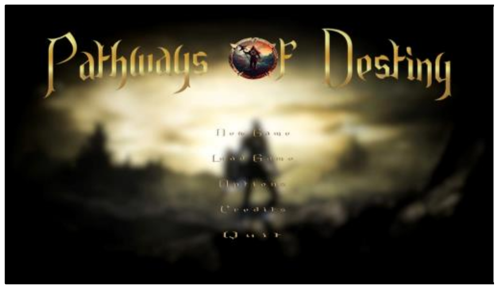
The diagram below shows a representation of how the system shall perform.

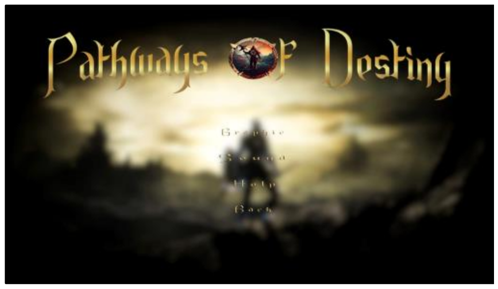


4.3 UI Design

Title Screen

Main Menu





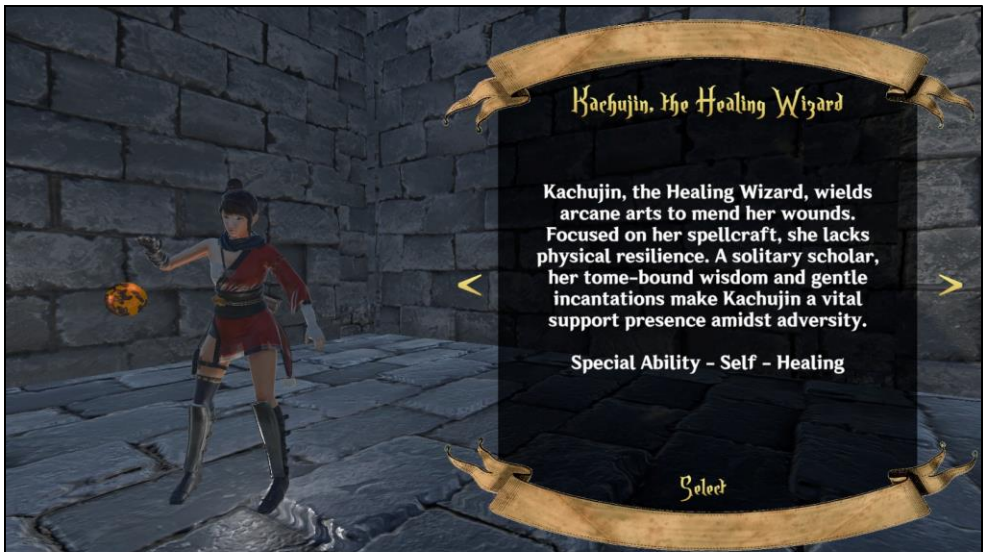
The Main Menu shall be display when the game starts and the following buttons will be available for interaction:

- New Game
- Load Game
- Options
- Credits
- Quit

The 'Load Game' button will load any previous existing game if any. If there is no Saved game, the no saved game panel shall be displayed

The 'Options' button will open another menu where the graphics, sound and help button will be available for changes the user wish to make or help.

After the 'New Game' button is pressed, the user will be taken to the character selection screen:



Kachujin, the Healing Wizard, wields arcane arts to mend her wounds. Focused on her spellcraft, she lacks physical resilience. A solitary scholar, her tome-bound wisdom and gentle incantations make Kachujin a vital support presence amidst adversity.

Special Ability - Self - Healing

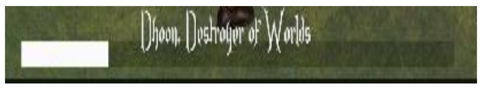
Pressing the 'Credits' button will play the credits scene.

The 'Quit' button closes the application window.

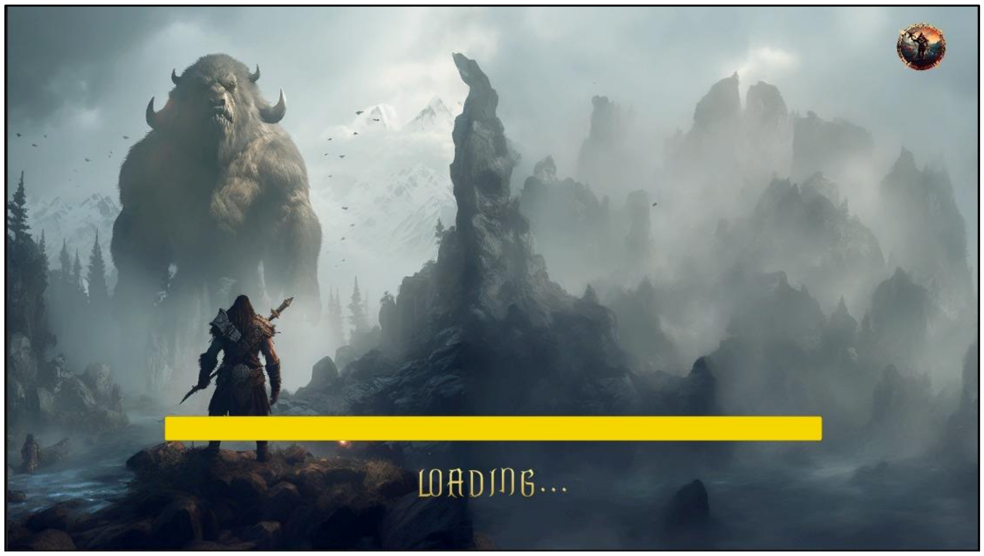
In addition to the Main Menu, there are also other UI design such as:

- Player Health

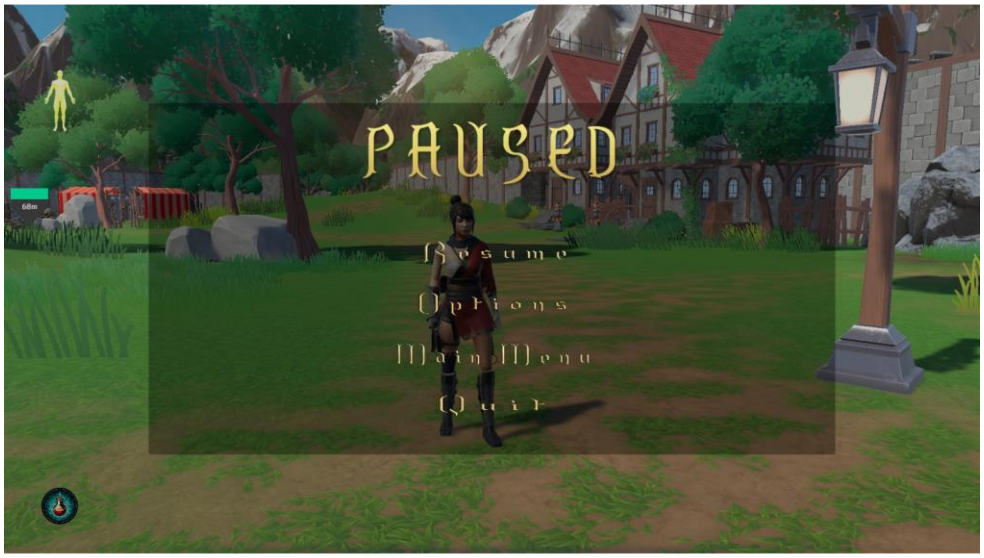
- Enemy Health



- NPC Dialogue
- Loading Screen



- Pause Menu



When approaching an enemy, the health bar will appear on top of it and will gradually decrease when it takes damage.

When approaching a non-playable character, a pop-up message will be displayed at the bottom of the screen with a continue button (for more messages to load) in order to read messages given by NPCs.

A Loading Screen will be made in order to change scenes, it will be comprised of an image background and a slider to represent the loading progress.

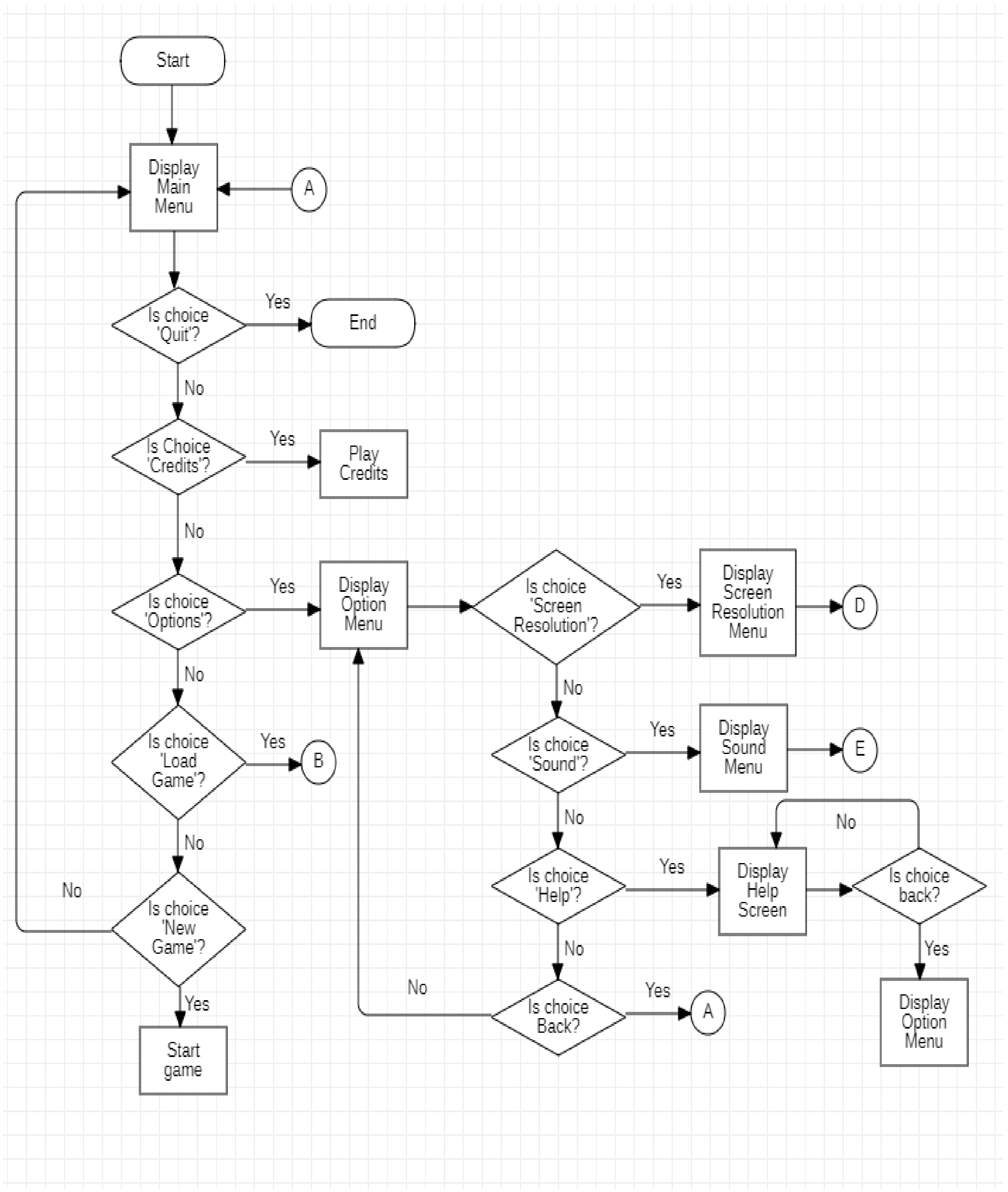
When the character’s health reaches zero, the player will die and a Death Screen indicating ‘YOU DIED’ will appear on screen.

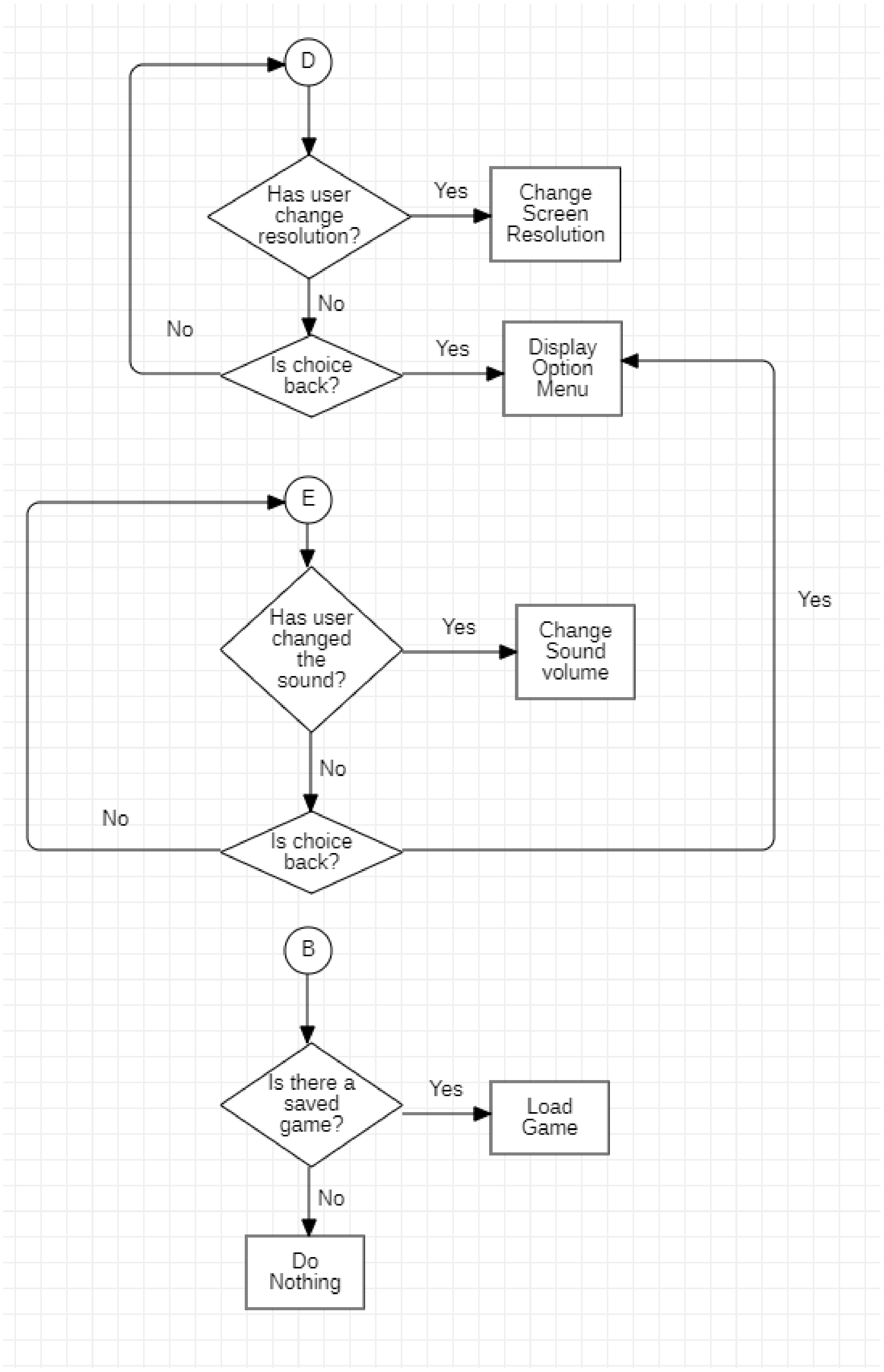
When the player has beaten the final boss of the game, a Winning Screen indicating ‘YOU WON!’ shall be displayed onto the screen.

The system shall also have a Pause Menu in order to pause the game for purposes like saving, change settings or quit the game. The Pause Menu allows to resume game.

4.4 Program Designs

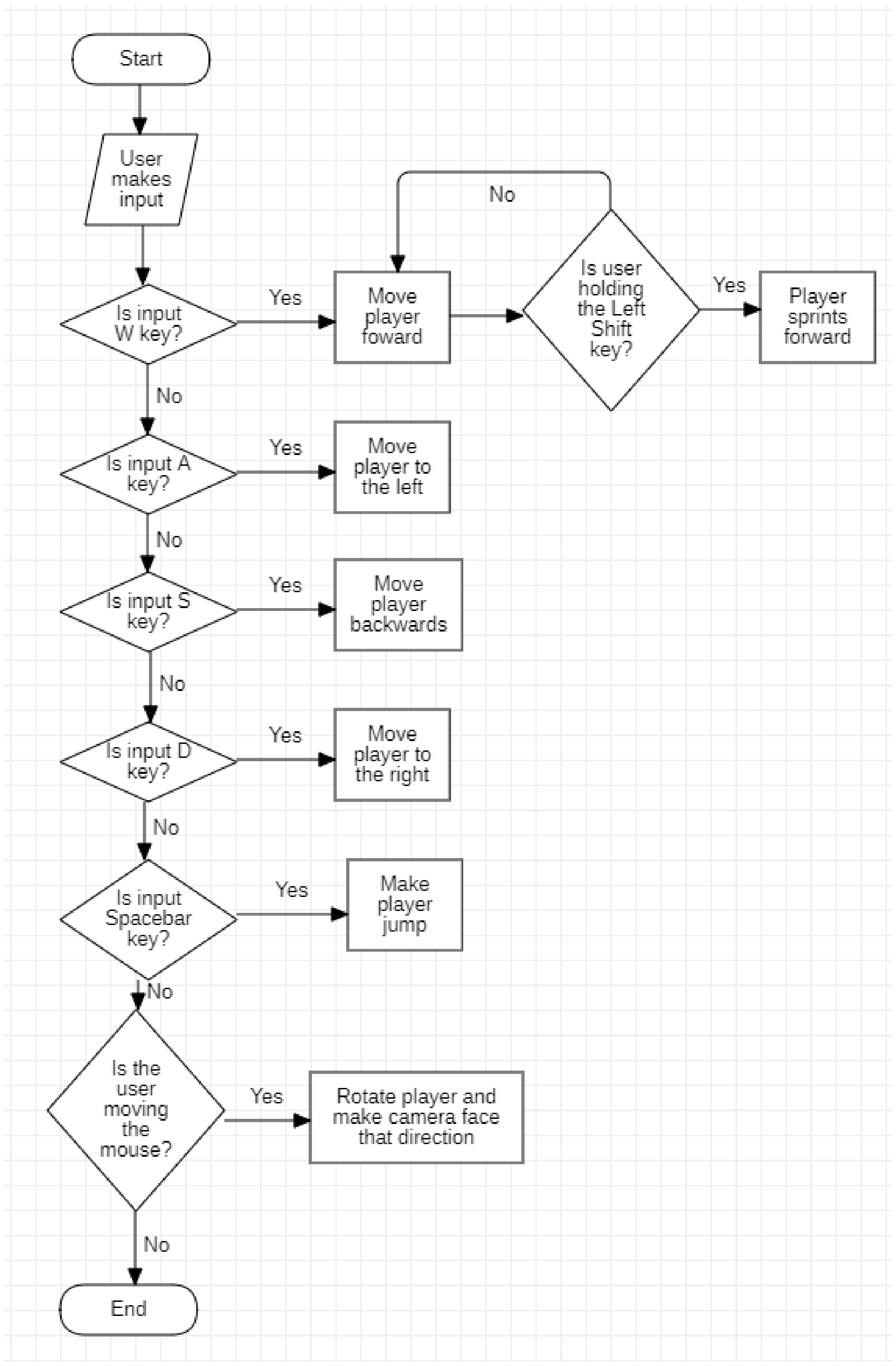
4.4.1 Menu Navigation Flowchart



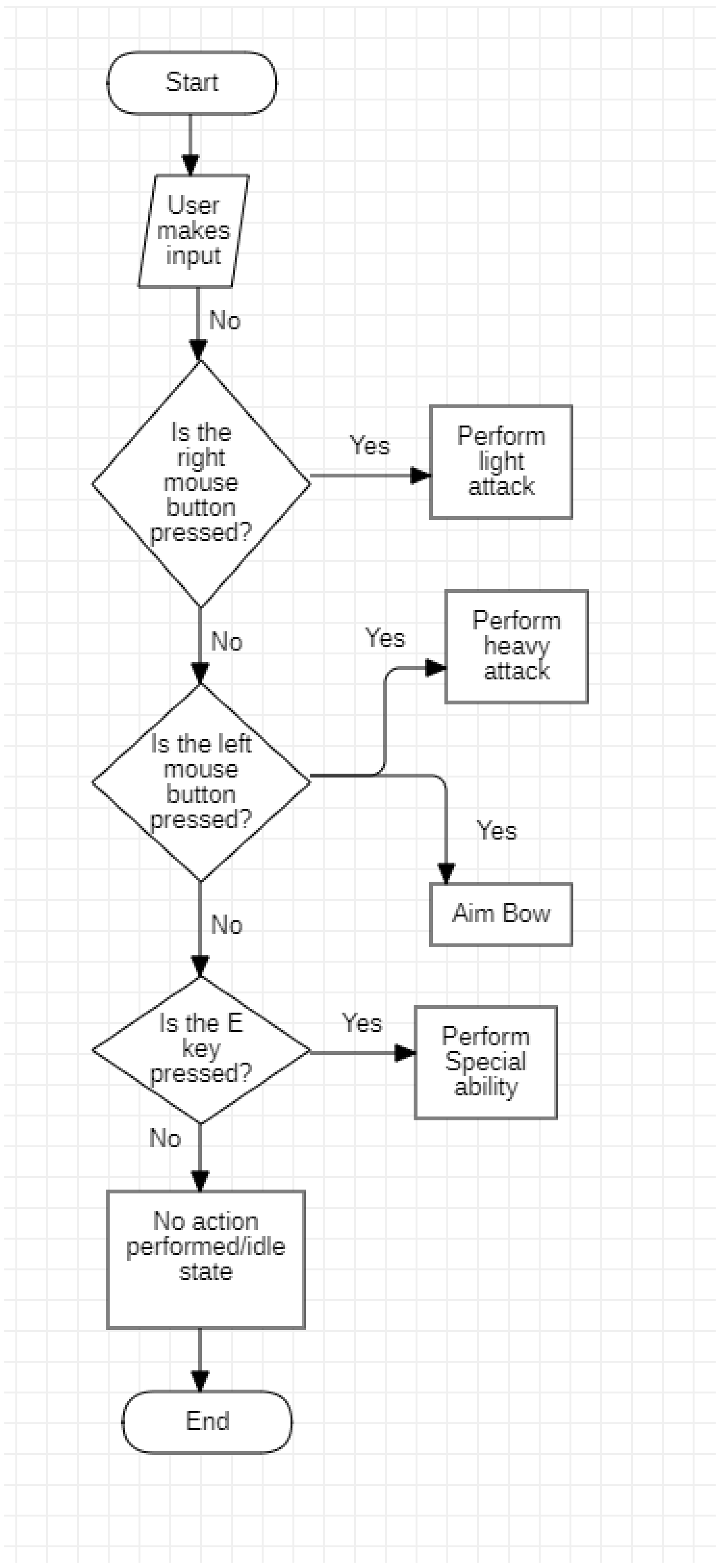

D
Has user change resolution?
Yes
Change Screen Resolution
No
No
Is choice back?
Yes
Display Option Menu
E
Has user changed the sound?
Yes
Change Sound volume
Yes
No
No
Is choice back?
B
Is there a saved game?
Yes
Load Game
No
Do Nothing

4.4.2 Player Movement Flowchart

The flowchart below represents the general movement that all characters in the game can do. Different characters can perform different tasks when user inputs a key. For example, the sword character rolls forward when the E key is pressed but the bow character does not.



4.4.3 Player Attack Flowchart



It is worth mentioning that similar to the Player Movement, the attack flowchart may differ for some characters as each character has some ability different from other. This is because in order to give the user a diverse play style while trying the 4 characters in the game. Nonetheless, the flowchart should englobe all the general attack schemas.

5. Implementation

5.1. System Requirements

● The system should have Windows 10 OS or a later version to be able to run the

application.

●The system should have a processor Intel Core i5 7th gen with 3.1 GHz or AMD

1400 and a memory of 3.2 GHz.

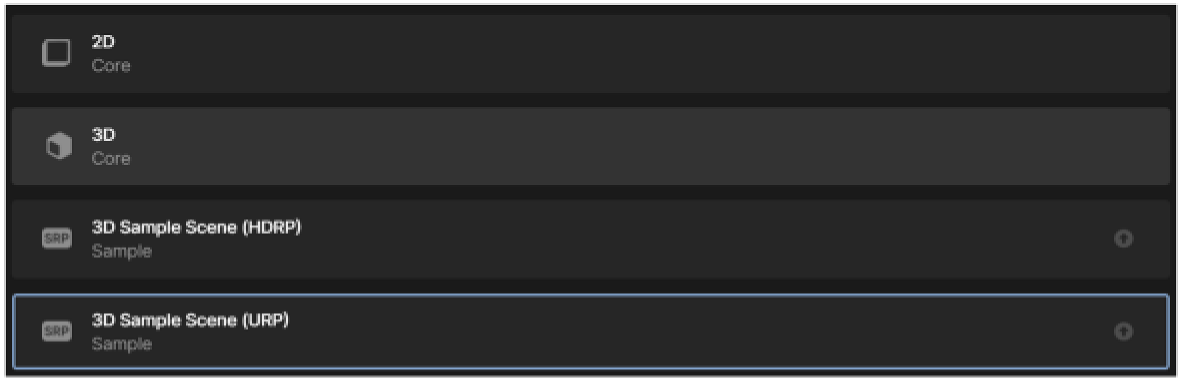
● The system requires a 4 GB RAM graphics, GTX 1050 TI

● The system should have 8 GB system ram

● The system requires a storage of 8 Gb.

5.2 System setup

When starting a project, unity allows us to choose different core package options. We chose to do our game on the 3D sample scene (URP) code package

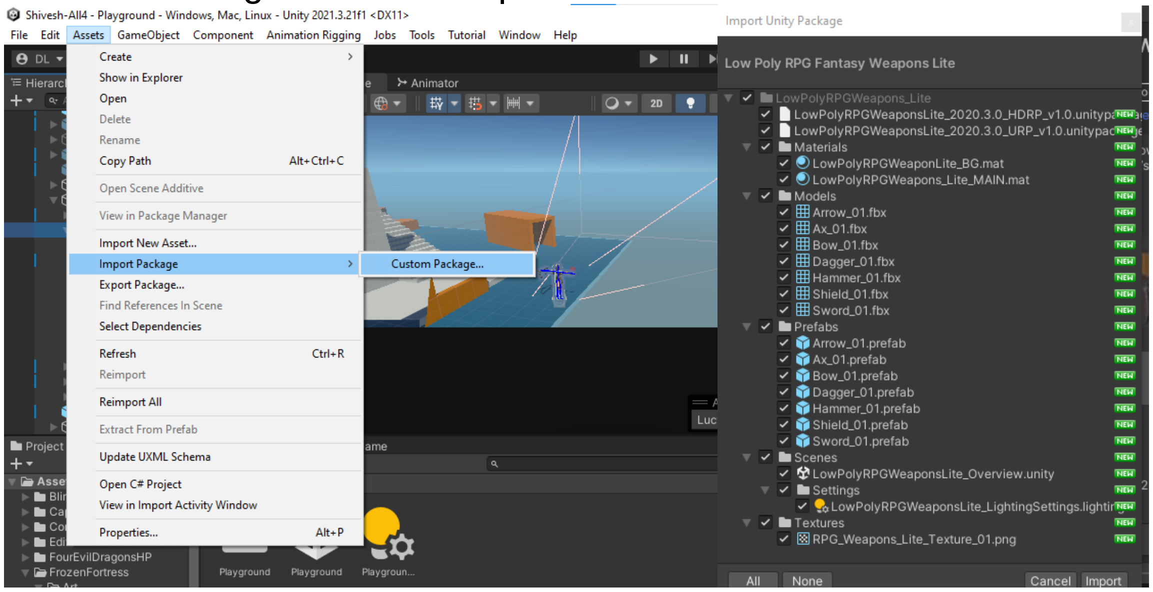


5.3 3D mode

- All assets imported will be considered 3D and the system will have a default orthographic view.
- The physics and rigid body will be in 3D
- The camera view is rendered from the Camera in 3D
- Allows to put 2D sprite for 2D interaction (Example: Menu)

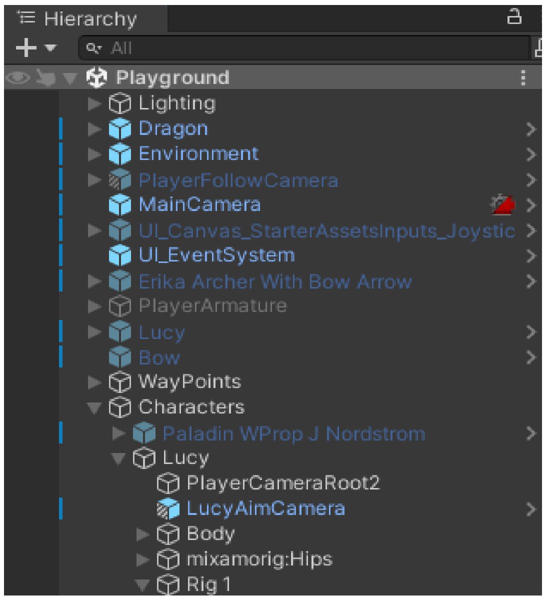
5.4 Import assets

Our game relies heavily on assets, which can be downloaded from the internet or created using a 3D modelling application. This will help us have a base for the game to build upon.



5.5 Hierarchy window

The hierarchy contain every game object in the current scene and allows us to manage all the objects found in a scene.



5.6 Our game consists of:

1. Players (4 different)
2. Special abilities
3. UI
4. NPC
5. Enemy
6. Boss
7. Crosshair (for aiming)

The Scripts and images on the following pages show the main components that are listed above. For some of them, part of the code will be shown.

5.7 Player

Paladin/ Sir Galahad:

Lucy:

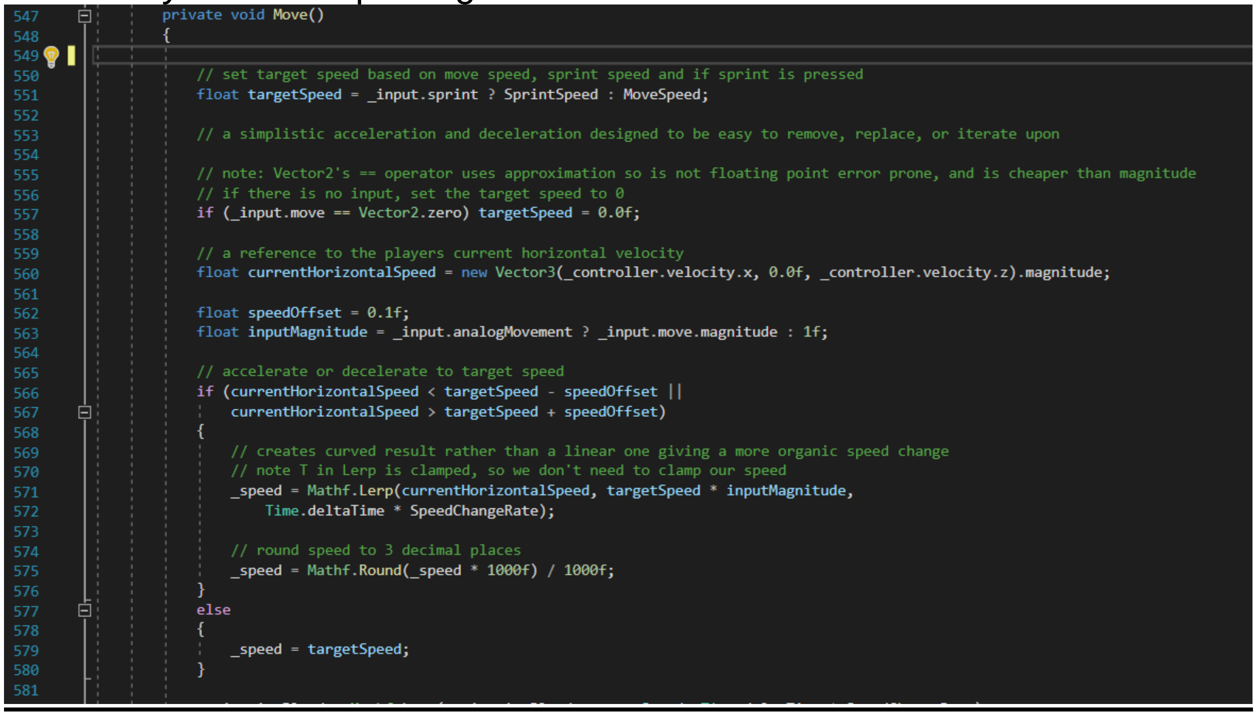
Warrior/Ragnar:

Wizard/ Kachujin:

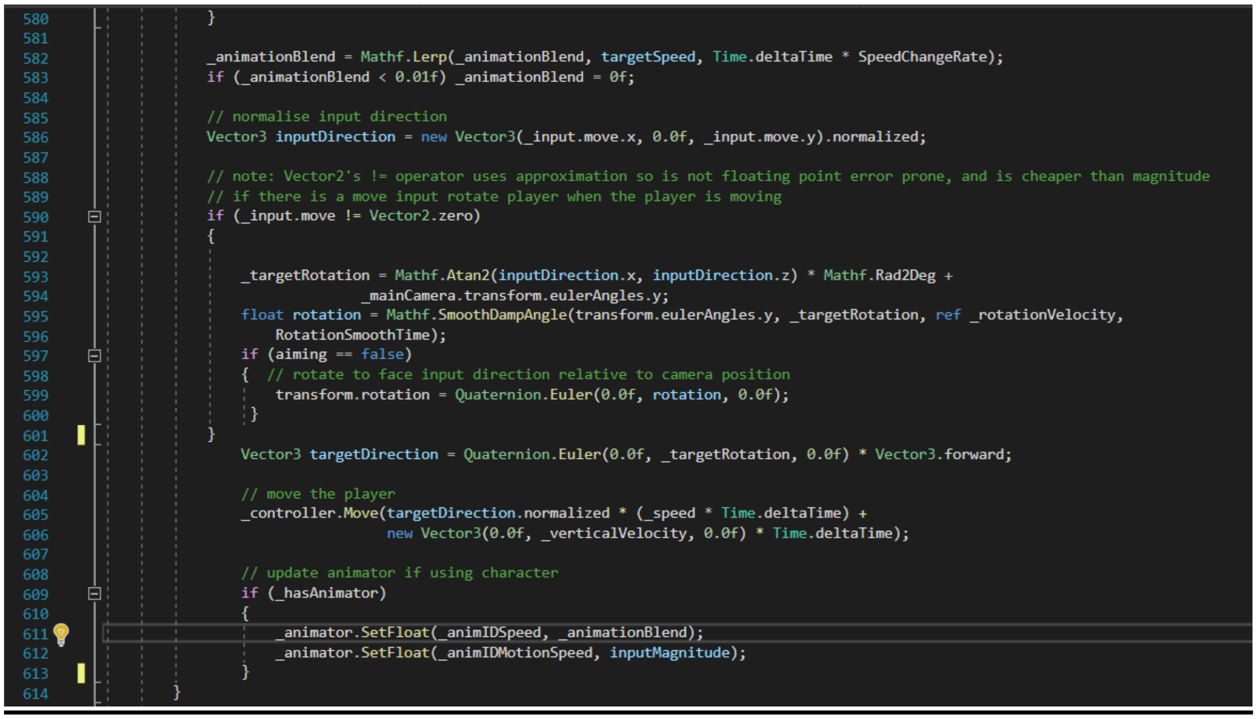
5.8 Player

Player controls

This code is a common one shared by all 4 character and is found in the basic unity controller package. We then added our code based on this



```
private void Move()
{

    // set target speed based on move speed, sprint speed and if sprint is pressed
    float targetSpeed = _input.sprint ? SprintSpeed : MoveSpeed;

    // a simplistic acceleration and deceleration designed to be easy to remove, replace, or iterate upon

    // note: Vector2's == operator uses approximation so is not floating point error prone, and is cheaper than magnitude
    // if there is no input, set the target speed to 0
    if (_input.move == Vector2.zero) targetSpeed = 0.0f;

    // a reference to the players current horizontal velocity
    float currentHorizontalSpeed = new Vector3(_controller.velocity.x, 0.0f, _controller.velocity.z).magnitude;

    float speedOffset = 0.1f;
    float inputMagnitude = _input.analogMovement ? _input.move.magnitude : 1f;

    // accelerate or decelerate to target speed
    if (currentHorizontalSpeed < targetSpeed - speedOffset ||
        currentHorizontalSpeed > targetSpeed + speedOffset)
    {
        // creates curved result rather than a linear one giving a more organic speed change
        // note T in Lerp is clamped, so we don't need to clamp our speed
        _speed = Mathf.Lerp(currentHorizontalSpeed, targetSpeed * inputMagnitude,
            Time.deltaTime * SpeedChangeRate);

        // round speed to 3 decimal places
        _speed = Mathf.Round(_speed * 1000f) / 1000f;
    }
    else
    {
        _speed = targetSpeed;
    }

```

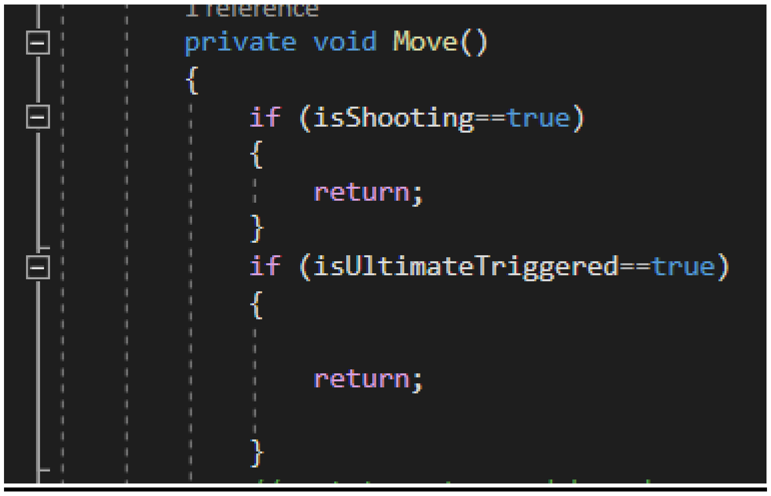


```
    }

    _animationBlend = Mathf.Lerp(_animationBlend, targetSpeed, Time.deltaTime * SpeedChangeRate);
    if (_animationBlend < 0.01f) _animationBlend = 0f;

    // normalise input direction
    Vector3 inputDirection = new Vector3(_input.move.x, 0.0f, _input.move.y).normalized;

    // note: Vector2's != operator uses approximation so is not floating point error prone, and is cheaper than magnitude
    // if there is a move input rotate player when the player is moving
    if (_input.move != Vector2.zero)
    {

        _targetRotation = Mathf.Atan2(inputDirection.x, inputDirection.z) * Mathf.Rad2Deg +
                          _mainCamera.transform.eulerAngles.y;
        float rotation = Mathf.SmoothDampAngle(transform.eulerAngles.y, _targetRotation, ref _rotationVelocity,
            RotationSmoothTime);
        if (aiming == false)
        {  // rotate to face input direction relative to camera position
            transform.rotation = Quaternion.Euler(0.0f, rotation, 0.0f);
        }
    }
        Vector3 targetDirection = Quaternion.Euler(0.0f, _targetRotation, 0.0f) * Vector3.forward;

        // move the player
        _controller.Move(targetDirection.normalized * (_speed * Time.deltaTime) +
                         new Vector3(0.0f, _verticalVelocity, 0.0f) * Time.deltaTime);

        // update animator if using character
        if (_hasAnimator)
        {
            _animator.SetFloat(_animIDSpeed, _animationBlend);
            _animator.SetFloat(_animIDMotionSpeed, inputMagnitude);
        }
}
```

5.9 Scripts

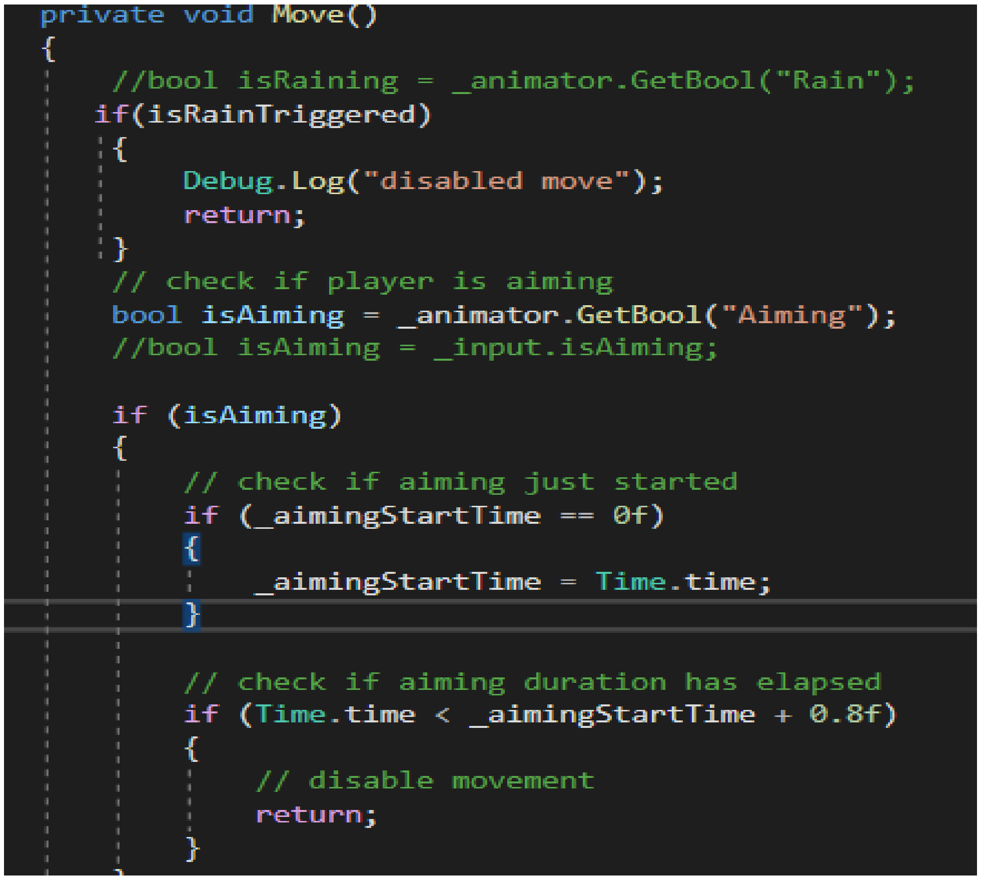
Disabling movement

The following prevent the wizard (Kachujin) character to move while it is shooting or using its ability



```
1 reference
private void Move()
{
    if (isShooting==true)
    {
        return;
    }
    if (isUltimateTriggered==true)
    {

        return;

    }
```

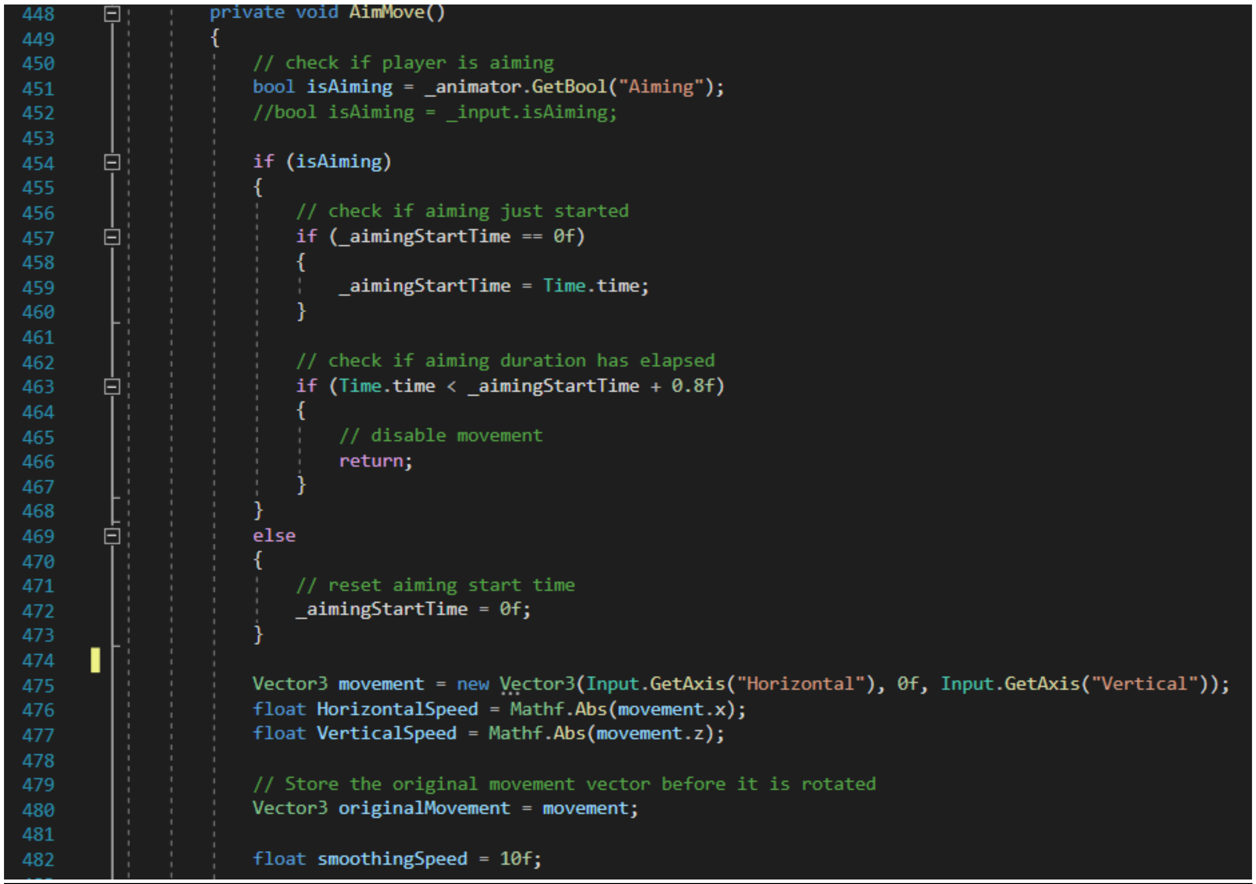
This code prevents the Lucy (archer) character from moving while it is using its special ability and while it is aiming. When aiming it switches to another type of movement which allows the character to perform strafe movement:



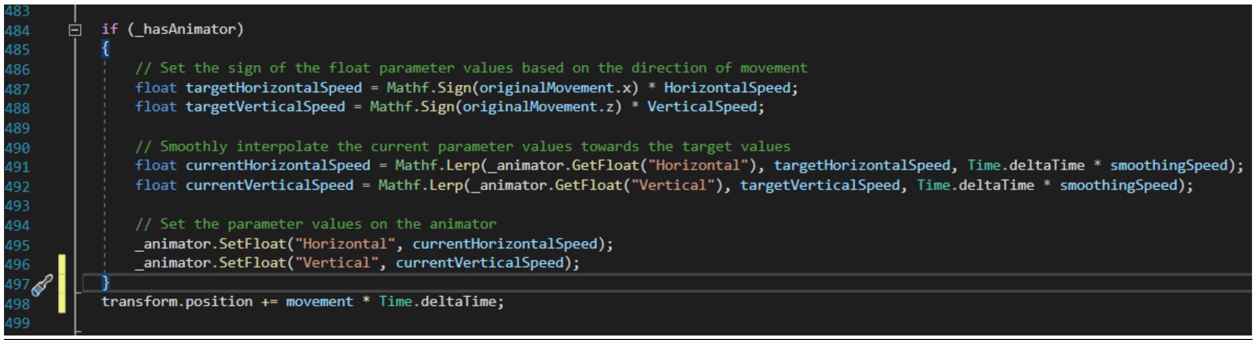
```
private void Move()
{
    //bool isRaining = _animator.GetBool("Rain");
   if(isRainTriggered)
    {
        Debug.Log("disabled move");
        return;
    }
    // check if player is aiming
    bool isAiming = _animator.GetBool("Aiming");
    //bool isAiming = _input.isAiming;

    if (isAiming)
    {
        // check if aiming just started
        if (_aimingStartTime == 0f)
        {
            _aimingStartTime = Time.time;
        }

        // check if aiming duration has elapsed
        if (Time.time < _aimingStartTime + 0.8f)
        {
            // disable movement
            return;
        }
```

Strafing movement when aiming

This script is enabled when Lucy character starts to aim



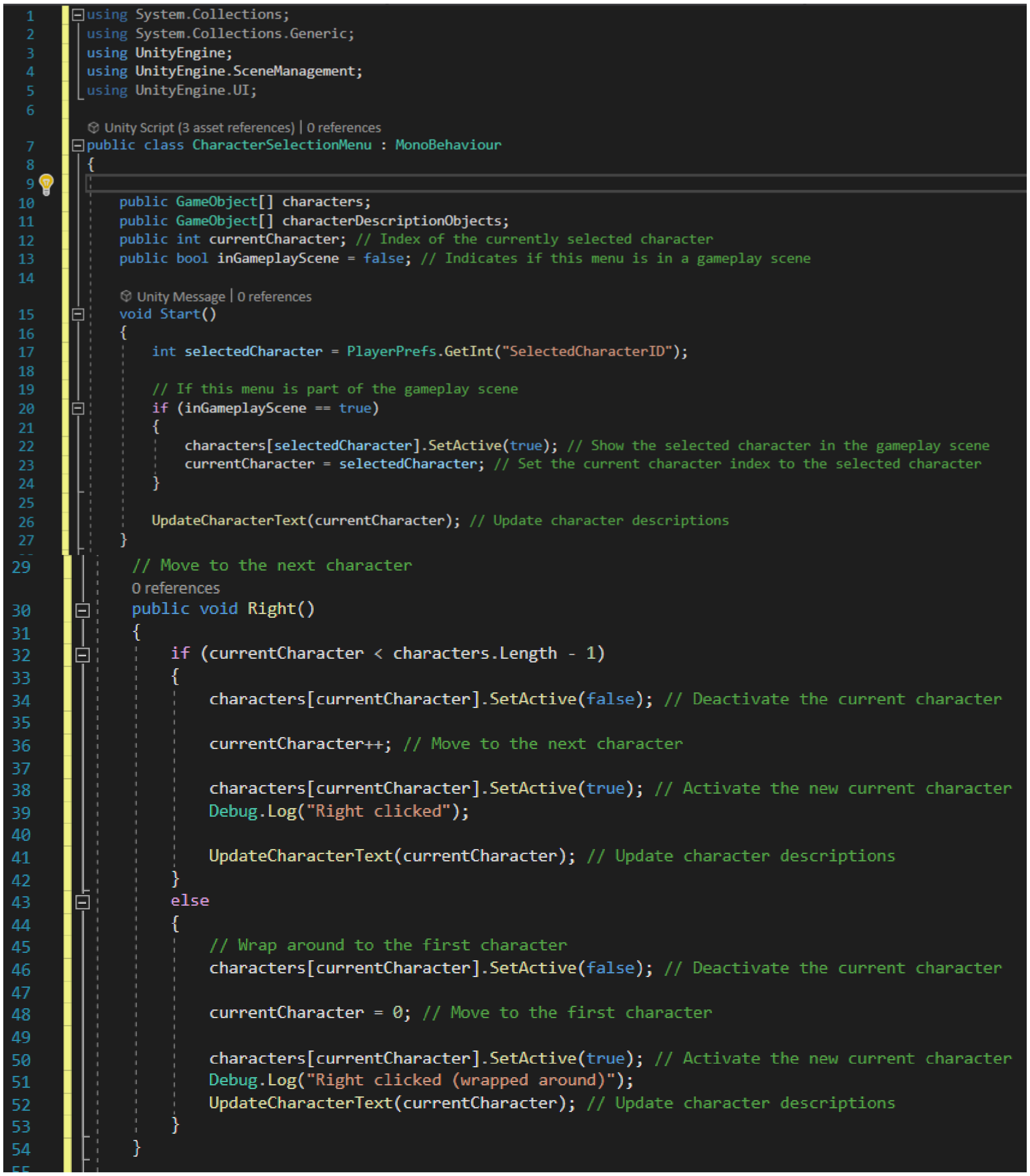
```
private void AimMove()
{
    // check if player is aiming
    bool isAiming = _animator.GetBool("Aiming");
    //bool isAiming = _input.isAiming;

    if (isAiming)
    {
        // check if aiming just started
        if (_aimingStartTime == 0f)
        {
            _aimingStartTime = Time.time;
        }

        // check if aiming duration has elapsed
        if (Time.time < _aimingStartTime + 0.8f)
        {
            // disable movement
            return;
        }
    }
    else
    {
        // reset aiming start time
        _aimingStartTime = 0f;
    }

    Vector3 movement = new Vector3(Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));
    float HorizontalSpeed = Mathf.Abs(movement.x);
    float VerticalSpeed = Mathf.Abs(movement.z);

    // Store the original movement vector before it is rotated
    Vector3 originalMovement = movement;

    float smoothingSpeed = 10f;
```



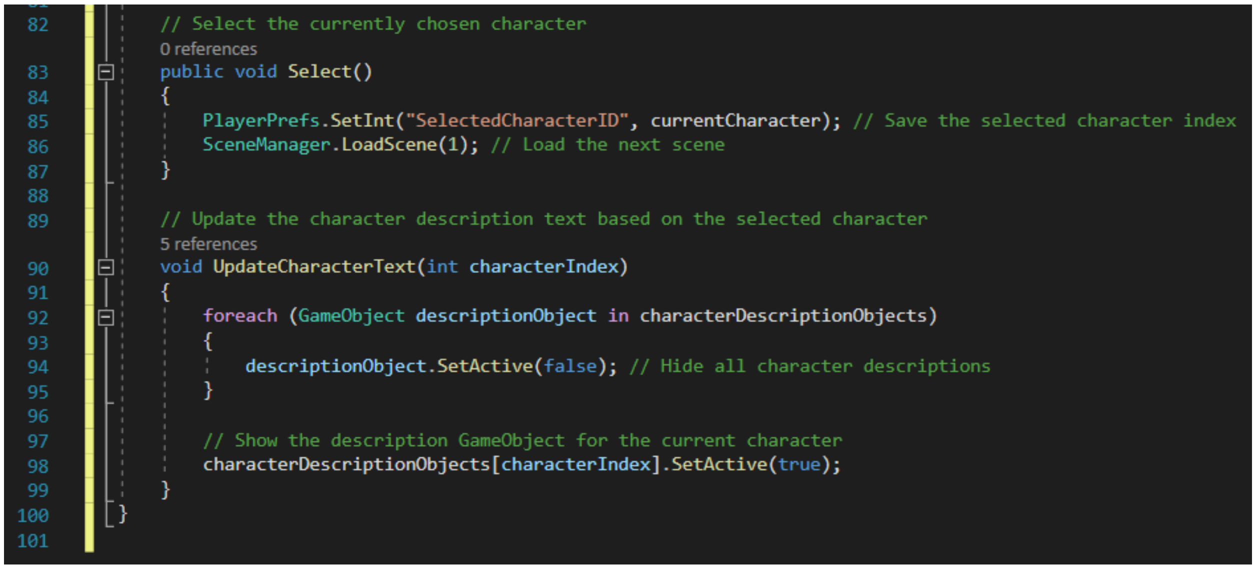
```
if (_hasAnimator)
{
    // Set the sign of the float parameter values based on the direction of movement
    float targetHorizontalSpeed = Mathf.Sign(originalMovement.x) * HorizontalSpeed;
    float targetVerticalSpeed = Mathf.Sign(originalMovement.z) * VerticalSpeed;

    // Smoothly interpolate the current parameter values towards the target values
    float currentHorizontalSpeed = Mathf.Lerp(_animator.GetFloat("Horizontal"), targetHorizontalSpeed, Time.deltaTime * smoothingSpeed);
    float currentVerticalSpeed = Mathf.Lerp(_animator.GetFloat("Vertical"), targetVerticalSpeed, Time.deltaTime * smoothingSpeed);

    // Set the parameter values on the animator
    _animator.SetFloat("Horizontal", currentHorizontalSpeed);
    _animator.SetFloat("Vertical", currentVerticalSpeed);
}
transform.position += movement * Time.deltaTime;
```

Character selection

The following script allows the selection of characters. It cycles through the characters and view its corresponding description and select it.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

Unity Script (3 asset references) | 0 references
public class CharacterSelectionMenu : MonoBehaviour
{

    public GameObject[] characters;
    public GameObject[] characterDescriptionObjects;
    public int currentCharacter; // Index of the currently selected character
    public bool inGameplayScene = false; // Indicates if this menu is in a gameplay scene

    Unity Message | 0 references
    void Start()
    {
        int selectedCharacter = PlayerPrefs.GetInt("SelectedCharacterID");

        // If this menu is part of the gameplay scene
        if (inGameplayScene == true)
        {
            characters[selectedCharacter].SetActive(true); // Show the selected character in the gameplay scene
            currentCharacter = selectedCharacter; // Set the current character index to the selected character
        }

        UpdateCharacterText(currentCharacter); // Update character descriptions
    }
    // Move to the next character
    0 references
    public void Right()
    {
        if (currentCharacter < characters.Length - 1)
        {
            characters[currentCharacter].SetActive(false); // Deactivate the current character

            currentCharacter++; // Move to the next character

            characters[currentCharacter].SetActive(true); // Activate the new current character
            Debug.Log("Right clicked");

            UpdateCharacterText(currentCharacter); // Update character descriptions
        }
        else
        {
            // Wrap around to the first character
            characters[currentCharacter].SetActive(false); // Deactivate the current character

            currentCharacter = 0; // Move to the first character

            characters[currentCharacter].SetActive(true); // Activate the new current character
            Debug.Log("Right clicked (wrapped around)");
            UpdateCharacterText(currentCharacter); // Update character descriptions
        }
    }
```

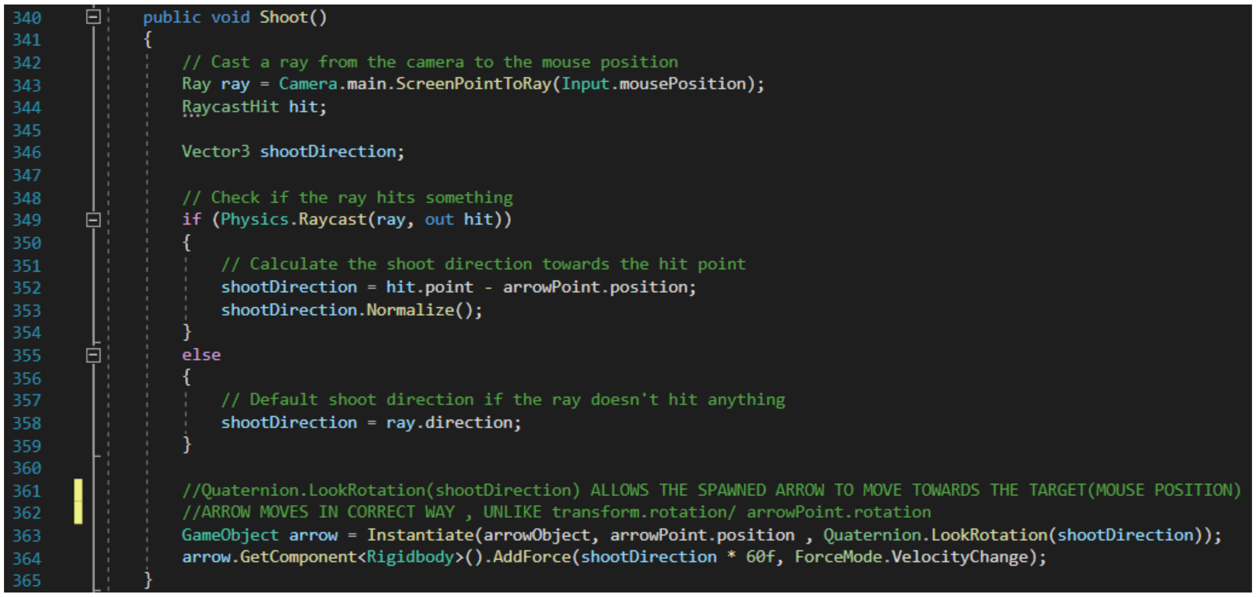


```
    // Select the currently chosen character
    0 references
    public void Select()
    {
        PlayerPrefs.SetInt("SelectedCharacterID", currentCharacter); // Save the selected character index
        SceneManager.LoadScene(1); // Load the next scene
    }

    // Update the character description text based on the selected character
    5 references
    void UpdateCharacterText(int characterIndex)
    {
        foreach (GameObject descriptionObject in characterDescriptionObjects)
        {
            descriptionObject.SetActive(false); // Hide all character descriptions
        }

        // Show the description GameObject for the current character
        characterDescriptionObjects[characterIndex].SetActive(true);
    }
}
```

Attacking

Lucy (Archer):

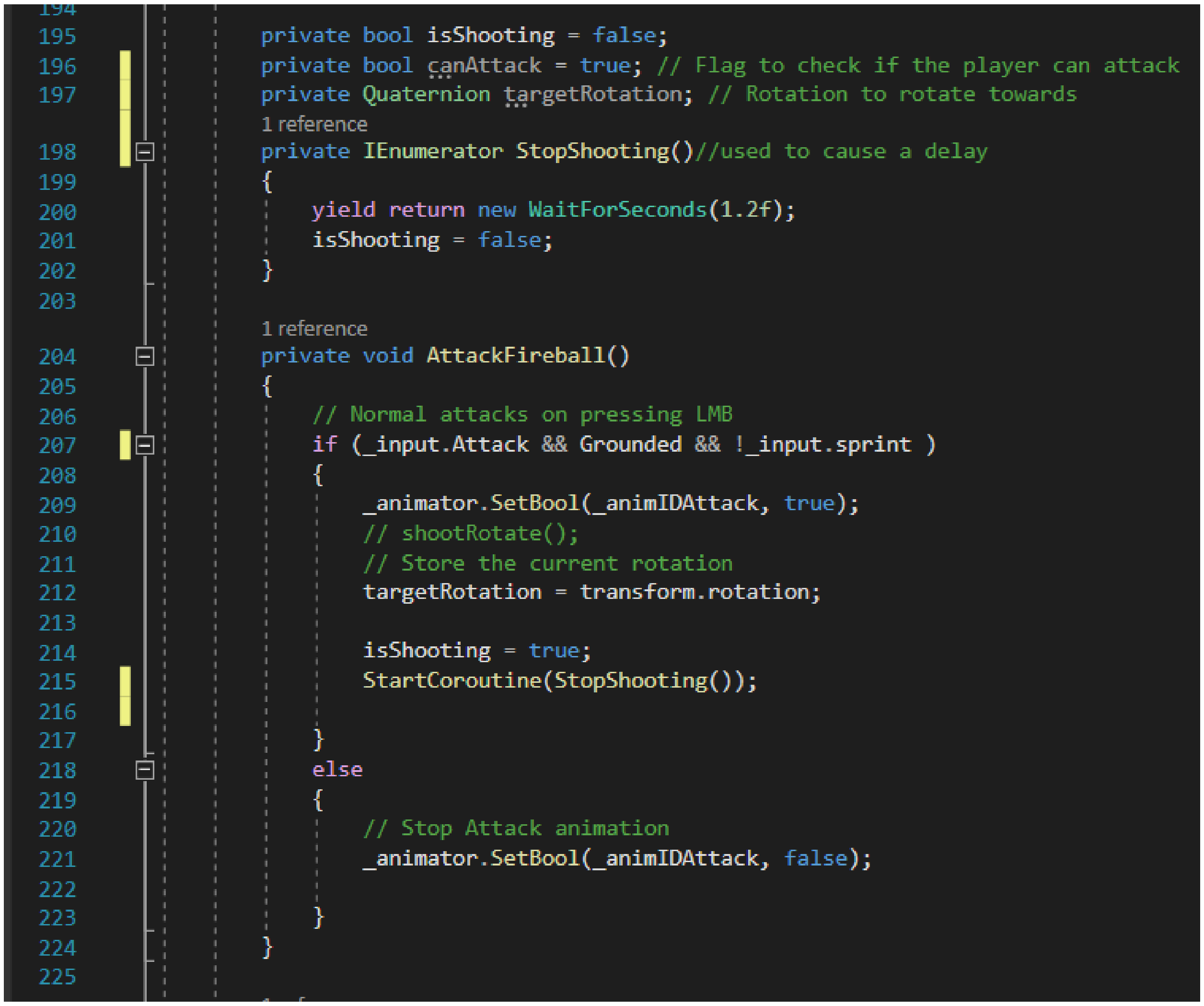
The following allows the archer to towards the center of the screen and spawn an arrow.



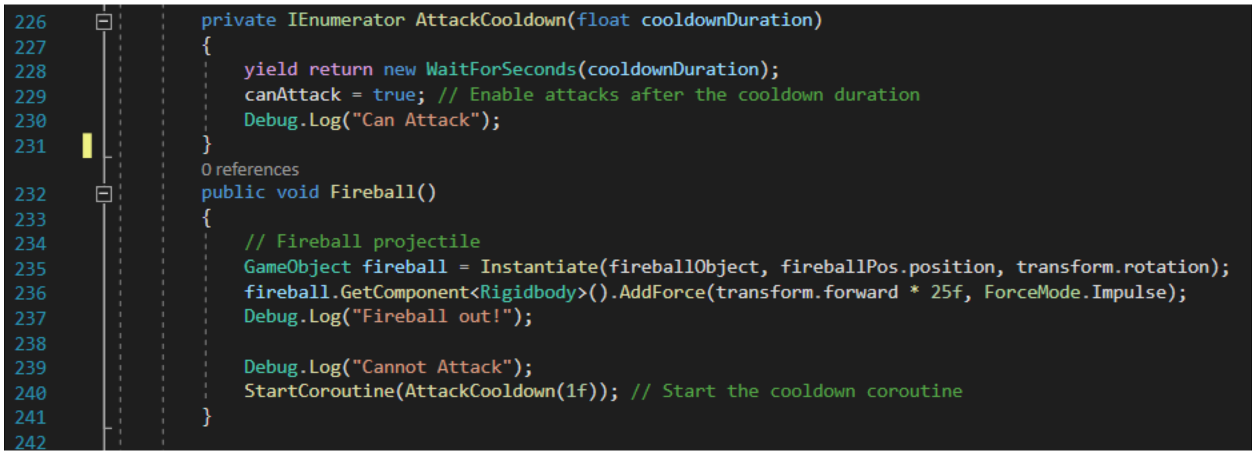
```
public void Shoot()
{
    // Cast a ray from the camera to the mouse position
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    Vector3 shootDirection;

    // Check if the ray hits something
    if (Physics.Raycast(ray, out hit))
    {
        // Calculate the shoot direction towards the hit point
        shootDirection = hit.point - arrowPoint.position;
        shootDirection.Normalize();
    }
    else
    {
        // Default shoot direction if the ray doesn't hit anything
        shootDirection = ray.direction;
    }

    //Quaternion.LookRotation(shootDirection) ALLOWS THE SPAWNED ARROW TO MOVE TOWARDS THE TARGET(MOUSE POSITION)
    //ARROW MOVES IN CORRECT WAY , UNLIKE transform.rotation/ arrowPoint.rotation
    GameObject arrow = Instantiate(arrowObject, arrowPoint.position , Quaternion.LookRotation(shootDirection));
    arrow.GetComponent<Rigidbody>().AddForce(shootDirection * 60f, ForceMode.VelocityChange);
}
```

Kagchujin (wizard):

The character can spawn a fireball, there is a delay before the animation is set to false. It ensures attack animations are triggered correctly, prevents rapid attacks, and launches fireball projectiles with cooldowns.



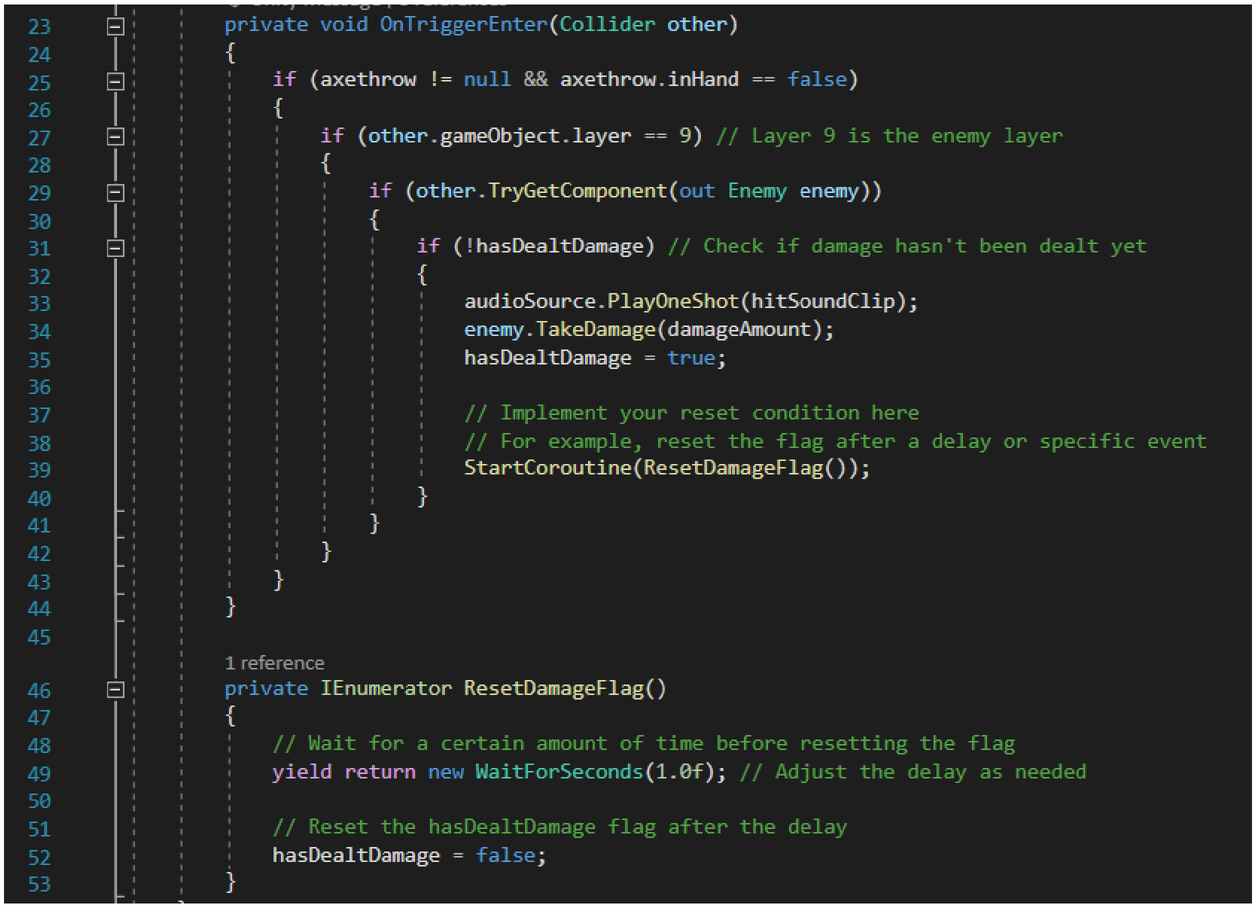
```
        private bool isShooting = false;
        private bool canAttack = true; // Flag to check if the player can attack
        private Quaternion targetRotation; // Rotation to rotate towards
        1 reference
        private IEnumerator StopShooting()//used to cause a delay
        {
            yield return new WaitForSeconds(1.2f);
            isShooting = false;
        }

        1 reference
        private void AttackFireball()
        {
            // Normal attacks on pressing LMB
            if (_input.Attack && Grounded && !_input.sprint )
            {
                _animator.SetBool(_animIDAttack, true);
                // shootRotate();
                // Store the current rotation
                targetRotation = transform.rotation;

                isShooting = true;
                StartCoroutine(StopShooting());

            }
            else
            {
                // Stop Attack animation
                _animator.SetBool(_animIDAttack, false);

            }
        }
```



```
        private IEnumerator AttackCooldown(float cooldownDuration)
        {
            yield return new WaitForSeconds(cooldownDuration);
            canAttack = true; // Enable attacks after the cooldown duration
            Debug.Log("Can Attack");
        }
        0 references
        public void Fireball()
        {
            // Fireball projectile
            GameObject fireball = Instantiate(fireballObject, fireballPos.position, transform.rotation);
            fireball.GetComponent<Rigidbody>().AddForce(transform.forward * 25f, ForceMode.Impulse);
            Debug.Log("Fireball out!");

            Debug.Log("Cannot Attack");
            StartCoroutine(AttackCooldown(1f)); // Start the cooldown coroutine
        }
```

Warrior:

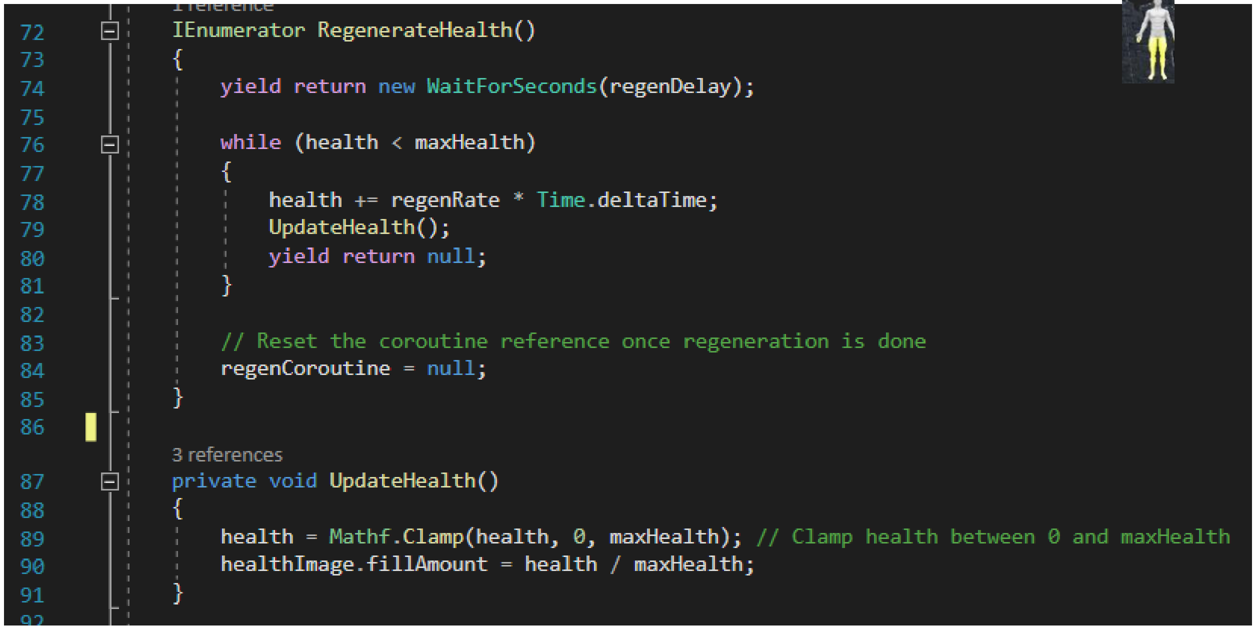
The character causes damage by detecting if the enemy has collided with the axe.



```
private void OnTriggerEnter(Collider other)
{
    if (axethrow != null && axethrow.inHand == false)
    {
        if (other.gameObject.layer == 9) // Layer 9 is the enemy layer
        {
            if (other.TryGetComponent(out Enemy enemy))
            {
                if (!hasDealtDamage) // Check if damage hasn't been dealt yet
                {
                    audioSource.PlayOneShot(hitSoundClip);
                    enemy.TakeDamage(damageAmount);
                    hasDealtDamage = true;

                    // Implement your reset condition here
                    // For example, reset the flag after a delay or specific event
                    StartCoroutine(ResetDamageFlag());
                }
            }
        }
    }
}

1 reference
private IEnumerator ResetDamageFlag()
{
    // Wait for a certain amount of time before resetting the flag
    yield return new WaitForSeconds(1.0f); // Adjust the delay as needed

    // Reset the hasDealtDamage flag after the delay
    hasDealtDamage = false;
}
```

Health System

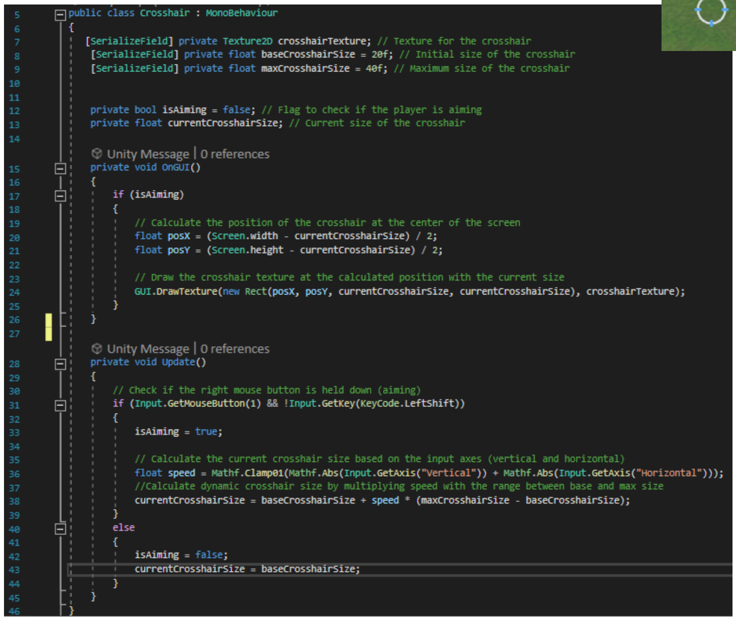
Incrementally increase health based on the regen rate until max health is reached. The health UI is also filled proportionately.



```
1 reference
IEnumerator RegenerateHealth()
{
    yield return new WaitForSeconds(regenDelay);

    while (health < maxHealth)
    {
        health += regenRate * Time.deltaTime;
        UpdateHealth();
        yield return null;
    }

    // Reset the coroutine reference once regeneration is done
    regenCoroutine = null;
}

3 references
private void UpdateHealth()
{
    health = Mathf.Clamp(health, 0, maxHealth); // Clamp health between 0 and maxHealth
    healthImage.fillAmount = health / maxHealth;
}
```

Crosshair (for archer and warrior)

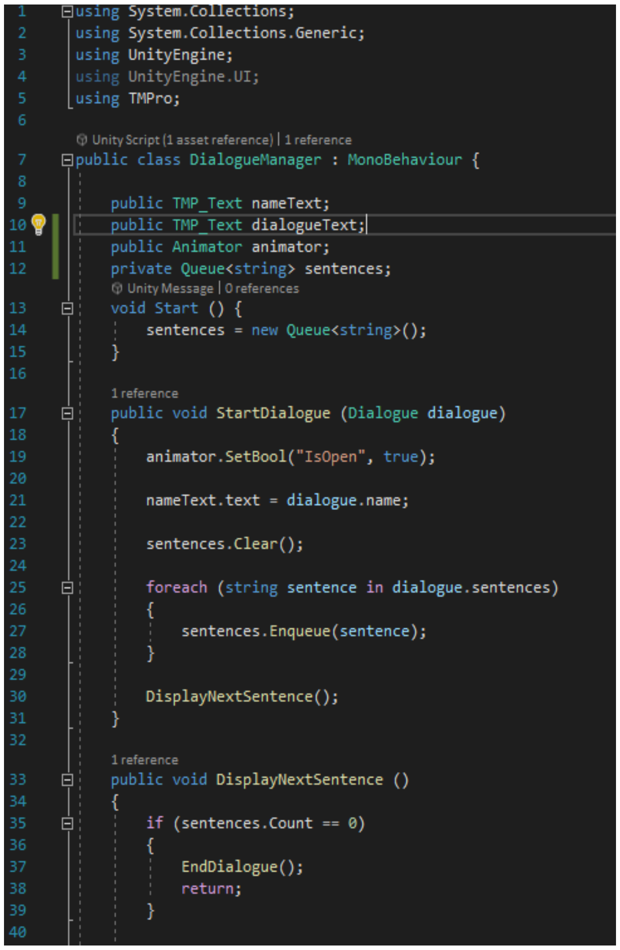
The crosshair decreases and increases in size depending on player movement.



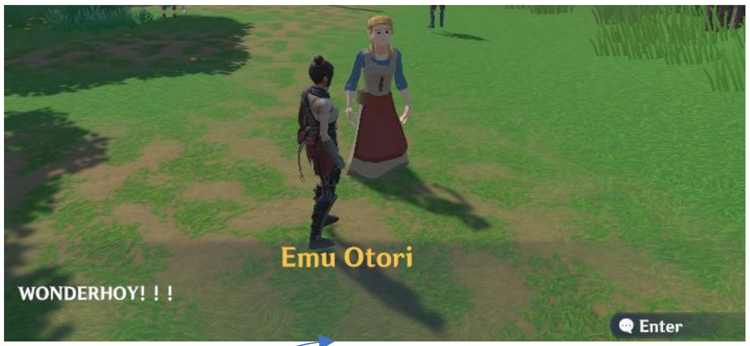
```
public class Crosshair : MonoBehaviour
{
    [SerializeField] private Texture2D crosshairTexture; // Texture for the crosshair
    [SerializeField] private float baseCrosshairSize = 20f; // Initial size of the crosshair
    [SerializeField] private float maxCrosshairSize = 40f; // Maximum size of the crosshair


    private bool isAiming = false; // Flag to check if the player is aiming
    private float currentCrosshairSize; // Current size of the crosshair

    Unity Message | 0 references
    private void OnGUI()
    {
        if (isAiming)
        {
            // Calculate the position of the crosshair at the center of the screen
            float posX = (Screen.width - currentCrosshairSize) / 2;
            float posY = (Screen.height - currentCrosshairSize) / 2;

            // Draw the crosshair texture at the calculated position with the current size
            GUI.DrawTexture(new Rect(posX, posY, currentCrosshairSize, currentCrosshairSize), crosshairTexture);
        }
    }

    Unity Message | 0 references
    private void Update()
    {
        // Check if the right mouse button is held down (aiming)
        if (Input.GetMouseButton(1) && !Input.GetKey(KeyCode.LeftShift))
        {
            isAiming = true;

            // Calculate the current crosshair size based on the input axes (vertical and horizontal)
            float speed = Mathf.Clamp01(Mathf.Abs(Input.GetAxis("Vertical")) + Mathf.Abs(Input.GetAxis("Horizontal")));
            //Calculate dynamic crosshair size by multiplying speed with the range between base and max size
            currentCrosshairSize = baseCrosshairSize + speed * (maxCrosshairSize - baseCrosshairSize);
        }
        else
        {
            isAiming = false;
            currentCrosshairSize = baseCrosshairSize;
        }
    }
}
```

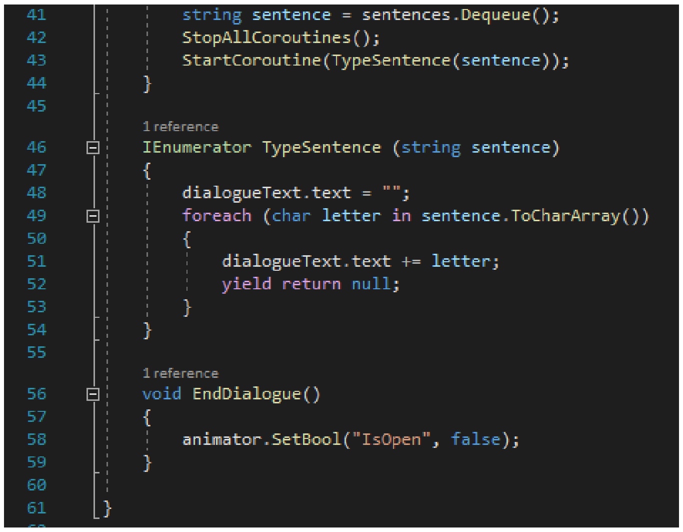
NPC

The NPC starts to interact with the player if it is within a close range. It displays a dialogue to guide the player in his quest.

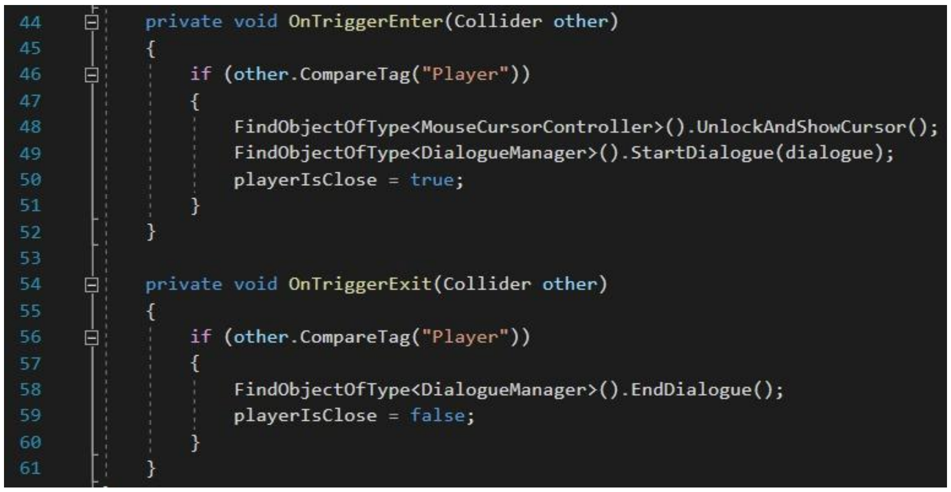


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

Unity Script (1 asset reference) | 1 reference
public class DialogueManager : MonoBehaviour {

    public TMP_Text nameText;
    public TMP_Text dialogueText;
    public Animator animator;
    private Queue<string> sentences;
    Unity Message | 0 references
    void Start () {
        sentences = new Queue<string>();
    }

    1 reference
    public void StartDialogue (Dialogue dialogue)
    {
        animator.SetBool("IsOpen", true);

        nameText.text = dialogue.name;

        sentences.Clear();

        foreach (string sentence in dialogue.sentences)
        {
            sentences.Enqueue(sentence);
        }

        DisplayNextSentence();
    }

    1 reference
    public void DisplayNextSentence ()
    {
        if (sentences.Count == 0)
        {
            EndDialogue();
            return;
        }

```





```
        string sentence = sentences.Dequeue();
        StopAllCoroutines();
        StartCoroutine(TypeSentence(sentence));
    }

    1 reference
    IEnumerator TypeSentence (string sentence)
    {
        dialogueText.text = "";
        foreach (char letter in sentence.ToCharArray())
        {
            dialogueText.text += letter;
            yield return null;
        }
    }

    1 reference
    void EndDialogue()
    {
        animator.SetBool("IsOpen", false);
    }

}
```



```
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            FindObjectOfType<MouseCursorController>().UnlockAndShowCursor();
            FindObjectOfType<DialogueManager>().StartDialogue(dialogue);
            playerIsClose = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            FindObjectOfType<DialogueManager>().EndDialogue();
            playerIsClose = false;
        }
    }
```

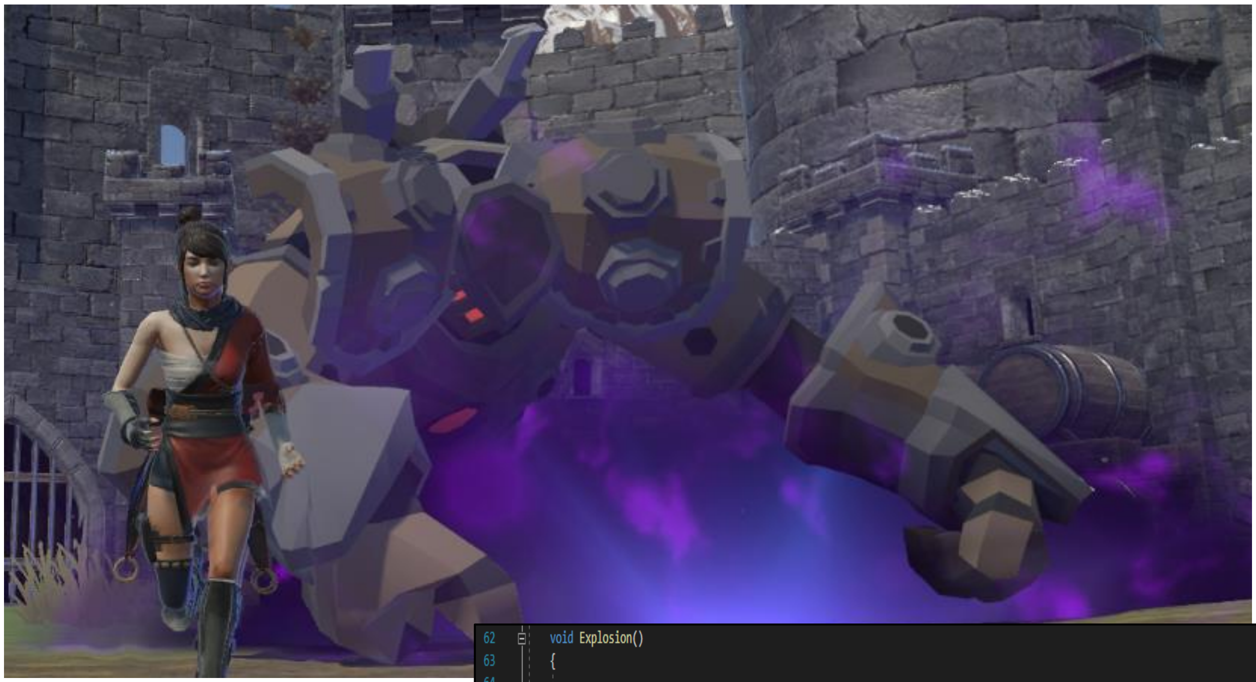
Normal Enemy(Ogres)

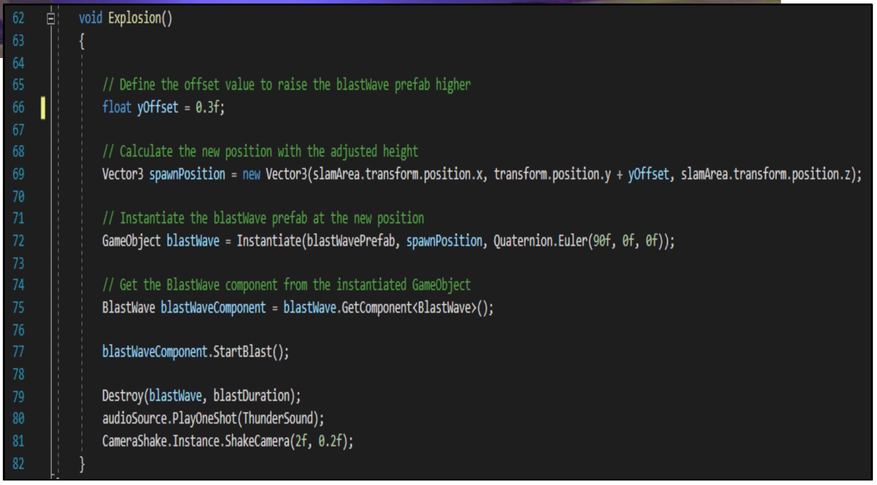
Collider is placed on the axe, when enemy reaches a certain distance to the player, the enemy performs an attacking animation. When the axe hits the player, damage is dealt.

Mini Boss

Final Boss

Explosion:

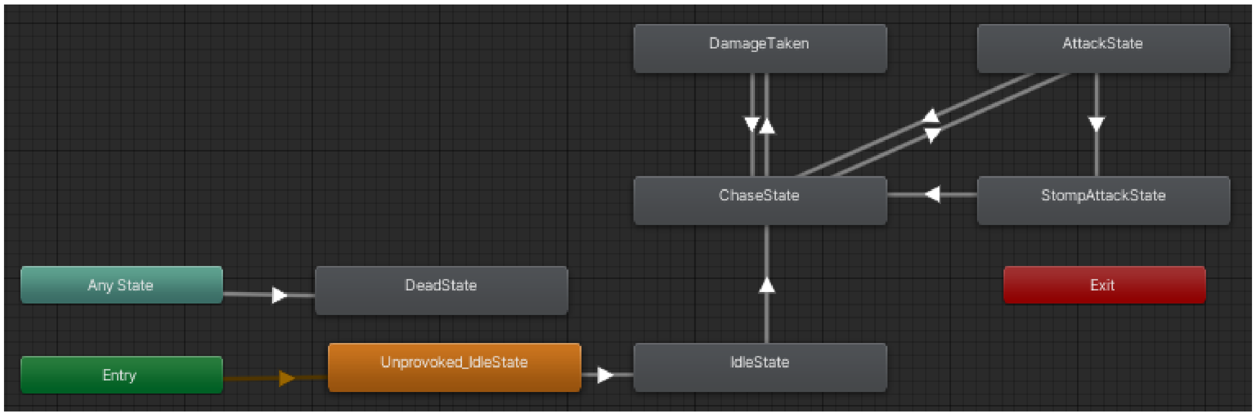


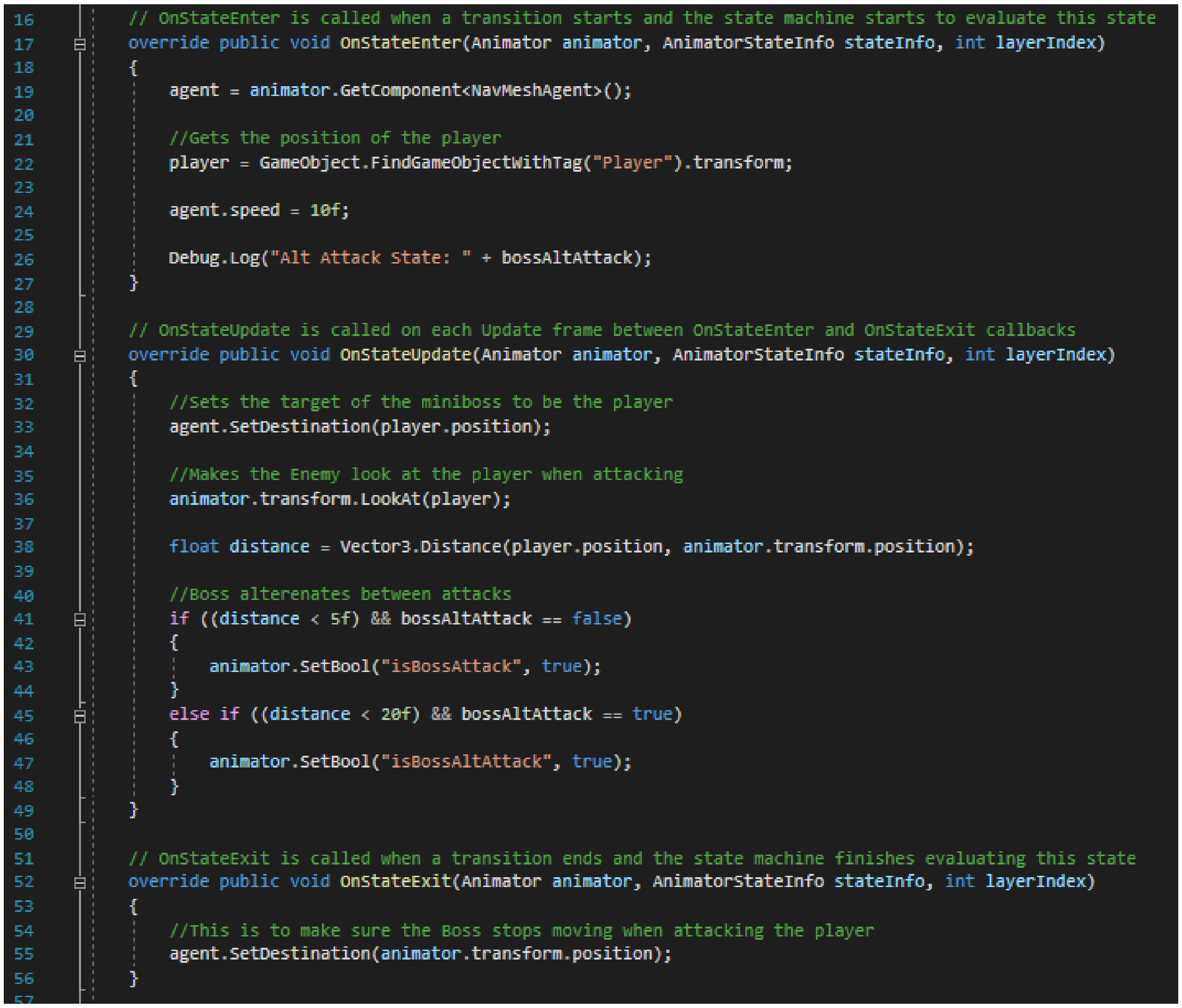


```
void Explosion()
{

    // Define the offset value to raise the blastWave prefab higher
    float yOffset = 0.3f;

    // Calculate the new position with the adjusted height
    Vector3 spawnPosition = new Vector3(slamArea.transform.position.x, transform.position.y + yOffset, slamArea.transform.position.z);

    // Instantiate the blastWave prefab at the new position
    GameObject blastWave = Instantiate(blastWavePrefab, spawnPosition, Quaternion.Euler(90f, 0f, 0f));

    // Get the BlastWave component from the instantiated GameObject
    BlastWave blastWaveComponent = blastWave.GetComponent<BlastWave>();

    blastWaveComponent.StartBlast();

    Destroy(blastWave, blastDuration);
    audioSource.PlayOneShot(ThunderSound);
    CameraShake.Instance.ShakeCamera(2f, 0.2f);
}
```

BossChaseState

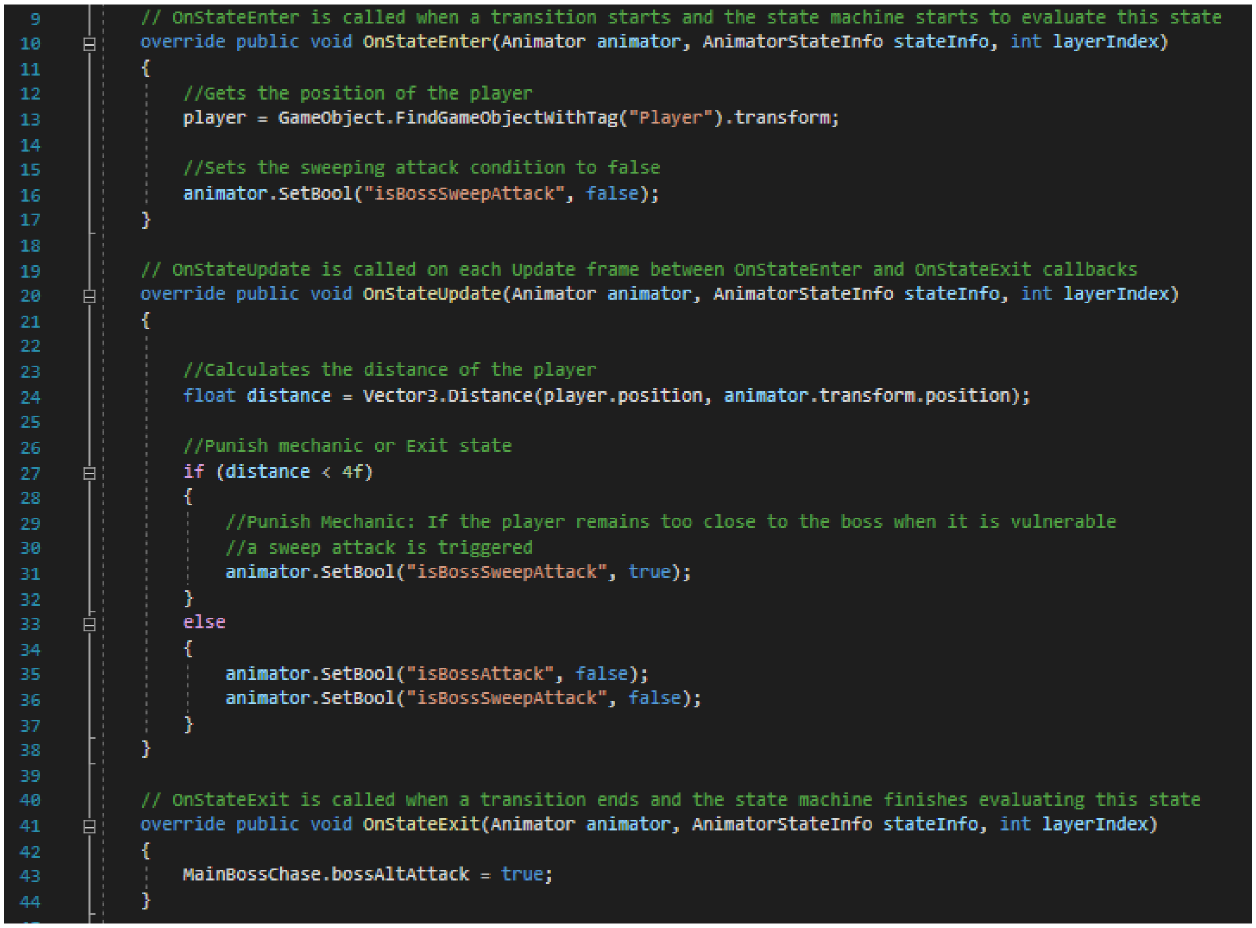
The boss chases the player when it is triggered by a collider found at the entrance of the arena.





```
    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent = animator.GetComponent<NavMeshAgent>();

        //Gets the position of the player
        player = GameObject.FindGameObjectWithTag("Player").transform;

        agent.speed = 10f;

        Debug.Log("Alt Attack State: " + bossAltAttack);
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        //Sets the target of the miniboss to be the player
        agent.SetDestination(player.position);

        //Makes the Enemy look at the player when attacking
        animator.transform.LookAt(player);

        float distance = Vector3.Distance(player.position, animator.transform.position);

        //Boss alterenates between attacks
        if ((distance < 5f) && bossAltAttack == false)
        {
            animator.SetBool("isBossAttack", true);
        }
        else if ((distance < 20f) && bossAltAttack == true)
        {
            animator.SetBool("isBossAltAttack", true);
        }
    }

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        //This is to make sure the Boss stops moving when attacking the player
        agent.SetDestination(animator.transform.position);
    }
```

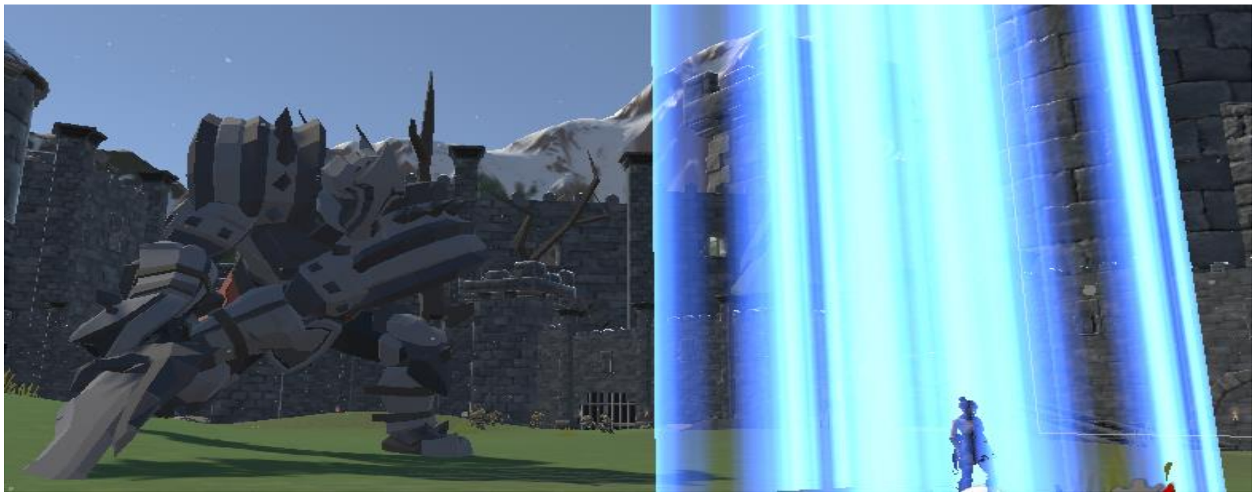
BossAttackState

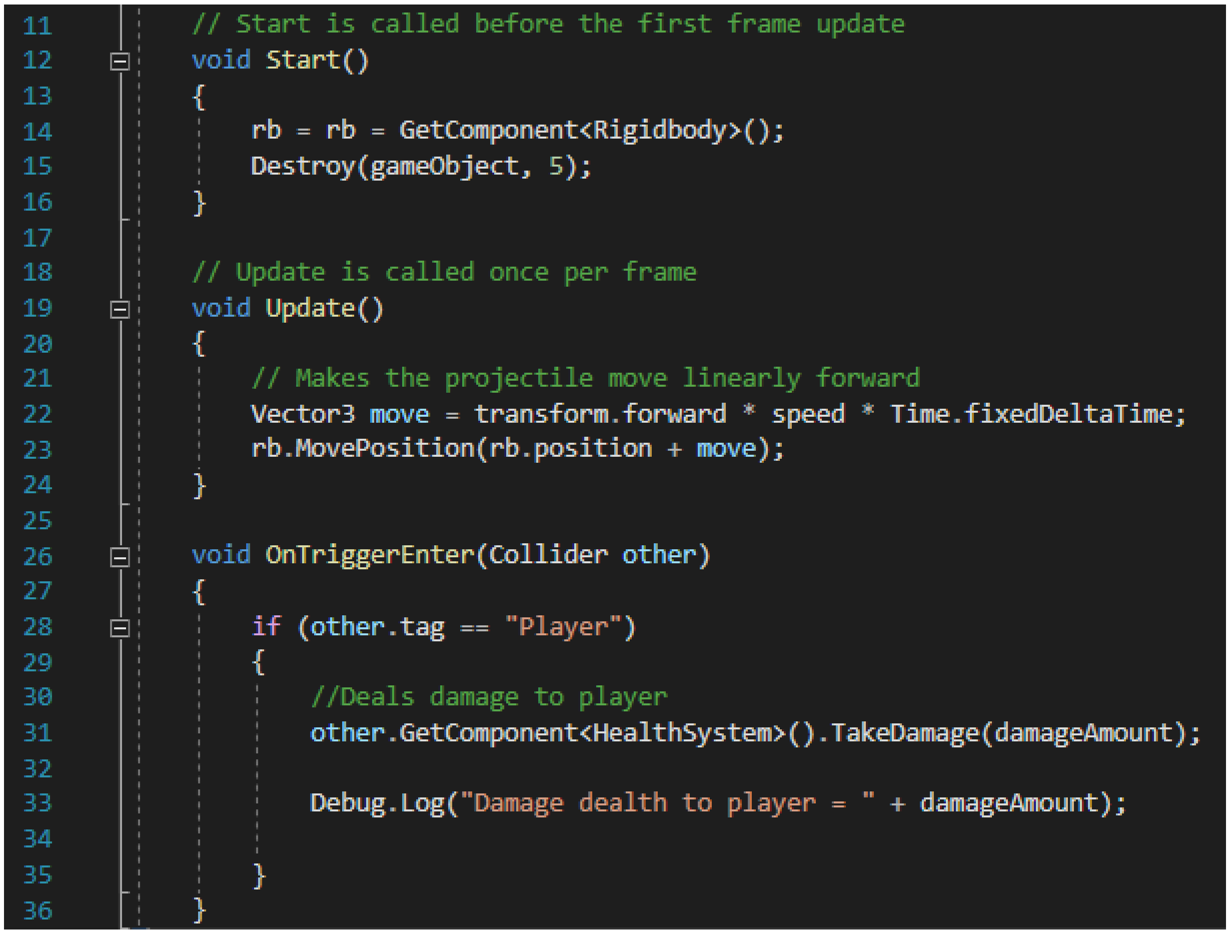


```
    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        //Gets the position of the player
        player = GameObject.FindGameObjectWithTag("Player").transform;

        //Sets the sweeping attack condition to false
        animator.SetBool("isBossSweepAttack", false);
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {

        //Calculates the distance of the player
        float distance = Vector3.Distance(player.position, animator.transform.position);

        //Punish mechanic or Exit state
        if (distance < 4f)
        {
            //Punish Mechanic: If the player remains too close to the boss when it is vulnerable
            //a sweep attack is triggered
            animator.SetBool("isBossSweepAttack", true);
        }
        else
        {
            animator.SetBool("isBossAttack", false);
            animator.SetBool("isBossSweepAttack", false);
        }
    }

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        MainBossChase.bossAltAttack = true;
    }
```

Boss Beam Attack

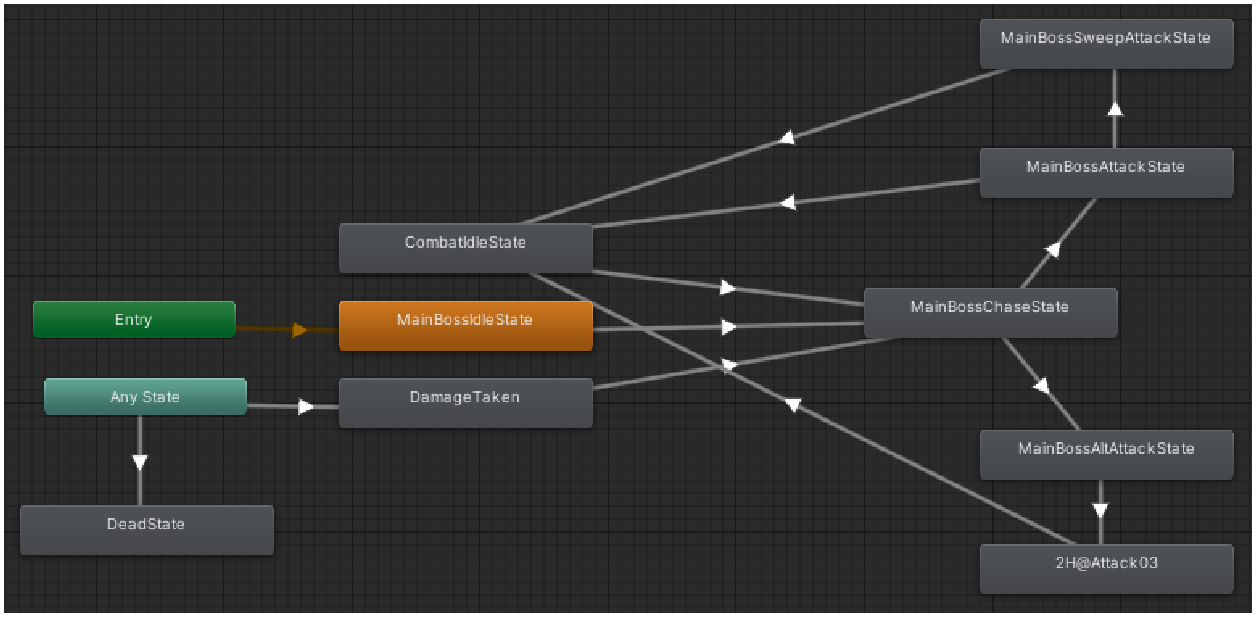
A line renderer is used to make the laser beam attack.The animation makes the laser beam move.If player is within that beam he will take damage.



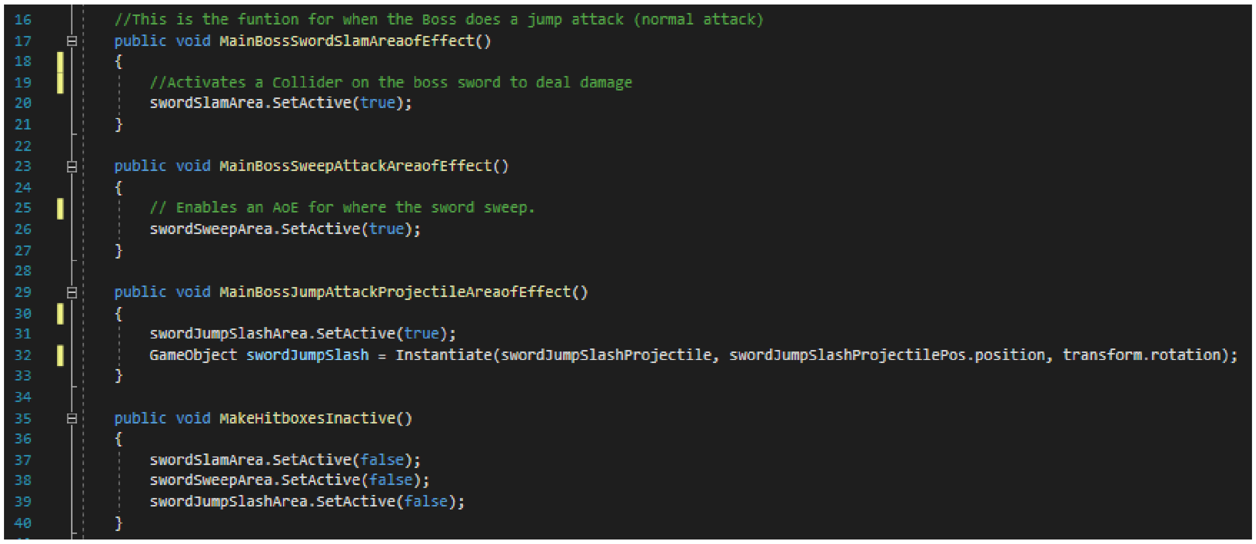


```
    // Start is called before the first frame update
    void Start()
    {
        rb = rb = GetComponent<Rigidbody>();
        Destroy(gameObject, 5);
    }

    // Update is called once per frame
    void Update()
    {
        // Makes the projectile move linearly forward
        Vector3 move = transform.forward * speed * Time.fixedDeltaTime;
        rb.MovePosition(rb.position + move);
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            //Deals damage to player
            other.GetComponent<HealthSystem>().TakeDamage(damageAmount);

            Debug.Log("Damage dealth to player = " + damageAmount);

        }
    }
```

Boss Animation Controller



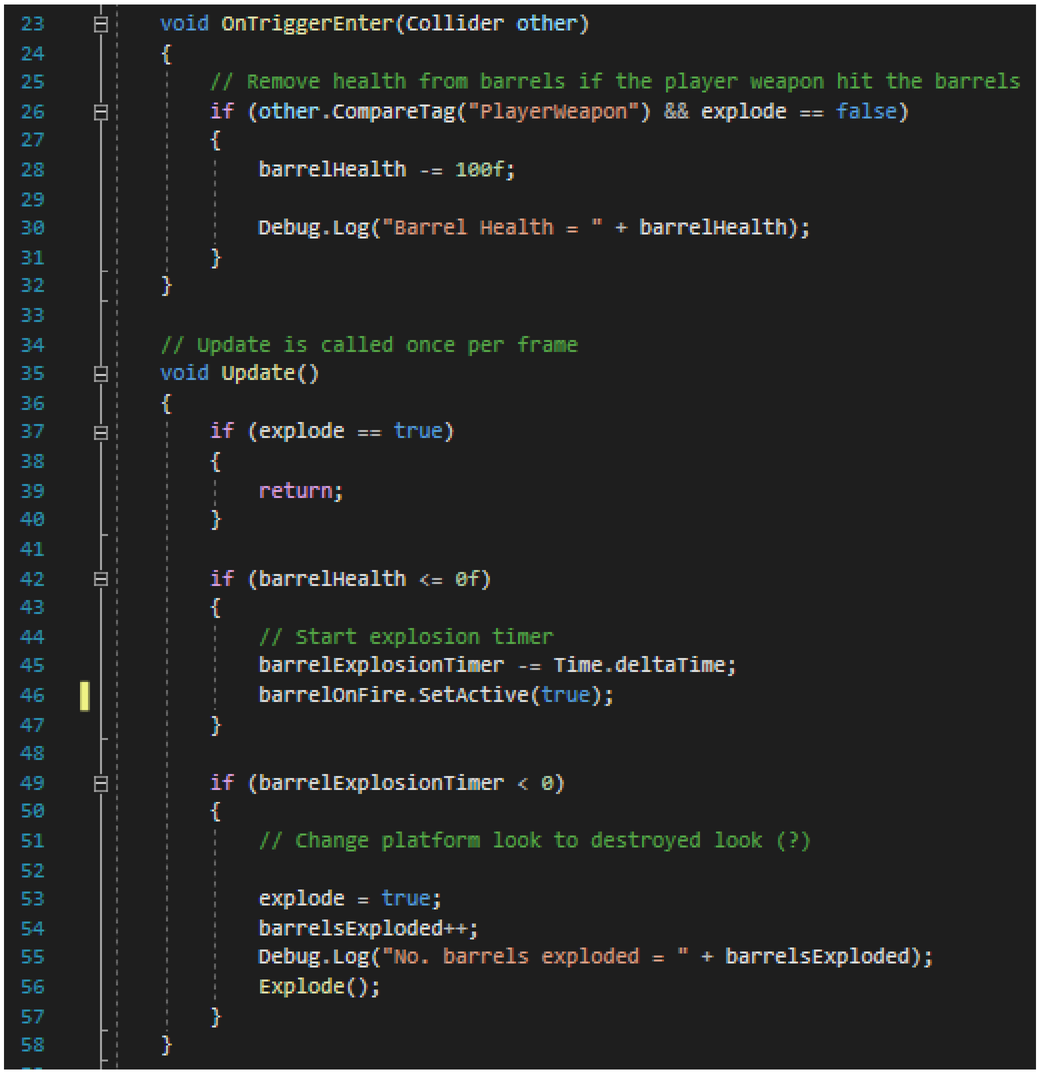
These functions are called by animation events when the boss is attacking. When the boss has finished attacking or when the player is hit by an attack, MakeHitboxesInactive() is called to disable the hitbox.



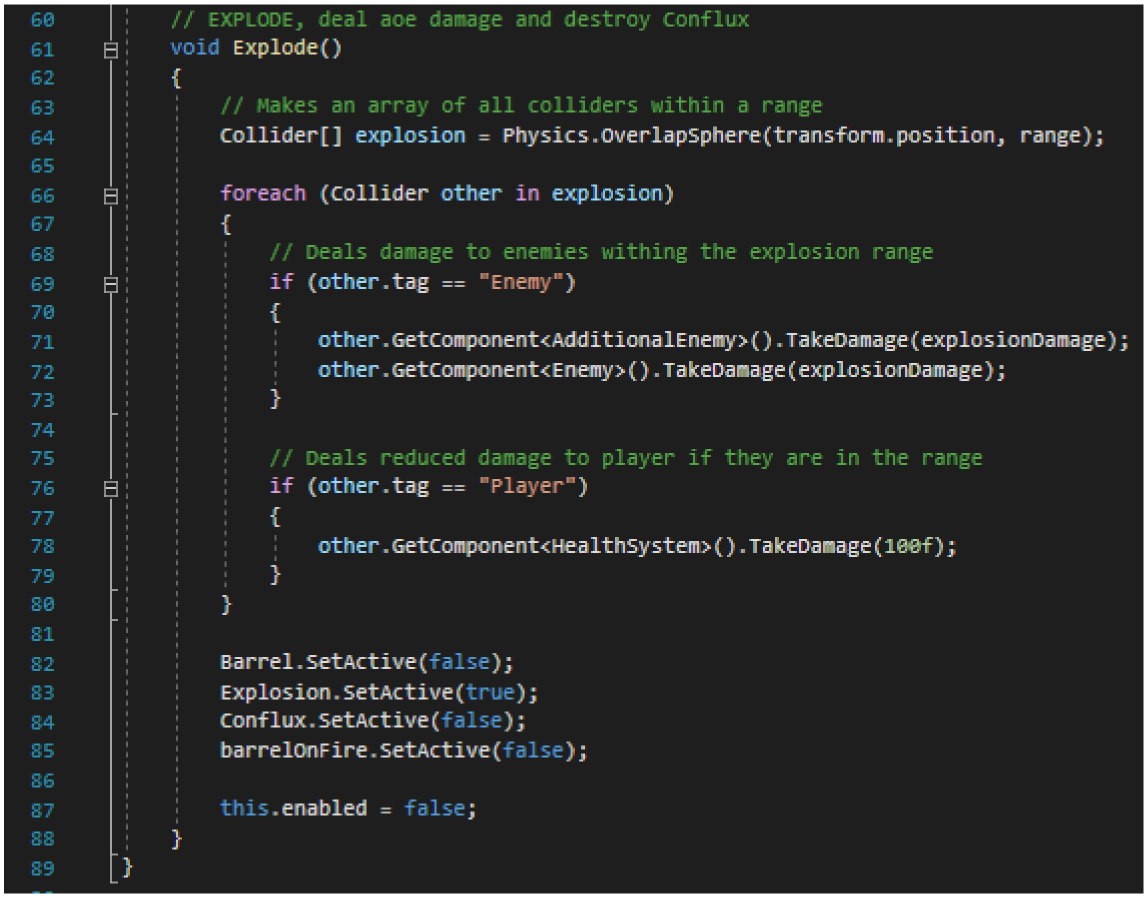
```
    //This is the funtion for when the Boss does a jump attack (normal attack)
    public void MainBossSwordSlamAreaofEffect()
    {
        //Activates a Collider on the boss sword to deal damage
        swordSlamArea.SetActive(true);
    }

    public void MainBossSweepAttackAreaofEffect()
    {
        // Enables an AoE for where the sword sweep.
        swordSweepArea.SetActive(true);
    }

    public void MainBossJumpAttackProjectileAreaofEffect()
    {
        swordJumpSlashArea.SetActive(true);
        GameObject swordJumpSlash = Instantiate(swordJumpSlashProjectile, swordJumpSlashProjectilePos.position, transform.rotation);
    }

    public void MakeHitboxesInactive()
    {
        swordSlamArea.SetActive(false);
        swordSweepArea.SetActive(false);
        swordJumpSlashArea.SetActive(false);
    }
```

Exploding Barrels

The barrel when hit will start a timer.When timer is reached it explodes. When all barrels on the arena are destroyed, the boss is no longer invincible.



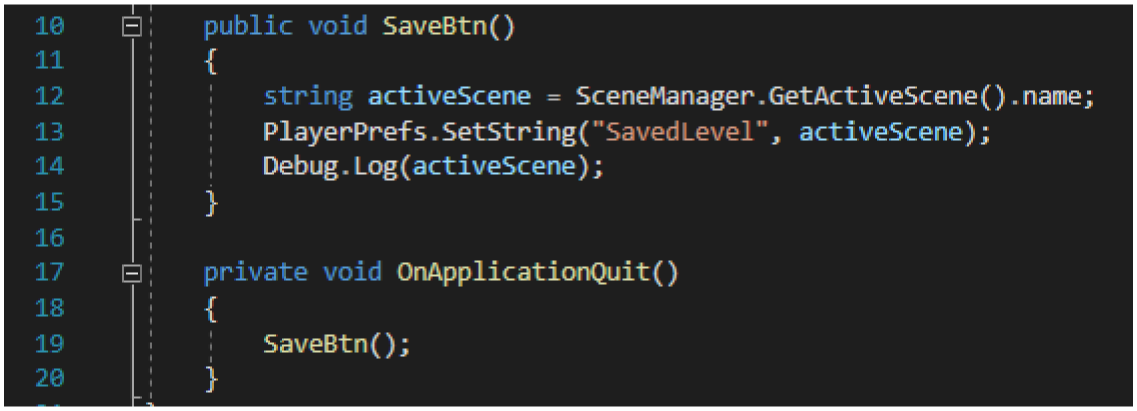
```
void OnTriggerEnter(Collider other)
{
    // Remove health from barrels if the player weapon hit the barrels
    if (other.CompareTag("PlayerWeapon") && explode == false)
    {
        barrelHealth -= 100f;

        Debug.Log("Barrel Health = " + barrelHealth);
    }
}

// Update is called once per frame
void Update()
{
    if (explode == true)
    {
        return;
    }

    if (barrelHealth <= 0f)
    {
        // Start explosion timer
        barrelExplosionTimer -= Time.deltaTime;
        barrelOnFire.SetActive(true);
    }

    if (barrelExplosionTimer < 0)
    {
        // Change platform look to destroyed look (?)

        explode = true;
        barrelsExploded++;
        Debug.Log("No. barrels exploded = " + barrelsExploded);
        Explode();
    }
}
```



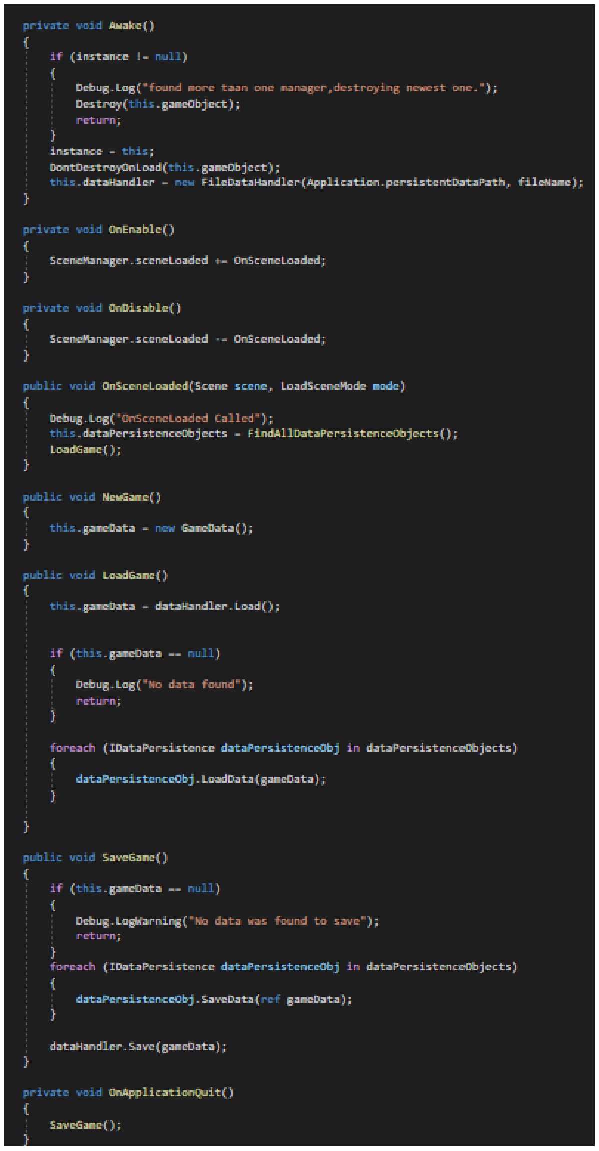
```
    // EXPLODE, deal aoe damage and destroy Conflux
    void Explode()
    {
        // Makes an array of all colliders within a range
        Collider[] explosion = Physics.OverlapSphere(transform.position, range);

        foreach (Collider other in explosion)
        {
            // Deals damage to enemies withing the explosion range
            if (other.tag == "Enemy")
            {
                other.GetComponent<AdditionalEnemy>().TakeDamage(explosionDamage);
                other.GetComponent<Enemy>().TakeDamage(explosionDamage);
            }

            // Deals reduced damage to player if they are in the range
            if (other.tag == "Player")
            {
                other.GetComponent<HealthSystem>().TakeDamage(100f);
            }
        }

        Barrel.SetActive(false);
        Explosion.SetActive(true);
        Conflux.SetActive(false);
        barrelOnFire.SetActive(false);

        this.enabled = false;
    }
}
```

Save game

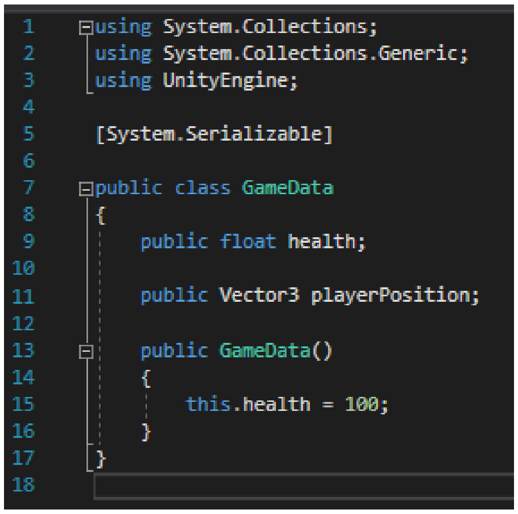
The selected player and its position in the current scene is saved automatically when the user returns to the main menu or quit the game.



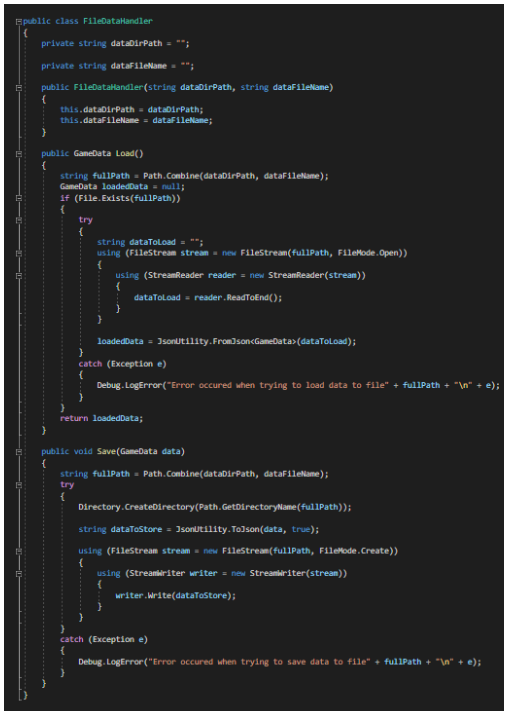
```
    public void SaveBtn()
    {
        string activeScene = SceneManager.GetActiveScene().name;
        PlayerPrefs.SetString("SavedLevel", activeScene);
        Debug.Log(activeScene);
    }

    private void OnApplicationQuit()
    {
        SaveBtn();
    }
```



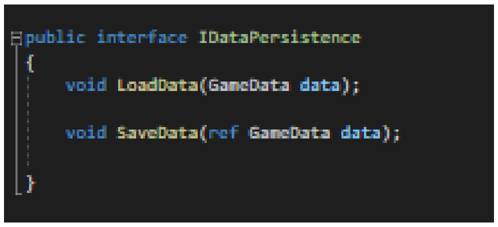
```
private void Awake()
{
    if (instance != null)
    {
        Debug.Log("found more taan one manager,destroying newest one.");
        Destroy(this.gameObject);
        return;
    }
    instance = this;
    DontDestroyOnLoad(this.gameObject);
    this.dataHandler = new FileDataHandler(Application.persistentDataPath, fileName);
}

private void OnEnable()
{
    SceneManager.sceneLoaded += OnSceneLoaded;
}

private void OnDisable()
{
    SceneManager.sceneLoaded -= OnSceneLoaded;
}

public void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    Debug.Log("OnSceneLoaded Called");
    this.dataPersistenceObjects = FindAllDataPersistenceObjects();
    LoadGame();
}

public void NewGame()
{
    this.gameData = new GameData();
}

public void LoadGame()
{
    this.gameData = dataHandler.Load();


    if (this.gameData == null)
    {
        Debug.Log("No data found");
        return;
    }

    foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
    {
        dataPersistenceObj.LoadData(gameData);
    }

}

public void SaveGame()
{
    if (this.gameData == null)
    {
        Debug.LogWarning("No data was found to save");
        return;
    }
    foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
    {
        dataPersistenceObj.SaveData(ref gameData);
    }

    dataHandler.Save(gameData);
}

private void OnApplicationQuit()
{
    SaveGame();
}
```



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]

public class GameData
{
    public float health;

    public Vector3 playerPosition;

    public GameData()
    {
        this.health = 100;
    }
}
```



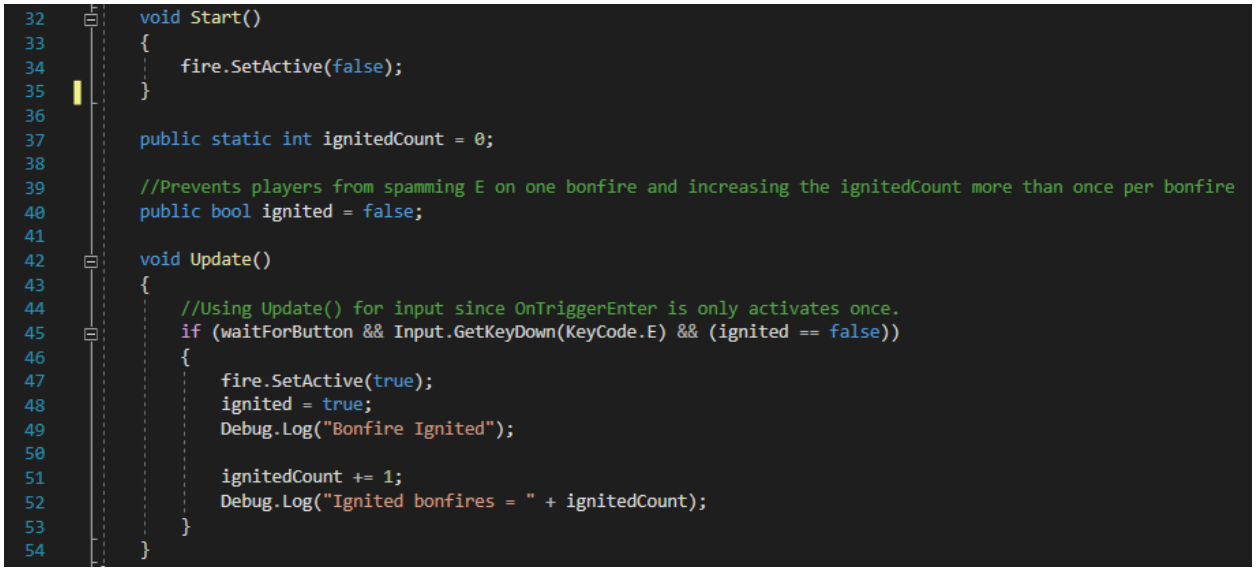
```
public class FileDataHandler
{
    private string dataDirPath = "";

    private string dataFileName = "";

    public FileDataHandler(string dataDirPath, string dataFileName)
    {
        this.dataDirPath = dataDirPath;
        this.dataFileName = dataFileName;
    }

    public GameData Load()
    {
        string fullPath = Path.Combine(dataDirPath, dataFileName);
        GameData loadedData = null;
        if (File.Exists(fullPath))
        {
            try
            {
                string dataToLoad = "";
                using (FileStream stream = new FileStream(fullPath, FileMode.Open))
                {
                    using (StreamReader reader = new StreamReader(stream))
                    {
                        dataToLoad = reader.ReadToEnd();
                    }
                }

                loadedData = JsonUtility.FromJson<GameData>(dataToLoad);
            }
            catch (Exception e)
            {
                Debug.LogError("Error occured when trying to load data to file" + fullPath + "\n" + e);
            }
        }
        return loadedData;
    }

    public void Save(GameData data)
    {
        string fullPath = Path.Combine(dataDirPath, dataFileName);
        try
        {
            Directory.CreateDirectory(Path.GetDirectoryName(fullPath));

            string dataToStore = JsonUtility.ToJson(data, true);

            using (FileStream stream = new FileStream(fullPath, FileMode.Create))
            {
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.Write(dataToStore);
                }
            }
        }
        catch (Exception e)
        {
            Debug.LogError("Error occured when trying to save data to file" + fullPath + "\n" + e);
        }
    }
}
```



```
public interface IDataPersistence
{
    void LoadData(GameData data);

    void SaveData(ref GameData data);

}
```

Bonfire Mini Quest

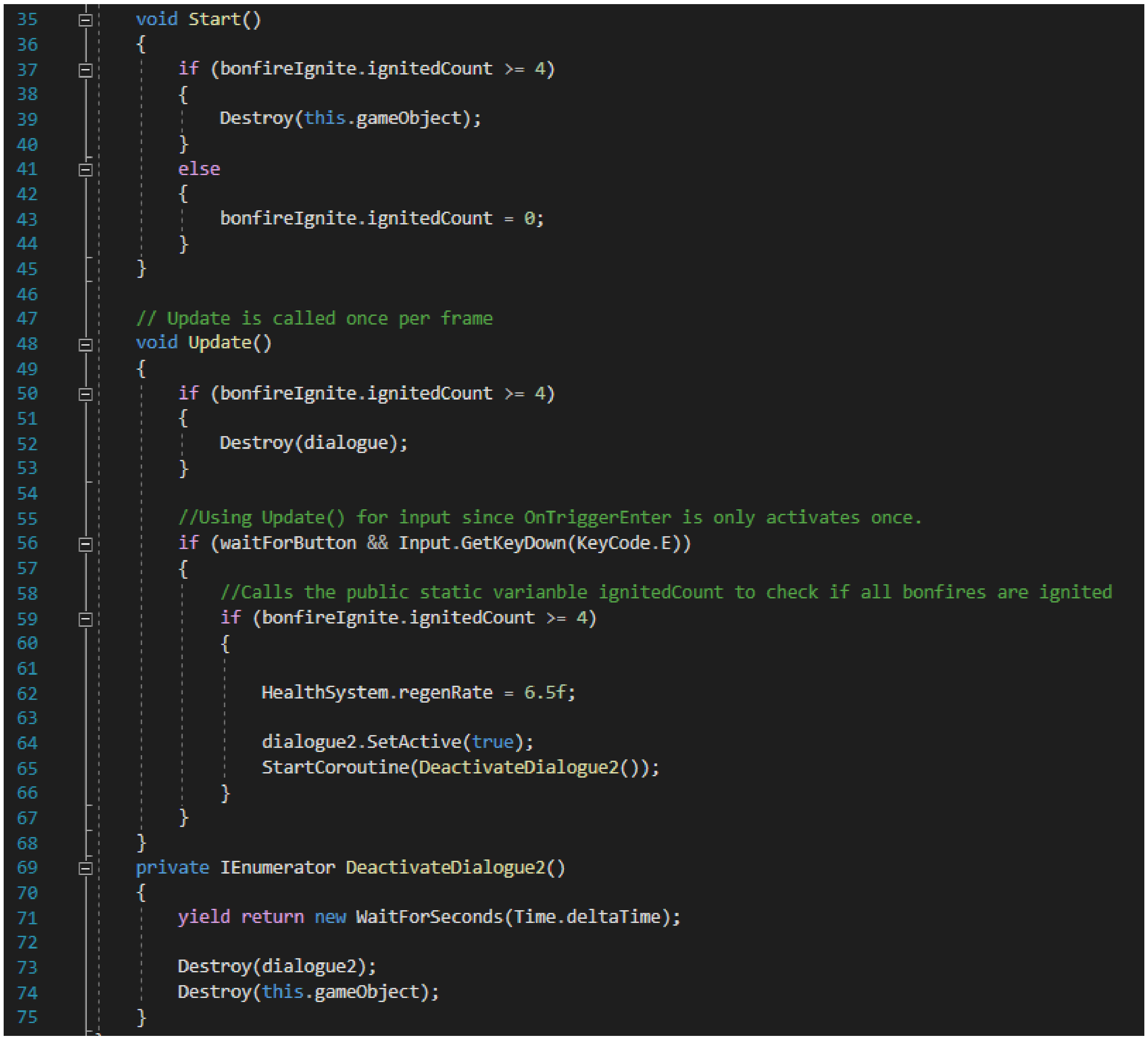
Bonfire



```
void Start()
{
    fire.SetActive(false);
}

public static int ignitedCount = 0;

//Prevents players from spamming E on one bonfire and increasing the ignitedCount more than once per bonfire
public bool ignited = false;

void Update()
{
    //Using Update() for input since OnTriggerEnter is only activates once.
    if (waitForButton && Input.GetKeyDown(KeyCode.E) && (ignited == false))
    {
        fire.SetActive(true);
        ignited = true;
        Debug.Log("Bonfire Ignited");

        ignitedCount += 1;
        Debug.Log("Ignited bonfires = " + ignitedCount);
    }
}
```

Chest

When the chest is open, the player is rewarded in the form of increased health regeneration.



```
    void Start()
    {
        if (bonfireIgnite.ignitedCount >= 4)
        {
            Destroy(this.gameObject);
        }
        else
        {
            bonfireIgnite.ignitedCount = 0;
        }
    }

    // Update is called once per frame
    void Update()
    {
        if (bonfireIgnite.ignitedCount >= 4)
        {
            Destroy(dialogue);
        }

        //Using Update() for input since OnTriggerEnter is only activates once.
        if (waitForButton && Input.GetKeyDown(KeyCode.E))
        {
            //Calls the public static varianble ignitedCount to check if all bonfires are ignited
            if (bonfireIgnite.ignitedCount >= 4)
            {

                HealthSystem.regenRate = 6.5f;

                dialogue2.SetActive(true);
                StartCoroutine(DeactivateDialogue2());
            }
        }
    }
    private IEnumerator DeactivateDialogue2()
    {
        yield return new WaitForSeconds(Time.deltaTime);

        Destroy(dialogue2);
        Destroy(this.gameObject);
    }
```

6. Testing

Test No	**1**
Component tested	**Character controls**
Purpose of test	To check if the following works: ● Walking ● Jumping ● Sprinting
Expected results	1. Characters must be able to walk in all directions by using the keys (W/A/S/D). 2. Character must be able to jump by using the key (SPACE) 3. Character must be able to run using the key (SHIFT) while moving
Actual results	The actual results matched the expected results
Tested by	Lovesh Dhoounmoon

Test No	2
Component tested	Abilities

Purpose of test	To check if the following work for each characters: 1. Invincibility 2. Rain of arrows 3. Heavy hit 4. Heal
Expected results	The characters must be able to activate their special ability using Q button(for female characters) or by holding shift and left click (male characters)
Actual results	1. Invincibility cause the character to scream and become temporarily invincible 2. Rain of arrows- 10 arrows are spawned and causes a radial damage to enemy found within that radius 3. heavy hit- creates a spark on the ground upon hitting and causes large radial damage to enemy 4. Heal- character instantly gain max health
Tested by	Lovesh Dhoounmoon

Test No	3
Component tested	Selection screen
Purpose of test	To check if 1. characters can be selected from the screen using select

	button 2. character on screen is changed to the next one upon pressing right button 3. character on screen is changed to the previous one upon pressing left button 4. Description of characters changes as other character is changed
Expected results	All buttons should work upon being pressed, description of characters should match character being shown on screen
Actual results	Everything button work as expected and description changes respectively
Tested by	Prithvi Jay Krishna Deelah

Test No	4
Component tested	Options Menu
Purpose of test	To check if the following work: ● Resolution settings ● Fullscreen/windowed ● Graphics Quality Settings ● Audio settings
Expected results	1. The user must be able to set the desired resolution 2. The user must be able to select either Fullscreen or windowed mode. 3. The user must be able to

	choose the game quality (low, medium, high) 4. The user must be able to adjust the audio settings.
Actual results	The actual results matched the expected results
Tested by	Shivesh Ramjeeawon

Test No	5
Component tested	Death Handler
Purpose of test	To check if the death handler script is working properly.
Expected results	1. Character must die if its health is <=0 2. Character must respawn at the last checkpoint saved.
Actual results	The actual results matched the expected results
Tested by	Shivesh Ramjeeawon

Test No	6
Component tested	Health bar UI
Purpose of test	To check if the health bar UI is working as intended.
Expected results	1. A health bar must be displayed on top of the game.

	2. The health bar must decrease if the player has taken damage from enemies/boss 3. The health bar must increase overtime.
Actual results	The actual results matched the expected results
Tested by	Shivesh Ramjeeawon

Test No	7
Component tested	Pause Menu
Purpose of test	To check if the following works: ● Resume game button ● Option button ● Main menu button ● Exit button
Expected results	1. User must be able to resume the game. 2. User must be able to access the settings panel. 3. User must be able to access the main menu. 4. User must be able to close the game
Actual results	The actual results matched the expected results
Tested by	Shivesh Ramjeeawon

Test No	8

Component tested	Enemy AI
Purpose of test	To check if the following works: • Enemy animation states (iddle, walking,attacking) • Enemy following range • Enemy attacking range • Damage dealt from enemy attack • Enemy health • Idle state after exiting following range • Enemy death
Expected results	1. Proper animations for the different states. 2. Enemies must be able to follow the player if either provoked (i.e shot at) or is in range of the target. 3. Enemies must be able to attack only if it is in the attacking range of the target. 4.Enemies must be able to inflict 40 damage per swing. 5.Enemy health must decrease if it is being shot at by the target with every weapon. 6.Enemy must stop following the player if the latter has stepped out of its following range. 7. Enemy shall die if its health is <= 0.
Actual results	The actual results matched the expected results
Tested by	Nileshwar Saleegram

Test No	9
Component tested	NPC dialogue
Purpose of test	Check if NPC can interact with player if it is within a range

Expected results	NPC interacts with player and shows correct dialogue
Actual results	NPC interacts with player correctly
Tested by	Prithvi Jay Krishna Deelah

Test No	10
Component tested	Gameplay UI (Ability)
Purpose of test	Check if ability synchronises with cooldown icon
Expected results	The cooldown icon should progressively light up as the cooldown is over.
Actual results	The actual results matched the expected results
Tested by	Lovesh Dhoounmoon

Test No	11
Component tested	Normal attack
Purpose of test	To check if each character can perform their regular attack
Expected results	1. Lucy (Archer)- On pressing right mouse button, the camera angle should change

	and left click should cause the character to shoot an arrow 2. Paladin (Knight)-on clicking left mouse button, player shall perform sword attack, double clicking should cause combo attack 3. Warrior-on clicking left mouse button, player shall perform axe swing attack, double clicking should cause combo attack 4. Kaguchi Jin (Wizard)- Left click should cause character to perform a throwing animation and spawn a fireball
Actual results	The actual results matched the expected results
Tested by	Nileshwar Saleegram

Test No	12
Component tested	Save & Load Game
Purpose of test	Check if • the selected character is saved • Last scene player was in when user quit the game or return to main menu is saved Player position is saved
Expected results	The saved character should be loaded in the previous scene at the saved

	position
Actual results	The actual results matched the expected results
Tested by	Lovesh Dhoounmoon

Test No	13
Component tested	UI Waypoint
Purpose of test	Check if • Waypoint disappear • Next waypoint shown • Distance text changes Waypoint icon move on side of screen
Expected results	• the waypoint must disappear when player is close • Next waypoint should appears • Distance text in waypoint changes based on player position from it Waypoint icon goes to left or right of screen if player is looking away
Actual results	The actual results matched the expected results
Tested by	Lovesh Dhoounmoon

Test No	14
Component tested	Boss attacks

Purpose of test	Check if • Smash attack is triggered • Stomp attack is triggered • Sweep attack is triggered • Jump-sweep attack is triggered Laser beam is triggered
Expected results	All damage hitboxes activate during the attacks and deactivate after finishing or on hitting the player.
Actual results	All attacks are triggered properly and causes damage to the player as needed.Results match expectation
Tested by	Nileshwar Saleegram

7. Conclusion

7.1 Achievements

1. Acquired the capability to work on different parts of the same project simultaneously.

2. Became more skilled in creating project designs and plans.

3. Learned to utilise various assets effectively.

4. Developed the ability to view projects from both user and developer perspectives.

5. Acquired skills in using C#.

6. Enhanced communication skills.

7. Gained proficiency in working within Unity's development environment.

8. Gained insights into creating multiple animations.

9. Developed understanding of implementing camera effects, like fog.

10. Adapted to working effectively under pressure.

11. Learned how to do a good and detailed report about a project

7.2 Challenges and problems encountered

The game was definitely filled with challenges, firstly it was a new programming language and new environment along with work style. Without YouTube tutorials, text tutorials on unity, we would not be able to complete our game project

- Lack of knowledge for implementing tricky features
- The engine requires vast knowledge about its properties, features and sections
- It requires a lot of time and dedication to create game as we have to work every single point in the game. Thus, a lot of time dedicated to build the game
- Making each character's abilities work as intended
- Merging everything together (UI, characters, map and enemies)

7.3 Future updates and improvements

This game is made for Windows, Linux and Mac at the present. There are various aspects of the game to improve some of them are listed below.

- To change the game art entirely as we are currently using free assets
- To add more levels and boss arena
- Add more enemies
- Make it multiplayer co-op
- Make lobby map more explorable
- Adding side quests
- More realistic behaviour from NPC
- Introduce new game features
- Use AI generated response for NPC dialogue
- More realistic graphics
- Attach health bar on every enemy and bosses
- Adding a minimap to guide player

9. References

1. https://docs.unity3d.com/Manual/index.html [Unity Documentation]
2. https://www.udemy.com/course/unitycourse2/ [Unity course, created by: Ben Tristem, Rick Davidson, GameDev.tv Team, Gary pettie 3/2023]
3. Starter Assets - Third Person Character Controller | URP | Essentials | Unity Asset Store [Date accessed: 15/03/23]
4. https://www.youtube.com/watch?v=j48LtUkZRjU&list=PLPV2KyIb3jR5QFsefuO2RlAg WEz6EvVi6 [Unity beginner tuto, created by: Brackeys 23/2017]
5. Mixamo [Date accessed: 03/04/23]
6. Low-Poly Simple Nature Pack | 3D Landscapes | Unity Asset Store[Date accessed: 08/04/23]
7. Low Poly Dungeons Lite | 3D Dungeons | Unity Asset Store [Date accessed: 16/04/23]
8. (698) How to make an enemy follow player - Unity NavMesh - YouTube [Date accessed: 1/05/23]
9. (698) Third Person (Archery / Bow and Arrow) System in Unity - Part 1 - YouTube [Date accessed: 13/05/23]
10. https://www.youtube.com/watch?v=ZzkIn41DFFo&ab_channel=HypedCloud [Date accessed: 22/05/23]
11. https://assetstore.unity.com/packages/tools/ai/ultimate-a-pathfinding-solution-224082 [Date accessed: 08/06/23]
12. https://assetstore.unity.com/packages/3d/characters/creatures/dragon-the-soul-eater-and-dragon-boar-77121 [Date accessed: 08/06/23]
13. (698) 3D ENEMY AI in UNITY - (E01): STATE MACHINE BEHAVIORS - YouTube [Date accessed: 10/06/23]
14. (698) How to Talk to NPCs! (or Interact with any Object, Open Doors, Push Buttons, Unity Tutorial) - YouTube [Date accessed: 1/07/23]
15. 201 Free Medieval Fonts · 1001 Fonts [Date accessed: 10/07/23]
16. Fantasy Dungeon Starter Kit | 3D Dungeons | Unity Asset Store [Date accessed: 15/07/23]

17. https://unityassetcollection.com/fantasy-adventure-environment-free-down1load/ [Date accessed: 1/08/23]
18. (698) Create a CHARACTER CREATION SCREEN In Unity - EP. 1 Character Preview - YouTube [Date accessed: 12/08/23]