National College of Ireland

Project Report

Broken Succession

15/05/2021

Bachelor's in computer sciences

Software Engineering

2020/2021

Liam Duggan

X17148529

X17148529@student.ncirl

Executive Summary

This project will consist of building a third person action role playing game in the vein of older titles such as Soul Reaver and Heretic 2. This game will allow the users to start a new game and explore the world, engaging with characters in that world, picking up quests, and fighting enemies. The game was built using the Unity Game engine

Section 1 – Introduction

1.1. Aims

My hope when I began this project was to work in depth with Unity to create a first person action game in the vein of older first person action games such as Blood and Hexen. But over the course of the project my aims shifted as I become more invested in the project. This lead to my reevaluating my original idea and attempting to pivot. My aim from then on was to create a third person adventure role playing game. My aim was to build a world where the player would be able to explore various environments, engage with

and accept quests from non-player characters, upgrade their character via rewards and experiences, and fight monsters within the world.

1.2. Technology

The primary technology of this project is Unity. Unity is a game development environment that is very popular with hobbyists and professionals alike. It offers very good functionality and ease of use when starting to build games, and dues to the lifespan and popularity it has amassed an enormous amount of user created plugins and tools that help with the designing and building process. Other tools that have been used include Blender, a 3D modeling and animation software that is freely available and widely supported, and has very easy integration of models and animations into Unity, MagickaVoxel which is a Voxel model creator, which allows the user to create models very easily, which can be imported into Blender for the purposes of animating them.

The primary coding language used in the project is C#, using the Visual Studio IDE. C# is the most supported language for unity, but previous versions also allowed for the use of a proprietary version of JavaScript known as UnityScript, which has since been deprecated.

1.3. Structure

In Section 1, the introduction, the background to this project was discussed, along with the aims I sought to fulfil, the technologies I utilised and why I chose them as opposed to competing software that is available in the field of the project I am undertaking.

Section 2 – System. In this section I will be explaining the requirements for my project in detail, and then describing the technical implementation of those requirements, and the rationale behind those implementations.

In the Functional Requirements subsection, I set out the base line must have functionality that the software must achieve and for each requirement provide a detailed itinerary of the progression through that requirement, alongside a Use Case Diagram, to help visualise the function.

In the Non-Functional Requirements subsection, I will be detailing the requirements for areas that while not fundamental to the base execution of the game, are still very important to how the game looks and feels to operate. This will include performance requirements, data requirements, and usability requirements.

In the implementation section I will explain some of the more elaborate and important components and game objects I created, and how they were built. I will go through some of the previous iterations of the systems, and how they came to be at this stage of the project.

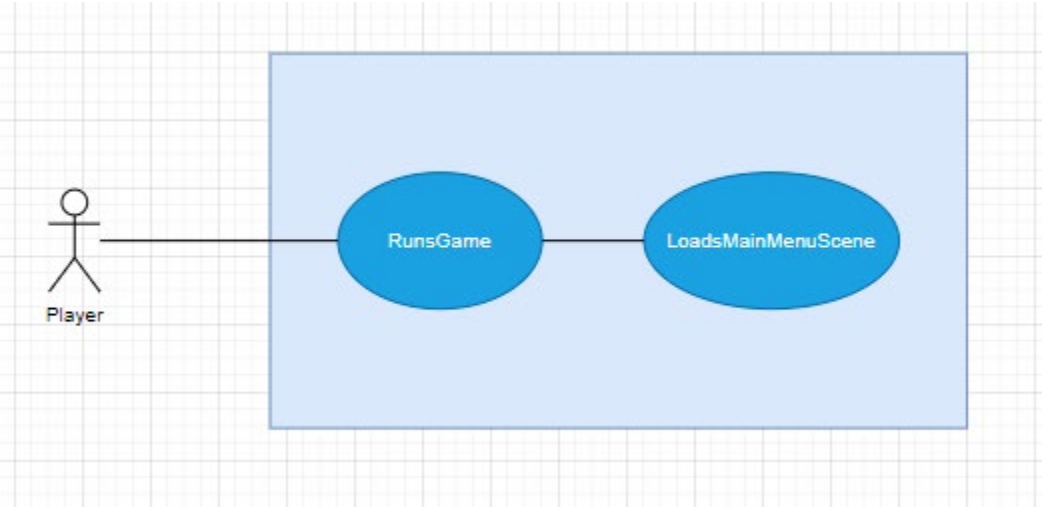
Section 2 – System

Functional Requirements

Requirement 1 – Player Opens Game Program

Figure 1: This use case diagram will show the actions performed as the user interacts with the system to perform the action of opening the game. This requirement is set as the top priority, as the user will not be able to play the game if there is an error or exception while performing this action, and if the game does not correctly route the player to the main menu scene the User Experience would be very adversely affected.

Use Case – Figure 1



Use Case Description

- **Name:** Start Game Use Case
- **Description:** This use case is related to the action of starting the game executable and being routed to the main menu scene.

Flow of Events

Activation: Player runs the BrokenSuccession.exe from their file system.

Basic Flow

1. The use case starts when the Player runs the BrokenSuccession.exe file from their file system.
2. The game starts and displays the startup credit sequence.
3. The game correctly loads the initial game scene, the Main Menu scene.
4. The Main Menu UI displays, showing the options ‘New Game’, ‘Load Game’, ‘Options’, ‘Exit Game’

Alternative Flow

Termination: Player is shown the main menu screen.

Special Requirements

None

Preconditions: Player must have BrokenSuccession.exe on the system.

Post-conditions:

Success Conditions:

1. The player can select one of the options available in the Main Menu UI.

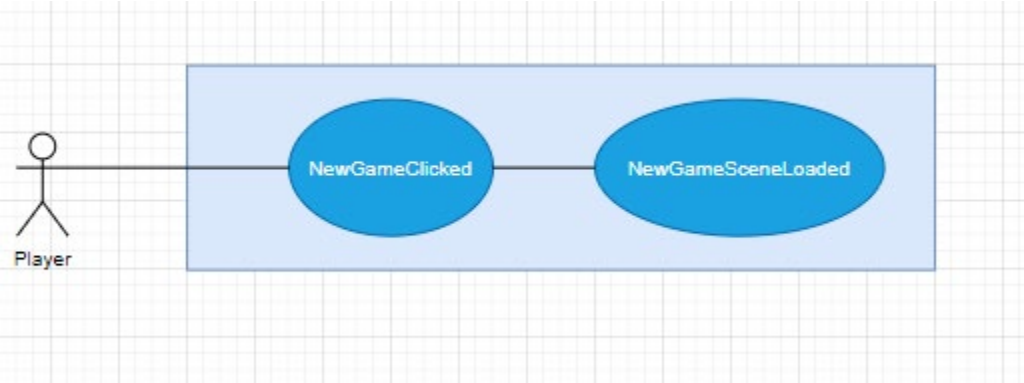
Failure Conditions:

1. Game loads a different Scene on startup.
2. Game finds no scenes in the build order, and proceeds to close.
3. The player in unable for whatever reason to interact or execute any of the options available on the Main Menu UI.

Requirement 2 – Player Selects New Game

Figure 2: This use case diagrams shows the way the system handles the interaction between itself and the player when the player selects the option 'New Game' from the Main Menu UI. This option is placed second in the hierarchy of requirements as it is what most Players will choose first when the open the game for the first time. Due to this, I placed it as a higher requirement than the other menu options.

Figure 2



Use Case Description

- **Name:** Select New Game Use Case
- **Description:** This use case is related to the action of choosing New Game from the Main Menu scene UI.

Flow of Events

Activation: Player Clicks New Game

Basic Flow

1. The use case starts when the Player clicks New Game on the main menu UI.
2. The game loads the First Level Scene
3. Player is given control of the PlayerCharacter in the scene

Alternative Flow

Termination: Player is in the first level scene and has control of the Player Character in the scene.

Special Requirements

None

Preconditions: Player must have successfully loaded the game.

Post-conditions:

Success Conditions:

1. Player is in the first level scene and has control of the Player Character in the scene.

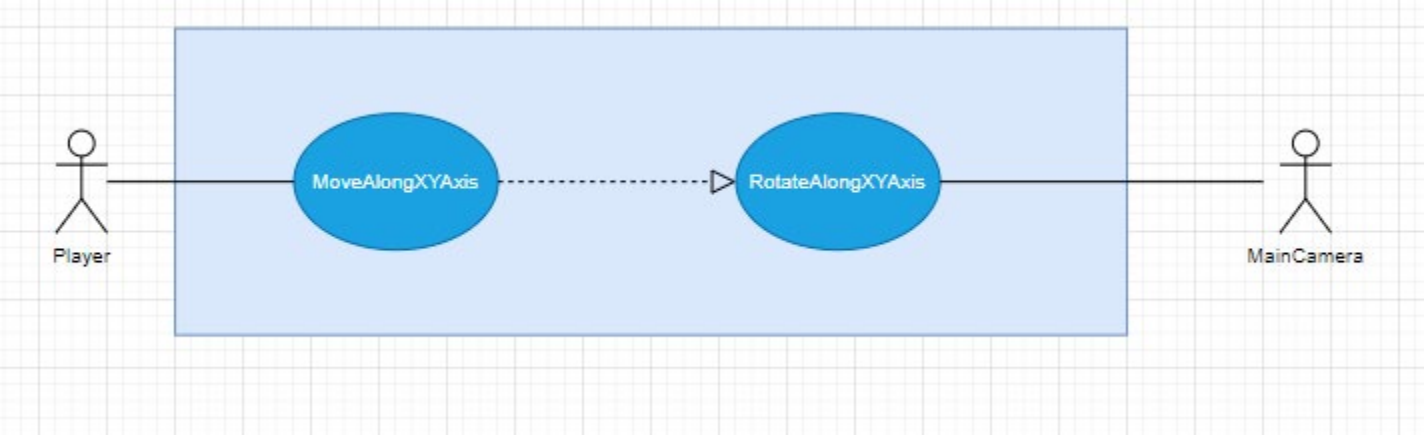
Failure Conditions:

1. Game loads a different Scene on New Game Selection.
2. Game executes the wrong function and opens another menu option.
3. Game loads correct scene, but Player does not have control.

Requirement 3 – Player can control the Main Camera in ThirdPerson Scene.

Figure 3: Figure 3 shows the interaction between the Player Actor, and the Main Camera actor in any scene that utilizes the third person perspective. This was chosen as the third in the priority list as it was considered the first part of "in game" control the Player would experience, and thus warranted the high priority.

Figure 3



Use Case Description

- **Name:** Player can control the Main Camera in ThirdPerson Scene.
- **Description:** This use case is related to the interaction between the Player moving their mouse along its XY axis in and the rotation of the Main Camera in the scene.

Flow of Events

Activation: Player moves their mouse along its X/Y axis

Basic Flow

1. The Player moves the mouse along its X/Y Axix
2. The MainCamera in the game scene rotates along that axis in response.

Alternative Flow

Termination: The Player ceases movement of the mouse.

Special Requirements :

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera prespective.

Post-conditions:

Success Conditions:

1. The Player is able to rotate the camera in the scene in response to their mouse movements.

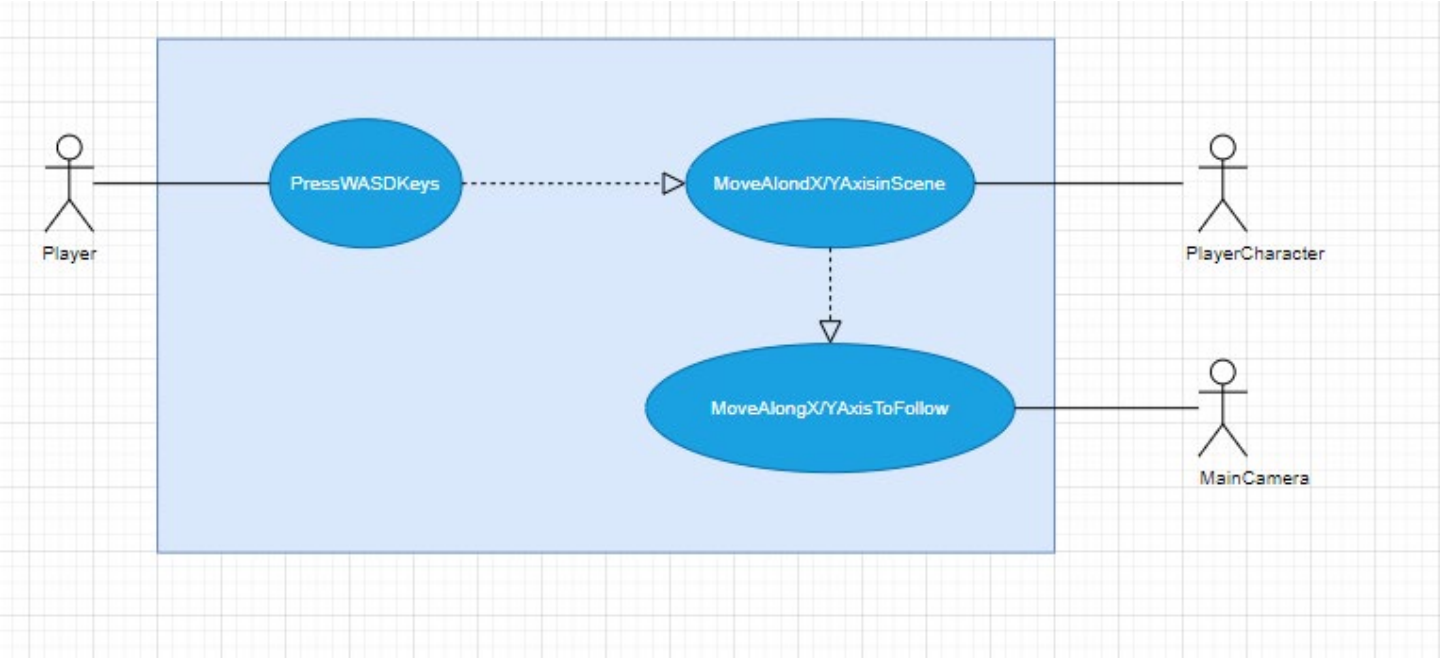
Failure Conditions:

1. The MainCamera doesn't respond to the mouse movement, leaving the camera at a static rotation.
2. The MainCamera object in the scene moves along its X/Y axis, rather than rotates.
3. The MainCamera object in the scene both moves and rotates on its X/Y axis.

Requirement 3 – Moving the Player Character in response to Player Keyboard action

Figure 4: Figure 5 shows the interaction between the Player Actor keyboard actions, and the Player Character actor in any scene that utilizes the third person perspective. This was chosen a fourth in the priority list as, similar to the camera control requirement above, moving the Player Character in a scene was considered one of the first pieces of control the Player would experience in the game.

Figure 4



Use Case Description

- **Name:** Moving the Player Character in response to Player Keyboard action
- **Description:** This use case is related to the interaction between the Player pressing the WASD keys on their keyboards and the PlayerCharacter in the scene moving along the X/Y axis in the scene.

Flow of Events

Activation: Player presses on of the WASD keys on their keyboard

Basic Flow`

1. Player presses on of the WASD Keys
2. The PlayerCharacter within the scene moves in the scene in the associated direction (W for forward, A for left, D for right, and S for back)
3. The MainCamera moves in the Scene to follow the PlayerCharacter

Alternative Flow

Termination: The Player stops pressing or holding down any of the WASD keys.

Special Requirements:

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera prespective.

Post-conditions:

Success Conditions:

1. The PlayerCharacter is now at the appropriate X/Y position based on the keyboard actions taken by the Player, with the MainCamera having followed with it.

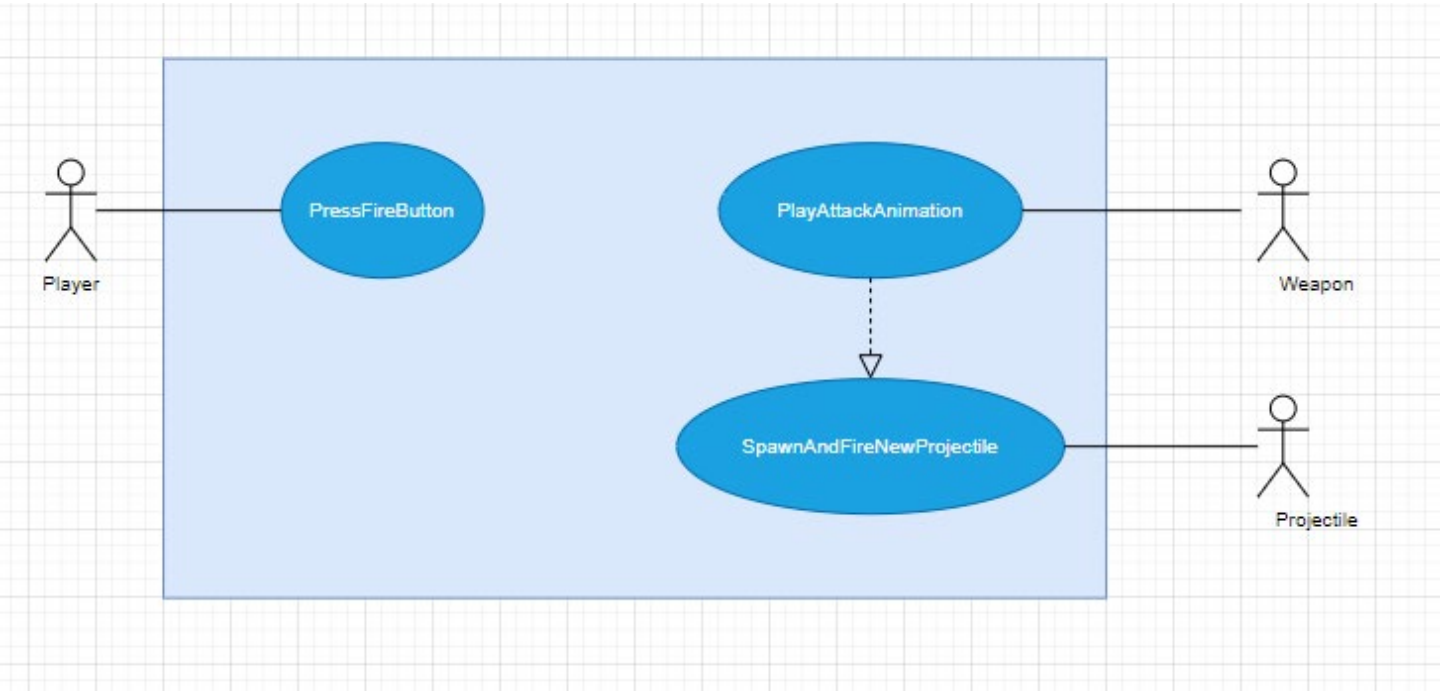
Failure Conditions:

1. The PlayerCharacter does not move in response to the keyboard actions taken by the Player.
2. The PlayerCharacter moves, but not in the correct direction. i.e. the Player presses W and the PlayerCharacter moves left in response.
3. The PlayerCharacter moves correctly in response to Player keyboard action, but the MainCamera does not move to follow them.

Requirement 5 – Activating Current Weapon

Figure 5: Figure 5 shows the interaction between the Player and the currently equipped player when the Player press the Fire button. This was chosen fifth in the priority list as the game is a third person action game, with activating the Players weapons being a key gameplay component.

Figure 5



Use Case Description

- **Name:** Activating the Players current weapon.
- **Description:** This use case is related to the interaction between the Player and the currently equipped Weapon when the Player presses the Fire key.

Flow of Events

Activation: Player presses the Fire key.

Basic Flow`

1. The Player presses the Fire key.
2. The Weapon plays its attack animation.
3. **In the case of a Projectile Weapon**
    a. The Weapon spawns a new projectile object.
    b. The Projectile is fired in the Direction the Player is pointing

Alternative Flow

1. No currently equipped weapon.

Termination: The Player stops pressing the Fire Key

Special Requirements:

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera perspective and be currently equipped with a weapon.

Post-conditions:

Success Conditions:

1. The currently equipped Weapon plays its attack animation
2. **In the event of a Projectile Weapon**
    a. The Weapon spawns a new Projectile object.
    b. The new Projectile is fired in the direction the Player is pointing.

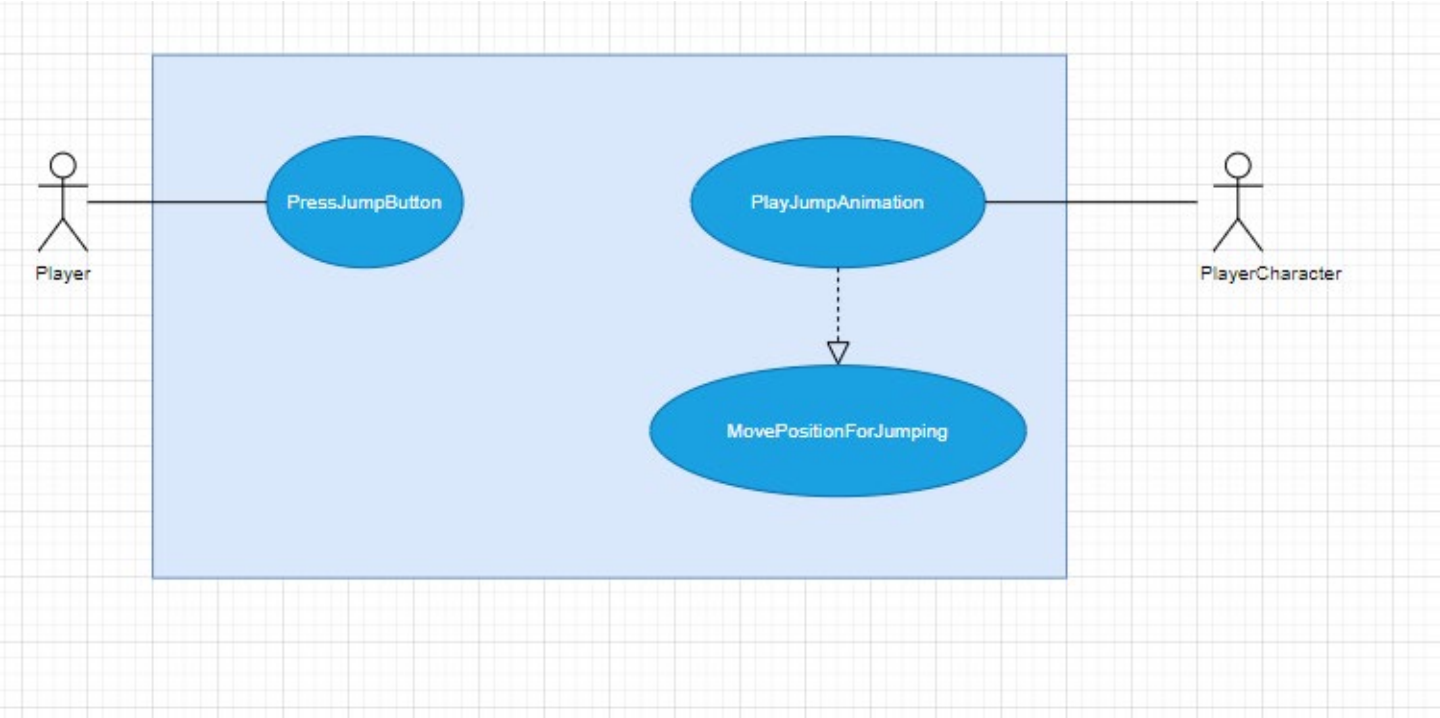
Failure Conditions:

1. The currently equipped weapon does not play its attack animation in response to Player pressing the fire Key
2. **In the event of a Projectile Weapon**
    a. The Weapon does not create a new Projectile Object.
    b. The new Projectile Object is created but is not fired in the correct direction.

Requirement 6 – Player Jump

Figure 6: Figure 6 shows the interaction between the Player and the PlayerCharacter object when then Player presses the jump key. This was chosen sixth in the priority list as the game is a third person action game, with jumping around the environment being a key gameplay component

Figure 6



Use Case Description

- **Name:** Player Jump
- **Description:** This use case is related to the interaction between the Player and the PlayeCharacter when the Player presses the Jump key.

Flow of Events

Activation: Player presses the Jump key.

Basic Flow`

1. The Player presses the Jump key.
2. The PlayerCharacter plays its jump animation.
3. The PlayerCharacter moves along its Y position to Jump.

Alternative Flow

1. The Player is already Jumping.

Termination: The Player stops pressing the Jump Key

Special Requirements:

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera perspective and on the ground.

Post-conditions:

Success Conditions

1. The PlayerCharacter plays the jump animation.
2. The PlayerCharacter moves along its Y position to jump.

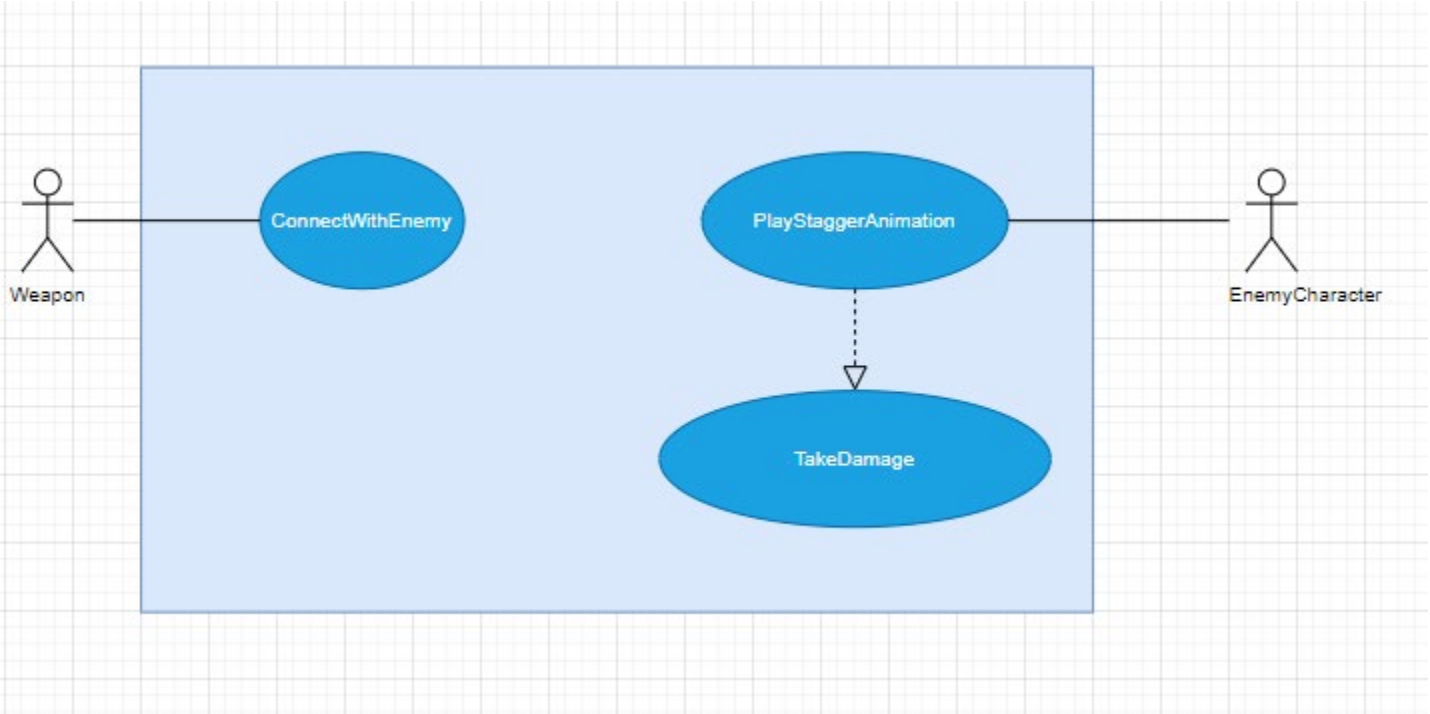
Failure Conditions:

1. The PlayerCharacter does not play the Jump animation or moves along its Y position to jump.
2. The PlayerCharacter does play the Jump animation, but does not movealong its Y position to jump
3. The PlayerCharacter moves along its Y position to jump, but does not play the jump animation.

Requirement 7 – Hurt Enemy

Figure 7: Figure 7 shows the interaction between the Weapon and the EnemyCharacter object when then Weapon comes into contact with the Collider object attached to the EnemyCharacter. The was chosen seventh in the priority list as the game is a third person action game and combat with enemies is a key component.

Figure 7



Use Case Description

- **Name:** Player hurts Enemy
- **Description:** This use case is related to the interaction between the Player Weapon, or a Projectile spawned from the Player weapon, intersects with an EnemyCharacter in the Scene.

Flow of Events

Activation: Player Weapon or Projectile intersects with an EnemyCharacter in the Scene.

Basic Flow`

1. Player Weapon or Projectile intersects with and EnemyCharacter.
2. EnemyCharacter loses health equal to the damage value of the Weapon/Projectile.
3. **In the event the EnemyCharacters remaining health > 0**
    a. EnemyCharacter plays its staggered animation.

b. EnemyCharacter proceeds to operate after staggered animation has completed playback.

4. **In the event he EnemyCharacters remaining health <=0**
    a. EnemyCharacter plays its Death animation.
    b. EnemyCharacter game object is removed from the scene upon the death animation ceasing.

Alternative Flow

Termination: The EnemyCharacter has died, or the Weapon/Projectile has ceased intersection with the EnemyCharacter.

Special Requirements:

Preconditions: Both the EnemyCharacter and Weapon must be active in the Scene.

Post-conditions:

Success Conditions

1. The EnemyCharacter loses the correct number of health points associated with the Weapon/Projectile.
2. **In the event the EnemyCharacters remaining Health points > 0**
    a. The EnemyCharacter plays its Staggered animation to completion.
    b. The EnemyCharacter resumes normal operation after animation completed.
3. **In the event the EnemyCharacters remaining health points <=0**
    a. The EnemyCharacter plays its Death animation to completion.
    b. The EnemyCharacter is removed from the Scene upon animation completed.

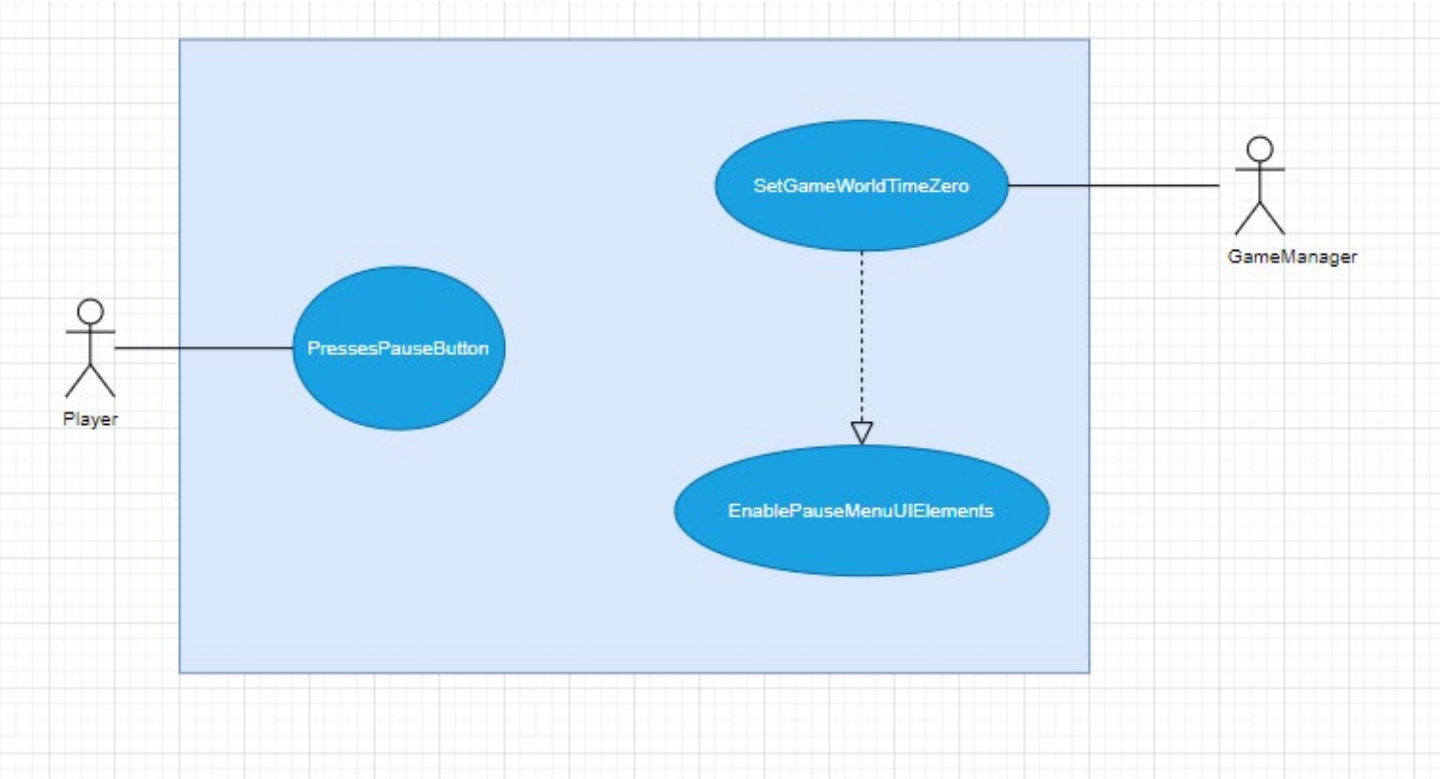
Failure Conditions:

1. The EnemyCharacter does not take any damage from the attack.
2. The EnemyCharacter does take damage, but does not play the correct animation.
3. **In the event the EnemyCharacters remaining health points > 0**
    a. The EnemyCharacter does not correctly leave the stagger animation.
    b. The EnemyCharacter does not correctly resume standard operation after completing the stagger animation.
4. **In the event the EnemyCharacters remaining health points <= 0**
    a. The EnemyCharacter is not correctly removed from the scene upon its death animation completing.
    b. The EnemyCharacter resumes normal operation after its death animation finishes despite being dead.

Requirement 8 – Pause Game

Figure 8: Figure 8 shows the interaction between the Player and the PauseMenu when the Player presses the pause button. The was chosen seventh in the priority list as the game is a third person action game and combat with enemies is a key component.

Figure 8



Use Case Description

- **Name:** Player pauses game
- **Description:** This use case is related to the interaction between the Player, GameManager, and PauseMenuUI when the Player presses the Pause button.

Flow of Events

Activation: Player Presses the Pause button.

Basic Flow`

1. Player presses the Pause button.
2. The GameManager sets the game world time to 0, meaning the game world is brought to a standstill.
3. The GameManager sets the PauseMenuUI to enabled within the Scene.

Alternative Flow

1. The game is already paused and the PauseMenuUI is displayed.
2. The Player presses the pause button.
3. The GameManager sets the PauseMenuUI to disabled within the Scene.
4. The GameManager sets the game world time to 1, bringing the game back to normal running speed.

Termination: Player presses Pause button again.

Special Requirements:

Preconditions:

Post-conditions:

Success Conditions

1. **In the event the Game has gone from In Progress to Paused**
    a. The game world speed has correctly been set at 0.
    b. The PauseMenuUI has correctly been set to enabled and is visible in the scene.
2. **In the event the Game has gone from Paused to In Progress**
    a. The game world speed has correctly been set at 1.
    b. The PauseMenuUI has correctly been set to disabled and is no longer visible in the scene.

Failure Conditions:

1. **In the event the Game has gone from In Progress to Paused**
    a. The game world speed has not correctly been set to 0.
    b. The PauseMenuUI has not been set to enabled, and is not visible in the scene.
2. **In the event the Game has gone from Paused to In Progress**
    a. The game world speed has not correctly been reset to 1.
    b. The PauseMenuUI has not correctly been set to disabled and is still longer visible in the scene despite the game no longer being paused.

Requirement 9 – Save Game

Figure 9: In this Use Case diagram, the relationship between the Player, GameManager, PlayerStatController, and FileSystem are examined when the Player chooses Save game from the in game Pause Menu.

Figure 9

Use Case Description

- **Name:** Player Saves Game
- **Description:** This use case is related to the interaction between the Player, GameManager, PlayerStatController, and FileSystem when the Player presses the Pause button.

Flow of Events

Activation: Player chooses the save game option from the Pause Game UI menu options.

Basic Flow`

1. The Player chooses an empty save slot to save their current progress to.
2. The GameManager calls the PlayerStatController to convert the current Player details into a writable format (e.g. JSON, XML, Binary etc.).
3. The PlayerStatController writes these contents to a file.
4. The FileSystem saves that file to maintain the changes.

Alternative Flow

5. The Player chooses a save slot that has already has a saved game.
6. The GameManager displays a warning that the Player will need to confirm the overwrite of the existing save data.
7. **In the event the Player confirms the overwrite**
    a. The GameManager calls the PlayerStatController to convert the current Player details into a writable format (e.g. JSON, XML, Binary etc.).
    b. The PlayerStatController writes these contents to a file.
    c. The FileSystem saves that file to maintain the changes.
8. **In the event the Player rejects the overwrite**
9. The GameManager disables the warning and returns to the Save Game screen

Termination: The Player leaves the Save Game screen.

Special Requirements:

Preconditions: The game is paused, and the PauseGameUIMenu is correctly displaying.

Post-conditions:

Success Conditions

1. The Players current details are correctly written to the save file and stored locally on the Players computer.

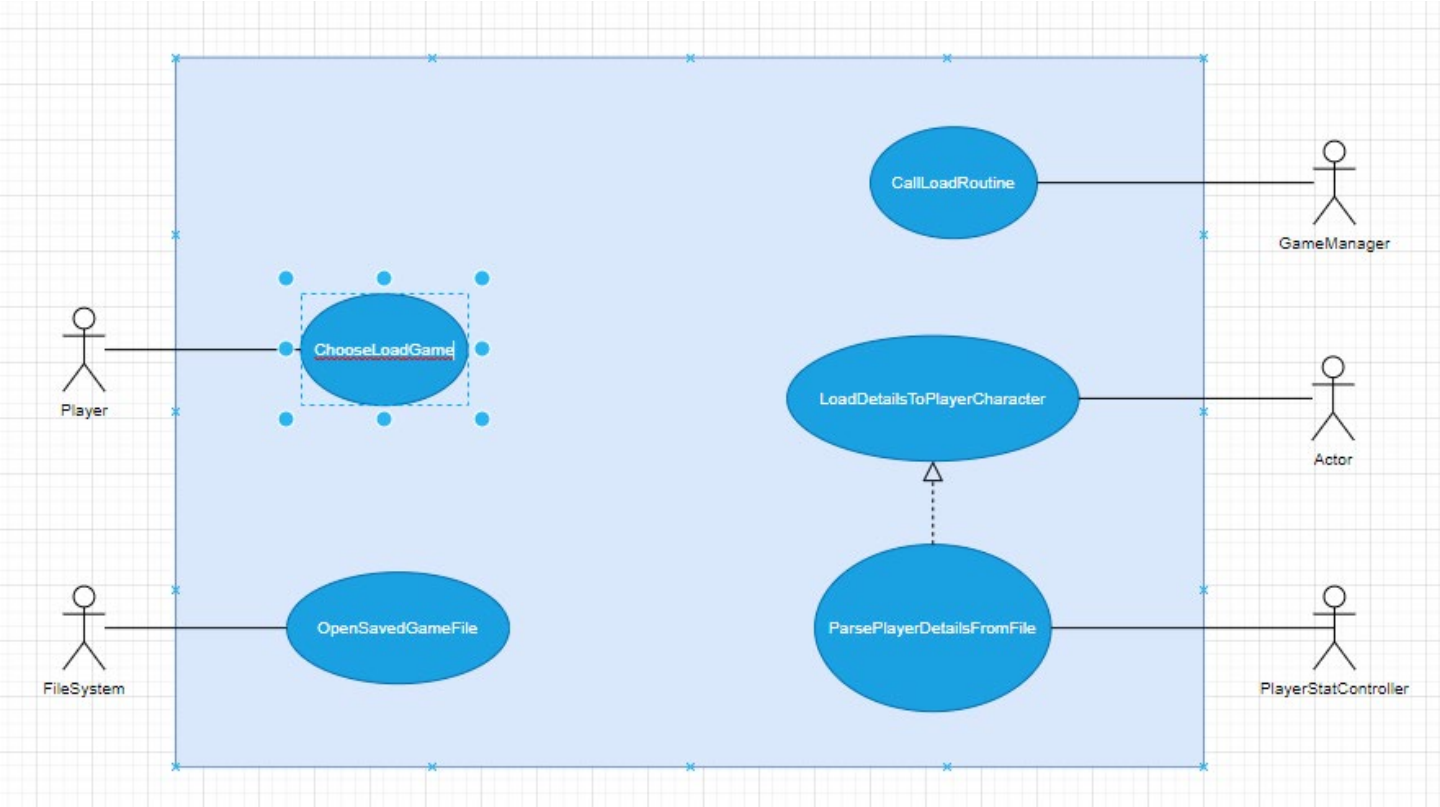
Failure Conditions:

1. The Players details are not correctly converted to the writable format, either omitting details or converting them incorrectly.
2. The Players new details are not saved to the file system, and thus can't be retrieved.
3. **In the event the Player chooses to overwrite a save**
4. The new Player details are not saved correctly to the save slot, but the Player details that were there are still lost.

Requirement 10 – Load Game

Figure 10: Figure 10 shows the interactions between the Player, GameManager, PlayerStatController, PlayerCharacter, and FileSystem when the Player chooses to open the Load Game screen, and loads and existing saved game.

Figure 10:



Use Case Description

- **Name:** Player Loads Game
- **Description:** This use case is related to the interaction between the Player, GameManager, PlayerStatController, PlayerCharacter, and FileSystem when the Player presses the Pause button.

Flow of Events

Activation: Player chooses the load game option from the Pause Game UI menu or Main Menu options.

Basic Flow`

1. The Player chooses a saved game from the Load Game list.
2. The GameManager calls the PlayerStatController to Parse the details from that chosen saved game file.
3. The FileSystem opens the chosen file of the local machine.
4. The PlayerStatController loads the data from the chosen file and parses it into the requisite format.
5. The PlayerStatContoller loads the parsed out details into the PlayerCharacter.

Alternative Flow

1. The Player is in the game when they choose to load a saved game.
2. The GameManager displays a warning that the player will lose unsaved progress if they confirm the load.
3. **In the event the player confirms they want to load the saved game**
    a. The GameManager calls the PlayerStatController to Parse the details from that chosen saved game file.
    b. The FileSystem opens the chosen file of the local machine.
    c. The PlayerStatController loads the data from the chosen file and parses it into the requisite format.
    d. The PlayerStatContoller loads the parsed-out details into the PlayerCharacter.
    e.
4. **In the event the Player rejects the load**

a. The GameManager disables the warning and returns to the Load Game screen

Termination: The Player leaves the Load Game screen.

Special Requirements:

Preconditions: The Player is on the Main Menu, or the Game has been paused and the Pause Game UI menu is correctly displayed.

Post-conditions:

Success Conditions

1. The Players game is successfully loaded from the File system.

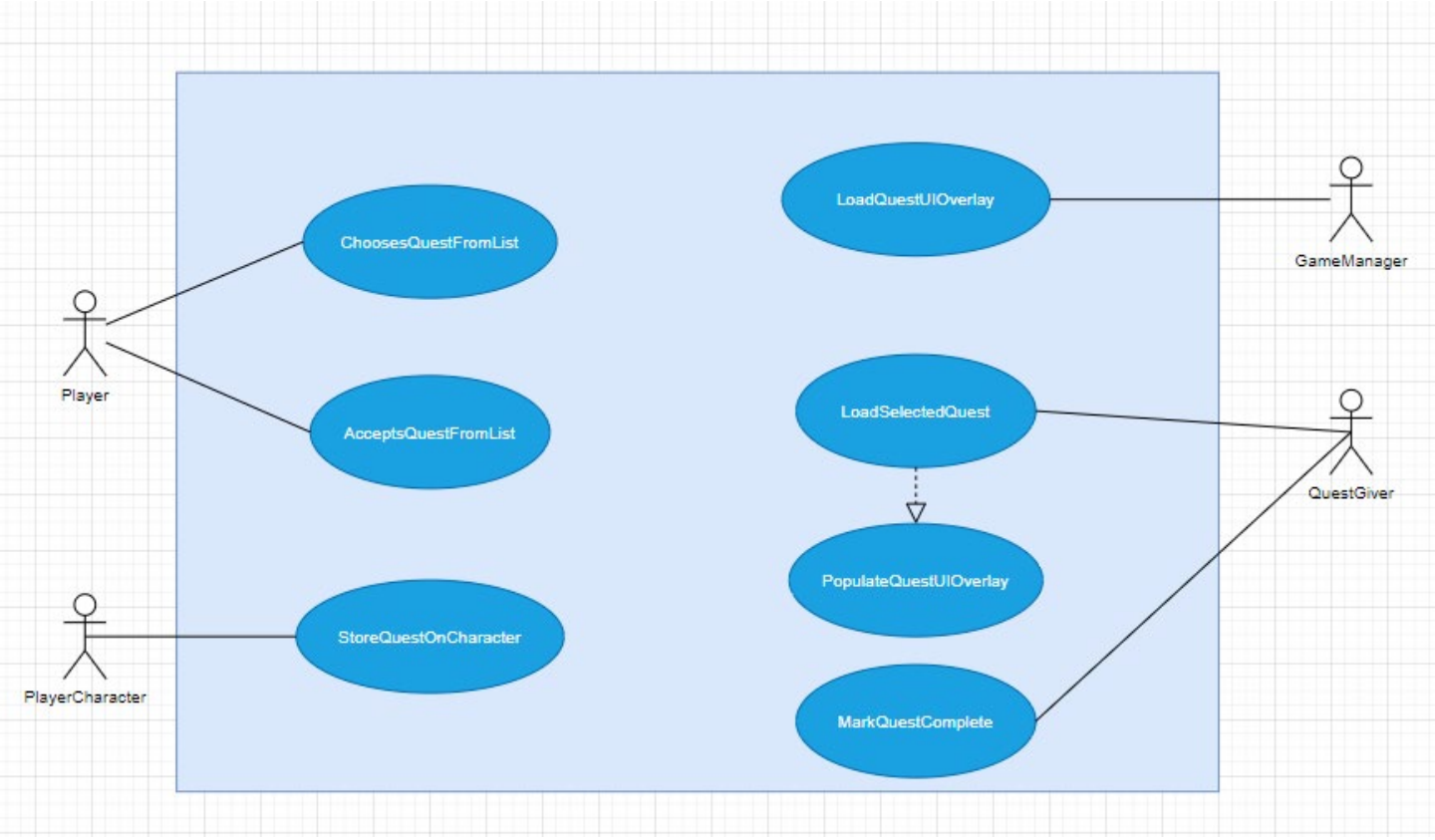
Failure Conditions:

1. The Players details are not correctly converted parsed from the saved game, leaving them with incorrect stats or in the incorrect location.

Requirement 11 – Engage with Quest Giver

Figure 11: Figure 11 illustrates the Interaction between the Player, PlayerCharacter, and QuestGiverCharacter when the Player chooses to engage with the quest giver.

Figure 11



Use Case Description

- **Name:** Player Engages with QuestGiver
- **Description:** This use case is related to the interaction between the Player, GameManager, QuestGiver, and PlayerCharacter when the Player engages with the QuestGiver.

Flow of Events

Activation: Player opts to talk to a QuestGiver NPC in game.

Basic Flow`

1. The Player chooses to talk to the QuestGiver in game.
2. The GameManager sets the QuestUIOverlay to enabled.
3. The QuestGiver load the quest from its list of quests.
4. The QuestGiver populates the QuestUIOverlay with the details of the quest.
5. The Player accepts the quest.
6. The Quest is added to the list of accepted quests attached to the PlayerCharacter.
7. The GameManager sets the QuestUIOverlay to disabled.

Alternative Flow

1. The Player chooses to talk to the QuestGiver in game.
2. The GameManager sets the QuestUIOverlay to enabled.
3. The QuestGiver load the quest from its list of quests.
4. The QuestGiver populates the QuestUIOverlay with the details of the quest.
5. The Player declines the quest.
6. The GameManager sets the QuestUIOverlay to disabled.

1. The Player has completed the quest and is returning it.
2. The Player talks to the QuestGiver.
3. The GameManager sets the QuestUIOverlay to enabled.
4. The QuestGiver searches the list of the Players accepted quests to find the quest.
5. The QuestGiver checks that the Player has completed the quest requirements.
6. The Player clicks complete quest.
7. The QuestGiver updates the PlayerCharacter with the rewards from the quest.
8. The GameManager sets the QuestUIOverlay to disabled.

1. The Player has not completed the quest and is talking to the QuestGiver again.
2. The Player talks to the QuestGiver.
3. The GameManager sets the QuestUIOverlay to enabled.
4. The QuestGiver searches the list of the Players accepted quests to find the quest.
5. The QuestGiver checks that the Player has not completed the quest requirements.
6. The QuestGiver updates the QuestUIOveylay with a message explaining the quest has not been completed.
7. The GameManager sets the QuestUIOverlay to disabled.

Termination: The Player accepts, declines, or completes a quest.

Special Requirements:

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera perspective and the PlayerCharacter must be in range of a QuestGiver.

Post-conditions:

Success Conditions

1. **In the event the Player Accepts or Completes a quest**

   a. The PlayerCharacter is correctly updated with the newly accepted Quest, or the rewards from the completed quest.
   b. The GameManager sets the QuestUIOverLay to disabled.
2. **In the event the Player Declines a quest**
   a. The GameManager sets the QuestUIOverLay to disabled.

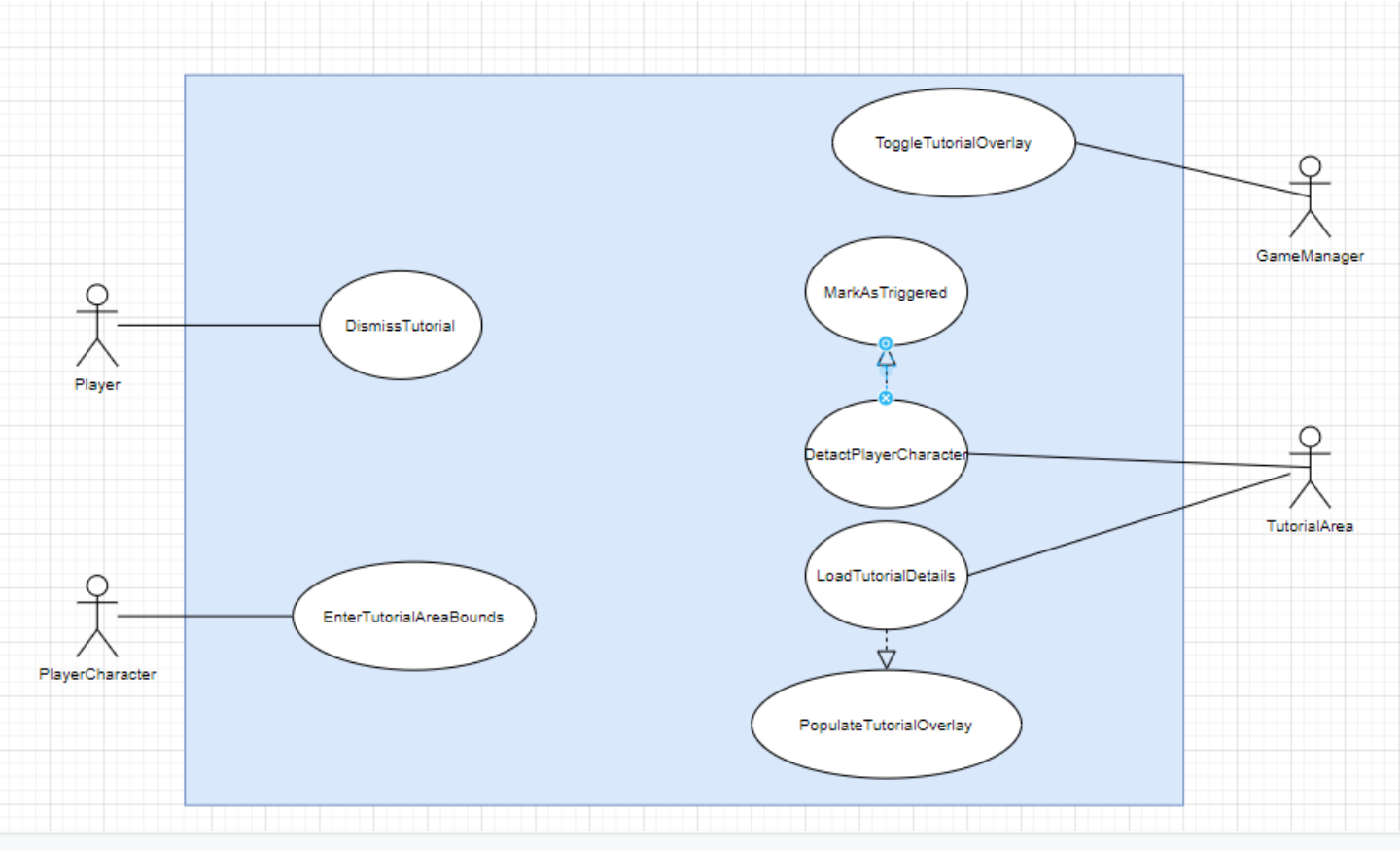
Failure Conditions:

3. **In the event the Player Accepts or Completes a quest**
   a. The PlayerCharacter is not correctly updated with the newly accepted quest.
   b. The PlayerCharacter is not correctly rewarded with the listed reward for the completed quest.
   c. The GameManager does not set the QuestUIOverLay to disabled, leaving it visible on screen even if the PlayerCharacter moves away from the QuestGiver.
4. **In the event the Player Declines a quest**
   a. The GameManager does not set the QuestUIOverLay to disabled, leaving it visible on screen even if the PlayerCharacter moves away from the QuestGiver.

Requirement 12 – Display Tutorial

Figure 12: Figure 12 show the interactions between the Player, PlayerCharacter, TutorialArea, and GameManager, when the PlayerCharacter enters the bounds of a TutorialArea within the Game World. This requirement was giving relatively low priority because while it is necessary to provide tutorials to the Player, the act of doing so dynamically within the game world as opposed to in a splash screen within a menu or manual is not as required.

Figure 12



Use Case Description

- **Name:** PlayerCharacter enters bounds of TutorialArea
- **Description:** This use case is related to the interaction between the Player, GameManager, TutorialAra, and PlayerCharacter when the PlayerCharacter crosses the bounds

Flow of Events

Activation: PlayerCharacter crosses the bounds of the TutorialArea.

Basic Flow`

1. The PlayerCharacter crosses into the bounds of the collider attached to the TutorialArea as a trigger.
2. The TutorialArea detects this collision and checks that the associated tutorial has not been triggered, and calls the GameManager to toggle the tutorial overlay.
3. The GameManager sets the game world time to 0, and sets the tutorial overlay to active within the scene.
4. The TutorialArea then loads the details of the tutorial to be displayed.
5. The TutorialArea populates the tutorial overlay with the contents of the associated tutorial.
6. The TutorialArea updates the tutorial to 'Triggered'.
7. The Player closes the TutorialOverlay.
8. The GameManager sets the TutorialOverlay to disabled, and the game world speed to 1.

Alternative Flow

1. The PlayerCharacter crosses into the bounds of the collider attached to the TutorialArea as a trigger.
2. The TutorialArea detects this collision and checks that the associated tutorial has been triggered, and thus returns without performing any other action.

Termination: The Player closes the tutorial overlay.

Special Requirements:

Preconditions: The Player must be in one of the Scenes that utilizes the third person camera perspective and the PlayerCharacter must pass the bounds of a TutorialArea

Post-conditions:

Success Conditions

1. **In the event the PlayerCharacter is entering an untriggered TutorialArea and closes the tutorial overlay**
    a. The TutorialArea is marked as triggered.
    b. The GameManager correctly sets the game world time to 1 and sets the tutorial overlay to disabled within the scene.
2. **In the event the PlayerCharacter is entering an untriggered TutorialArea**
    a. The TutorialArea correctly detects that it has been triggered, and does not perform any further action, leaving the Player uninterrupted.

Failure Conditions:

3. **In the event the PlayerCharacter is entering an untriggered TutorialArea and closes the tutorial overlay**
    a. The TutorialArea is not marked as triggered.
    b. The GameManager does not correctly sets the game world time to 1 or leaves the tutorial overlay enabled within the scene.

4. **In the event the PlayerCharacter is entering an untriggered TutorialArea**
    a. The TutorialArea incorrectly detects that it has not been triggered and tries to go through the basic flow, interrupting the Players user experience with a tutorial they have already seen.

Data Requirements

This project utilizes the local file system to save player data, in order to have persistence between sessions. The game transposes the details of the players statistics and progress into binary, and then saves that binary data to a file on the user's machine. The project does not need to implement any specific database implementation, or require the user to have any database software installed on their machine in order to function, which is good for release purposes, as it does not exclude users who do not have the knowledge or comfort with database software from downloading and playing the game.

Non-Functional Requirements

Performance Requirements

Due to video games being a predominantly visual medium, being able to achieve and maintain a reliably high framerate and visual fidelity is an incredibly important consideration when designing and building a game. Because of this, I set to establish a performance requirement of, at minimum, a consistent 60 frames per second.

https://www.youtube.com/watch?v=pfiHFqnPLZ4

The above link provides a very good visual representation of the difference between a high a low frame per second animation.

To ensure that I would be able to maintain a good frame per second performance for my game, one of the main sources of information and guidance I utilized was the Unity documentation for best practices.
https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity.html

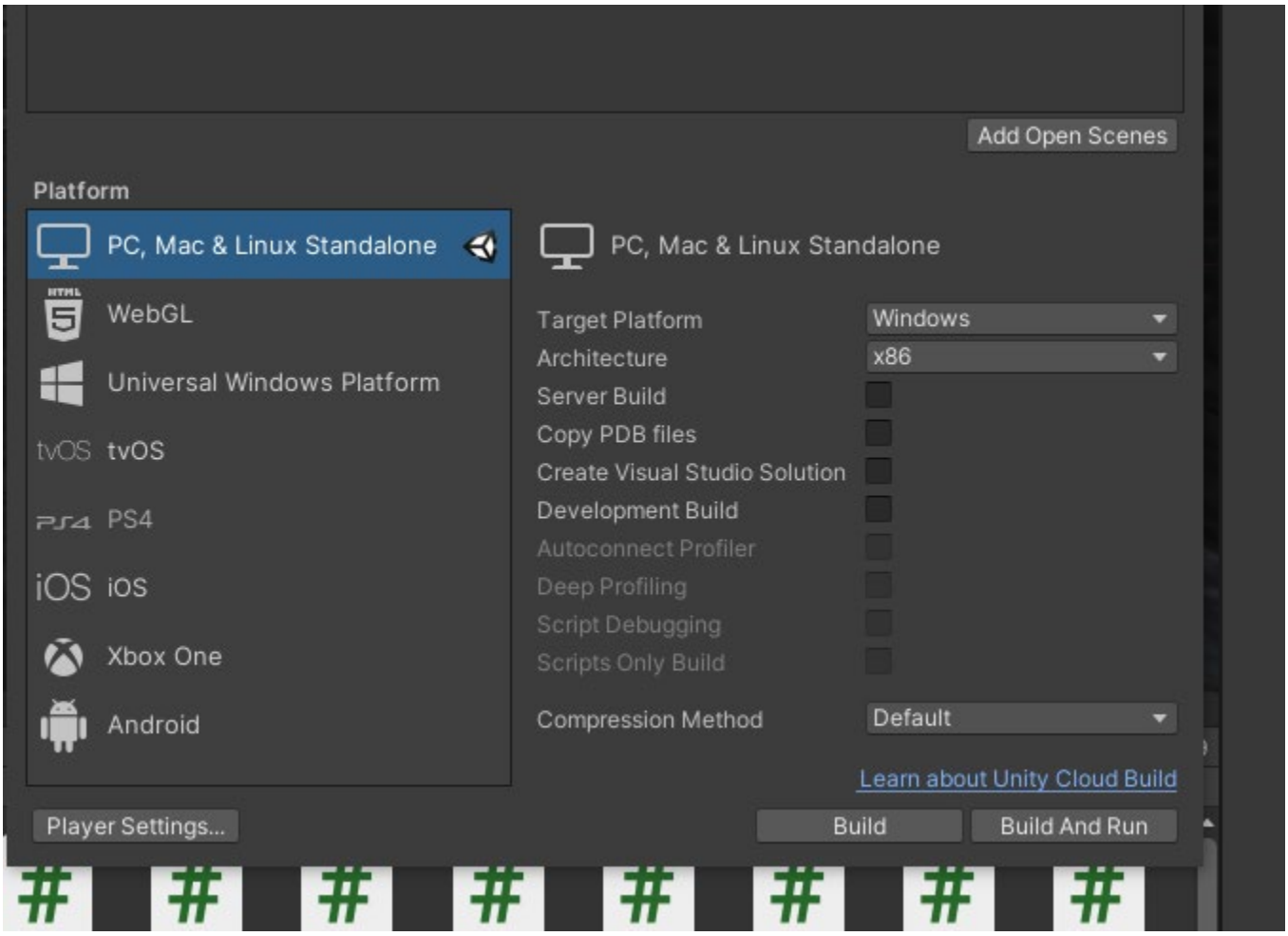
It outlines many of the main contributors to project bloat, memory leaks or memory hogs, and general hits to performance. For example, early in development, I was using the Resources folder within my project to store miscellaneous files and assets, believing it to be a general use folder. By reading through the documentation, I realized that the Resources folder within a Unity project actually has some special attributes, related to the Unity Resources API.

- "Use of the Resources folder makes fine-grained memory management more difficult
- Improper use of Resources folders will increase application startup time and the length of builds
    - As the number of Resources folders increases, management of the Assets within those folders becomes very difficult
- The Resources system degrades a project's ability to deliver custom content to specific platforms and eliminates the possibility of incremental content upgrades
    - AssetBundle Variants are Unity's primary tool for adjusting content on a per-device basis", Unity Technologies (2020)

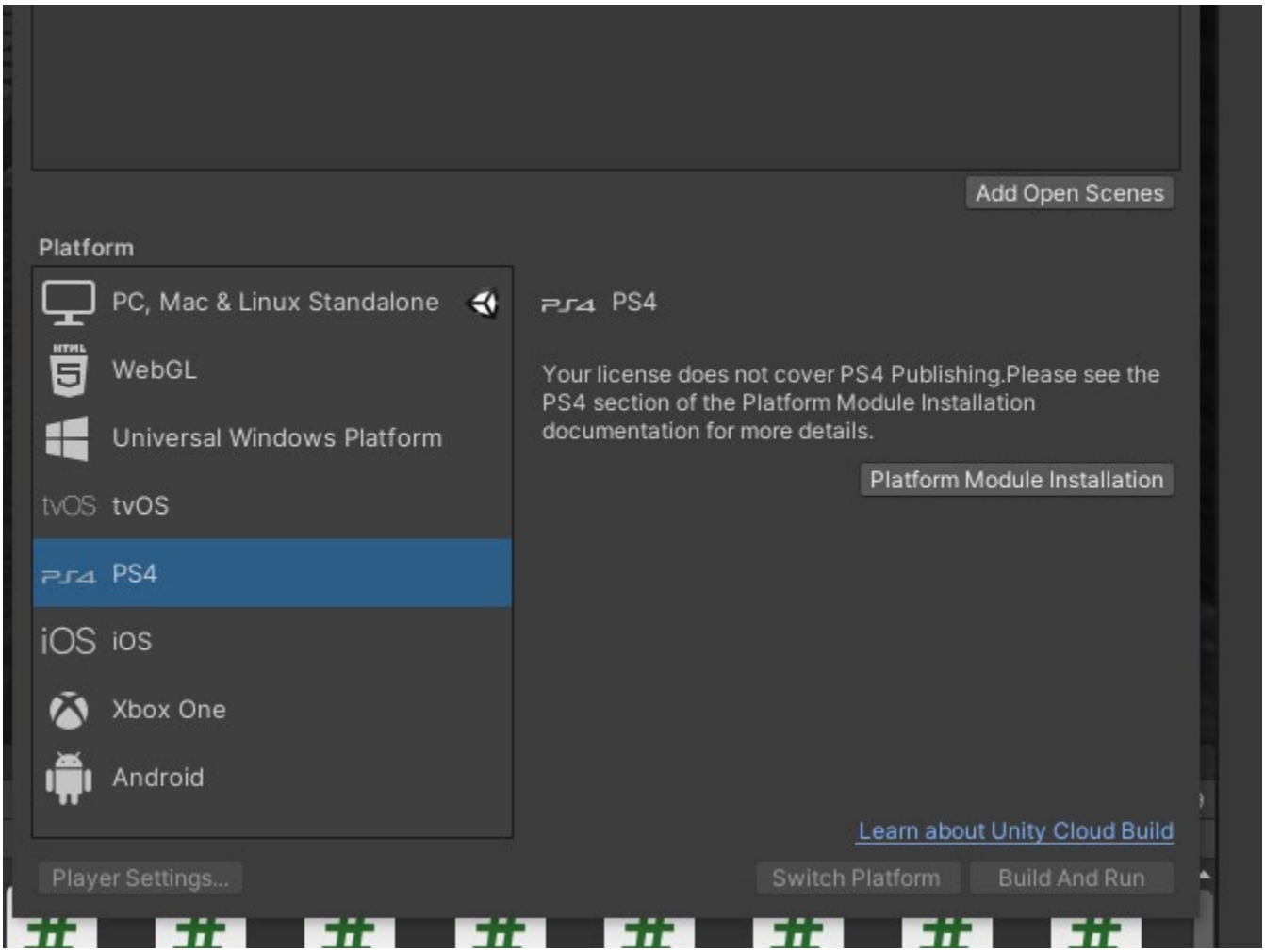
This documentation also helped to increase the load times when the scene was being transition, for example when the Player selects New Game on the Main Menu screen, the load times are quite short, a benefit that could have been hampered had I not been referring to the documentation.

Operating System Requirements

The user must have access to a desktop machine (Linux, Windows, or Apple) to play this game. This is due to the licensing for Unity the various compilation services that are offered by Unity. By this I mean that Unity offers different services to compile a project within Unity to run on different operating systems.



You can't build a game to run on every platform in one button, you need to built it for each platform in sequence, and Unity does not offer the license to use the different build services by default. They offer some for free but in many cases, you must purchase the license in isolation, or subscribe to one of the higher tiers of Unity subscription, which were too financially severe to use for this project.



Security Requirements

Security requirements were not a significant concern for this project. The most pressing area of security in this project is the files that players save their progress and characters to. This is because many games that allow for local file saves, the saved games or characters are written to the files in a format that's easy to write, map, and parse, such as JSON or XML. This makes it easy to write and store the saves, but it also allows for the player to easily modify the files on their machine, as all that would be required on their end is a basic text editor, even a tool as basic as notepad or vim. This can be a significant issue in online multiplayer games, as the ability for players to simply modify their characters in a text editor would potentially destroy the game balance and negatively impact the perception and reputation of the game.

In a single player game such as this however, that threat is nowhere near as significant. A player manipulating their character here only upsets the balance of the intended progression but has no knock-on impact on other players. In saying that however, I did implement a solution to write the players character to a file, while limiting their ability to modify their characters, which I will discuss more in the implementation section.

Reliability Requirements

The reliability of a game is very important, particularly in a game where a key component is progressing through a story and upgrading a character. If a player has made a lot of progress and the game were to fail in some capacity, and they lost their progress due to the games lack of reliability, it would drastically hamper their enjoyment of the game. To ensure the reliability and robustness of this game, I utilized various testing methods to try my best to cover as much of the game as I could to weed out as many bugs and issues as I could.

Maintainability Requirement

The ability to maintain and scale the project in the future is an important one, as I intent to continue working on this game going forward as it is a project I am passionate about. To the end of maintainability, I have tried to make my code as readable as possible for myself, so that in the event that I need to go back a patch an error in a system or expand a system to include new features or requirements, I can easily find the part of the code that is related to that fix or updated and start quickly to make the changes that are necessary.

For the purposes of making more content for the game, I have brought in some of the more popular content development pipelines and tools that exist for Unity. For instance, I made the decision to invest in the Dungeon Architect toolkit. This toolkit provides some extremely helpful and timesaving functions for level building Unity. Unfortunately, I implemented these relatively late in the project, so I was unable to utilize it to the most potential for the final version provided with this report.

Reusability Requirement

Reuse of code is a harder ideal to maintain in my experience with game development. Due to the nature of having many disparate classes and entities interacting with one another, it makes it hard to enforce inheritance, or to be able to rely on multiple classes being able to have the same method implementation. Games also require having more scripts and events that are one shot, that will occur once and the not again. I tried my best to make my code as general as possible to cut down on the amount of reused code and keep development time down.

User Requirements

The requirements for a user to run this game are as follows

Operating system	**Windows:** 7 SP1+, 8, 10, 64-bit versions only **macOS:** 10.12+ **Linux:** Fixed at: Ubuntu 16.04, 18.04 and CentOS 7 Server versions of Windows and OS Xare untested.
CPU	SSE2 instruction set support.
GPU	Graphics card with DX10 (shader model 4.0) capabilities.
Storage	400mb

Environmental Requirements

In this section I will detail the requirements I had to develop this project.

- **Windows PC:** A windows machine to Unity & Visual Studio
- **Multiple Windows Devices:** For testing the project on different hardware configurations.
- **Unity:** The primary technology for the project. Unity was used to develop and integrate the project.
- **Visual Studio:** Visual Studio was used as the dedicated IDE for the C# scripts used for the project.
- **Blender:** Blender was used to sculpt and manipulate some 3d objects for the project.
- **MagickaVoxel:** MagickaVoxel was used to create Voxel models for the project.

- **Internet Access:** Internet access was required to download and install the various software packages required, as well as many if the assets used in the project.

Usability Requirements

In order to play the game and get the most out of the project, the player must be informed of some of the less clear mechanics in the game, such as the stealth mechanics. To this end, a tutorial section was implemented to provide messages to the player that will inform them of the controls and the specifics of the mechanics.

Design & Architecture

The game was built in the Unity game development environment, using Visual Studio as the dedicated IDE. The primary development language used was C#.

Implementation

The aim of the heading is to explain and highlight the technologies and solutions used in the development of this project. I will be explaining the way I set up the most important and significant Game Objects within the scene, and what the various attached components are doing and why. I will explain it in this way as it is difficult to explain the actions of the scripts without the context of the game object to which it is attached.

GameManager

The game manager script is a script used to manage the state of the scene. It contains a lot of code to handle events such as pausing the game, handling the interactions between the game objects in the scene and the UI, and maintaining the connection between the controllable PlayerCharacter, and the static PlayerStats object. The GameManager adheres to the singleton pattern. The reason for this particular structure is that I wanted to ensure that there was only ever one GameManager object within the scene, as many of the other scripts and components reference it. If there had been multiple instances of the GameManager component in the scene, it could lead to conflicts and issues maintaining the integrity of the state.

To set up the GameManager as a Singleton, the class has a public static GameManager member variable call instance. In the Awake method, that instance member variable is set to the keyword this. This gives the GameManager class a static reference to the GameManager component in the scene.

```
public static GameManager instance;
```

```
Unity Message | 0 references
private void Awake() {
    GameManager.instance = this;
}
```

The Awake function is a function that is included in the parent class MonoBehaviour, which every unity script inherits from by default. This function the first method called on all MonoBehaviour objects when they are initialized or set to enabled for the first time. This makes it the best option for populating static or class variables like this.

"Unity calls Awake only once during the lifetime of the script instance. A script's lifetime lasts until the Scene that contains it is unloaded. If the Scene is loaded again, Unity loads the script instance again, so Awake will be called again. If the Scene is loaded multiple times additively, Unity loads several script instances, so Awake will be called several times (one on each instance)." Unity Documentation

An issue that arose while developing the GameManager script came from the different order of execution between the Awake and Start functions. The PlayerController script users the reference to the GameManager to help update the UI, to refresh the current health bar for example. The issue was that I was setting GameManager.instance = this in its Start function, but the PlayerController was trying to reference it in its Awake function. Because of the order of execution, Awake executing for all GameObjects before any Start methods are called, this was leading to a Null Pointer Exception in the PlayerController. This was fixed by moving the GameManager.instance = this into the Awake method.

PlayerController

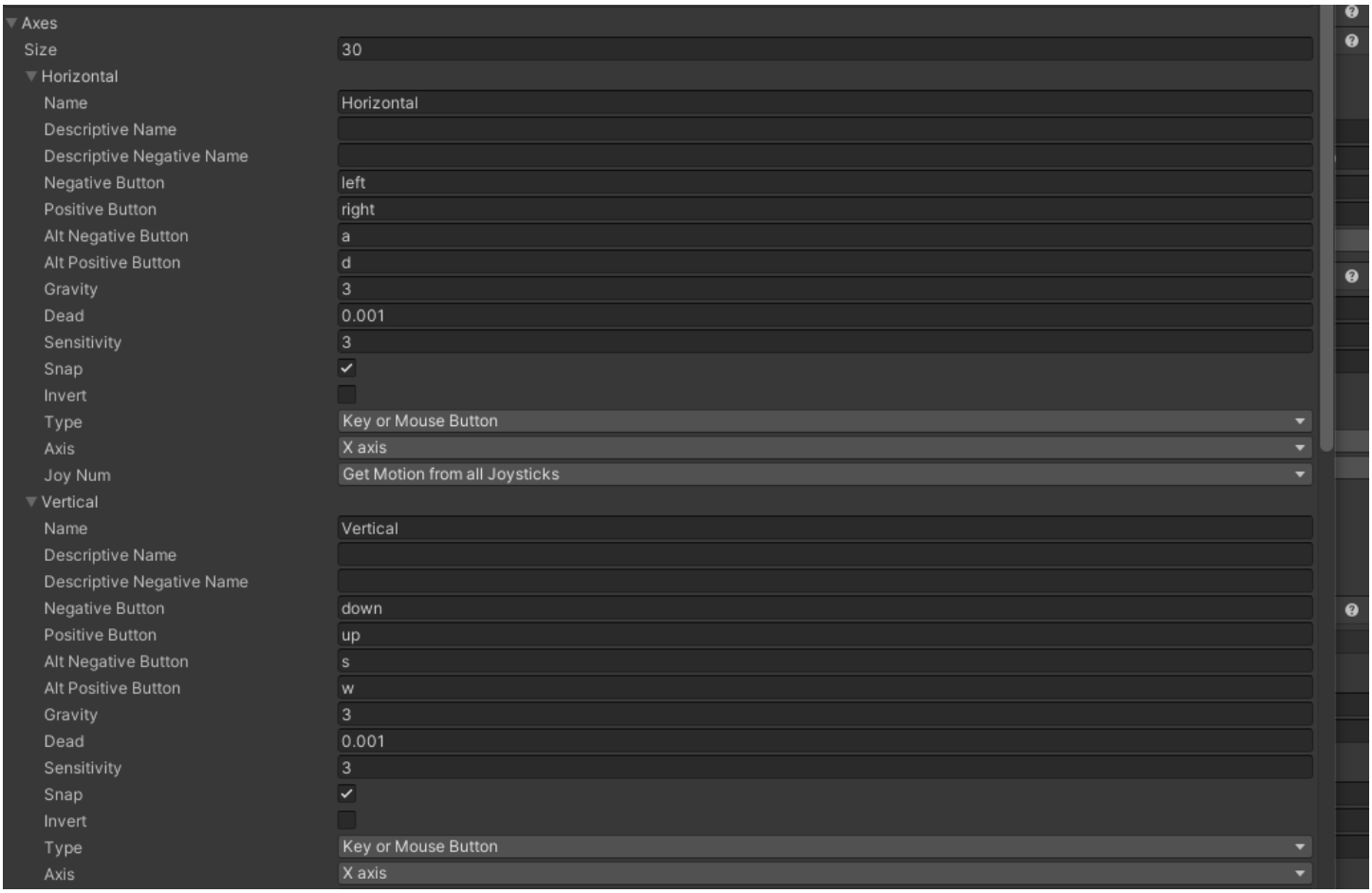
The Player was the object in the scene that handles most of the second by second interaction between the game and the player. It is the game object that handles user input for movement, camera rotation, and weapon attacks.

The main component attached to the PlayerController, which was a C# script written by me that performed most of those actions within the Update method. The Update function is a function that that is called every frame, which makes it perfect for code that is waiting for player input, as any frame where the player is making an input, such as a mouse movement or keystroke, will be captured and run within the Update function.

For example, the code for moving the player in response to key strokes in contained within the aptly named movePlayer method, which is called from the Update method.

```
float xMovement = Input.GetKey(KeyCode.LeftShift) ? Input.GetAxis("Horizontal") * (moveSpeed * sprintSpeed) : Input.GetAxis("Horizontal") * (moveSpeed);
float zMovement = Input.GetKey(KeyCode.LeftShift) ? Input.GetAxis("Vertical") * (moveSpeed * sprintSpeed) : Input.GetAxis("Vertical") * (moveSpeed);
bool isRunning = Input.GetAxis("Vertical") !=  0f || Input.GetAxis("Horizontal") != 0f;
anim.SetBool("Running", isRunning) ;
anim.SetInteger("RunningXDir", (int)Input.GetAxis("Horizontal"));
anim.SetInteger("RunningYDir", (int)Input.GetAxis("Vertical"));
if (Input.GetButtonDown("Jump")) {
    tryJump();
}
Vector3 dir = transform.right * xMovement + transform.forward * zMovement;
dir.y = rigidbody.velocity.y;
rigidbody.velocity = dir;
```

The first two lines in the above screen shot are capturing the inputs, by listening for input on the horizontal or vertical axis. The keyboard values for these axes are set in the Project Settings menu.



The return value from the GetAxis method is a float value, between -1 and 1, depending on the whether the player presses the negative or positive button respectively. This makes the GetAxis method double useful, because not only can you get the input from the player, you can use that value to directly influence values like the player speed, player direction, or what animation to play.

To control the main camera, the camLook method is used, which is also called from the Update method.

```
1 reference
void camLook() {
    float yLook = Input.GetAxis("Mouse X") * lookSensitivity;
    rotX += Input.GetAxis("Mouse Y") * lookSensitivity;
    rotX = Mathf.Clamp(rotX, yRotationClamps[1], yRotationClamps[0]);
    mainCamera.transform.localRotation = Quaternion.Euler(-rotX, 0, 0);
    transform.eulerAngles += Vector3.up * yLook;
}
```

This method is similarly using the GetAxis method to listen for input on the players mouse X and Y axes. This lets me change the rotation of the main camera around the player character in the scene. An early unsuccessful attempt at capturing the mouse movement to set the camera used the input to move the camera, giving it a new position. This proved to be very jarring and disorienting, as even the slightest movement of the move led the camera to move very far very quickly.

Another issue that arose with the camera movement was clipping. Because the camera doesn't have a physical body within the scene, it would pass right through the objects that made up the game world, like the walls and ceiling. To solve this issue, I added a collider component to the main camera. This solved that problem but introduced another one. Because colliders are by default physical objects adding one to the main camara caused it to collide with the other game objects that had colliders which would in turn move the player because the player was the parent of the camera. The camera acted almost the fulcrum of a level whenever it collided with the environment.

The solution to this problem was to make the camera collider a trigger, a write specific code to change the cameras position in response to the environment colliders entering the trigger. This code was written in the ThirdPersonCameraControl script.

When an environment collider enters the camera trigger, it calls the cutCameraDifference method. This method finds the position of the other collider and that colliders width, and subtracts it from the cameras global position. It then uses that resulting value to populate the Z value in the new local position of the camera. It has to use the local position, because it is setting the new position of the camara in relation to the player, not the scene.

```
2 references
void cutCameraDifference(Collider otherCollider) {
  // Debug.Log(transform.position.z);
    Transform parentRransform = gameObject.transform.parent.transform;
    Transform otherObject = otherCollider.transform;
    float newZ = (otherObject.position.z - transform.position.z) - otherObject.localScale.x;
    newZ = newZ > 0? newZ * -1: newZ;
    transform.localPosition = new Vector3(transform.localPosition.x, transform.localPosition.y, newZ);
    lastEnteredColliderTime = Time.time;
}
```

To reset the camara position to its original when the camara is no longer close to the environment, the Update method lerps the camera local position to its original values after the camera has not been in contact with the environment for 3 seconds.

```
if (Time.time - lastEnteredColliderTime >= colliderTimer) {
    //transform.localPosition = new Vector3(transform.localPosition.x, transform.localPosition.y, -2.5f);
    transform.localPosition = Vector3.Lerp(transform.localPosition, new Vector3(transform.localPosition.x, transform.localPosition.y, -2.5f), lerpTime / 2f);
    lerpTime += Time.deltaTime;
} else {
    lerpTime = 0;
}
```

It uses lerp to give the transition a smooth flow, as opposed to an abrupt single change to the camera position.

Weapon & Projectile

The Weapon and Projectile components are the components that handles the functionality of weapons within the game. They are written in such a way that they can be applied to both the players weapons, and the weapons used by the enemy, rather than having two separate sets of scripts to handle the weapons used by enemy characters and weapons used by the player.

Weapons are broken down into two types, Melee and Projectile, determined by the enum value chosen on the Weapon game object in the scene. Depending on the weapon type value, different actions happen when the weapon is activated.

When a Melee weapon is fired, the weapons swing animation is triggered, and the capsule collider attached to the weapon is set to active, and a coroutine is called. This is done so that when the collider, which is set as a trigger, intersects with another game object, it can detect that intersection. When the collider intersects, it checks the tag of the other game object. If the tag is different from the weapon character type value, which is used to identity if the weapon is used by the player or and enemy, the takeDamage method is called on the other object. Then the coroutine that was called executes, disabling the capsule collider again, so that the weapon is effectible inert until it is activated again.

```
Unity Message | 0 references
private void OnTriggerEnter(Collider other) {
    if (Time.time - lastHurtTime >= shootRate) {
        string type = "";
        switch (charType) {
            case CharacterType.ENEMY: type = "Enemy"; break;
            case CharacterType.PLAYER: type = "Player"; break;
            case CharacterType.NPC: type = ""; break;
        }
        Debug.Log(other.name + " layer : " + other.gameObject.layer);
        if (!other.CompareTag(type) && other.gameObject.layer ==10) {
            Debug.Log(other.name + " layer : " + other.gameObject.layer);
            other.gameObject.GetComponent<Damagable>().takeDamage(meleeDamage);
        }
        lastHurtTime = Time.time;
    }
    return;
}
```

In the case that the weapon is a projectile weapon, the script instead looks to the Projectile value attached to the Weapon object. The Projectile is a prefab object with a Projectile component attached to it. The Weapon component gets its muzzles transform position and instantiates a new Projectile. It then determines the direction to launch that new projectile. If it is an enemy, if determines the direction the player is in from the muzzle, and launches the projectile in that direction, whereas if it is the player, it will launch forward from the muzzle.

```
2 references
private void fireProjectile() {

    --curAmmo;
    // GameObject projObj = Instantiate(projectile, muzzle.position, muzzle.rotation);
    GameObject projObj = ammoPool.GetGameObject();
    projObj.transform.rotation = muzzle.rotation;
    projObj.transform.position = muzzle.position;
    if (playerTarget == null) {
        playerTarget = FindObjectOfType<PlayerController>();
    }
    if (charType == CharacterType.ENEMY) {
        Vector3 playerDir = (muzzle.transform.position - playerTarget.transform.position).normalized;
        projObj.GetComponent<Rigidbody>().velocity = playerDir * projectileSpeed;
    } else {
        projObj.GetComponent<Rigidbody>().velocity = muzzle.forward * projectileSpeed;
    }
    //set the velocity

    if (!gameObject.CompareTag("Enemy"))
        GameUI.instance.updateAmmoText(curAmmo, maxAmmo);
}
```

EnemyController

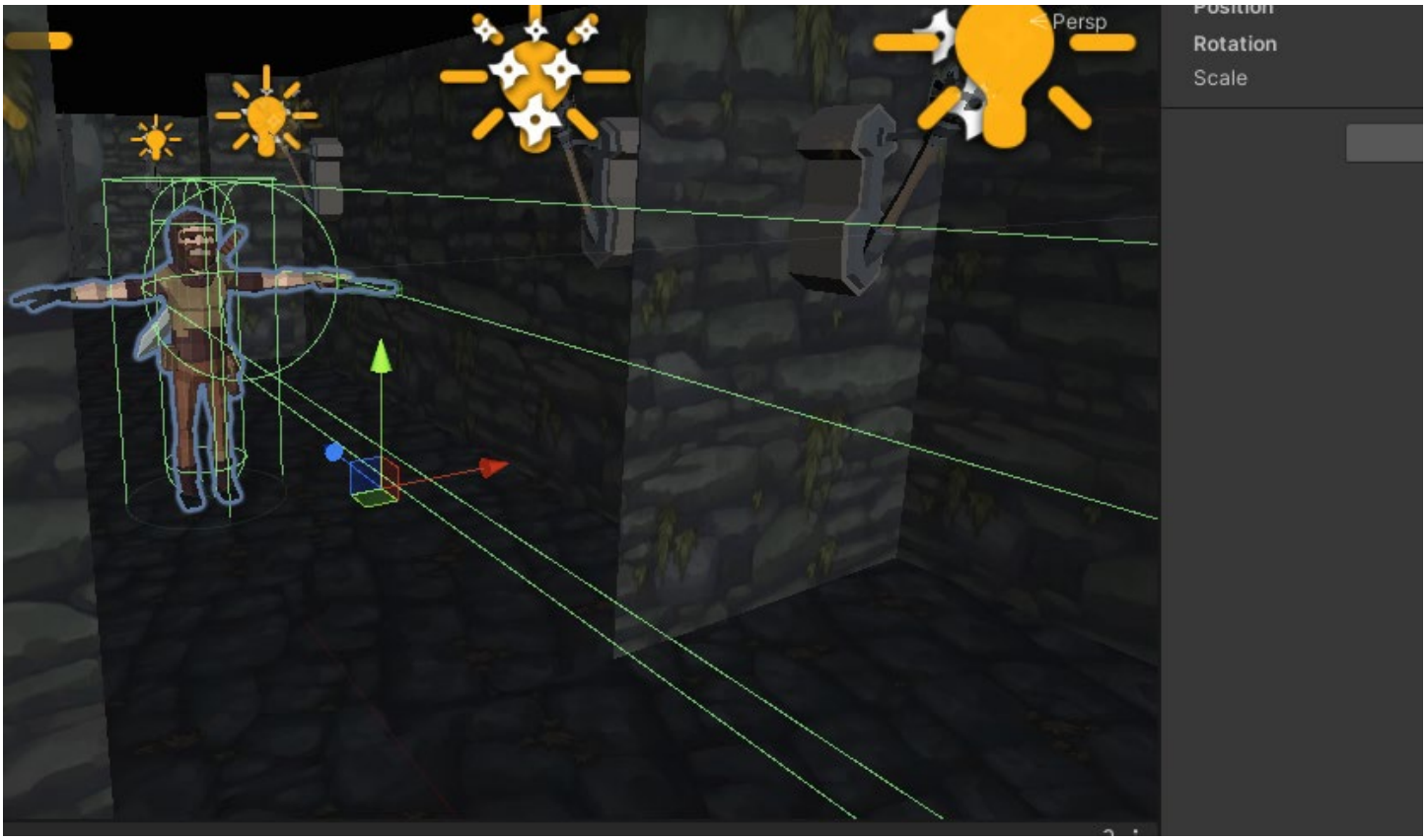
The EnemyController component handles most of the code relating to the behaviour of the AI enemies in the game. It handles moving them, playing their animations, firing their weapons, and integrates their associated Viewport objects. The Viewport component is what is used to determine whether the enemy has spotted the player or not.

Enemies have two behaviors when they haven't spotted the player. They will either patrol between a set of patrol points in the scene, or stand guard in set areas. The enemy moves between these patrol points using the NavMeshAgent component. This component works in conjunction with the Unity AI navigation mesh that I created for each scene to determine the route the enemy would follow to reach its destination. When it reaches its destination, it simply sets the new destination to the next element in the collection of patrol points.

```
void patrol() {
    if (patrolPoints.Length == 0) {
        return;
    }
    Debug.Log("Patrolling");
    agent.destination = patrolPoints[nextPatrolPoint].position;
    nextPatrolPoint = (nextPatrolPoint + 1) % patrolPoints.Length;
}
```

The enemy will repeat this behaviour until the player is spotted, at which point it will give chase. The way the AI determines the route to the player is using the AI NavMesh. It references the mesh, and determine the way to the player, avoiding any obstacles in the enemies path. This path information is updated every frame that the enemy is chasing the player, meaning that they will always have up to date information on how to track the players movements.

To give the sense that the enemies aren't just automatons who will chase the player until either they or the enemy is killed, I implemented a system where if the enemy hasn't scene the player within certain time frame, it will exit its alert status and return to its patrol. This is done by using the ViewportController script, which I wrote to detect if the enemy has scene the player. This ViewportController is a component that is attached to the enemy and position in front of the enemies face, to give them and artificial eyeline.



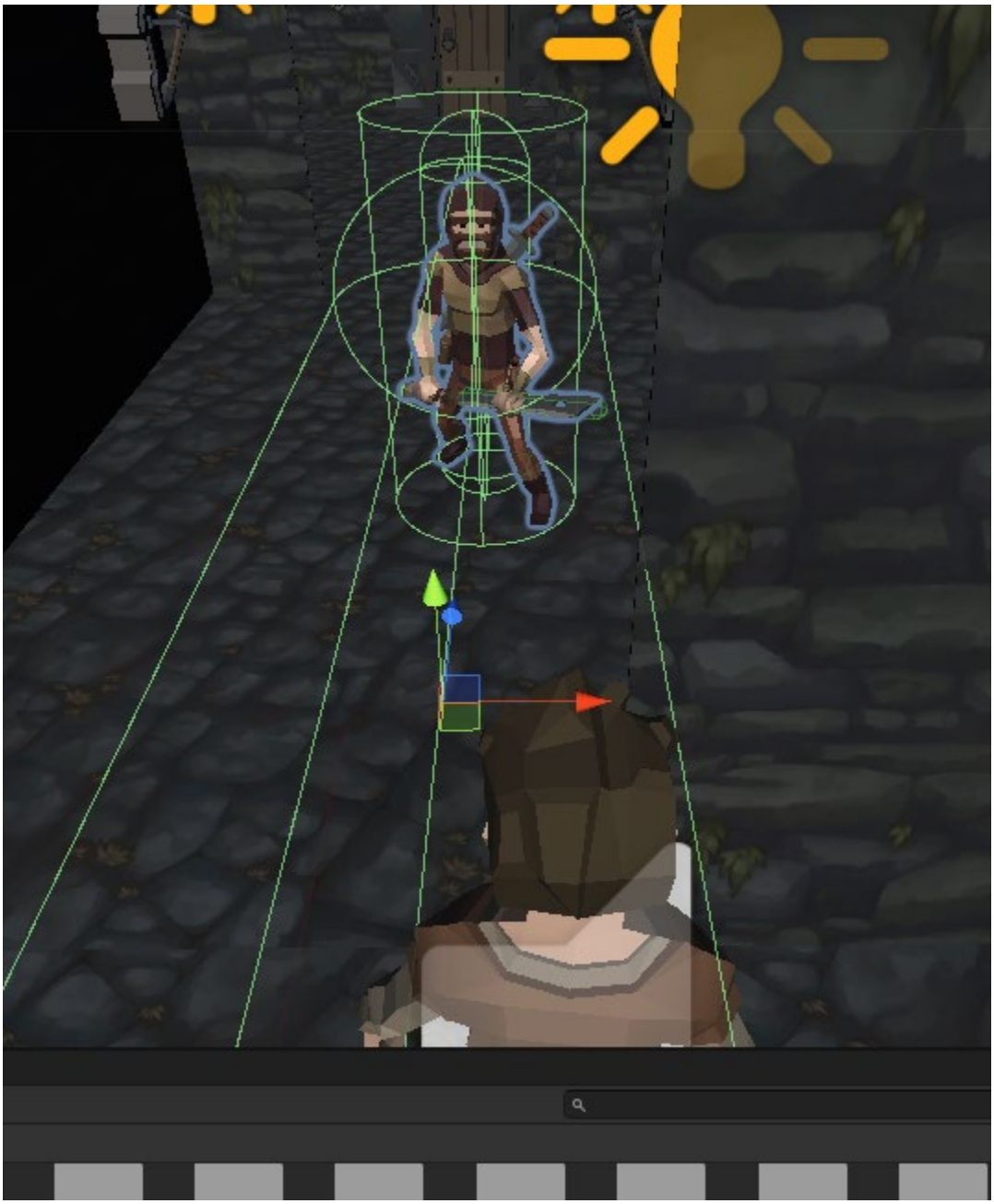
This object is then giving a capsule collider that is set to be a trigger. Then the player enters the bounds of this trigger, it sets the enemies alert value to true, and the enemy begins to chase the player. This code took several iterations before it reached the point that it was ready. The initial versions worked, with the enemy detecting the enemy and giving chase, but the enemies 'vision' of the player was not obstructed by the environment. They enemy could see the player through walls. This is because the OnTriggerEnter method does not have any context for the order or relation in which objects enter its bounds. It just activates executes whenever something enters. Because of this, if there was a wall in between the enemy and the player, the method would just see them as two completely disconnected events. In order to get around this issue, I first implemented a RayCast. This would cause the ViewportController to emit a Ray directly out in front of the enemy when the Player passed into the bounds of the trigger. My idea here was that if the Ray hit a piece of

the environment first, I could return from the method, as my logic was that the Ray would hit the player first if their wasn't any terrain in the way.

```
Ray targetRay = new Ray(new Vector3(parentTransform.position.x, parentTransform.position.y + 1f, parentTransform.position.z) + forwardDirection, forwardDirectio
for (int i = 0; i < connections.Length; ++i) {
    Debug.Log(connections[i].collider.gameObject.name);
    if (connections[i].collider.gameObject.CompareTag("Player")) {
        // Vector3 playerDir = (parentTransform.transform.position - connections[i].collider.gameObject.transform.position).normalized;
         Debug.DrawRay(new Vector3(parentTransform.position.x, parentTransform.position.y + 1f, parentTransform.position.z), playerDir, Color.green,1f);
    }
}
if (!connections[1].collider.gameObject.CompareTag("Player"))
    return;
```

This would end up not being a workable solution. The problem was the a RayCast has no width or thickness, so it would only ever collide with the player if they were in the dead center in front of the enemy, so I had replaced my initial problem with another one.

My next solution was to use a SphereCast. With a SphereCast, I could set the radius of the cast, and match it to the ViewportController trigger bounds. With this, I could detect the player with a SphereCast, even if they were at the edges of the trigger bounds. This looked like a good solution, but after some playtesting I noticed an issue with it. While the logic behind the fix was sound, its result on the actual gameplay was disappointing. The problem was that I was still looping through all the objects the SphereCast hit as it was cast out, which means that any environment objects would be hit regardless of how little of them was hit, like the player. That lead to situations like the below.



In this situation, the enemy should be reasonably expected to spot the player and give chase. But the slightest edge of the wall is in between them, and it is being hit by the SphereCast first, and so the method returns without setting the enemy to alert.

The third solution I used was a relatively simple modification of my original RayCast idea. I would cast out the Ray, and see if any terrain was hist before the player, but this time I would not send the RayCast out just directly ahead, I would use find the Player, and use the transform position of both the Player and the Enemy to determine what direction to cast the Ray. By doing this I was able to create a good facsimile of the enemy having real perception. They wouldn't be able to see the player through walls, but they would be able to see the player if their cover wasn't good enough.

```
private void OnTriggerEnter(Collider other) {
    if (other.CompareTag("Player")) {
        Transform parentTransform = transform.parent;
        //Vector3 forwardDirection = transform.TransformDirection(Vector3.forward);
        Debug.Log(transform.localScale.y);
        float length = gameObject.GetComponent<CapsuleCollider>().height;
        Debug.Log(length);


        Vector3 playerDir = (parentTransform.transform.position - other.transform.position).normalized;
        RaycastHit[] connections = Physics.RaycastAll(new Vector3(parentTransform.position.x,parentTransform.position.y+1f, parentTransform.position.z), -playerDir, length);
        for (int i = 0; i < connections.Length; ++i) {
            Debug.Log(connections[i].collider.gameObject.name);
            if (connections[i].collider.gameObject.CompareTag("Player")) {
                // Vector3 playerDir = (parentTransform.transform.position - connections[i].collider.gameObject.transform.position).normalized;
                 Debug.DrawRay(new Vector3(parentTransform.position.x, parentTransform.position.y + 1f, parentTransform.position.z), playerDir, Color.green,1f);
            }
        }
        if (!connections[1].collider.gameObject.CompareTag("Player"))
            return;
        lastSpottedTime = Time.time;
        attachedEnemy.spottedPlayer = true;
    }
}
```

GameUI

The last of the components that I will explain is the GameUI component. The GameUI component is the component used to handle the various UI elements in the Game, such as the health meter, ammo counter, and the various UI menus that are displayed, such as the pause and quest interfaces.

The most notable piece of code within the GameUI from my perspective was the toggleInventory method. This method is used to populate the inventory screen with the various pieces of equipment the player has acquired. It does this by reading the list of equipment attached to the PlayerController instance in the scene. It them enables the inventory screen, and creates a series of new objects of the inventory object UI prefab. This prefab has text views for the name and details of the weapon, a sprite image for the weapon, and a button to equip the weapon.

```
1 reference
public void toggleInventory(List<GameObject> weapons) {
    GameManager.instance.toggleQuest();
    equipmentScreen.SetActive(true);
    foreach (GameObject obj in weapons) {
        Weapon wep = obj.GetComponent<Weapon>();
        GameObject newSpan = Instantiate(weaponPrefab, weaponSpan.transform);
        TextMeshProUGUI[] textBoxes = newSpan.GetComponentsInChildren<TextMeshProUGU
        textBoxes[0].SetText(wep.weaponName);
        newSpan.GetComponentInChildren<Image>().sprite = wep.weaponIcon;
        Weapon spanWep = newSpan.AddComponent<Weapon>() as Weapon;
        spanWep = wep;
        Debug.Log(spanWep);
        Button button = newSpan.GetComponentInChildren<Button>();
        button.onClick.AddListener(delegate { equipWeapon(spanWep); });
    }
}
```

References

Unity Tecnologies - Assets, Resources and AssetBundles – 2020, https://learn.unity.com/tutorial/assets-resources-and-assetbundles#

Unity Documentation – MonoBehaviour.Awake() – 2020, https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

Addendum

Proposal

National College of Ireland

Project Proposal

Unconfirmed Fantasy Themed Adventure Game

December 22nd 2020

Bachelor of Computing

Software Engineering

2020

Liam Duggan

X17148529

X17148529@student.ncirl.ie

Contents

1.0 Objectives 35

2.0 Background 1

3.0 Technical Approach 1

4.0 Special Resources Required 1

5.0 Project Plan 1

6.0 Technical Details 1

7.0 Evaluation 1

8.0 Invention Disclosure Form (Remove if not filled) 1

2.0 Objectives

My objective with this project is to create a fantasy themed game built in the Unity game engine.

The game will be split into two primary gameplay loops, the level gameplay, and the Overworld gameplay.

The level gameplay will take the form of a first person shooter, inspired by the gameplay of older first person shooters games such as Duke Nukem 3d or Shadow Warrior. The player will progress through levels, attempting to defeat the enemies within, and complete their quest objectives.

The secondary gameplay loop will take place on the Games overworld. In these segments, the player will be able to travel the world of the game, visiting various towns or locales. In these areas, the player will be able to interact with other characters in the game to pick up new quests, get clues to the weaknesses of certain monsters, or to purchase new items and equipment.

On the Overworld, the Player will also be able to level up their character and spend skill points from levelling up or completing mission objectives. These points can be spend to unlock new magic abilities or to upgrade the player characters statistics, such as their total health or maximum speed.

3.0 Background

In the last five years there has been a resurgence of a style of First-Person Shooter game that reached an extreme level of popularity and acclaim in the mid-90s to early 2000s. Games such as Dusk, Project WARLOCK, and AMID EVIL have all been met with critical and commercial. This was of great interest and excitement to me, as I was a great fan of those old first-person shooters, and I find myself less and less interested in the more mainstream first-person shooters, such as Battlefield or Call of Duty.

This has lead me to be very interested in creating my own old school shooter, both from a position of personal interest and passion, and a believe that the resurging genre has openings to capitalise on by releasing a product that has a similar gameplay feel, but adds additional mechanics and options, and takes place in a wildly different setting to what is currently available.

4.0 Technical Approach

This project will be undertaken using the AGILE methodology, using two week long sprints, with the aim to have each sprint end with a prototype that has one notable addition or fix from the end of the previous sprint phase. My current aim is to have the follow a very basic flow with regards to how each area in the game is designed. That flow is as follows

1. Area General Design (Colour Schemes, tone, lore)
2. Enemy Design - Appearance
3. Enemy AI Development
4. Map building (Pen & Paper)
5. Map Implementation in Unity
6. Integration

Step 3 will be done using a Debug level, just so I have a place to test AI or attacks without needing a full mapped out and developed map.

The assets for enemies will primarily be done using Magica Voxel. This is as time permits. As the project goes on, I may have to utilise third party asset packs to deepen the depth of the enemy roster.

5.0 Special Resources Required

A number of external Unity assets packs will be required in the completion of this project, as the time requirement for manually generate those assets will be too severe to generate them entirely by myself, particularly in the area of music or sound effect assets, as I currently do not possess or have access to any real sound mixing or recording facilities. The asset packs currently include 'The Archive Volume 1' and 'The Archive Volume 2' from Dark Fantasy Studios at http://darkfantasystudio.com/

6.0 Project Plan

Gantt chart using Microsoft Project with details on implementation steps and timelines

7.0 Technical Details

The primary coding language that will be used is the C# language, as this is the language that Unity uses. A significant amount of work will also be done using Blender and MagicaVoxel, in order to generate assets for in game use.

8.0 Evaluation

This project will require a significant amount of testing, both in a functional and technical sense, but also for play testing and 'look and feel purposes'. The functional tests will be run by myself and a handful of volunteer testers to determine whether all functional requirements of the project have been met.

The technical tests will be done by myself, using the inbuilt analytic tools present in Unity. This testing will be done to ensure that the performance of the game is maintained at a high level, e.g. higher than 60fps using industry standard performance benchmarks for common user hardware configurations.

The playtesting tests will be performed initially by myself to get a baseline of the way the game will play, e.g. level of difficulty, speed that the game progresses, etc. In the later parts of development however, that focus will shift to volunteer play testers.

Liam Duggan

Reflective Journals

Introduction

Hello, my name is Liam Duggan, and this journal will be documenting my process and work on my final year project for my Bachelors Degree in Computer Science(Software Engineering). I am currently in full time employment as a Software developer with an Irish stockbroking company and I have been working from home due to the restrictions imposed by COVID 19 this year.

My goal with this journal is to provide a detailed and reflective account of my progress through this project over the course of the next school year.

My hope for this journal is that it will allow me to counter some of the weaknesses I have had with project work in previous years of this course. These weaknesses would be

1. Time Management
2. Documentation
3. Explaining my work

1. I have always had problems with time management in projects because I generally work with peaks and valleys in activity on a project. I have peaks were I will put in significant work into a project and make a lot of progress, and then valleys where my interest or need moves to a different project or pursuit. I am hoping this journal will help me keep a more consistent approach to my work.
2. On my previous projects I have tended to leave my documentation to the end, as a reaction to the finished project. This has worked sufficiently but some aspect of the work always falls through the cracks and is not reflected in the documentation.
3. On this point, I am hoping that by writing out what I am doing, and why, it will help improve my ability to break down, rationalize, and explain what it is I am doing, what particular tricks or workarounds I am using to achieve my goals, and why I think my approach is superior to other options that I have at the time.

My strengths in my work are generally on the technical side of things, with my weaknesses on the logistical side, in areas such as timekeeping, end goal setting, and scope management.

October 10th

I am beginning to look at what I want to do for my final year project. I have two ideas that have come to the forefront amongst the others. These two ideas are, I feel, the best cross section of what I'm passionate about, what I'm good at, and what I feel like I can deliver as a full fledged product within the time I have.

These ideas are

1. Portfoliu.com(working title)
2. Unnamed Fantasy Game

1. Portfoliu.com is an online platform where users can log in and create their own 1 to 3 page portfolio site that they can customize the content of, the layout, and the functionality. The idea will be that the site will provide a robust editor that users will be able to use to populate their portfolio with html content sections and widgets, and when they are satisfied the site will serve them the generated html file/s. The editor will allow the user to switch between desktop, mobile, and tablet views, to provide fully responsive portfolios to the user.
    a. The aim is to build the site using react.js, and host the site on a python server using the Django framework for handling requests.
    b. Some of the potential issues will be how to serve the pages to the user. There will be a fair amount of css and JS to provide the kind of responsiveness and functionality that I would like to offer. Best practice would be to have them as external files but how would those individual files be deliver?

2. The Unnamed fantasy game is a game idea I have been building on for a little while that I am hoping to expand on throughout this year even if I don't choose it as my final year project. The idea of this game is that it will be a first-person shooter that takes place in a high fantasy world that the user can explore. The way the player will explore that world is through an overworld system, similar to an older RPG game. Players will be able to move around the map to reach regions that they can explore, and when they choose one the game will shift into the first-person world of that area. I am hoping to make the gameplay of the first-person sections reminiscent of older first persons shooters, such Blood or Duke Nukem 3d, with a focus on speed and movement as opposed to cover based shooters.

October 15th

I am moving forward with the game as my final year project. I've chosen the game because of the two ideas I was between it is the one I'm more passionate about and interested in. I have chosen to build the game in the Unity game engine.

I chose Unity because I have some experience in building projects in Unity, albeit those projects were significantly smaller than what I am aiming for with this game. It's a very good option for game development in my opinion and I think it will be 100% sufficient for every part of the game I am planning to implement.

I have started work on a prototype that I can feature in my proposal pitch video.

October 17th

The prototype is going well. I have the player movement implemented, so the player can move with the keyboard, and look with the mouse. I'll have to refine the values for the players movement speed to get it to the point I'm happy with it. I've also implemented the first enemy, a very simple zombie type enemy that chases the player and makes an attack when they are in range. Its pretty basic so far, it can only really chase the player and move around objects in the environment, but it gave me an intro into the AI system and nav mesh system within Unity that I'm going to have to look at in a lot more detail as I try to bring in more enemy types.

October 19th

Short update for today. I've written my script for my project proposal video, I just need to record it later today so I can submit it.

In terms of updates for the game itself I have implemented the weapon and health system for the enemy and players. So now both the player and the enemies have health values that can be increased or reduced depending on what events occur, and should the players health reach 0 or below, the game will end. When the health of an enemy reaches 0 they are simply removed from the game.

The weapon system was a fair bit more complex. The first thing I need was to implement a way for the player to cycle between the weapons they have, so when they chose to use one weapon, the other weapons they have are deactivated. This also had to take into account how the weapons are fired, whether they were melee or ranged, what kind of projectile they shoot, etc.

I've had to look at how best to differentiate weapon types within Unity and what the best practices are regarding things such as inheritance and polymorphism.

And the final big thing was monitoring how many projectiles exist within the scene, and how they are disposed of or reused. Because there is the potential for a lot of projectiles to fill up the scene between what the players and the enemies are shooting, if the projectile objects are left alone and not properly disposed of there could be significant performance issues in the long term. I've implemented a Singleton object to monitor the amount of projectiles in the scene for the moment, to help monitor and reuse existing projectile objects, rather than have the game make a new object whenever a weapon is fired.

Obtober 22nd

I have started looking into what I can use to make models and assets for the game going forward. At the moment the game is just using basic Unity 3d objects to represent items or creatures in the world. Stuff like capsules for monsters or cubes to represent big rocks or whatever. I've been doing a pretty good course in how to use MagicaVoxel, a free editor for creating voxel art and assets.

The benefit to using voxels is that I'm not great with curved 3d modelling, but with voxels the objects can be made up of individual cubes to form an elaborate design. MagicaVoxel also allows you to export assets directly into file formats that can be imported into Unity, which maintains the shape and colour of the asset within MagicaVoxel. There are also a lot of good resources on how to use blender to build an animation rig for an object created MagicaVoxel, which will help a lot in the long term.

October 25th

I have finished of the projectile system, and implemented 4 different weapons the player can use, that all have distinct ways of firing and resources they use. I'm having some issues with the way one of the projectiles mesh renderer interacts with the terrain. Sometimes it bounces of the terrain, and sometimes it just passes through. I haven't figured out where the issue is stemming from yet, so it will take a while longer to resolve that.

The next thing I want to start working on is the overworld traversal system. This won't be an exhaustive build, it's simply the system that controls bringing the player out to the overworld scene when they are in a level, and allowing them to choose a place to explore when they are on the overworld. The big thing here is figuring out how to maintain persistence of the players stats as they traverse the various scenes.

I'm off for two weeks starting tomorrow so I'm going to take a few days off from this project.

October 29th

Back to it, and I'm starting on the scene traversal system. I currently have the ability to move between scenes. For example the player is able to open the pause menu within the game and return to the main menu, or load the overworld, but I haven't implemented the persistence yet. At the moments its simply loading a new instance of the player object whenever a new level scene is loaded.

I'm looking at guides and stack overflow questions to figure out the best way to do this at the moment. I'm think of a static class to represent the Player. This class won't inherit from anything, it will just be a static class that stores the players current stats, and will provide methods for improving those stats as the player levels up.

November 2nd

I've started working on the Character Stat class that will maintain the stats the player has built up over time. I've gotten it to the point that the stats I populate that object with will persist on scene transition, but I'm having trouble thinking of a good way to integrate it with the Player Character script, and more particularly how to get the player experience to increment in the Character Stat object when the Player Character does something.

November 5th

I've moved away from the technical side for the moment to work on getting more knowledge and experience in level design, so I can start drawing out some preliminary maps for the levels that I'm going to build for the first demo of the game. I'm looking to have at least one map that has the structure completed out with assets and enemies and the like by the end of December, so I have to get started on that.

November 9th

I've been working through tutorials and articles on https://www.worldofleveldesign.com/ to get my head around world design, mostly drawing out maps in pencil in my notebook, and thinking of what feels like satisfying placement of enemies and pickups and the like.

November 15th

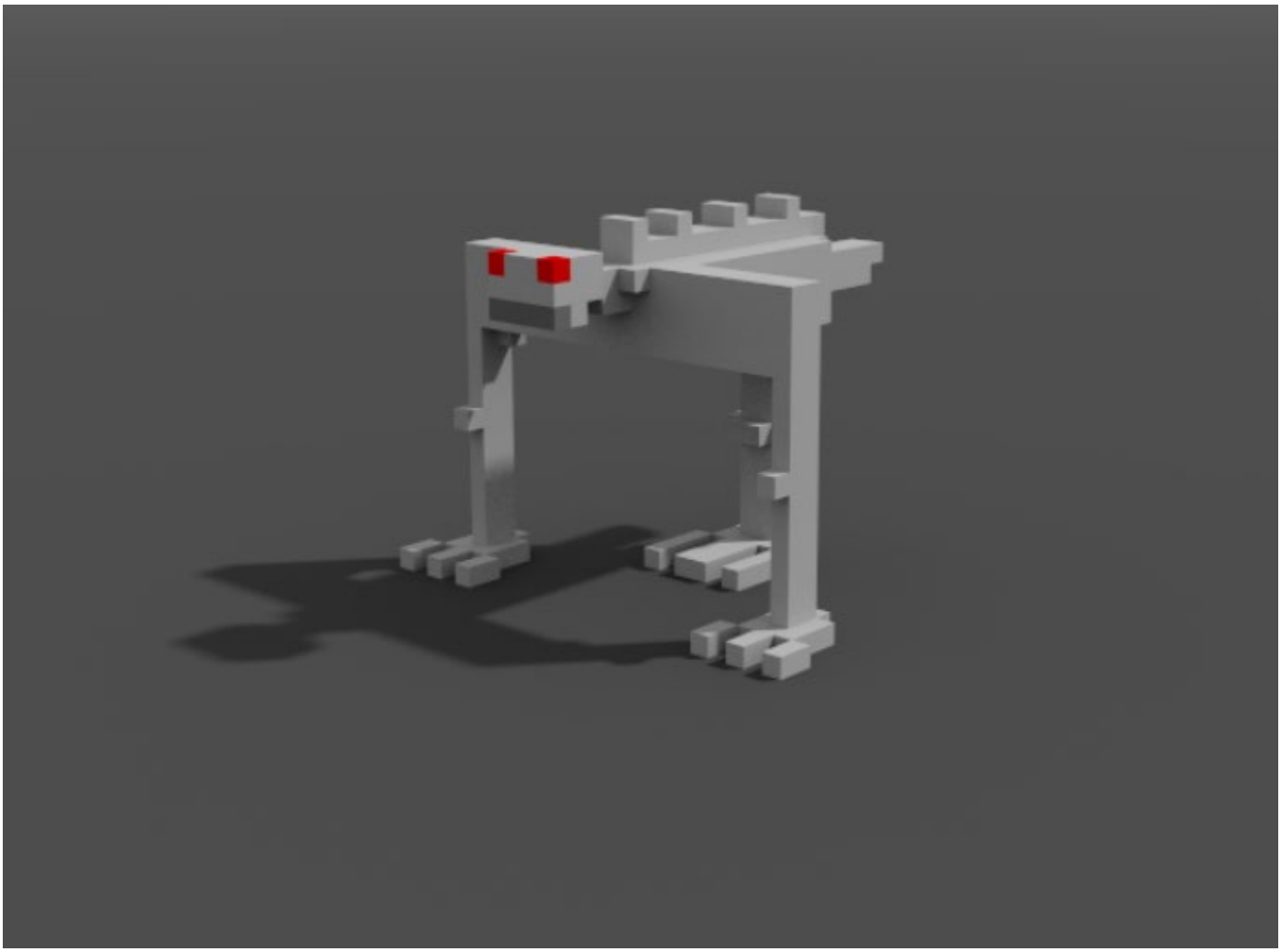
There's a lot of project work from my other classes coming up over the next forthnight so I'm going to have to put the game on the backburner until I have them submitted. I don't want to end up leaving them too late and finding myself having to cut corners or leave features or requirements out.

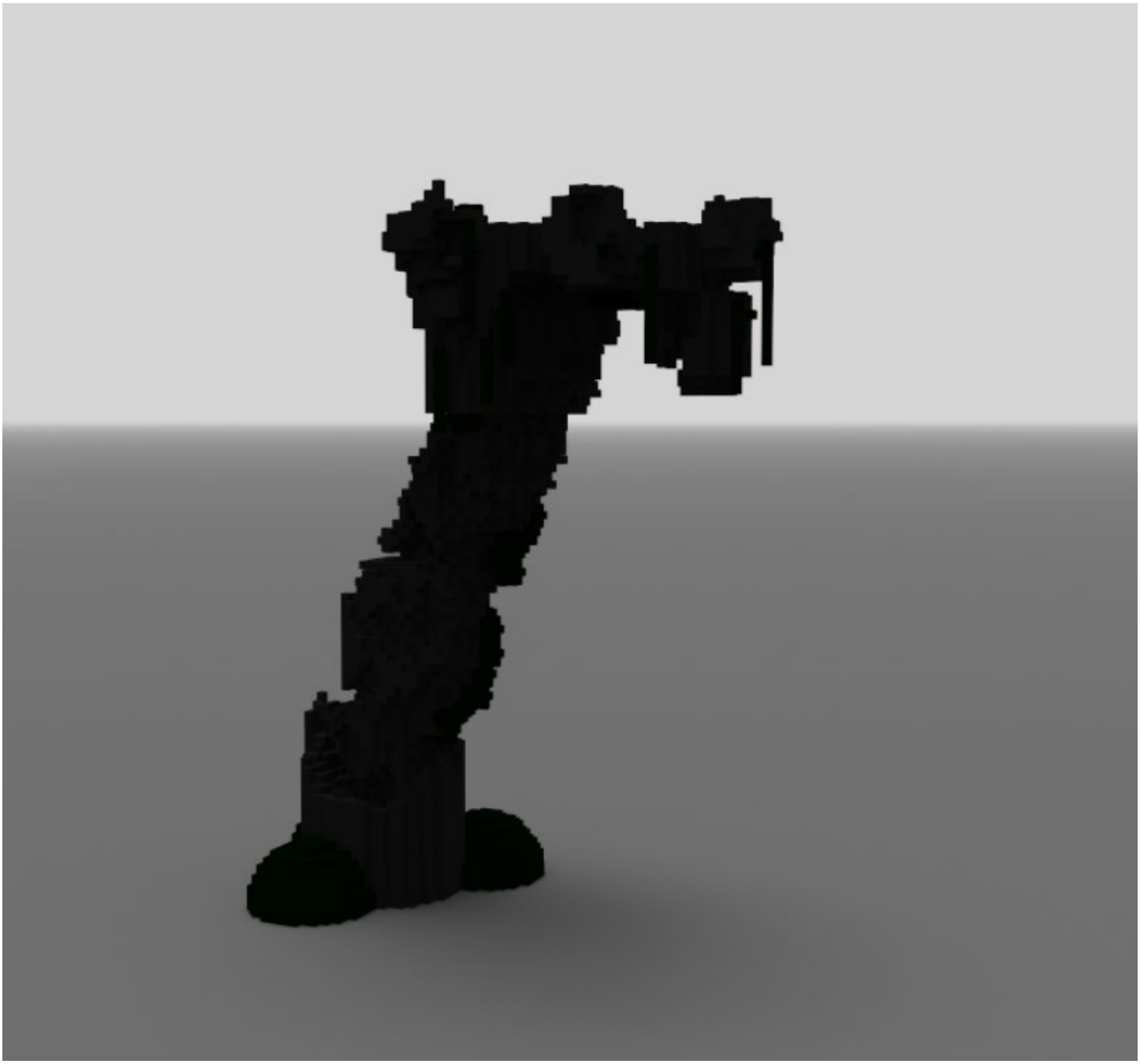
November 22th

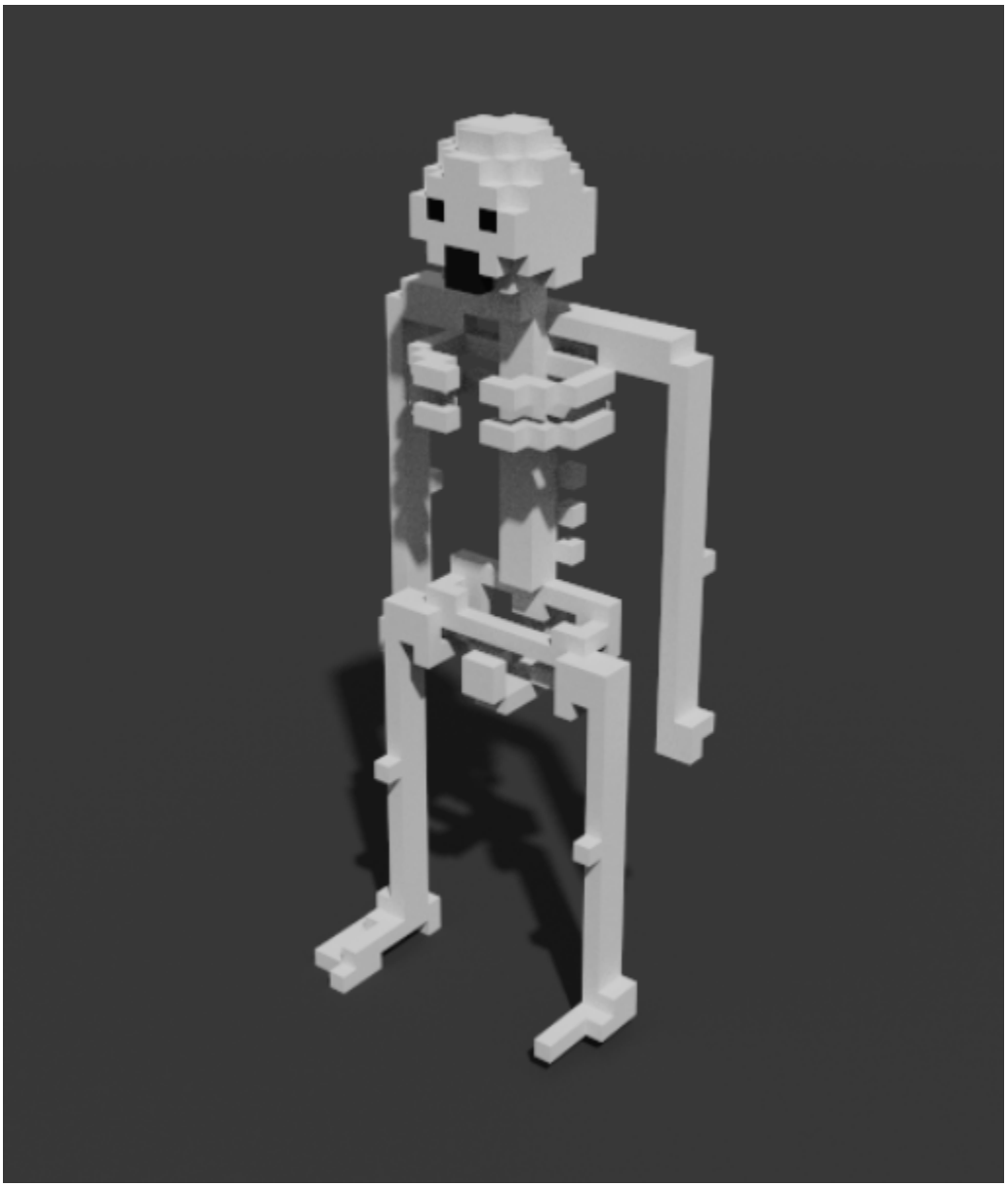
The last week or so has been fairly hectic. I've had my web services project due, with my Data Applications one due today, which have both eaten into my free time, so I haven't made much progress on the game. All I've managed to do was generate some very basic models in MagicaVoxel and try to attach an animation rig in blender. The animation rig didn't really work but I have an idea of where I'm going wrong. I'm fairly certain the issue is that the bones weren't properly connected in the rig, which meant they weren't moving in tandem correctly.

November 29th

Finally managed to get back with some progress on the game, mainly in the form of some models.







The above images are the oldest to the newest models I've made. They're not amazing, but I think they'll do for a first try. The last one is in the t pose because I read it makes it easier to do animation when the model is built in that pose.

February Journal

4th February

After the fairly demoralizing lack of progress that I made on my project in January, I'm off to a decent start in February. I am looking at how to build out a dialogue system that will allow the player to interact with non-player characters in the world. I've tried to implement this in previous projects I've worked on but I've always run into issues at runtime when I try and populate the UI from a collection of dialogue options attached to the NPC object in the unity scene. Typically, what ends up happening is the game ends up in some form of infinite loop as it tries to load the text from the game object, which leads to the game running into a stack overflow exception.

The plan for the dialogue system is to implement a sort of visual novel style of interaction between the player and the NPCs, similar to something like the Ace Attorney series, where the conversation take place from a first person perspective.

The conversations will include an image of the NPC, that will animate and change facial expressions based on the emotion of the scene, which will be handled using Unity's 2d animation system, which is thankfully quite simple and robust.

Each NPC will have an associated data structure containing a set of dialogue lines for them, and a set of choices that the player can make in response at certain points. I'm also hoping to be able to implement some sort of hook into my planned quest system. Something like a numerical ID attach to certain conversations that hooks into a dictionary of Quests that I have attached to the game manager object that is passed from scene to scene.

9th February

I made a breakthrough on the dialog system. I took a break for a few days to work on my project for Clou Application Development, and when I came back to the project I started looking at the way I was implementing the dialog system, and the issue was on which side I was handling the process. I was trying to handle the system on the player side, with all the code being maintained on the PlayerController. The problem with this was it got me into the mind frame of using the in built methods on a Unity Game object, such as Update and Awake. I was attempting to use the update method to handle player action, e.g. responding to a player click on a dialog option. The problem with this was that the update function is called once per frame, which meant that either it was occurring so frequent that the player couldn't reliably hit it, or it was leading to a loop. To fix this, I started movingthe code to handle the dialog system to the UI controller object, which I was using to handle button presses in the UI, for the pause menu and the like. This allowed me to make put the code handling dialog options on at will buttons in the UI, rather than trying to handle it in the Player or NPC objects.

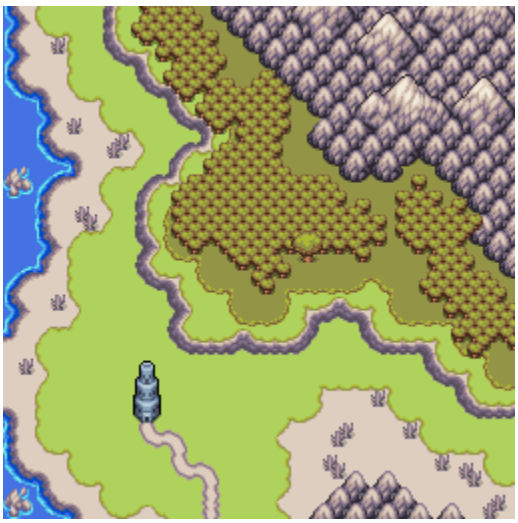
I also started using Unity Serialised Objects to store the dialog script, and dialog options. Serialised objects are a very useful feature in Unity, that I have been using as the data structures for my NPC dialogs. I can store all the lines of dialog, and also give the Serialised Object a set of methods to control the flow of the conversation. In addition to this, its very easy to attach a serialized object to a game object in the scene and get that Serialised object by calling a get method from the UI controller.

I've been testing this system using some dummy NPC game objects just to make sure my methods for controlling the flow of the conversation and handling player responses are working correctly. My next step is to figure out how to pull in a quest and attach it to the players list of quests from the UI controller.

16th February

I've started work on building out how the player will traverse the overworld. My initial intent was just to have the player click on items in the overworld to move there, but that really hampered my attempts to create a feeling of a large open world, because it removed any sense of time to the journey, since click on an icon moved the player there immediately. It really just made the Overworld more of a level select screen, which I wasn't happy with.

To combat this, I've decided to implement something similar to the type of Overworlds present in old Japanese roleplaying games, such as the early Final Fantasy or Breath of Fire games.



These are some of the parts of the overworld that I have been working with. The idea is that the player will move around the overworld like an old RPG, and enter the towns and locales from game objects that are placed on the map at those areas on the map. These game objects will contain methods for loading the various dungeon or village scenes.

This will allow me to create a feeling that the player is actually travelling from place to place, rather than just choosing the next map or story scene from a menu system.

I'm also looking at potentially implementing some form of randomly generated content, such as bandit attacks or injured travelers that the player can interact with as they travel across this map. I think this would add some excitement

to the process of travelling and give the player more opportunities to gain items and experience, but it's essentially a stretch goal at the minute.

23rd February

Progress on the overworld has progressed steadily, which I'm happy with. I've gotten the controls for moving around on the overworld working correctly now. It took a little bit of doing to get it to correctly and routinely differentiate between using the first-person controls and using the new 2d controls. But I have that working now. I just need to work on getting all the game objects in place for the various interactable objects and locales in the game world, which is a task that I'm not entirely sure the timeframe off. I've been trying to scope it out, but it is fairly reliant on having the levels and content to back them up, which I think I am going to make my focus throughout March.

27th February

Just a short update for today, as I've been trying to make more time for my distributed systems and Cloud app projects and reports that are coming up. I'm happy with where I have the game at the moment, so I'm more comfortable leaving it to sit for a few days while I catch up with those two classes.