

National College of Ireland
BSc in Computing
2015/2016

Michael Kilfeather
12420472
x12420472@student.ncirl.ie

Equilibrium

Technical Report



Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: Michael Kilfeather
Student ID: 12420472
Supervisor: Eugene McLaughlin

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarized overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: _____

Date: _____

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is

completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks

and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- Prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- The requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

Table of Contents

Executive Summary.....	9
1 Introduction.....	10
1.1 Background	10
1.2 Aims	10
1.3 Technologies	11
1.3.1 Unity.....	11
1.3.2 Microsoft Visual Studio 2015 IDE.....	11
1.3.3 C#	12
1.3.4 RAIN AI.....	12
1.3.5 Dialogue System for Unity.....	13
1.3.6 Mixamo Fuse	13
1.3.7 InControl	14
1.3.8 ProBuilder Basic	14
2 System	15
2.1 Requirements	15
2.1.1 User Requirements Definition	15
2.1.2 Requirements Specification	15
2.1.3 Functional requirements	16
2.1.4 Non-Functional Requirements	30
2.1.5 Data requirements	32
2.1.6 User requirements	32
2.1.7 Environmental requirements	32
2.1.8 Usability requirements	32
2.2 Design and Architecture	34
2.2.1 Use Case Diagram.....	34
2.2.2 Class Diagram	34
2.2.3 Logical View.....	39
2.2.4 Software Architecture.....	39
2.2.5 Performance	40
2.3 Implementation	41
2.3.1 UserInput.cs.....	41

2.3.2	CharacterMovement.cs	43
2.3.3	CharacterStats.cs	44
2.3.4	GameManager.cs	45
2.3.5	WeaponManager.cs.....	46
2.3.6	WeaponControl.cs	47
2.3.7	GainReward.cs	48
2.3.8	IncrementOnDestroyModified.cs.....	48
2.3.9	EnemyAI.cs.....	49
2.3.10	GraphicsMenu.cs	50
2.3.11	SkillTreeNode.cs.....	52
2.3.12	MyCharacterActions.cs	53
2.3.13	FreeCameraLook.cs	54
2.3.14	PersistentPlayerData.cs.....	54
2.3.15	ChooseRandomLocation.cs	56
2.3.16	Dialogue Tree (Dialogue System for Unity).....	57
2.3.17	Behaviour Tree (RAIN AI)	59
2.4	Testing.....	60
2.4.1	Unit Testing.....	60
2.4.2	Black-Box / Functional Testing.....	61
2.4.3	Performance Testing.....	62
2.5	Graphical User Interface (GUI) Layout	63
2.5.1	Pause Menu.....	63
2.5.2	Quest Log Window.....	64
2.5.3	Skill Tree panel	65
2.5.4	Player HUD	65
2.5.5	Main Menu	66
2.5.6	Options Menu.....	67
2.5.7	Weapon store panel.....	68
2.5.8	Dialogue UI	69
2.6	Customer testing	69
2.7	Evaluation.....	71
3	Conclusions.....	74
4	Further development or research	76

5	References	77
6	Appendix	78
6.1	Project Proposal	78
6.1.1	Objectives.....	79
6.1.2	Background	80
6.1.3	Technical Approach.....	81
6.1.4	Special resources required.....	82
6.1.5	Project Plan	83
6.1.6	Technical Details	83
6.1.7	Evaluation.....	84
6.2	Project Plan	85
6.3	Requirement Specification.....	86
6.3.1	Document Control	86
6.3.2	Revision History	86
6.3.3	Distribution List	86
6.3.4	Related Documents	86
6.3.5	Introduction	88
6.3.6	User Requirements Definition	90
6.3.7	Requirements Specification	91
6.3.8	Interface Requirements.....	103
6.3.9	System Architecture.....	107
6.3.10	System Evolution	108
6.4	Product Design Specification.....	109
6.4.1	Introduction	111
6.4.2	General Overview and Design Guidelines/Approach.....	111
6.4.3	Architecture Design.....	111
6.5	Monthly Journals	115
6.5.1	Reflective Journal – September.....	116
6.5.2	Reflective Journal – October.....	117
6.5.3	Reflective Journal – November.....	118
6.5.4	Reflective Journal – December.....	120
6.5.5	Reflective Journal – January.....	121
6.5.6	Reflective Journal – February	123

6.5.7	Reflective Journal – March.....	124
6.6	Other Material Used	125
6.6.1	Evaluation Surveys	125

Executive Summary

Equilibrium is a cyberpunk-themed third-person action role-playing stealth game made in Unity 3D. The objective of the game is for the player to complete missions assigned by an NPC. Upon completion of these missions, the player will be rewarded with money and skill points. The majority of these missions take place in a central mission area, with each mission having different objectives. Enemies will be powered by behaviour trees, in order to implement more complex AI into the game. With the money that the player acquires throughout the game, they will be able to enter a shop in the 'hub world' of the game and purchase new weapons and ammunition.

The game was developed using C# in the Microsoft Visual Studio 2015 IDE. The game is primarily aimed at the Windows PC platform due to its graphical fidelity. 3D models for the game were acquired from various websites e.g. TF3DM as well as the Unity asset store and some 3D character models were created through 'Mixamo Fuse' software. Data is saved and stored on the user's PC using player preferences in Unity through the use of the Dialogue System plugin's save and load feature.

The target audience for this game is aimed towards those aged 16 and over, due to the game's mature content, including violence.

1 Introduction

1.1 Background

The games industry has grown exponentially in the past couple of years. In Ireland alone, it has made an estimated €206 million in revenue. In the past, the development process for a game was a monumental task, often requiring a large studio comprised of programmers, artists, sound designers and more. Nowadays, many games development tools / engines are now available for free to the general public such as the Unity game engine and more recently, the Unreal engine. This now makes the process of developing a game from start to finish a lot more feasible, with just a small independent studio.

The stealth genre of video games is considered a niche market. True stealth games such as Metal Gear Solid or Splinter cell catered to a specific audience, often due to their level of difficulty and learning curve behind them meaning that they generally didn't appeal to the casual gamer. In more recent years, these types of games have tried to break away from this mould and have become more 'action-centric' and cinematic in feel in the hopes of appealing to a wider audience. To achieve this, many stealth games of today have incorporated open-world environments and RPG elements, seen in extremely successful franchises such as Grand Theft Auto, Fallout and Skyrim. Previously, stealth games were very linear in nature. This has now changed, as evidenced by the recently released Metal Gear Solid V: The Phantom Pain, where the player can approach missions in creative ways in a vast, open environment, with a plethora of weapons and gadgets at their disposal.

1.2 Aims

The aim of this project is to create an immersive third-person action stealth RPG. The game will comprise elements seen in games like Watch Dogs, Deus Ex and Mass Effect such as interactive NPC's, skill tree's, realistic weapon mechanics and in-game item purchasing such weapons, ammunition and silencers.

The game will allow the player to give their character new abilities by acquiring skill points through the completion of objectives and purchasing these skills through the Skill Tree menu. The player will be able to enter a shop in the 'hub world' of the game and purchase weapons and ammunition, by using credits they have earned in the game.

My goal for this project is to retain the elements of what make stealth games so successful, to cater to fans of the stealth genre, while also incorporating elements seen in other genres such as RPG's to appeal to an even wider audience.

1.3 Technologies

1.3.1 Unity

Unity is a game engine used to develop both 3D and 2D games for various platforms such as PC, consoles, mobile devices and websites. Unity was chosen as the primary technology for this project. I chose the Unity game engine as the primary technology for this project as I was already very comfortable working with it prior to starting this project due during work placement, having originally learnt it myself in self-study. A great benefit of Unity is that it has excellent support available from the community through official unity forums as well as YouTube tutorials. It also has an asset store which contains a large selection of both paid and free assets that can be used in projects such as plugins and 3D models.

The version of Unity that is currently being used for development of this project is Unity 5. Unity 5 is a significantly upgraded version of the game engine as it introduced many new features, mainly related to graphics such as Real-time Global Illumination, HDR Reflection Probes and a Physically-based Standard Shader.

1.3.2 Microsoft Visual Studio 2015 IDE

Microsoft Visual Studio is an integrated development environment (IDE) primarily used to develop computer programs and web applications e.g. ASP.NET. It

supports plenty of programming languages such as C, C++, Visual Basic.NET, C# and more.

Visual Studio was chosen as the IDE for my project due to the many benefits it provides. Visual Studio works seamlessly with Unity. It is great for debugging a Unity game that is either running in the Unity Editor or in the Unity Player, or even debugging an external managed DLL in a Unity Project.

Code can also be written much quicker due to Visual Studio's IntelliSense, refactoring and code browsing capabilities.

1.3.3 C#

C# (pronounced "C-sharp") is an object-oriented programming language from Microsoft and is one of the programming languages designed for the Common Language Infrastructure (CLI). It is generally used in conjunction with the Microsoft Visual Studio IDE. I chose C# as the primary programming language for this project as it is the recommended programming language of choice to use with Unity.

Unity games can be programmed in three different languages; C#, UnityScript (JavaScript) and Boo. C# is recommended for Unity as it supports many more features, faster script compilation and better supported documentation. Prior to the development of this project, I had worked with the C# programming language and Unity game engine over the course of my internship, so I was already very familiar with the language.

1.3.4 RAIN AI

RAIN AI is a powerful AI engine used to create complex behaviour for any character in any game and on any platform. It uses behaviour trees for implementing artificial intelligence. In terms of AI, a behavior tree controls the flow of decision making of an agent. They describe switching between a finite set of

tasks in a modular way. Behaviour Trees came to fruition from the games industry as a powerful tool to model the behaviour of non-player characters (NPCs).

I chose RAIN AI for many reasons. For one, it is completely free and there are no license fees. It is very quick and easy to use, once it is set up and the basic fundamentals understood. It also offers full access to a moderated support forum, which I have used a few times over the course of this project and which has been very helpful. I used RAIN to give enemy characters in my game complex behaviour through various states such as patrolling, detecting the player, attacking the player and searching for the player. RAIN AI uses sensors, both audio and visual that can be attached to the enemy / NPC. Audio sensors can be used to detect sounds from the player or other characters within a certain radius of the enemy. Visual sensors are used for the enemy's line of sight.

1.3.5 Dialogue System for Unity

Dialogue System for Unity is a powerful Unity plugin that allows the user to implement interactive dialogue and quests into the game. It uses a visual node-based editor so that the user can create branching dialogue. It also allows for the implementation of cutscenes, quest logs, save/load functionality and more. I chose this plugin due to the fact that it is simple to use, very efficient and provides plenty of documentation and moderated forum support. This plugin is necessary as dialogue will play a big part in the game as it is used to set the context of the story for the game as well as a means of triggering missions for the player to partake in.

1.3.6 Mixamo Fuse

Mixamo Fuse is 3D character creation software developed by Mixamo in collaboration with Adobe. It allows the user to create a plethora of unique character designs. The user can customize values to change the shape, texture and clothing of the character. Once the user is satisfied with the design of their character, the character model can be uploaded to the Mixamo website, where it will be automatically rigged and animated. This software was chosen for use in this project

as it is a big timesaver for adding characters and animating them. Manually rigging and animating 3D models is a very complex and time consuming process.

1.3.7 InControl

InControl is an input manager for the Unity game engine. It provides input mappings for most gamepads and controllers such as the PS4 Dualshock controller, Xbox One gamepad and more. The asset is written in C# and makes it easy to add cross-platform controller/gamepad support to your game. Actions can be binded to controls. These controls can also be rebinded at runtime. This asset was chosen for the project as it is far more efficient than binding controls through the Unity game engine itself.

1.3.8 ProBuilder Basic

ProBuilder is a 3D tool that enables a developer to build and modify geometry within the Unity editor itself. In other words, there is no need for the developer to use external 3D modeling software such as Autodesk Maya or 3DS Max. ProBuilder Basic is the free version of ProBuilder Advanced, created by ProCore. The main purpose of ProBuilder Basic is really for prototyping basic levels and is used early on in the development process of the game for constructing basic geometry and creating “white-box” scenes. These are scenes that don’t have any texture or detail and use simple geometry. In this instance, ProBuilder Basic is the perfect tool. Vertices and faces of shapes can be modified with this tool and this is something that the default Unity editor does not provide.

2 System

2.1 Requirements

2.1.1 User Requirements Definition

The objective of the game is for the player to play through missions that are assigned to the character throughout the game. Upon completing these missions, the player will gain money and skill points which can be used to purchase new weapons and unlock abilities.

The user can save their progress whenever they are in the 'hub world' of the game. They can load their save data at any time during the game and system will load the 'hub world' level and load all of the character's stats and other items they had at the point they saved.

The user will be able to unlock new skills through the use of a skill tree system. The skill tree can be accessed through the pause menu. The user will be able to purchase new weapons for their character when they enter a shop. The user can quit the game whenever they wish. The user can also adjust graphical options in order to get the best performance possible out of their system.

2.1.2 Requirements Specification

After no more than 10 – 15 minutes of playing through the game, the user should know how to access and use all of the game's features with ease. Specific actions will show button prompts so the player will know how to perform the action in the game. A menu can be accessed in-game which will show all of the controls in detail. Response time should be very quick as it should take the user no more than 1 minute to start the game upon execution.

2.1.3 Functional requirements

New Game

The user should be able to start the game from the beginning when starting the application for the first time. The user can start a new game by selecting the 'New Game' button on the main menu of the game. This requirement has not changed from the original requirements specification document.

Start Mission

This requirement enables the player to start a mission in the game. To do this, the player can open up the 'Quest Log' window by accessing it from the pause menu. From here, the user can view the mission and its description. The user approaches an NPC to trigger a mission. Once the player's conversation has ended with an NPC, the mission will be added to the player's Quest Log window where they can view the mission details.

Unlock Skill

This requirement allows the player to unlock new skill for their character to utilize. Skills can be unlocked through the use of a skill tree which can be accessed from the pause menu in-game. The player will accumulate skill points over the course of the game. When they earn enough points for a skill, it will be available for purchase. Skill points are awarded through completion of quests. This requirement remains unchanged.

Change Settings

This requirement enables the user to modify various settings from the main menu of the game such as the graphics level they wish to run the game on, resolution, anti-aliasing levels and more.

This is a newly added requirement as I feel that it is important for the player to be able to customize the game to their liking, in the event that their system cannot run the game at optimal settings.

Purchase Weapon

This requirement allows the user to purchase new weapons for their character. The player can enter a shop from the in-game city and purchase a variety of different weapons. The player uses in-game 'credits' to purchase these items. Credits are acquired through the completion of missions and through other various conditions.

This requirement has been changed to just weapon purchasing. Prior to this, clothing customization was also intended to be added, but due to time constraints, this had to be cut down to weapon purchasing.

Save Game

This requirement gives the user the ability to save their progress whenever they are in the 'hub-world' of the game. To do this, the user opens the pause menu. From here, the user presses the 'Save Game' button. This will then store the save game data to a file. This is done through the Dialogue System for Unity plugin, which saves data to 'PlayerPrefs'. On Windows, these PlayerPrefs are stored in the registry under HKCU\Software\[company name]\[product name] key, where the company and product names are the names set up in 'Project Settings' in the Unity Editor.

This requirement has been altered, due to some setbacks with the 'Load Game' requirement. Previously, the requirement should have worked in a way that the user can save at any time. This is explained in detail in the 'Conclusions' section of the report.

Load Game

The requirement will allow the user to pick up from where they had last saved the game. All of the player's progress such as money earned, skill points attained and missions completed should remain intact when they load the game.

This requirement has been altered. Previously, it was intended for the user to be able to load the game from the main menu. Due to some setbacks, the requirement had to be altered so the user can only load the game during gameplay. This is further explained in the 'Conclusions' section of the report.

Quit Game

This requirement lets the player quit the game in two different ways, and is now slightly different to the original 'Quit Game' requirement as a result. One way in which the player can quit the game is by pausing the game. This will open up the pause menu. From here, the player can quit the game and this will then bring them back to the main menu, but does not close the application entirely. Another way of quitting the game is through both the main menu and pause menu. From the main menu, the user simply clicks on the 'Quit Game' button and the application will exit to the desktop. From the pause menu, the player can click the 'Quit to Desktop' button and this will carry out the same procedure.

2.1.3.1 Requirement 1<New Game>

2.1.3.1.1 Description & Priority

This allows the player to start a new game. This requirement is vital as it is required for the player to begin playing the game for the first time.

2.1.3.1.2 Use Case

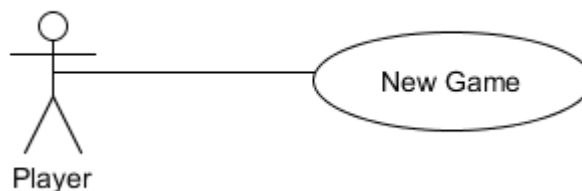
Scope

The scope of this use case is to allow the player to start a new game.

Description

Describes the process by which the player starts a new game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode

Activation

This use case starts when the player starts a new game.

Main flow

1. The system identifies the player.
2. The Player starts a new game.
3. The system loads the opening level of the game.

Alternate flow

N/A

Exceptional flow

N/A

Termination

The system presents the next screen to the player.

Post condition

The system goes into a wait state

2.1.3.2 Requirement 2<Start Mission>**2.1.3.2.1 Description & Priority**

The player commences a mission. This is required so that the player can progress through the game by completing missions.

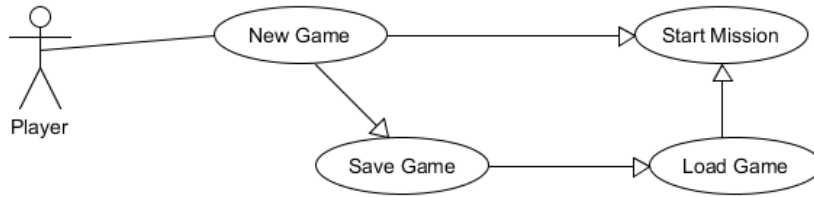
2.1.3.2.2 Use Case**Scope**

The scope of this use case is to allow the player to start a mission.

Description

This use case describes the means by which a player can start a mission.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

This use case starts when the player starts a mission.

Main flow

1. The player is in the main town area.
2. The player approaches the NPC in the town area.
3. The player selects the missions presented to them during the conversation from the Dialogue user interface.

Alternate flow

N/A

Exceptional flow

N/A

Termination

The system loads a new scene, where the mission will take place.

Post condition

The system goes into a wait state

2.1.3.3 Requirement 3<Unlock Skill>

2.1.3.3.1 Description & Priority

The player unlocks a new skill / ability for them to use during missions. This is important as it will help players approach missions in different ways.

2.1.3.3.2 Use Case

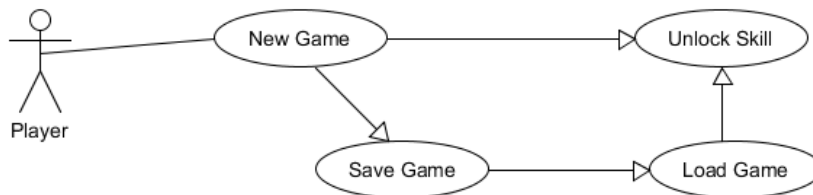
Scope

The scope of this use case is to allow the player to gain a new skill.

Description

This use case describes the means by which a player gains a skill.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

This use case starts when the player unlocks a Skill.

Main flow

1. The player opens the Skill Tree menu.
2. The player selects the skill they wish to unlock.
3. The system unlocks the skill from the skill tree.

Alternate flow

N/A
Exceptional flow

N/A

Termination

The system stays on the Skill Tree menu, unless the player decides to exit the Skill Tree menu.

Post condition

The system goes into a wait state

2.1.3.4 Requirement 4<Change Settings>

2.1.3.4.1 Description & Priority

The player can access the options menu in order to adjust graphical settings as well as controls to suit the user's needs. This is vital especially if the user is having difficulty in running the game at a playable framerate.

2.1.3.4.2 Use Case

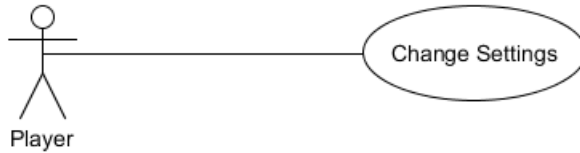
Scope

The scope of this use case is to allow the player to change various settings such as graphics or controls.

Description

This use case describes the process by which the player can change the graphical options of the game.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state

Activation

The use case starts after the player opens the 'Options' menu.

Main Flow

1. The player opens the options menu from the main menu.
2. The player selects the value of the graphical setting they wish to change.
3. The system changes the quality of the graphics.

Termination

The system stays on the options menu until the player presses the 'back' button to return to the main menu.

Post Condition

The system goes into a wait state.

2.1.3.5 Requirement 5<Purchase Weapon>

2.1.3.5.1 Description & Priority

The player can purchase new weapons for the character from a shop in the game. New weapons are used to help the player out during enemy encounters.

2.1.3.5.2 Use Case

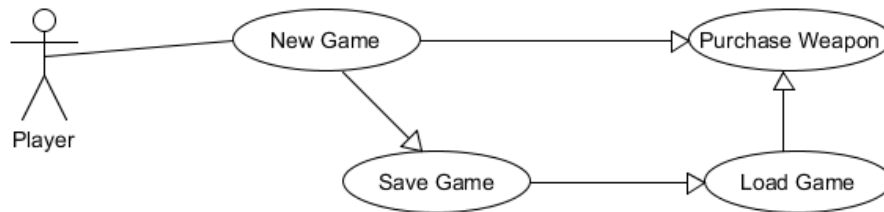
Scope

The scope of this use case is to allow the player to purchase new weapons for their character.

Description

This use case describes the process by which the player can purchase weapons.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

The use case starts when the player enters a shop.

Main flow

1. The player enters a shop.
2. A menu opens, displaying weapons that the player can purchase.
3. The player selects the item they want.
4. Money is taken from the player's inventory.

Termination

The system stays on the menu unless the player selects the 'close button' on the menu.

Post Condition

The system goes into a wait state.

2.1.3.6 Requirement 6<Save Game>

2.1.3.6.1 Description & Priority

This requirement allows the player to save at a certain point within the game. This is required so that the player can load back to a stage within the game, in case they need to go back to that stage for whatever reason.

2.1.3.6.2 Use Case

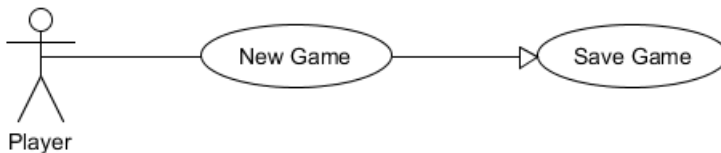
Scope

The scope of this use case is to allow the player to save their progress mid-game.

Description

This use case describes the process by which the player saves the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

The use case starts when the player saves the game.

Main flow

1. The player opens the pause menu.
2. The player selects the 'Save Game' option.

3. The system stores the save game data to a file.

Termination

The system returns the player to the pause screen.

Post Condition

The system goes into a wait state.

2.1.3.7 Requirement 7<Load Game>

2.1.3.7.1 Description & Priority

The player can load the game at a certain point. This use case is required so the player can load the current state of where they are in the game after they have saved it.

2.1.3.7.2 Use Case

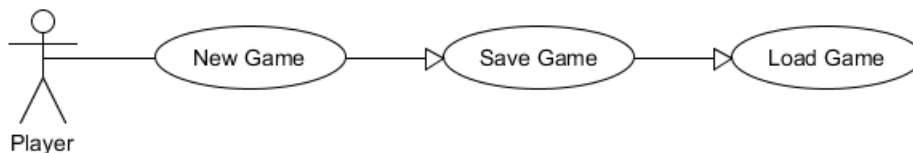
Scope

The scope of this use case is to allow the player to load the game.

Description

This use case describes the process by which the player loads the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

The use case starts when the player loads the game.

Main Flow

1. The player first saves the game from the pause menu.
2. The player then opens up the pause menu.
3. The player selects the 'Load Game' option from the pause menu.
4. The state of the game that was saved will then be Loaded.

Termination

The system presents the next screen to the player.

Post condition

The system goes into a wait state.

2.1.3.8 Requirement 8<Quit Game>

2.1.3.8.1 Description & Priority

The player quits the game. This function is required so that the player can shut down the application.

2.1.3.8.2 Use Case

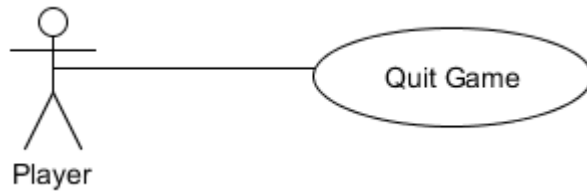
Scope

The scope of this use case is to allow the player to quit the game.

Description

This use case describes the process by which the player quits the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

This use case starts when the player quits the game.

Main Flow

1. The player selects the 'Quit Game' option from the main menu.
2. The system exits the application.

Alternate Flow

A1: <Quit game from Pause Menu>

1. The player opens the Pause Menu.
2. The player selects the 'Quit Game' option from the pause menu.
3. The system exits the application

Termination

The system closes the application.

Post condition

The system is off.

2.1.4 Non-Functional Requirements

2.1.4.1 Performance/Response time requirement

The game should run at 60 FPS on a user's PC or Laptop. However this will largely depend on the graphics card that's installed in the user's computer. For optimal performance and graphical fidelity, a dedicated graphics card would be recommended e.g. Nvidia, AMD Radeon, as opposed to integrated graphics cards such as Intel HD Graphics. Graphical settings can be adjusted before the player launches the game so the user should still be able to play the game regardless of their computer specifications. Response time should be almost instantaneous with no input lag or delay when the user performs an action within the game.

2.1.4.2 Availability Requirement

The game will be available to users at all times and can be accessed by launching an .exe file.

2.1.4.3 Robustness Requirement

Lots of beta testing and bug fixing will be done prior to the completion of the project so that the user won't encounter any game breaking bugs, glitches or crashes.

2.1.4.4 Reliability Requirement

The game should be available for the user to run successfully at all times, especially if they have the game downloaded onto their computer.

2.1.4.5 Maintainability Requirement

The game should be supported after its release. If there are any game breaking bugs or glitches that may have gone unnoticed during beta testing, these will be rectified.

2.1.4.6 Portability Requirement

The user will be able to play the game on both desktop PC's and laptops. The user can keep the game stored on a USB flash drive / external hard drive, or even through cloud storage (Google Drive, Dropbox) and transfer it to another computer and play it on that system if they wish.

2.1.4.7 Extendibility Requirement

Additional content may be added at a later date after the game's initial release which may possibly introduce new gameplay features which will add replayability to the game.

2.1.4.8 Reusability Requirement

If the player has finished all of the missions, they can return to the mission area of the game at any time and face enemies to try out new weapons they have purchased. The player can also earn more money within the game to purchase these new weapons.

2.1.4.9 Resource utilization requirement

The game should use as many resources as it can from what the user's PC / Laptop is capable of providing which will have an impact on performance and graphical fidelity e.g. RAM size, Graphics Card, CPU frequency.

2.1.5 Data requirements

Users require the ability to save their current progress they have made in the game. This data is stored using 'player preferences' which is held in the registry of the user's computer. However, this data won't be large and the user will only need a small amount of free space on their hard drive to save data.

For the final build of the game, the user should require no more than 1GB of space on their hard drive in order for them to run and access all of the data associated with game.

The game is saved through the Dialogue System for Unity plugin, using Unity's PlayerPrefs. This stores and accesses player preferences between different game sessions. On Windows, this is stored in the Windows registry, under the HKCU\Software\[company name]\[product name] key.

2.1.6 User requirements

The user must have either a desktop PC or laptop capable of running a modern version of Windows OS, preferably Windows 10, or Windows 7 as the minimum. The user should also have internet access in order to download the application.

2.1.7 Environmental requirements

The application must run on a fully stable working environment in order to ensure the game runs smoothly. The application was designed with Windows OS in mind. The game has been tested and confirmed to work on both Windows 7 and Windows 10. Other versions of Windows have not been tested with the game such as Windows 8.1, but should run without any issues.

2.1.8 Usability requirements

There are a number of different usability requirements that are relevant to this project. The following requirements should be adhered to:

Understandability:

- UI elements such as menus, fonts and in-game HUD (heads-up display) should be clearly visible to the player and easy to understand.
- The player should understand how to play through the main portion of the game and also how to access secondary features of the game.

Learnability:

- Special actions should be context sensitive e.g. button prompts
- Tips or hints should be provided in the game to aid the player

Operability:

- All actions described in documentation and through in-game control options should work as described.
- Options such as graphics should be customizable in order to meet the user's needs.

Attractiveness:

- Graphical fidelity should be set to a certain standard e.g. shading, lighting, textures
- UI elements should be visually appealing to the user.

2.2 Design and Architecture

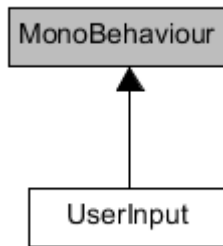
2.2.1 Use Case Diagram



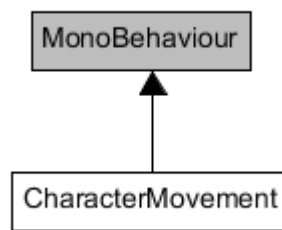
2.2.2 Class Diagram

This class diagram displays each main class individually and shows the relationship between each of these classes. The majority of these classes inherit from the MonoBehaviour class, which is a default Unity class. Some classes, however, inherit from custom made classes.

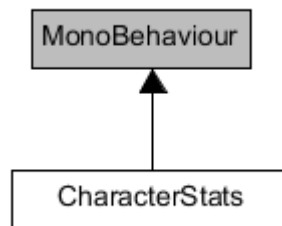
UserInput.cs



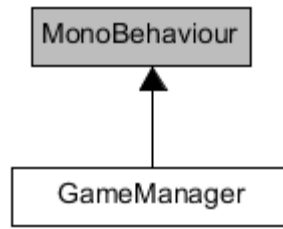
CharacterMovement.cs



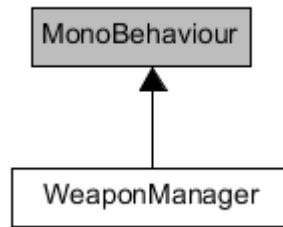
CharacterStats.cs



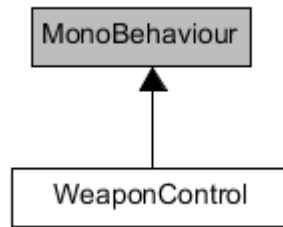
GameManager.cs



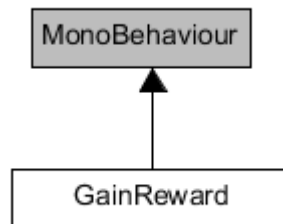
WeaponManager.cs



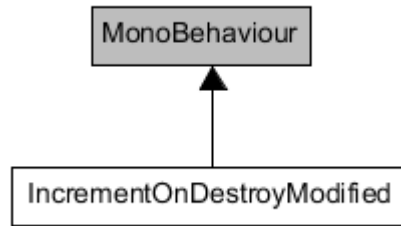
WeaponControl.cs



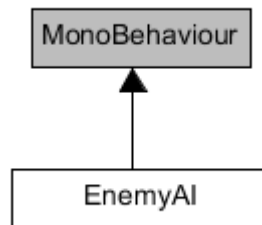
GainReward.cs



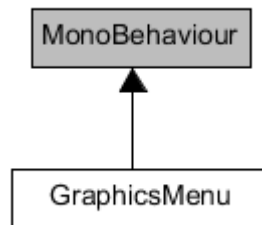
IncrementOnDestroyModified.cs



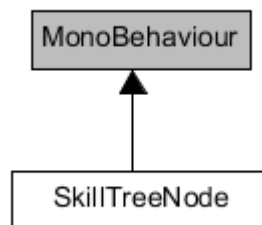
EnemyAI.cs



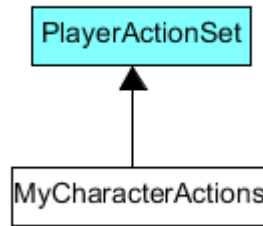
GraphicsMenu.cs



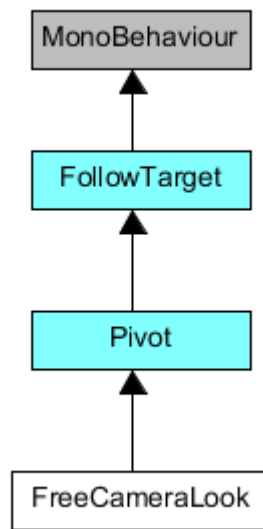
SkillTreeNode.cs



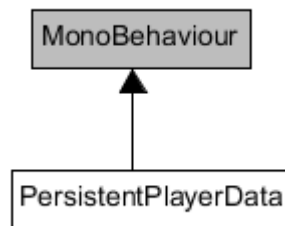
MyCharacterActions.cs



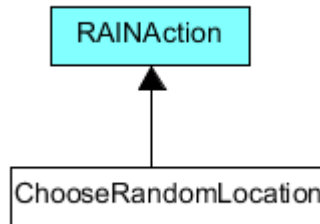
FreeCameraLook.cs



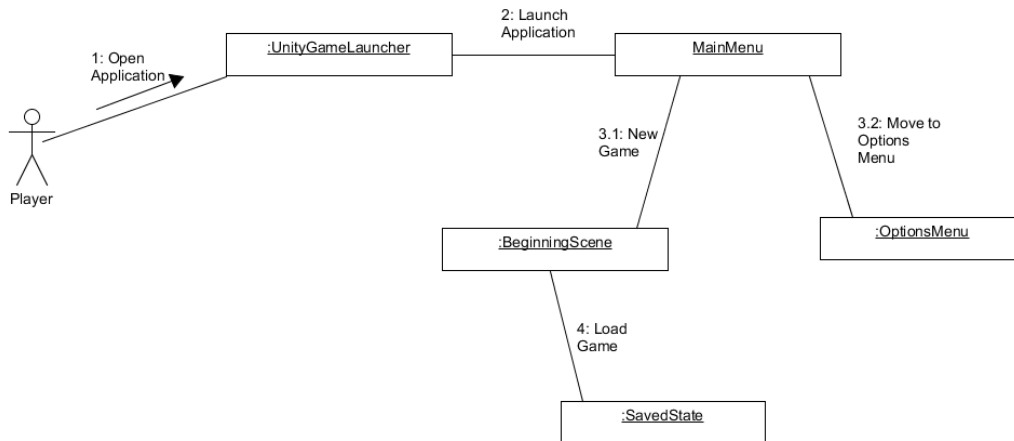
PersistentPlayerData.cs



ChooseRandomLocation.cs



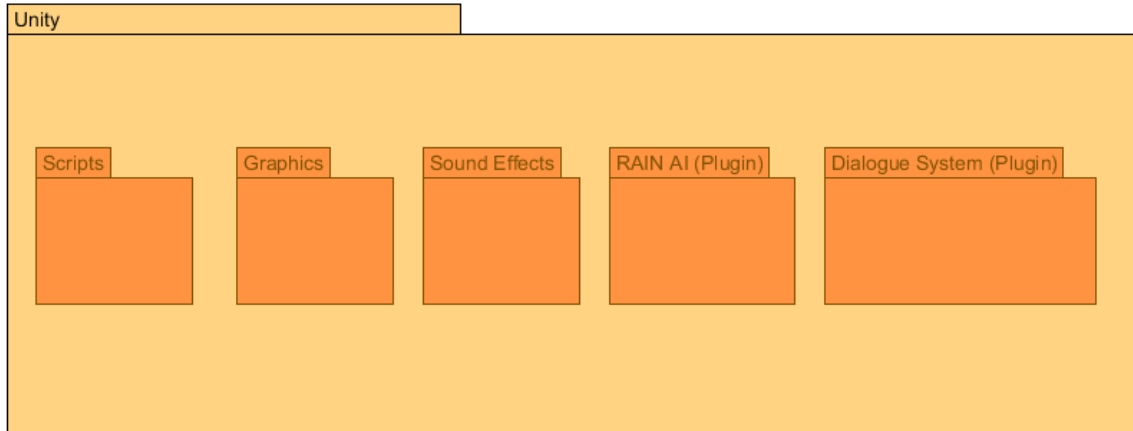
2.2.3 Logical View



A simple logical view of how the user can launch the game and choose between starting a new game or moving to the options menu to adjust graphical settings. Once the player starts the game, they can also save and load the game.

2.2.4 Software Architecture

A simple diagram detailing the software (Unity) and the primary assets (packages) it contains is outlined below.



2.2.5 Performance

In terms of in-game performance, I am currently aiming for a consistent 60 FPS. Assets will be added to the project overtime such as detailed geometry for levels, lighting effects e.g. light probes and reflection probes which can be taxing on the game's performance. Towards the end of the project's lifecycle, I will be optimizing the game to ensure that the FPS is smooth and consistent for the user. Graphical Settings can also be adjusted before the player starts the game, so they can adjust the performance of the game on their computer.

Save times and Load times should be fast so that there is not a long wait for the user to resume playing.

The Unity Profiler tool will be used to analyse performance and gather usage statistics e.g. CPU usage, RAM, GPU, Physics, animation. This tool is used to help optimize in-game performance.

2.3 Implementation

2.3.1 UserInput.cs

This class is pivotal to the game as it handles most of the player's actions through inputs by using the keyboard, mouse, or PS4 controller. It handles many actions including and not limited to: aiming, crouching, switching weapons, picking up weapons as well as special actions like rolling and slow motion.

```
if (aim)
{
    Camera.main.GetComponent<DepthOfField>().enabled = true;

    if (weaponManager.ActiveWeapon.weaponType != WeaponManager.WeaponType.Unarmed)
    {
        bool canFire = SharedFunctions.CheckAmmo(weaponManager.ActiveWeapon);

        if (!weaponManager.ActiveWeapon.CanBurst)
        {
            if (characterActions.Fire.WasPressed && !anim.GetCurrentAnimatorStateInfo(2).IsTag("Reload"))
            {
                if (canFire)
                {
                    anim.SetTrigger("Fire");
                    weaponManager.FireActiveWeapon();

                    if (!weaponManager.ActiveWeapon.silenced)
                    {
                        weaponManager.ActiveWeapon.muzzle.SetActive(true);
                    }

                    cameraFunctions.WiggleCrosshairAndCamera(weaponManager.ActiveWeapon, true);
                    // weaponManager.ActiveWeapon.curAmmo--;
                }
            }
        }
    }
}
```

This block of code is from the Update() method in the UserInput class. It is essential for allowing the player to fire their weapon. It checks if the weapon can fire automatically or semi-automatically (CanBurst) and if the player pressed the 'fire' button and the character is not currently reloading their weapon. It also checks to see the weapon is silenced or not. If it's not silenced, the weapon will emit a muzzle flash.

```

void CorrectIK()
{
    weaponType = weaponManager.weaponType;

    if(!ik.DebugAim)
    {
        switch(weaponType)
        {
            case WeaponManager.WeaponType.Pistol:
                ik.aimingZ = 362.37f;
                ik.aimingX = 3.6f;
                ik.aimingY = 35.7f;
                break;
            case WeaponManager.WeaponType.Rifle:
                ik.aimingZ = 360.7f;
                ik.aimingX = 8f;
                ik.aimingY = 46.7f;
                break;
        }
    }
}

```

This is another method in the UserInput script which is very important as it allows the user to aim correctly. It gets the spine from the character's skeleton and applies the x, y and z coordinates, defined in the script, depending on the type of weapon that the character is aiming with.

```

void VisionAction()
{
    if(characterActions.Vision.WasPressed && charStats.VisionMode)
    {
        if(!vision)
        {
            Camera.main.GetComponent<TonemappingColorGrading>().enabled = true;

            StartCoroutine(visionSoundSeq());

            foreach (GameObject go in gM.enemies)
            {
                go.GetComponentInChildren<VisionSkill>().rend.material = go.GetComponentInChildren<VisionSkill>().mats[1];
            }
        }
        else if(vision)
        {
            Camera.main.GetComponent<TonemappingColorGrading>().enabled = false;

            foreach (GameObject go in gM.enemies)
            {
                go.GetComponentInChildren<VisionSkill>().rend.material = go.GetComponentInChildren<VisionSkill>().mats[0];
            }

            audS.clip = null;
        }

        vision = !vision;
    }
}

```

This is the 'VisionAction' method. This method works if the 'Vision' skill is unlocked by the player from the skill tree. The method works as a toggle,

using a boolean value. If the player presses the button used to activate the skill, it checks if the boolean is set to 'false'. If so, the skill will activate. Otherwise, the skill will be disabled.

2.3.2 CharacterMovement.cs

This class is responsible for handling the character motor and the physics of the character. It is also used to initialize and update the Animator component attached to the player character. It handles extra features such as footsteps, using Inverse Kinematics (IK). It determines which foot is stepping on the floor when the player moves and plays a sound accordingly. This class is also used to handle the 'Stealth Takedown' system which allows the character to approach an enemy from behind and knock them out, without having to use a weapon.

```
Vector3 relativePos = takeDownTarget.InverseTransformPoint(transform.position);  
  
// Debug.Log(relativePos);  
  
distance = Vector3.Distance(transform.position, takeDownTarget.position);  
// Debug.Log(distance);  
  
if (Vector3.Angle(transform.forward, takeDownTarget.forward) < 90.0f)  
{  
    // Debug.Log("angle");  
  
    if (relativePos.z < 0.0)  
    {  
        if (canTakeDown && takeDownTarget != null && takeDownTarget.GetComponent<EnemyAI>() && distance <= 2.0f && takeDownTarget.GetComponent<EnemyAI>().health > 0)  
        {  
            if (uS.characterActions.Fire)  
            {  
  
                wM.ActiveWeapon.equip = false;  
  
                wM.weaponNumber = 0;  
  
                takingDown = true;  
                createRanInt = false;  
  
                enAnim.gameObject.GetComponent<EnemyAI>().takenDown = true;  
                enAnim.gameObject.GetComponent<EnemyAI>().health = 0;  
                enAnim.gameObject.GetComponent<EnemyAI>().rig.AI.WorkingMemory.SetItem("health", enAnim.gameObject.GetComponent<EnemyAI>().health);  
  
            }  
        }  
    }  
}
```

This is a code block contained within the 'Update' method of the CharacterMovement class. It is used for the Stealth Takedown system. It's used to calculate the angle between the player character and the enemy that the player is facing. It checks to see if the player is facing the back of the enemy. If so, the takedown sequence is initiated and the animations for the player and enemy are called.

```

// Check if the foot position is
bool CheckFootPosition(bool left)
{
    AvatarIKGoal ikGoal = left ? AvatarIKGoal.LeftFoot : AvatarIKGoal.RightFoot;
    float footBottomHeight = left ? anim.leftFeetBottomHeight : anim.rightFeetBottomHeight;

    Vector3 footPos = anim.GetIKPosition(ikGoal);

    footPos -= transform.position;

    return footPos.y <= footBottomHeight + 0.003f;
}

```

This boolean method is used to check the height of the player character's feet and calculates the position of the foot relative to the position of the character. The first line of code in this method uses a ternary operator which is basically a shorter way of using if and else statements. This line of code checks to see if the 'ikGoal' is equal to 'left'. If so, then the 'AvatarIKGoal.LeftFoot' is left.

2.3.3 CharacterStats.cs

This class handles all base stats of the player such as their health, skill points, money as well as the abilities that the player has unlocked. It also has some functions which are used for health regeneration over time, when the player is hurt as well as bringing the player back to the 'hub world' scene if they die during a mission.

```

void PlayerDying()
{
    dead = true;

    StartCoroutine(PlayOneShot("Dead"));
    DialogueManager.PlaySequence("Fade(out, 3)");

    if (!playAudio)
    {
        GetComponent().PlayOneShot(deathSound);
        playAudio = true;
    }
}

```

This method handles the procedure for when the player character dies during a mission. If an enemy kills the player and their health reaches 0, the screen fades to black over 3 seconds.

```

void LevelReset()
{
    deathTimer += Time.deltaTime;
    if(deathTimer >= resetAfterDeathTime)
    {
        if (dead)
        {
            gM.LoadNewScene();
        }
    }
}

```

This is the next method that is called when the player character dies. A timer starts. After 5 seconds, the 'hub world' scene will then load.

2.3.4 GameManager.cs

The GameManager class is used for a couple of different functions, mainly for when the player moves between scenes and when the player dies when facing an enemy.

```

void OnLevelWasLoaded()
{

    Player.GetComponent<CharacterStats>().Health = 100f;
    Player.GetComponent<CharacterStats>().dead = false;
    Player.GetComponent<CharacterStats>().deathTimer = 0f;

    if(Player.GetComponent<UserInput>().CanPickUp == true)
    {
        Player.GetComponent<UserInput>().CanPickUp = false;
    }

    SpawnPoint = GameObject.FindGameObjectWithTag("SpawnPoint").transform;
    PersistentObjects[2].transform.position = SpawnPoint.position;
    AddEnemies();
}

```

This method, 'OnLevelWasLoaded' in the GameManager class, is inherited from the base class 'MonoBehaviour' which is a default Unity class. This method will always be called after a new scene has loaded. I used this method to handle any conflicts that occurred, for example, when the player dies, I need to make sure that when the player moves into a new scene, after dying, that they are alive again, so I used this method to handle the logic.

This method was also useful for setting the point that the player should spawn in, whenever a new scene was loaded.

```
void AddEnemies()
{
    enemies = GameObject.FindGameObjectsWithTag("Enemy");
}
```

This method is used to add enemies to a list in this class. It is called in the 'OnLevelWasLoaded' method. This 'AddEnemies' method works by essentially searching for GameObjects in the scene that are tagged with the name 'Enemy'. These enemy GameObjects populate the 'enemies' list inside the GameManager class. The purpose of adding enemies to this list is so that whenever the player wishes to use their special ability (Vision ability), the effect of this ability can be applied to each and every enemy on the list.

2.3.5 WeaponManager.cs

WeaponManager is a base class for handling all of the weapons that the player carries. Weapons are stored in a list and weapon types are stored in an 'enum'. This class also handles the logic for switching weapons as well as reloading the current weapon that the player is carrying. It's also used for handling the Inverse Kinematics (IK) of the character's left hand, so the hand can be positioned depending on what weapon it's carrying.

```
void OnAnimatorIK()
{
    if (weaponType != WeaponType.Unarmed)
    {
        anim.SetIKPositionWeight(AvatarIKGoal.LeftHand, IKweight);
        anim.SetIKRotationWeight(AvatarIKGoal.LeftHand, IKweight);

        Vector3 pos = ActiveWeapon.HandPosition.transform.TransformPoint(Vector3.zero);

        anim.SetIKPosition(AvatarIKGoal.LeftHand, ActiveWeapon.HandPosition.transform.position);
        anim.SetIKRotation(AvatarIKGoal.LeftHand, ActiveWeapon.HandPosition.transform.rotation);
    }
}
```

This method is used with the 'Mecanim' Animator system in Unity for the player character. It checks to see if the player is carrying a weapon, and if so, it will adjust the position and rotation of the character's left hand, depending on the transform coordinates defined in the inspector in the Unity editor.

2.3.6 WeaponControl.cs

WeaponControl is more of a sub class to WeaponManager. It is attached to each and every single weapon that the character can use. It's used to determine the amount of ammo that's currently in the weapon, it's clip size, the firing rate, the position of where the bullet should spawn from, sound effects as well as the position of the weapon when it's equipped (hands) and the position of the weapon when it's holstered (back / leg).

```
public RestPosition restPosition;
public enum RestPosition
{
    RightHip,
    Waist
}
```

To decide what position the weapon should be holstered in, we use an enum, called 'RestPosition', with the two rest positions being right hip and waist.

```
switch (restPosition)
{
    case RestPosition.RightHip:
        transform.parent = transform.GetComponentInParent<WeaponManager>().transform.GetComponent<Animator>().GetBoneTransform(HumanBodyBones.RightUpperLeg);
        break;
    case RestPosition.Waist:
        transform.parent = transform.GetComponentInParent<WeaponManager>().transform.GetComponent<Animator>().GetBoneTransform(HumanBodyBones.Spine);
        break;
}
```

Inside the Update() method in the script, we use a switch statement along with the enum to position the weapon on either the right hip or waist. To do this, we get the transform of the spine (Waist) and transform of the right upper leg (RightHip).

```
transform.parent = transform.GetComponentInParent<WeaponManager>().transform.GetComponent<Animator>().GetBoneTransform(HumanBodyBones.RightHand);
transform.localPosition = EquipPosition;
transform.localRotation = Quaternion.Euler(EquipRotation);
```

These lines of code are used to correctly assign the position of the weapon, when the weapon the character is using is equipped. It sets the position of the weapon to the characters right hand and also sets the rotation of the weapon to whatever coordinates are defined by the developer in the inspector.

2.3.7 GainReward.cs

This is a custom class that coincides with the 'Dialogue System for Unity' plugin. The 'PixelCrushers.DialogueSystem' library is imported into this class to access all the features of the Dialogue System Plugin.

```
public void GiveReward(double amount, double cash)
{
    charStats.SkillPoints += (int) amount;
    charStats.Credits += (int) cash;
}
```

This is the main method of the class. It uses parameters, in this case, skill points and credits (money) parameters using the 'double' data type. Using the Dialogue System plugin, you input the amount of credits / skill points the user should be rewarded upon completing a mission.

2.3.8 IncrementOnDestroyModified.cs

This class is based on the 'IncrementOnDestroy' class that is included with the Dialogue System for Unity plugin that I modified, for the purposes of this project. This class is attached to gameobjects that are part of a mission in the game that may need to be destroyed by the player. The primary purpose of the class is to check when an object has been destroyed. If so, an integer is incremented.

```
/// <summary>
/// When this object is destroyed, increment the counter and update the quest tracker.
/// </summary>
public void OnDestroy() {
    if (!listenForOnDestroy) return;
    int oldValue = DialogueLua.GetVariable(ActualVariableName).AsInt;
    int newValue = Mathf.Clamp(oldValue + increment, min, max);
    DialogueLua.SetVariable(ActualVariableName, newValue);

    if (newValue >= max && QuestLog.GetQuestState(questName) == QuestState.Active)
    {
        QuestLog.CompleteQuest(questName);
        FindObjectOfType<GainReward>().GiveReward(skillPointReward, creditReward);
        DialogueManager.ShowAlert(alertMessageComplete, 7f);
        AudioManager.instance.PlaySoundOneShot(FindObjectOfType<GameManager>().GetComponent<AudioSource>(), missionComplete);
    }

    DialogueManager.SendUpdateTracker();
    if (newValue < max)
    {
        if (!(string.IsNullOrEmpty(alertMessage) || DialogueManager.Instance == null))
        {
            DialogueManager.ShowAlert(alertMessage);
        }
    }
}
```


This is the main method of the class. The method constantly checks to see if the gameobject has been destroyed. It uses a condition which checks to see if the integer value is greater than or equal to the maximum value of objects that need to be destroyed. If so, then the Quest Log is updated and sets the current mission to 'complete'. The player's 'CharacterStats' script is then updated, by increasing the skill points and credits in the class, as a reward for completing the mission. The Dialogue Manager object will then display an alert with a string that can be defined by the developer in the inspector in the Unity editor.

If, however, the object with this script is destroyed and the integer value is less than the maximum value, the Dialogue Manager will display an alert detailing the current amount of objects that have been destroyed and the amount remaining that need to be destroyed.

2.3.9 EnemyAI.cs

This class is essentially the main class for all enemies in the game. This class also refers to variables stored in the AI rig of the enemy which is used with the RAIN AI plugin. The weapon system works differently for AI agents, so variables such as 'attackRate' and 'attackTimer' which are 'float' data types are stored here as well as the current ammo. Agents don't require complex weapon systems that the player character requires.

The 'Attack' logic is also stored in this class which allows the enemy to fire bullets whenever the player is in its line of sight.

```
if (rig.AI.WorkingMemory.GetItem<bool>("canFire") == true && !anim.GetCurrentAnimatorStateInfo(2).IsTag("Reload"))
{
    Attack();
}
```

This condition checks if the boolean "canFire" is true, which is stored in the 'memory' of the agent's AI rig. It also checks the agent's animator component to see if the "Reload" state is not currently running. If both of these conditions return true, the Attack() method is called.

```

void Attack()
{
    enemyToAttack = rig.AI.WorkingMemory.GetItem<GameObject>("varPlayer");

    weaponManager.aim = true;

    Vector3 direction = enemyToAttack.transform.position - transform.position;
    float angle = Vector3.Angle(direction, transform.forward);

    transform.LookAt(enemyToAttack.transform.position);

    attackTimer += Time.deltaTime;

    if (attackTimer > attackRate && enemyToAttack.GetComponent<CharacterStats>().Health > 0)
    {
        ShootRay();
        attackTimer = 0;
    }
}

```

This is the Attack() method. It assigns the GameObject variable “enemyToAttack” as the ‘varPlayer’ GameObject variable that is stored in the memory of the agent’s AI rig. This method also calculates the current direction of the player and the agent will look at wherever the player is moving. The agent will then fire, as long as the player’s health is greater than 0.

2.3.10 GraphicsMenu.cs

The GraphicsMenu script is used for changing the settings of various graphical options, from the main menu of the game. It allows the user to modify the resolution, overall graphics quality, anti-aliasing and v-sync. These are modified by selecting values on a dropdown list.

```

qualitySettingsDropdown.onValueChanged.AddListener(
    delegate //sets option by calling 'SetQualityLevel();
    {
        SetQualityLevel();
    }
);

antiAliasingDropdown.onValueChanged.AddListener(
    delegate
    {
        SetAntiAliasing();
    }
);

vSyncCountDropdown.onValueChanged.AddListener(
    delegate
    {
        SetVSyncCount();
    }
);

resolutionDropdown.onValueChanged.AddListener(
    delegate
    {
        SetResolution();
    }
);

```

These 'delegates' are initialized in the Start() method of the script meaning that they will be called when the scene that has this script is loaded. These delegates are used for encapsulating references to the methods that are inside the delegate objects. These delegate objects can then call the referenced method. The purpose of these delegate objects for this script is so that the different dropdown lists that need to have a number of different values can be populated instantly.

```

public void SetAntiAliasing()
{
    switch(antiAliasingDropdown.value)
    {
        case 0:
            QualitySettings.antiAliasing = 0;
            break;
        case 1:
            QualitySettings.antiAliasing = 2;
            break;
        case 2:
            QualitySettings.antiAliasing = 4;
            break;
        case 3:
            QualitySettings.antiAliasing = 8;
            break;
    }
}

```

This is one of the methods called in one of the delegate objects (antiAliasingDropdown.OnValueChanged.AddListener). It uses a switch statement. The switch is the value that is to be assigned, in this case, each value in the drop down list.

2.3.11 SkillTreeNode.cs

The SkillTreeNode script is a component that is attached to each 'skill button' in the skill tree user interface. Its primary function is to call a method that unlocks the skill when the user clicks on the button. It's also used to display information about the skill such as skill points required to unlock it.

```
public void PurchaseSkill()
{
    if(charStats.SkillPoints >= cost && !unlocked)
    {
        charStats.SkillPoints -= cost;
        unlocked = true;
        lockIcon.SetActive(false);
        switch (skillname)
        {
            case "Dodge Roll":
                FindObjectOfType<CharacterStats>().DodgeRoll = true;
                break;
            case "Vision":
                FindObjectOfType<CharacterStats>().VisionMode = true;
                break;
            case "Sleight of Hand":
                FindObjectOfType<UserInput>().GetComponent<Animator>().SetFloat("ReloadSpeed", charStats.sleight);
                break;
            case "Rapid Regen":
                FindObjectOfType<CharacterStats>().regenPoints = 2.0f;
                break;
            case "Focus":
                FindObjectOfType<CharacterStats>().FocusMode = true;
                break;
        }
    }
}
```

This is the method used to unlock the skill that is selected by the user. It uses a condition to check first that the player has enough skill points to unlock the desired skill and checks the boolean (unlocked) to make sure that the player can't keep clicking the button as they will keep losing skill points even though they already unlocked the skill.

In order to determine which skill should be unlocked when the user clicks on the skill button, this method uses a switch statement and uses a string variable as the switch. For example, if the user clicks on the button and string 'skillname' is

“Vision”, it will choose ‘case “Vision”’ from the switch statement as the skill to be unlocked.

2.3.12 MyCharacterActions.cs

MyCharacterActions is a custom class made to inherit from the ‘PlayerActionSet’ class provided by the ‘InControl’ plugin, which contains a set of player actions that the player can bind to their input device e.g. PS4 controller, Keyboard.

```
public PlayerAction Left;
public PlayerAction Right;
public PlayerAction Up;
public PlayerAction Down;
public PlayerAction Aim;
public PlayerAction Fire;
public PlayerAction Focus;
public PlayerAction Roll;
public PlayerAction SwitchWeapon;
public PlayerAction Pickup;
public PlayerAction Pause;
public PlayerAction Vision;
public PlayerAction Reload;
public PlayerAction Interact;
public PlayerAction ToggleSilencer;
```

This is from the MyCharacterActions script, where the character actions are created. Each action is created from the ‘PlayerAction’ class which is from the ‘InControl’ library.

```
public MyCharacterActions()
{
    Left = CreatePlayerAction("Move Left");
    Right = CreatePlayerAction("Move Right");
    Up = CreatePlayerAction("Move Up");
    Down = CreatePlayerAction("Move Down");
    Aim = CreatePlayerAction("Aim");
    Fire = CreatePlayerAction("Fire");
    Roll = CreatePlayerAction("Roll");
    Focus = CreatePlayerAction("Focus");
    Vision = CreatePlayerAction("Vision");
    Interact = CreatePlayerAction("Interact");
}
```

This method: MyCharacterActions, is where each of the actions from above are constructed and given a name, using a string. This class is instantiated and these actions in this method are called from the UserInput class.

2.3.13 FreeCameraLook.cs

The FreeCameraLook script is used to handle most of the main camera functions. It's used to get the camera in the scene to lock on to the player, follow the player whenever the player moves and also handle camera rotation. It's also used in conjunction with the WeaponControl script for weapon recoil.

```
void HandleRotationMovement()
{
    if (canControl) // if (fDemo.isMenuOpen == false && canControl)
    {
        handleOffsets();

        var inputDevice = InputManager.ActiveDevice;

        float x = inputDevice.RightStickX + offsetX;
        float y = inputDevice.RightStickY + offsetY;

        float xMouse = Input.GetAxis("Mouse X") + offsetX;
        float yMouse = Input.GetAxis("Mouse Y") + offsetY;

        if (turnsmoothing > 0)
        {
            smoothX = Mathf.SmoothDamp(smoothX, x, ref smoothXvelocity, turnsmoothing);
            smoothY = Mathf.SmoothDamp(smoothY, y, ref smoothYvelocity, turnsmoothing);

            smoothX = Mathf.SmoothDamp(smoothX, xMouse, ref smoothXvelocity, turnsmoothing);
            smoothY = Mathf.SmoothDamp(smoothY, yMouse, ref smoothYvelocity, turnsmoothing);
        }
    }
}
```

This is the method used for handling camera rotation, whenever the user moves the mouse or analog stick on the PS4 controller. This is done by assigning x and y 'float' variables to the x and y Inputs of both the mouse and controller.

2.3.14 PersistentPlayerData.cs

PersistentPlayerData is the main script used for Save and Load operations within the game. Its purpose is to keep objects and variables defined in the script persistent. For example, when the player saves, it'll record the current value of all of the variables defined in the script at that point in time. Whenever the player loads the game, it will apply the saved value of those variables defined in the script.

The script works in conjunction with the 'Game Saver' component which is part of the Dialogue System for Unity plugin. This Game Saver component is attached to the 'Dialogue Manager' gameobject in the scene. The PersistentPlayerData script

is also attached to this same gameobject with the Game Saver component so the PersistentPlayerData script can find the Game Saver component easily.

There are two main methods used in the PersistentPlayerData script: OnRecordPersistentData() and OnApplyPersistentData().

```
public void OnRecordPersistentData()
{
    if (!FindComponents()) return; //[PixelCrushers]

    // Add your code here to record data into the Lua environment.
    // Typically, you'll record the data using a line similar to:
    // DialogueLua.SetActorField(name, "myFieldName", myData);

    DialogueLua.SetActorField(playerActorName, "health", charStats.Health);
    DialogueLua.SetActorField(playerActorName, "skill", charStats.SkillPoints);

    DialogueLua.SetVariable("credits", charStats.Credits);
    DialogueLua.SetVariable("vision", charStats.VisionMode); //
    DialogueLua.SetVariable("focus", charStats.FocusMode);
    DialogueLua.SetVariable("sleight", charStats.sleight);
}
```

In the OnRecordPersistentData() method, we set the values of the variables we want to save and assign them names, using strings. These are then recorded and stored in the 'Lua' environment in the Dialogue System database.

```
public void OnApplyPersistentData()
{
    if (!FindComponents()) return; //[PixelCrushers]

    // Add your code here to get data from Lua and apply it (usually to the game object).
    // Typically, you'll use a line similar to:
    // myData = DialogueLua.GetActorField(name, "myFieldName").AsSomeType;

    charStats.Health = DialogueLua.GetActorField(playerActorName, "health").AsFloat;
    charStats.SkillPoints = DialogueLua.GetActorField(playerActorName, "skill").AsInt;

    charStats.Credits = DialogueLua.GetVariable("credits").AsFloat;
    charStats.VisionMode = DialogueLua.GetVariable("vision").AsBool;
    charStats.FocusMode = DialogueLua.GetVariable("focus").AsBool;
    charStats.sleight = DialogueLua.GetVariable("sleight").AsFloat;
}
```

In the OnApplyPersistentData() method, we get the variables from the object that we stored in the Dialogue System Lua environment and apply those values that were stored to the object. We can also choose what type of data type to apply them as e.g. AsFloat, AsInt, AsBool.

2.3.15 ChooseRandomLocation.cs

ChooseRandomLocation is an example of a custom action that is created for use in the RAIN behaviour tree, for an agent. The main function of this script is to allow the AI agent to search the map, within a certain distance defined by the developer, when the player character is out of the line of sight of the enemy after the player is discovered.

This script uses a timer, which is created by using a float variable called `_startTime`.

```
public override void Start(AI ai)
{
    base.Start(ai);

    _startTime += Time.time;
}
```

On this start method, the timer will then increment by `Time.time`, which is the time at the beginning of the frame.

```
public override ActionResult Execute(RAIN.Core.AI ai)
{
    // Chooses any graphs at our AI's position
    List<RAINNavigationGraph> tGraphs = NavigationManager.Instance.GraphForPoint(ai.Kinematic.Position);
    if (tGraphs.Count == 0)
        return ActionResult.FAILURE;

    // Guaranteed to be a navigation mesh
    NavMeshPathGraph tGraph = (NavMeshPathGraph)tGraphs[0];

    // Look for polys within the given distance
    float tDistance = 20;
    if (Distance.IsValid)
        tDistance = Distance.Evaluate<float>(ai.DeltaTime, ai.WorkingMemory);

    // Uses OctTree (RAIN.Utility.OctTree)
    List<NavMeshPoly> tPolys = new List<NavMeshPoly>();
    tGraph.PolyTree.GetCollisions(new Bounds(ai.Kinematic.Position, Vector3.one * tDistance * 2), tPolys);

    if (tPolys.Count == 0)
        return ActionResult.FAILURE;

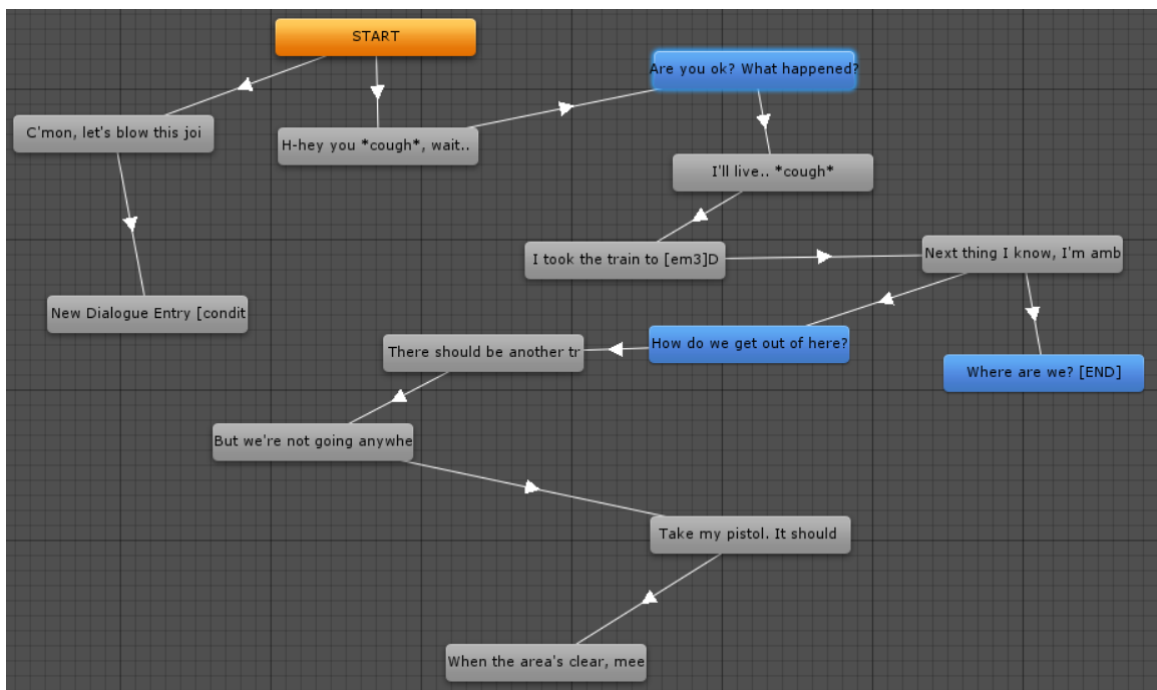
    // Picks a random node
    NavMeshPoly tRandomPoly = tPolys[UnityEngine.Random.Range(0, tPolys.Count - 1)];

    // If the user set a Target variable, use it
    if (Target.IsVariable)
        ai.WorkingMemory.SetItem<Vector3>(Target.VariableName, tRandomPoly.Position);
    // Otherwise just use some default
    else
        ai.WorkingMemory.SetItem<Vector3>("randomLocation", tRandomPoly.Position);
}
```


This is part of the main method of the script. This method (ActionResult Execute), runs whenever the action node that has this script attached to it runs. The purpose of this is to find polys on the navigation mesh on the map, within the specified distance by the developer, and then assign a target position for the AI agent to move towards on this navigation mesh. A navigation mesh is essentially, a flat plane made out of polygons that specifies which parts of the map / terrain can be traversed by the agent.

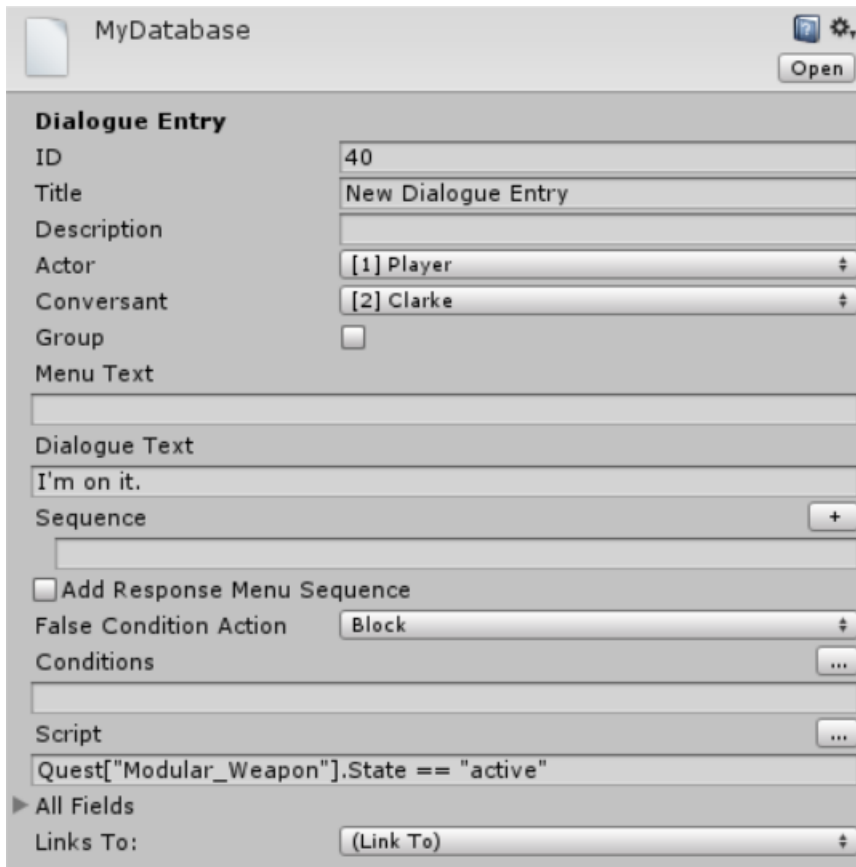
Once the timer in this script exceeds the value specified by the developer, then the agent moves from its search state to the patrol state.

2.3.16 Dialogue Tree (Dialogue System for Unity)



This is a Dialogue Tree that's used with the 'Dialogue System for Unity' plugin. It allows the player to interact with an NPC. For this game, the intended use of this Dialogue Tree is for the player to be able to trigger missions by interacting with the NPC. Each node in the Dialogue Tree contains text fields. The developer inputs

whatever text they want to appear into the node. Whenever the player interacts with the NPC and a node fires, this text that was put into the node will be displayed. The blue nodes in the Dialogue Tree are nodes that the player can choose to select when the player is given the option during conversation, usually as a response.

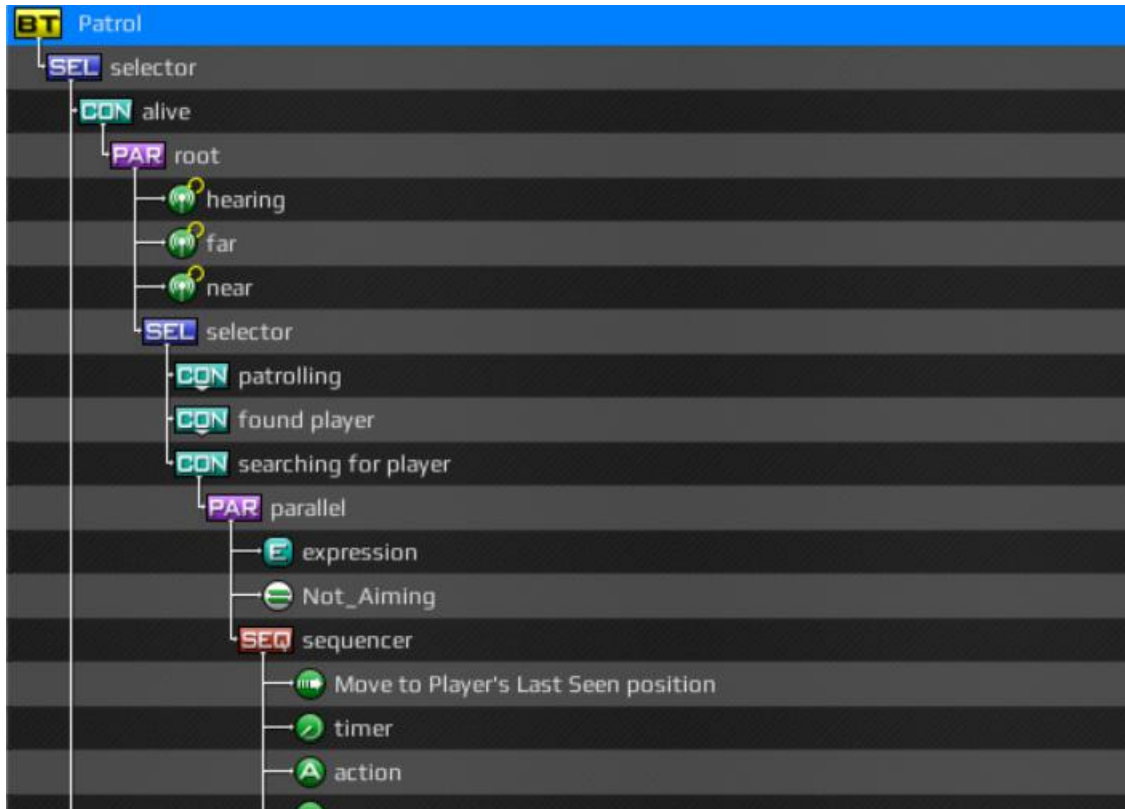


The image shows a Unity inspector window titled "MyDatabase" with an "Open" button. The main section is "Dialogue Entry" and contains the following fields:

- ID: 40
- Title: New Dialogue Entry
- Description: (empty)
- Actor: [1] Player
- Conversant: [2] Clarke
- Group:
- Menu Text: (empty)
- Dialogue Text: I'm on it.
- Sequence: (empty) with a "+" button
- Add Response Menu Sequence
- False Condition Action: Block
- Conditions: (empty) with a "..." button
- Script: (empty) with a "..." button
- Script text: Quest["Modular_Weapon"].State == "active"
- ▶ All Fields
- Links To: (Link To)

When a node is selected from the Dialogue Tree, this inspector is displayed in the Unity editor. It allows the developer to modify the contents of the node in the tree. Fields in this node can be modified. For example, in the 'Conditions' field, the developer can set it so that this node will only become visible in the dialogue tree for the player if a certain condition is met e.g. the state of a quest is set to 'success'. This inspector also contains a field called 'Sequence'. With this sequence field, you can input code into the field, by using 'Sequencer Commands', which are built-in commands from the Dialogue System for Unity plugin. An example of a Sequencer Command would be enabling an object in the game scene e.g. `SetActive(object_name, true);`

2.3.17 Behaviour Tree (RAIN AI)



This is a segment of the behaviour tree used for the enemy characters in the game. The behaviour tree is composed of nodes such as sequencers, constraints, selectors and parallels. Sequencer nodes run actions in sequence e.g. character moves to target position. Character then pauses for 5 seconds. Character plays an animation. Parallel nodes allow actions to run at the same time e.g. enemy walks towards player while firing weapon. Other nodes, such as Audio and Visual, allow the AI agent to hear entities within a specified radius and see entities within a specified line of sight. For more complex behaviour that cannot be done through the default nodes used with RAIN, custom action / decision nodes need to be added to the behaviour tree. These actions / decisions are essentially C# scripts, but instead of inheriting from MonoBehaviour, the script uses the RAIN library to

inherit from scripts such as 'RAINAction', if the custom node is an action node, or 'RAINDecision', if the custom node is a decision node.

2.4 Testing

2.4.1 Unit Testing

Testing was a crucial part of the development process of this project, as there were many variables and functions involved. Unit testing was a part of the development process from the start. Every time I implemented a new function or added a variable such as a boolean that would fire every time the function executed, I would add a 'Debug.Log' message to the function.

Example:

```
if (Physics.Raycast(ray, out hit, weaponManager.ActiveWeapon.range))
{
    Debug.Log(hit.collider.name);

    float distance = Vector3.Distance(weaponManager.ActiveWeapon.bulletSpawn.transform.position, hit.point);

    RaycastHit[] hits = Physics.RaycastAll(startPos, hit.point - startPos, distance);
```

The line 'Debug.Log(hit.collider.name);' essentially, sends a message to the console in the Unity editor, displaying a message. With this function in the screenshot, the player's weapon fires a 'raycast' which is an invisible line. Without using Debug.Log messages, it's very difficult to determine if the 'raycast' actually hit anything or not. So, to check if the 'raycast' did hit something, we add a Debug.Log message of 'Debug.Log(hit.collider.name)'. When a 'raycast' is fired and the 'raycast' hits an object that has a collider attached to it, the Debug.Log will

print the name of the object that was hit by a 'raycast' that has a collider attached to it.

As part of the Unit Testing process, I also took advantage of Unity and Visual Studio's collaborative features. Visual Studio 2015 IDE now has dedicated Unity features, so I decided to make use of this. To apply this, you select the "Debug > Attach Unity Debugger in Visual Studio" option in the Visual Studio IDE. You then set a breakpoint on the method that you need to check. When you start running the game, once the method in the script that has the breakpoint set is called, Unity will pause and the Debugging feature will start running. You then open the IDE and press F11 to move through each line of code and see what's happening. The console window in the Visual Studio IDE will display the status of each gameobject with the script that's attached to the gameobject and display variables and other data types that are contained within the script, depending on what line of code is highlighted after pressing F11. I found this to be an effective method of testing for problems in scripts in the event that there were no error messages, but functions were not working as intended. This allowed me to isolate individual lines of code and see what is occurring as each line is executed.

2.4.2 Black-Box / Functional Testing

The purpose of Black-Box testing, was for me to see how the game functioned without observing the internal workings of the game. In other words, I tested it from the point of view of a customer as opposed to a developer, testing the playability of the game. This involved multiple playthroughs of the game, where I would keep an eye on certain functions and take note. For example, I had some trigger boxes placed throughout a scene. I needed to ensure that whenever the player character walks into that trigger, the function executes as intended. This could be something simple as a pop up window displaying how to exit the level whenever the player is inside a trigger or something more complex such as a cut scene playing out with multiple camera angles running in sequence whenever the player enters a trigger.

Extensive testing of the user interface took place, especially in regards to the weapon shop in the game, where the logic needed to work as intended. Whenever the player enters the weapons shop and approaches the counter and press a button, a panel will appear, and the user can navigate around the panel, highlighting different buttons which display the weapon to purchase. Whenever the player selects the button, the gun will be purchased and the player's current amount of money will be depleted by whatever the cost of the weapon was. If they didn't have enough money, the player would be unable to receive the weapon. I set a large value of money in the Unity editor for the player, before starting the game and repeatedly purchased various weapons and kept track of the amount of money the character had and ensured that when they had no money, they could no longer purchase. I applied this same principle to the skill tree also, as it carried the same logic. Instead of money, the player spends skill points.

The artificial intelligence was also tested extensively. I did this by observing the behaviour of each agent in the scene, ensuring that they patrolled their assigned waypoints, paused periodically and searched the area when in their search state after discovering the player and attacked the player when in their line of sight.

Black-Box testing was also an integral part of customer testing. If any of the assigned testers encountered any bugs during a playthrough of the game, this was noted and the bug was then rectified.

2.4.3 Performance Testing

For performance testing, I took advantage of Unity's built-in Profiler tool. It is used as a means for helping narrow down your search for performance bottlenecks by generating usage and statistics reports on all components in the game scene during runtime. The Profiler tool monitors various aspects during runtime such as CPU usage, Rendering, Memory, Physics processing and more. Functions in scripts that are constantly running via an 'Update' method in a script are displayed in an Overview window in the Profiler tool and it gives the total percentage of its usage in aspects such as CPU usage. This makes it easier for the developer to see which function could possibly be slowing the game down as some functions

that run in Update can cause slowdown and cause framerate drops, which affects how smooth the game runs. Performance testing was a vital part of the testing process as I needed to ensure that the game played at a consistent 60 frames per second as this is seen as the standard for almost all PC games nowadays. Many console games run at 30 frames per second, due to hardware limitations. PC games are capable of much higher framerates so it was important that I achieved this level of performance for the game in order to provide a smooth gameplay experience for the user.

2.5 Graphical User Interface (GUI) Layout

2.5.1 Pause Menu



This is the pause menu that the user can activate at any time during the game. It allows the user to access the quest log window, skill tree panel, save the game, load the game and check the controls for the game. The user can also close the

pause menu and resume playing the game or quit the game and return to the main menu.

2.5.2 Quest Log Window



This is the Quest Log window, which is part of the Dialogue System for Unity plugin. It is accessed through the pause menu. The user is able to keep track of all quests they have completed as well as quests that are currently active. Depending on the type of mission, the player can track the status of the mission e.g. amount of objects destroyed.

2.5.3 Skill Tree panel



This is the Skill Tree panel. It is also accessed through the pause menu. From here, the player can view the current amount of skill points they have, the cost of the skill they want to acquire, the name of the skill and its description. When the user's mouse pointer hovers over a skill node, the skill's name and cost will be displayed along with a skill preview video that plays in the Skill Tree menu.

2.5.4 Player HUD



This is the main HUD (Heads-Up Display) for the player character. It displays the current type of weapon the player has equipped, the number of bullets per clip, health bar and 'Focus' bar.



This is the silencer gauge which is also part of the HUD. It displays the durability of the silencer that is attached to the current weapon that the player is using. It uses a slider component from the Unity UI system. Whenever the player fires a shot with their weapon, the slider will decrease. Once the value of the slider reaches zero, the player's silencer will be destroyed and the weapon can then be heard by enemies when fired.

2.5.5 Main Menu



This is the main menu screen. From here, the player can start a new game, access the options menu or exit the game which will close the application.

2.5.6 Options Menu



This is the options menu, accessed from the main menu. From this menu, the player can alter the level of various graphical options. Graphical settings that can be modified include: overall graphics quality, anti-aliasing, v-sync and resolution.

2.5.7 Weapon store panel



This is the UI for the weapon store, where the player can visit to purchase weapons, ammunition and silencers. The price and name of the weapon / item is displayed using text components. The current amount of credits that the player has is also displayed in the panel. This UI is displayed on a 'World Space' canvas instead of the usual 'Screen Space – Overlay' canvas which is generally used for UI elements such as the player HUD e.g. health, ammo.

2.5.8 Dialogue UI



This is the UI that is visible to the player when they engage in conversation with an NPC. This UI is part of the Dialogue System for Unity plugin. Over the course of the game, the player interacts with a key NPC where they will assign missions to the player. The player approaches this NPC to select the mission they wish to partake in. The rewards for the completion of the mission are described in this Dialogue UI. Once the player selects the mission, they will be given a choice whether they wish to start the mission or not. If they accept the mission, the character is moved to the mission scene.

2.6 Customer testing

A number of tests were carried out towards the end of the development timeline of the project. I had 4 participants test out various aspects of the project. In terms of demographics, 2 of the users were avid gamers and had experience with games of this genre and various other genres e.g. PC and console games (Fallout 4, GTA V, Metal Gear Solid), while the other 2 users were more casual gamers, who would be more accustomed to 'pick up and play' games e.g. mobile games (Candy Crush, Clash of Clans).

Each participant was asked to fill out a survey, which I created using SurveyMonkey. The individual ratings of each survey as well as the link to the

survey can be viewed in the appendix, in the 'Other Material Used' section. The main purpose of this customer testing was for a way to stress test the game, so that the overall performance of the game could be evaluated (e.g. framerate). It was also a good way of testing for any game-breaking bugs / glitches that may have possibly gone unnoticed, and lastly, a way to gain feedback on the game as a whole.

After initial testing, reception was fairly positive. Different aspects such as the user interface, weapon mechanics and more were well received. These aspects are further elaborated in the 'Evaluation' section.

There was a bug that occasionally occurred, but only when the build of the project was being tested and went unnoticed when tested in the Unity editor. The issue was in relation to the 'Dialogue System for Unity' plugin whereby a cut scene sequence can be triggered when the player enters a trigger collider. This allows the camera in the scene to move to angles specified by the developer, for the cut scene. This worked fine in the Unity editor, but in the project build, it had issues and the camera in the scene would essentially break, rendering the game unplayable. This issue has been rectified since and no longer occurs.

Another issue that cropped up during customer testing was a performance related issue. This was in relation to the skill tree system, whereby the user can open up a panel and purchase a skill. When a skill button on the panel is highlighted, a preview video of the skill is displayed, using a 'MovieTexture' object which allows videos to be played in Unity. The problem was that whenever some skill buttons were highlighted on the panel, and videos were displayed, there would be a massive dip in the framerate of the game, so much so, that it would drop from 60 frames per second to below 30 frames per second. This of course affected the playability of the game. I was able to find the root cause of this myself and rectify the issue by using the Unity Profiler tool, which is used to measure performance and help optimize the game. This tool is further explained in the 'Evaluation' section of the report.

2.7 Evaluation

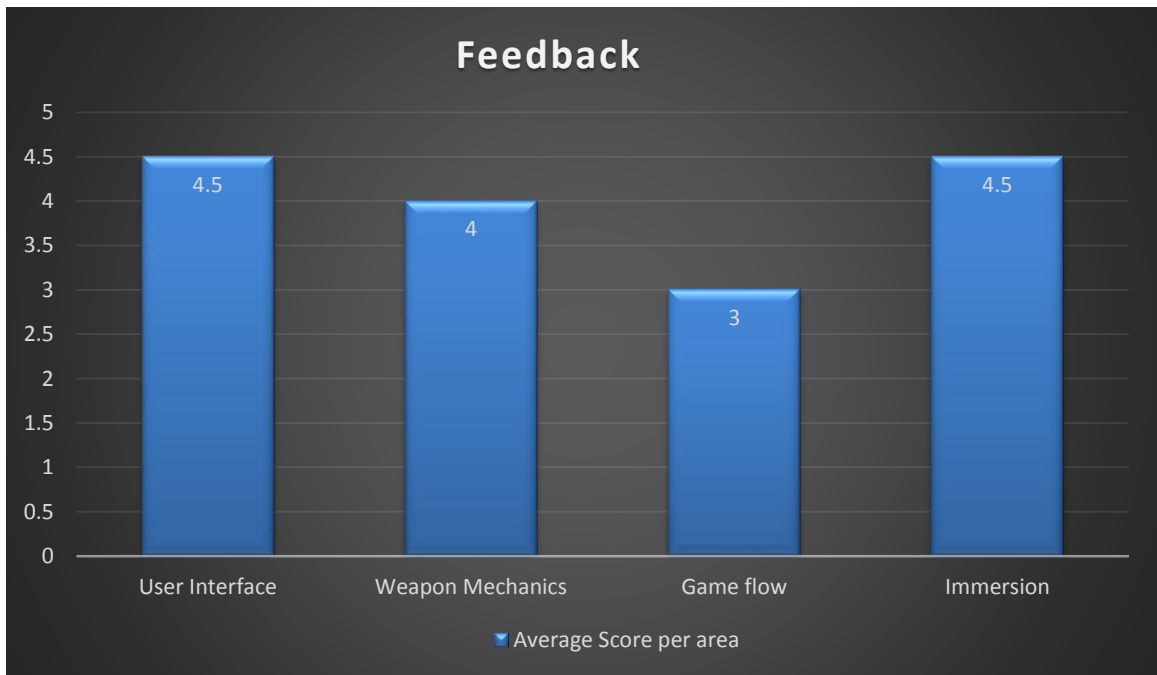


Figure 1. Customer Feedback - Using a bar chart to show average score per aspect

For customer feedback, I decided to assess four different areas of the project to get a general idea on what users felt about the game, after playing through the game for a certain period of time.

The four areas assessed were: User Interface, Weapon Mechanics, Game Flow and Immersion. After receiving the results of the survey from the participants, I got the rating of each aspect from each participant and calculated the average of each individual aspect.

For the User Interface, reception was very positive. Users felt that it was visually appealing and it blended in with the overall theme of the game and was responsive to the user's inputs e.g. little to no input lag / delay.

Weapon mechanics were well received by users also. This was an important area as the weaponry is integral to the game. Weapon mechanics include factors such as firing rate, recoil, bullet spread and more, for each weapon.

Game flow was seen as positive by users, however, it was the lowest scoring area. Some users felt that the pacing of the game wasn't quite right, in the sense that the story felt a little bit rushed e.g. through dialogue. The reason for this is mainly due to constraints such as time and the fact that other areas of the game had a bigger priority, so the story and its pacing needed to be cut down. With a full development team and more time, this is an area that could be expanded upon and one that would have more depth.

Immersion was also well received by users, in particular, elements such as cinematic cutscenes as well as the futuristic environment that kept with the tone of the game.

Other means of evaluation were also carried out during the development of the project.

After adding 'Debug.Log' messages to important methods in scripts, I would frequently play through parts of the game to ensure that no major errors appeared in the console and that there were no game breaking bugs whereby the player could not progress any further or get stuck in a loop e.g. the player gets caught in a death animation and the character cannot control the player anymore. These issues that appeared would then be rectified.

To monitor performance, I made use of the Unity Profiler tool. This is a built-in tool for Unity that helps with the optimization of your game. It gives percentages on aspects such as rendering, animations, CPU usage as well as methods that run in your code. What was also useful about this tool was that it could run in tandem with the game, in real-time in the Unity editor. So while I play tested the game, I could monitor the status of the Unity profiler and check if there was anything that was specifically causing dips in the frame rate of the game. An instance where I required the use of the profiler tool was during customer testing of the game, as mentioned in the 'Customer Testing' section, whereby the playable videos in the

skill tree panel would cause massive dips in framerate. Not all videos were causing this however, so thanks to the Unity profiler tool, I could see which video was causing the drop in framerate. I was then able to successfully find which video caused the issue and compress the video quality so that there would be no drop in framerate anymore.

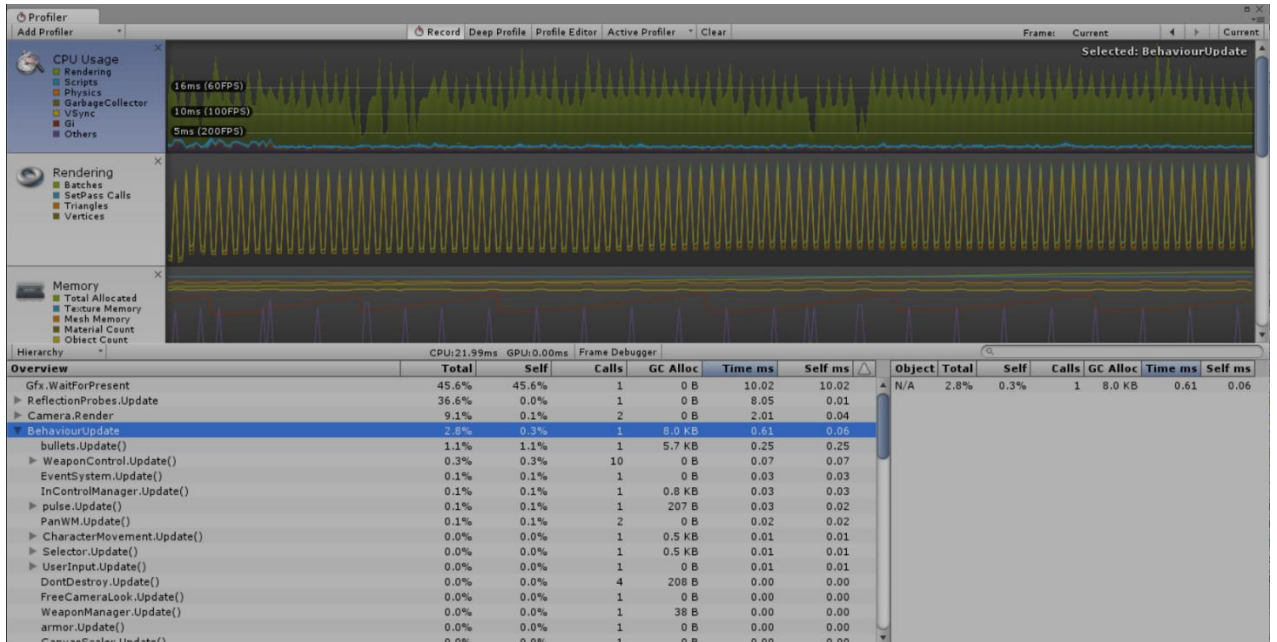


Figure 2. Screenshot of the Unity Profiler tool used at runtime

3 Conclusions

Overall, I found the development of this project to be both a challenging and enjoyable experience. My skills with the C# programming language have most certainly improved and I learned a lot from using new plug-ins that I had never used prior to the development of this project such as the RAIN AI plugin and Dialogue System for Unity.

I also learned a lot about the Unity 'Event System'. This is used for managing inputs from the keyboard / gamepad for the Unity UI. In order for the 'InControl' plugin to work correctly with menus, I needed to have an understanding of how Unity UI events work. After studying with this for some time and referring to Unity documentation, I now have a more solid understanding as to how the Unity UI Event System works.

A setback I encountered during project development was the implementation of a fully-fledged save and load system. Initially, I had planned to have it work so that the player could save their progress during the game, exit the game and be able to load their progress from the main menu, even after they had closed the application and started it again. Unfortunately, I encountered an issue whereby I could not get the game to load properly, the way I wanted it to from the main menu. I was able to load the player character and other variables that were required, but could not get important content to load such as the current weapons the player had equipped at the time that the game was saved. I suspect that the issue stemmed from a conflict within the 'WeaponManager.cs' class and not the actual 'PersistentPlayerData.cs' script which is used for saving and loading operations. So unfortunately, due to time constraints, I had to scale this feature down a bit. The alternative is now that the player can only save and load mid-game. So if the player saved in a level, they may load back at the point in which they saved, but this cannot be done from a main menu or after the application is closed.

One big limitation of the project was most notably the size of the development team. Since this project was an individual project, it was limited to one developer and was to be developed over the course of 9 months. Realistically, a fully-fledged

game of this scale and ambition would not be achievable within the allotted timeframe, especially with only one developer handling all areas of the project such as UI design, level design, gameplay mechanics and more. As a result of this, some features had to be omitted that were originally planned. Usually, within a games development team, there are different areas that handle specific aspects of the game e.g. level design department, programming department. If there were multiple developers working on this project, a lot more could have been accomplished and it could have been developed to a standard suitable for sale as a full-fledged game.

Other features that I would have liked to have implemented into the project, with more time would be clothing customization for the character as well as a more well-developed story for the game with additional side missions for the player to partake in. Due to time constraints however, this was not possible as more important features needed to be focused on.

Despite these limitations, I am satisfied that the principal outcomes have been achieved with the realisation of an immersive third-person action stealth RPG complete with interactive NPC's, a skill tree system, realistic weapon mechanics and in-game item purchasing such as weapons, ammunition and silencers.

4 Further development or research

With more resources I would have liked to implement a multiplayer mode. With this game mode, the player could take part in online death matches against other players. Before a match starts, each player could choose the type of weapons they would like to use for the match in order to suit their play style. A number of maps would be developed for this mode, each one different from the last with dynamic features such as a destructible environment which can be used for players to help their team gain the upper hand. There would be a range of different game modes such as: capture the flag and team deathmatch, for example.

I also feel that with a development team, it would be possible to create a more open-world type of stealth RPG game. The player would be able to explore various cities and interact with dozens of NPC's. This would also allow for a large variety of side quests as well as just main missions.

5 References

C sharp accent tutorials. Youtube channel. Available at:

https://www.youtube.com/channel/UCq9_1E5HE4c_xmhzD3r7VMw

(Last accessed: March 10th 2016)

Pixel Crushers. Youtube channel. Available at:

<https://www.youtube.com/channel/UCzjJjdKeVL0AkHJPZ5ybnKw>

(Last accessed: 28th April 2016)

Pixel Crushers. Forum. Available at:

<http://www.pixelcrushers.com/phpbb/viewforum.php?f=3>

(Last accessed: 30th March 2016)

General discussion and Troubleshooting - Rival{Theory}. Forum. Available at:

<http://rivaltheory.com/forums/forum/rain/troubleshooting/>

(Last accessed: 17th March 2016)

TF3DM. 3D models (resources). Available at:

<http://tf3dm.com/>

(Last accessed: 9th February 2016)

6 Appendix

6.1 *Project Proposal*

Project Proposal

Equilibrium

Michael Kilfeather, 12420472, x12420472@student.ncirl.ie

BSc (Hons) in Computing

Gaming and Multimedia

21/09/2015

6.1.1 Objectives

(Max. 1 Page)

The aim of this project is to create a large-scale 3D game using the Unity game engine. The primary platform it will be developed for is the PC. The game I'm developing will be, at its core, a third-person stealth action game with RPG (role-playing game) elements. In a stealth game, the primary objective for the player is to be able to make it through a level without being detected and killed by the enemy, while also completing various objectives. Nowadays, most modern stealth games are more open-ended than ever. Players can tackle various mission objectives in one level and approach missions in a variety of ways. I hope to replicate these gameplay mechanics which can be seen in other highly successful stealth games such as 'Metal Gear Solid' and 'Splinter Cell' for example.

The player will start with a pre-set character, but will be able to customize the character's abilities as the player progresses. The game will have plenty of customization options such as character clothing and weapon customization as well as a skill tree system which will allow the player to unlock new special abilities and improve their default ones. RPG elements such as these can be seen in many open-world games such as Watch Dogs, GTA V, Skyrim and so on.

The player will have a list of main missions to complete which progress the story. These missions will have various objectives which need to be fulfilled. There will also be side missions available. Many of these side missions can be triggered when the player talks to an NPC. Completion of these side missions may result in unlocking new weapons for purchase and more.

When the player is not out on missions, they will be able to explore a large city area and approach NPC's (non-player characters) and have the ability to talk to them through the use of a dialogue system asset for Unity. The player will also be able to enter shops where they can purchase new weapons / gadgets and customize their clothing.

6.1.2 Background

The Gaming Industry has grown exponentially over the past decade. It currently outperforms the Film industry, in terms of revenue and is expected to be worth more than \$100 billion in the next three years. Mobile gaming is a major driving force behind generating massive revenue through in-app purchases. The Games Development sector in Ireland has also seen rapid growth over the past few years, with various startup companies setting up in the heart of Dublin city as well as the big players such as EA Bioware which currently operates in Galway and Activision Blizzard, as well as Middleware companies such as Havok and Demonware in Dublin city centre.

I chose this project due to my immense interest in Games Development as well as gaming in general. I find the process of developing games both fascinating, challenging but also rewarding. I've had an interest in games development for many years and often enjoy working on games in my free time as a hobby using the Unity game engine. In my third year of studies, for a group project, I developed a Unity game with three other team members which had a very positive reception during development and upon completion. As part of third year, for work placement, for six months, I worked as a trainee programming intern for an indie games development studio where I gained invaluable hands-on experience and learned as much as possible with Unity. After these experiences, I feel that I am now well prepared to take on this project.

I chose this type of game as RPG's and Stealth games are some of my favourite video game genres. Both genres are massively popular and have evolved so much over such a short space of time. Some examples of popular RPG's include The Elder Scrolls series and the Fallout series, both created by Bethesda Game Studios. Both of these RPG series have such a huge fan base because of the immersive worlds and memorable characters and engaging storylines that these developers have created.

The Stealth genre is no different. Prime examples of the genre's success include the renowned Metal Gear Solid series, a Japanese video game series, created by

mastermind Hideo Kojima which has seen massive success over in the west. This stealth game series was well known for being way ahead of its time, with revolutionary game mechanics which really pushed the stealth genre forward.

I chose Unity as the game engine for creating the game. Unity is an easy to use but very powerful games development tool that is capable of producing Triple-A quality games. It provides plenty of features that can be seen in other popular game engines, such as the Unreal Engine. Lots of these features such as physics and lighting can be applied to the game by simply dragging and dropping components into the scene, with no coding needed. However, it's up to the user to fine-tune these components through code if they want added complexity from these features. Unity also has its own built-in terrain editor, which for many developers, eliminates the need for environments to be created through dedicated 3D modelling software such as Autodesk Maya or 3DS Max.

6.1.3 Technical Approach

Brief description of the approach to be followed (Max. 1 Page), Research, literature review, requirements capture, implementation etc...

Research

For the type of software / engine used for the project, there was no research required, as I had already settled on using the Unity game engine. I had been using Unity for over 6 months while on placement and was also teaching myself how to use Unity prior to that so I am already quite familiar with how the engine and its components work.

I spent a good deal of time deciding on what type of game I would make for this project. I researched some examples of more complex projects made in Unity by looking on Youtube, Google as well as Unity forums. I eventually came to the decision that I would create a third-person stealth action game, with RPG elements.

I then began researching various game mechanics seen in third-person games, stealth games as well as RPG's, by looking at games such as Metal Gear Solid,

Shenmue, Uncharted, Watch Dogs and more. I wanted to see if it would be feasible to replicate mechanics from these games and if they could be achieved using the Unity engine.

Implementation

Implementation of the project will be done through the use of the Unity Game Engine. Coding will be done in C# using the Visual Studio IDE. Throughout the lifecycle of the project's development, there will be various stages to help break up important tasks. I will first start by implementing a character controller into the game, which will control the character's general movement such as walking, sprinting, crouching, aiming, sneaking and more. Once this has been completed, I will then move onto the implementation of core features such as enemy and NPC AI, dialogue, side missions, customization features as well as the user interface. AI and Dialogue features will be implemented through the use of plugins from the Unity Asset Store.

Once these core features have been successfully implemented, I will move on to level design. I will create the areas that the missions will take place in and I will also design the main area that the character can traverse when they are not undertaking missions.

Once all of these features have been successfully implemented, I begin the testing stage. For this, I will let users try out the game and gain feedback from them, while also fixing bugs and optimizing the game.

6.1.4 Special resources required

If applicable, e.g., books, hardware, etc.

Certain system requirements will need to be met, in order to run the game smoothly at optimal settings. Since updating to Unity 5, the latest version, more demanding graphical features will be utilized such as Depth of Field, real-time reflection probes

and more, which can be achieved to the best effect with a graphics card that supports DirectX 11 and up. However, this shouldn't be an issue for most people who wish to play the game as most modern laptops / desktops have graphics cards which support DirectX 11 and higher. Graphical settings can be raised or lowered before playing the game, so anyone can play the game at a framerate which best suits their system's specifications.

If the player does not wish to use the traditional keyboard and mouse control setup, then a PS4 controller and micro-usb will be required for user input. Throughout project development, I will need to make use of various Youtube tutorials and other online resources such as Digital-Tutors, part of the Pluralsight subscription given to students by the college. Any tutorials used, will be referenced clearly in the project.

6.1.5 Project Plan

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

Please see .mpp file attached

6.1.6 Technical Details

Implementation language and principal libraries

The language that I will be using for this project is C#. This is the primary programming language used for the Unity game engine. The IDE which I will be using for coding is Microsoft Visual Studio 2013.

Throughout development, I will make use of the various libraries that Unity has to offer as well as the large amount of plugins which can be used for development from the Unity Asset Store. For level design, I will be using the 'Prototype' plugin by ProCore3D. This is essentially a free version of 'ProBuilder' for Unity. It allows you to build and edit geometry, within the Unity editor itself. It's an efficient tool to use for early level designs for the game. I may also resort to Autodesk Maya for more detailed level designs, if needs be.

I will also be implementing a dialogue system into the game, through the use of the 'Dialogue System for Unity' plugin. This is a code-free plugin which allows you to add interactive dialogue and quests to your game.

For AI implementation, I will use the 'RAIN AI' plugin. This will be used for enemies and NPC's in the game and I will be making use of behaviour trees for logic and a waypoint system for movement paths.

6.1.7 Evaluation

I plan to evaluate my game through testing which will be done by end users of the game. I plan on hosting my game on 'Game Jolt' for members of the public to test. GameJolt is a Freeware games host with a large user base. It allows independent developers to upload builds of their projects for other people to download and test. Projects can be uploaded even if they are still currently in a 'work-in-progress' state, which makes it a great platform for gaining feedback from the community on the site.

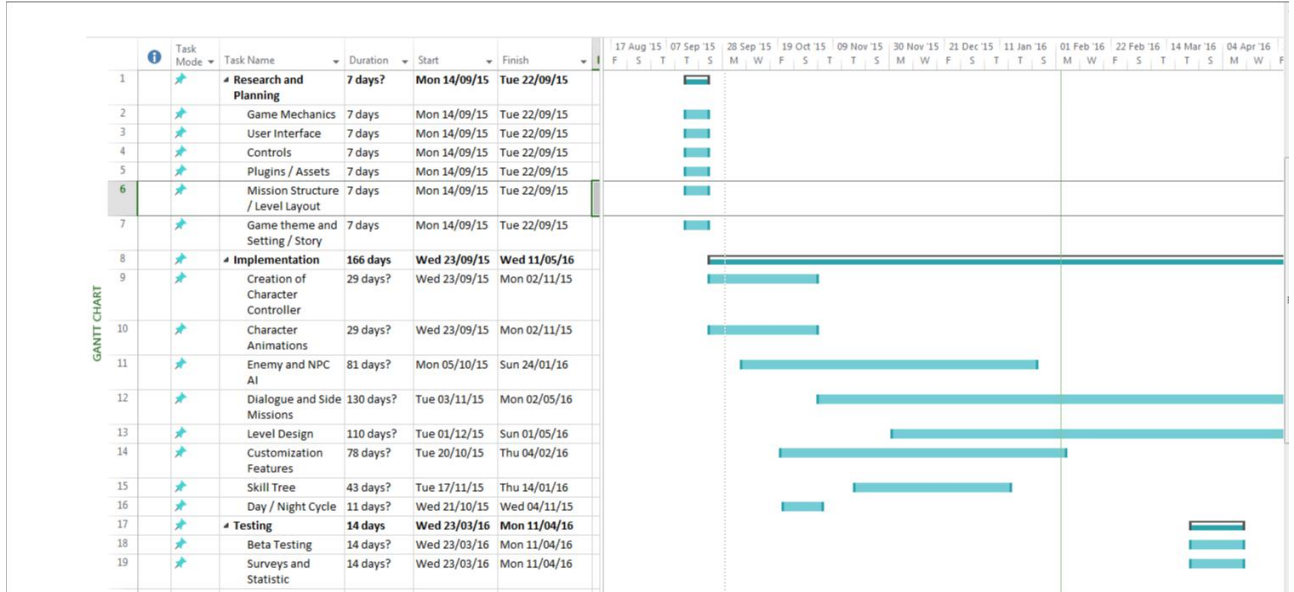
I also plan on carrying out surveys for testers to fill out so that I can gain insight and feedback on which game mechanics need tweaking or polishing, difficulty level and more. This will be done using SurveyMonkey.

Once I have gathered enough data and statistics from testers that have played the game, I will get the average rating from all testers and what parts of the game need to be improved the most.

____Michael Kilfeather 02/10/2015_____

Signature of student and date

6.2 Project Plan



6.3 Requirement Specification

6.3.1 Document Control

6.3.2 Revision History

Date	Version	Scope of Activity	Prepared	Reviewed	Approved
19/10/2015	1	Create	AB	X	X
03/02/2016	2	'Change Settings' Requirement added to 'Functional Requirements', new images added to GUI in 'Interface Requirements'.	CD		

6.3.3 Distribution List

Name	Title	Version
Eamon Nolan	Lecturer	

6.3.4 Related Documents

Title	Comments
Title of Use Case Model	
Title of Use Case Description	

Table of Contents

- Document Control.....
- Revision History
- Distribution List
- Related Documents
- 1 Introduction.....
 - 1.1 Purpose
 - 1.2 Project Scope
 - 1.3 Definitions, Acronyms, and Abbreviations
- 2 User Requirements Definition.....
- 3 Requirements Specification.....
 - 3.1 Functional requirements
 - 3.1.1 Use Case Diagram.....
 - 3.1.2 Requirement 1 <New Game>
 - 3.1.3 Requirement 2 <Start Mission>.....
 - 3.1.4 Requirement 3 <Unlock Skill>.....
 - 3.1.5 Requirement 4 <Customize/Purchase>
 - 3.1.6 Requirement 5 <Save Game>
 - 3.1.7 Requirement 6 <Load Game>.....
 - 3.1.8 Requirement 7 <Quit>.....
 - 3.2 Non-Functional Requirements.....
 - 3.2.1 Performance/Response time requirement
 - 3.2.2 Availability requirement.....
 - 3.2.3 Recover requirement
 - 3.2.4 Robustness requirement.....
 - 3.2.5 Reliability requirement
 - 3.2.6 Maintainability requirement
 - 3.2.7 Portability requirement
 - 3.2.8 Extendibility requirement.....
 - 3.2.9 Reusability requirement
 - 3.2.10 Resource utilization requirement.....

4	Interface requirements.....
4.1	GUI.....
4.2	Application Programming Interfaces (API).....
5	System Architecture
6	System Evolution.....

6.3.5 Introduction

6.3.5.1 Purpose

The purpose of this document is to set out the requirements for the development of a game created with the Unity Game Engine. At its core, the game will be a stealth game, with RPG elements. The player will complete various missions in order, unlocking new items, weapons and money as they progress through the game. There will also be side missions that the player can participate in which can be triggered through conversations with NPC's, using the Dialogue System for Unity. Before the player begins a mission, they will be able to explore a small town / city area where they can enter shops. In these shops, the player can talk to NPC's, customize clothing, weapons and also purchase new weapons and items.

The game will also include many dynamic features, seen in many games such as weather change and day and night cycle. At random times, when the player is traversing the town, when not playing through missions, the weather can dynamically change at any moment.

The player will also be able to upgrade their skills overtime using the Skill Tree system. Skills that the player can unlock / upgrade include stealth takedowns, slow motion ability and more.

The intended customers are fans of the Stealth and RPG genres. The game will appeal to fans of popular games series such as: Deus Ex, Metal Gear Solid, Watch Dogs and Mass Effect.

The game will be set in an alternate future, with a cyberpunk setting. The look of the characters, weapons and areas will reflect that.

The primary platform for the game will be PC. However, it's possible that the game could become cross platform in the future and make its way to mobile devices such as Android phones and tablets.

6.3.5.2 Project Scope

The scope of the project is to develop a game with the Unity Game Engine. The game will be programmed in C# with the Microsoft Visual Studio IDE. Character and weapon models will be downloaded from online sources including the Unity Asset Store. Level design will be created using 'ProBuilder Basic', which is great for rapid prototyping of level designs. Photoshop will be used for textures and sprites.

Schedules: The game will be developed over the course of 9 months. There will be various phases and stages of development, starting with Planning and Research, followed by Implementation and ending with testing.

Costs: The majority of development will be free. However, there may be some occasional purchases throughout development from the Unity Asset Store for plugins and models.

Software Engineering environment: Requirements diagrams will be made using UMLet, a free UML tool for fast UML diagrams.

6.3.5.3 Definitions, Acronyms, and Abbreviations

FPS – Frames per Second

RPG - Role Playing Game

Side Mission - Missions that the player can participate in which may give the player rewards upon completion.

NPC - A non-player character in a game is any character that is not controlled by the player. Usually controlled through artificial intelligence.

Unity – A 3D cross-platform game engine

Visual Studio – An IDE from Microsoft used for developing computer programs, web sites and games. Supports various programming languages such as C#, C++ and Visual Basic.

DLC – Downloadable content

Skill Points – Points that the player can gain from achieving objectives during missions. These can be used to unlock skills in the Skill Tree.

6.3.6 User Requirements Definition

The objective of the game is for the player to play through missions that the character is assigned throughout the game. Upon completing these missions, the player will gain money which can be used to unlock new weapons and equipment.

Once the user starts the game, they will have the option of saving the game at any time through a pause menu. The user will also be able to load their save data and pick up from where they left off when they start up the game again. The user can load the game from the main menu screen.

The user will be able to upgrade their character's current abilities and also unlock new ones through a Skill Tree system. The skill tree can be accessed through a separate menu in-game. The user should also be able to quit the game whenever they wish.

The user will also be able to customize and purchase new clothing designs and weapons. The player can enter a shop where they can carry out purchasing and customization features.

6.3.7 Requirements Specification

After no more than an hour of playing through the game, the user should know how to access and use all of the game's features with ease. The player will be able to access an in-game tips menu which will show them how to access features such as customization, shops and the skill tree for example. Specific actions will show button prompts so the player will know how to perform the action in the game. A menu can be accessed in-game which will show all of the controls in detail. Once the user has checked the controls and inputs as well as the game tips, I would expect the user to encounter no more than 5 errors per gameplay session. Response time should be very quick as it should take the user no more than 1 minute to start the game upon execution.

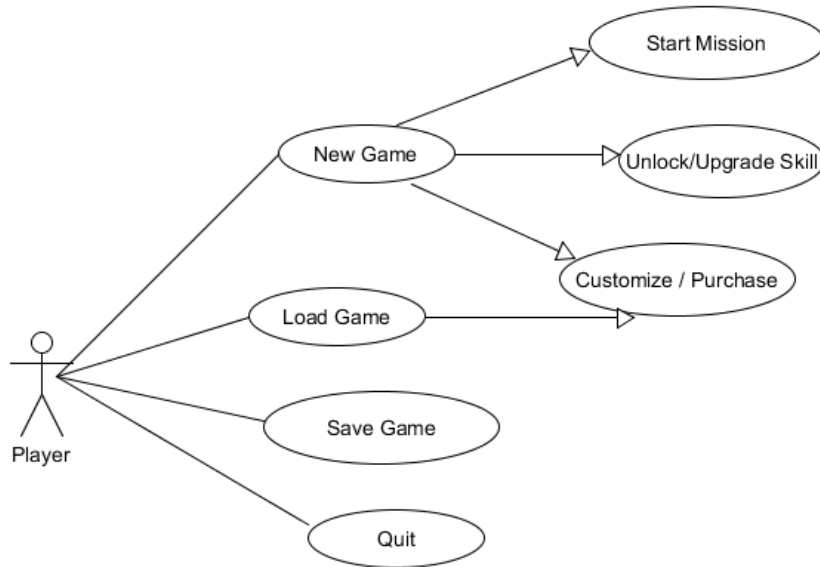
6.3.7.1 *Functional Requirements*

The following list of functional requirements defines the core functions of the system, ranked in order. Each functional requirement describes a specific behavior of the system.

1. New Game
2. Start Mission
3. Unlock Skill
4. Customize/Purchase
5. Save Game
6. Load Game
7. Quit

6.3.7.1.1 Use Case Diagram

The Use Case Diagram provides an overview of all functional requirements.



6.3.7.1.2 Requirement 1 <New Game>

6.3.7.1.2.1 Description & Priority

This allows the player to start a new game. This requirement is vital as it is required for the player to begin playing the game for the first time.

6.3.7.1.2.2 Use Case

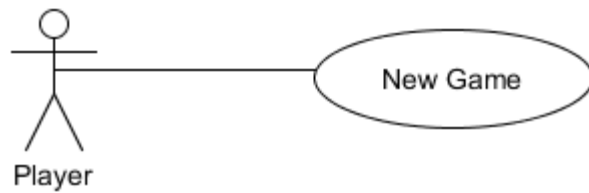
Scope

The scope of this use case is to allow the player to start a new game.

Description

Describes the process by which the player starts a new game.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode.

Activation

This use case starts when the player starts a new game.

Main flow

1. The system identifies the player.
2. The Player starts a new game.
3. The system loads the opening level of the game.

Alternate flow

N/A

Exceptional flow

N/A

Termination

The system presents the next screen to the player.

Post condition

The system goes into a wait state

6.3.7.1.3 Requirement 2 <Start Mission>

6.3.7.1.3.1 Description & Priority

The player commences a mission. This is required so that the player can progress through the game by completing missions.

6.3.7.1.3.2 Use Case

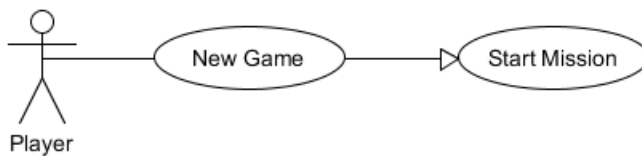
Scope

The scope of this use case is to allow the player to start a mission.

Description

This use case describes the means by which a player can start a mission.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

This use case starts when the player starts a mission.

Main flow

4. The player is in the main town area (See A1).
5. The player opens the in-game menu.
 6. The player opens the 'Mission Menu' in the in-game menu.
 7. The player selects the mission from the 'Mission Menu'.

Alternate flow

A1: <Player triggers mission after talking to a specific NPC>

1. The player approaches an NPC.
2. The player starts a conversation with the NPC.

3. The player triggers a new mission through the conversation.

Exceptional flow

N/A

Termination

The system moves to the loading screen, loading the scene where the mission will take place.

Post condition

The system goes into a wait state

6.3.7.1.4 Requirement 3 <Unlock Skill>

6.3.7.1.4.1 Description & Priority

The player unlocks a new skill / ability for them to use during missions. This is important as it will help players approach missions in different ways.

6.3.7.1.4.2 Use Case

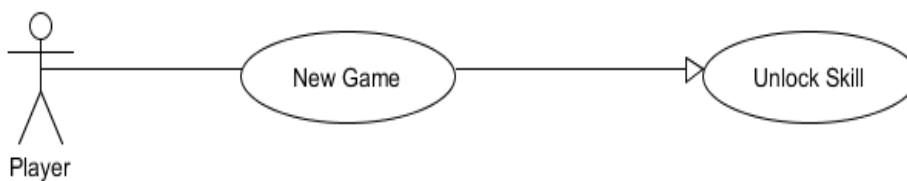
Scope

The scope of this use case is to allow the player to gain a new skill.

Description

This use case describes the means by which a player gains a skill.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

This use case starts when the player unlocks a Skill.

Main flow

1. The player opens the Skill Tree menu.
2. The player uses skill points to purchase a skill.
3. The system unlocks the skill from the skill tree.

Alternate flow

N/A

Exceptional flow

N/A

Termination

The system stays on the Skill Tree menu, unless the player decides to exit the Skill Tree menu.

Post condition

The system goes into a wait state.

6.3.7.1.5 Requirement 4 <Customize / Purchase>

6.3.7.1.5.1 Description & Priority

The player can purchase and customize new weapons as well as clothing designs for the character. New and customized weapons will help the player out during enemy encounters.

6.3.7.1.5.2 Use Case

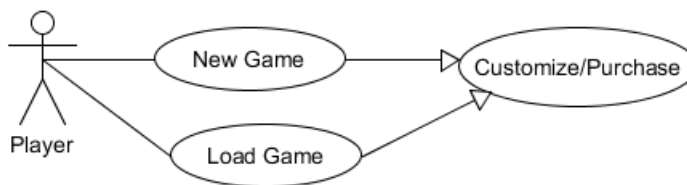
Scope

The scope of this use case is to allow the player to purchase or customize new weapons and clothing for their character.

Description

This use case describes the process by which the player can purchase or customize weapons and clothes.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the player starts the game.

Activation

The use case starts when the player enters a shop.

Main flow

1. The player enters a shop.
2. A menu opens, displaying weapons / clothes that the player can purchase or customize.
3. The player selects the item they want.
4. Money is taken from the player's inventory.

Termination

The system stays on the menu unless the player exits the menu themselves.

Post Condition

The system goes into a wait state.

6.3.7.1.6 Requirement 5 <Save Game>

6.3.7.1.6.1 Description & Priority

The player can save their progress in the game. This is required so that the player can continue from where they had left off when they launch the game again.

6.3.7.1.6.2 Use Case

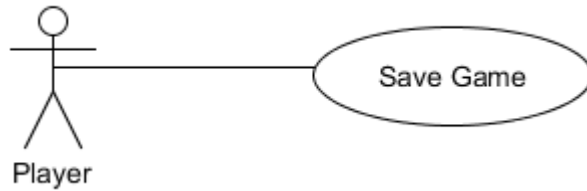
Scope

The scope of this use case is to allow the player to save their progress in the game.

Description

This use case describes the process by which the player saves the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

The use case starts when the player saves the game.

Main flow

1. The player opens the pause menu.
2. The player selects the 'Save Game' option.
3. The system stores the save game data to a file.

Termination

The system returns the player to the pause screen.

Post Condition

The system goes into a wait state.

6.3.7.1.7 Requirement 6 <Load Game>

6.3.7.1.7.1 Description & Priority

The player can load the game. This use case is required so the player can load the current state of where they are in the game after they have saved it, even after they've closed and relaunched the game.

6.3.7.1.7.2 Use Case

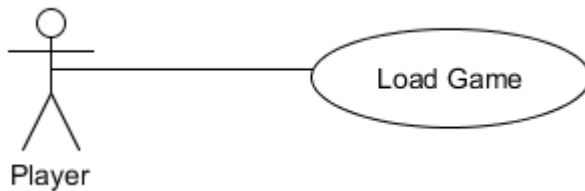
Scope

The scope of this use case is to allow the player to load the game.

Description

This use case describes the process by which the player loads the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

The use case starts when the player loads the game.

Main Flow

1. The player clicks the 'Load Game' option from the main menu.
2. The system opens the loading screen.

Termination

The system presents the next screen to the player.

Post condition

The system goes into a wait state.

6.3.7.1.8 Requirement 7 <Quit>

6.3.7.1.8.1 Description & Priority

The player quits the game. This function is required so that the player can shut down the application.

6.3.7.1.8.2 Use Case

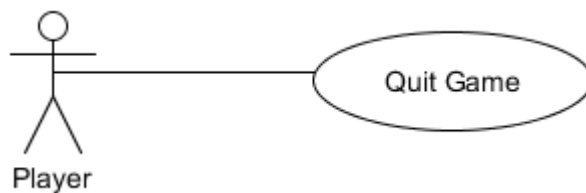
Scope

The scope of this use case is to allow the player to quit the game.

Description

This use case describes the process by which the player quits the game.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode.

Activation

This use case starts when the player quits the game.

Main Flow

1. The player selects the 'Quit Game' option from the main menu.
2. The system exits the application.

Termination

The system closes the application.

Post condition

The system is off.

6.3.7.2 *Non-Functional Requirements*

6.3.7.2.1 Performance/Response time requirement

The game should run at 60 FPS on a user's PC or Laptop. However this will largely depend on the graphics card that's in the user's computer. For optimal performance and graphical fidelity, a dedicated graphics card would be recommended e.g. Nvidia, AMD Radeon, as opposed to integrated graphics cards such as Intel HD Graphics. Graphical settings can be adjusted before the player launches the game so the user should still be able to play the game regardless of their computer specifications. Response time will be immediate and there should be no input lag or delay when the user performs an action within the game.

6.3.7.2.2 Availability Requirement

The game will be available to users at all times and can be accessed by launching a .exe file or through the use of Unity Web Player.

6.3.7.2.3 Recovery Requirement

In terms of recovery, the game will have a save system so the user can save their progress at any time. In the event that the game may crash unexpectedly, the user will be able to resume their progress when they load the game from the main menu.

6.3.7.2.4 Robustness Requirement

Lots of beta testing and bug fixing will be done prior to the completion of the project so that the user won't encounter any game breaking bugs, glitches or crashes.

6.3.7.2.5 Reliability Requirement

The game should be available for the user to run successfully at all times, especially if they have the game downloaded onto their computer.

6.3.7.2.6 Maintainability Requirement

The game will be supported after its release. If there are any game breaking bugs or glitches that may have gone unnoticed during beta testing, these will be rectified.

6.3.7.2.7 Portability Requirement

The user will be able to play the game on both desktop PC's as well as laptops. The user can keep the game stored on a USB flash drive / external hard drive and transfer it to another computer and play it on that system if they wish.

6.3.7.2.8 Extendibility Requirement

Additional content may be added at a later date after the game's initial release which may possibly introduce new gameplay features which will add replayability to the game.

6.3.7.2.9 Reusability Requirement

The game will have side missions / quests that the player can access throughout the game which they may have missed during their first playthrough.

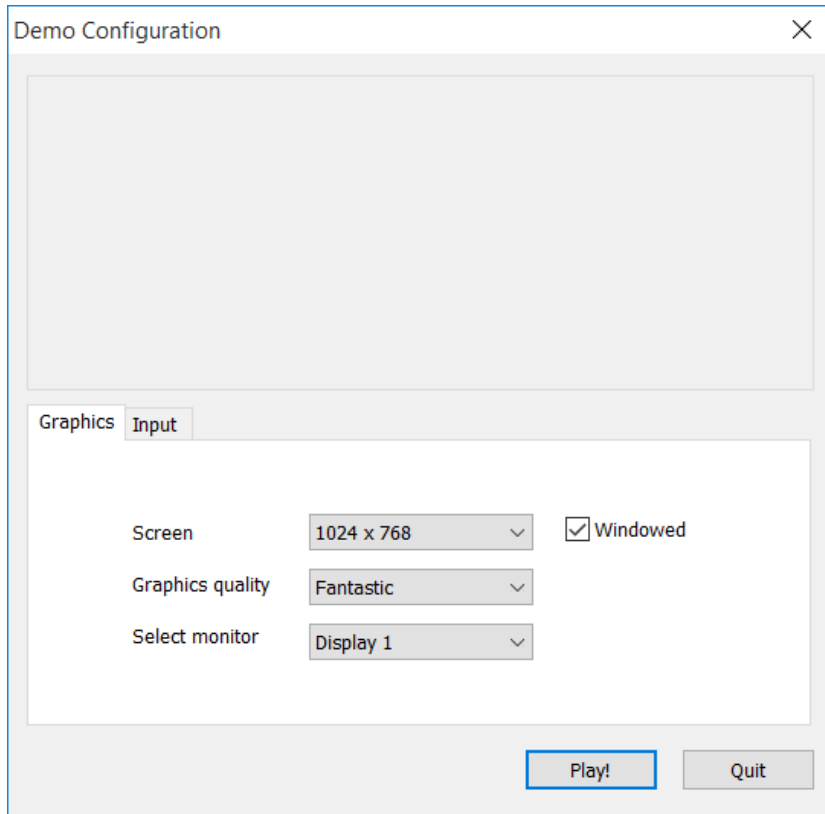
6.3.7.2.10 Resource utilization Requirement

The game will use as many resources as it can from what the user's PC / Laptop is capable of providing which will have an impact on performance and graphical fidelity e.g. RAM size, Graphics Card, CPU frequency.

6.3.8 Interface Requirements

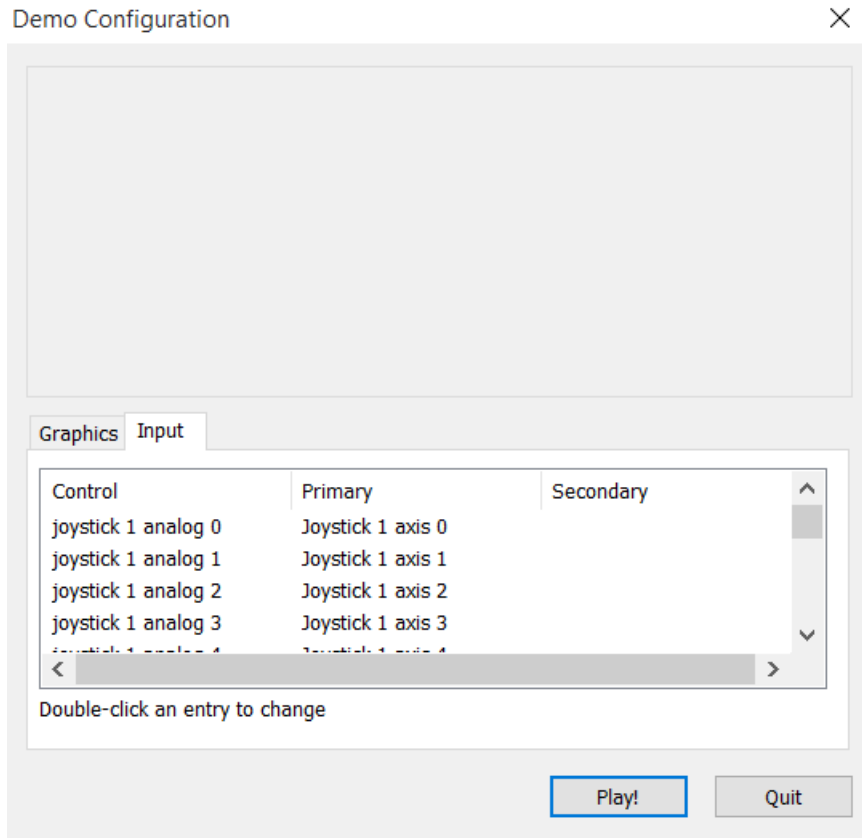
6.3.8.1 *GUI*

Game Launch Screen – Graphics Settings



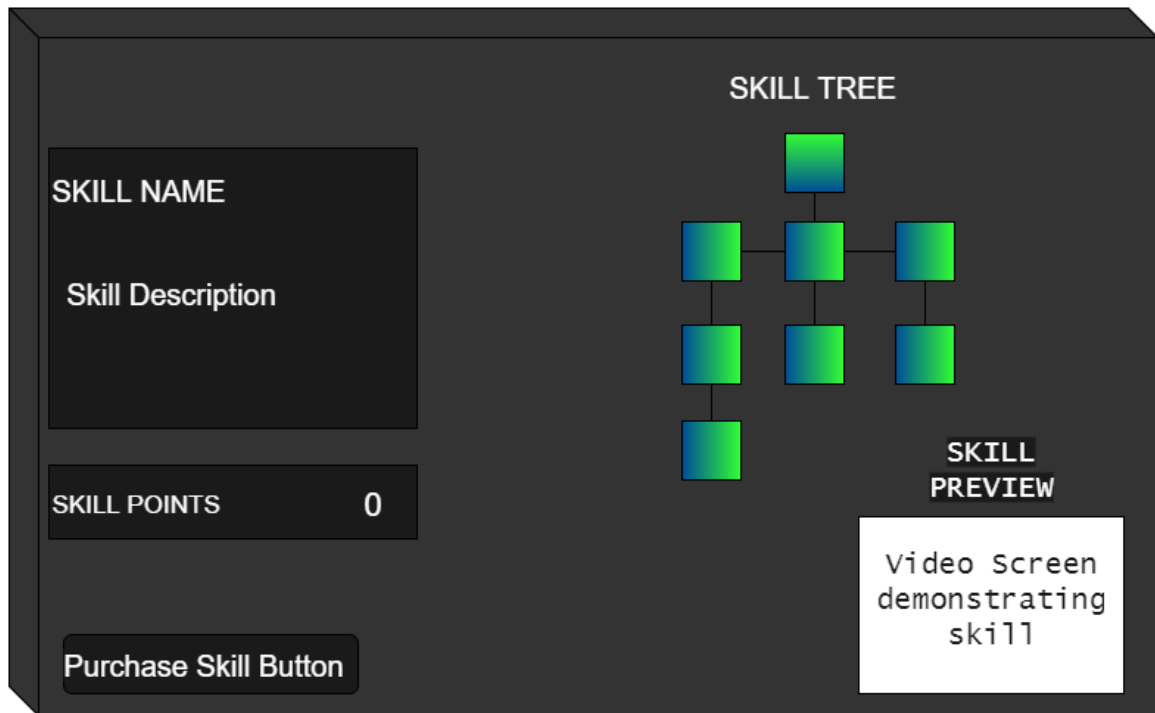
This is the default opening screen that the player is presented with after launching the game. From here, both the 'New Game' and 'Quit' requirements take place. Both 'New Game' and 'Quit' requirements will also be used in the main game menu which is displayed after this screen. In this screen, the player can also adjust the graphical settings of the game, if their system is unable to run the game at its highest settings. The player can adjust the resolution of the game screen, the overall graphics quality and even monitor display, if the player happens to be using more than one monitor.

Game Launch Screen – User Input Settings



From this screen, the player will be able to modify the controls of the game. The game is currently set up so that the player can use a PS4 controller in the game, as well as keyboard and mouse. PS4 controls can be modified in this screen as well as the traditional keyboard and mouse controls.

In-Game Screen – Skill Tree Menu (Mock-up)

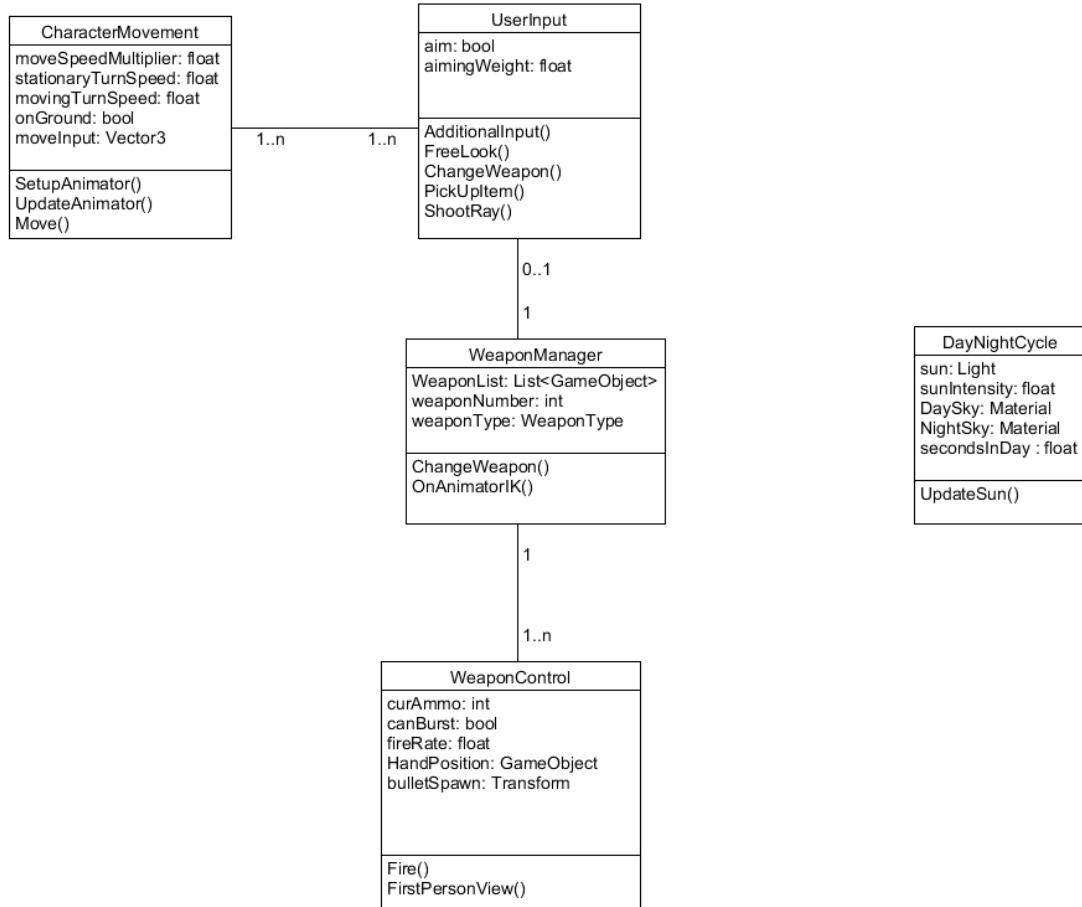


This in-game menu deals with the 'Unlock Skill' requirement. The player can access this menu at any time during the game and purchase new skills that they can make use of during missions, by using skill points that they have acquired. This image is a rough mock-up of the GUI for the Skill Tree menu and may have some design changes over the course of project development.

6.3.8.2 *Application Programming Interfaces (API)*

The Unity Game Engine is the primary API that will be used for developing the game. Other API's that will be used include the 'RAIN AI' API, which is used for creating behaviour trees for characters and other entities in Unity as well as the 'Dialogue System for Unity' API which allows for the creation of branching conversation trees using a visual node-based editor and quest creation.

6.3.9 System Architecture



This architecture I feel, best represents the type of game I plan on creating. The diagram, at its core, represents how the player can interact with the world as well as their own character's inventory, in this case, weapon management. The `CharacterMovement` class will be used to set the player's movement speed, turning speed and will be used to set up the player's animation controller. This will be linked to the `UserInput` class, which allows the player to control their movement, aiming, weapon switching, firing and various other actions including crouching and rolling. The `WeaponManager` class handles all of the weapons that the player is carrying, the type of weapon as well as methods for handling weapon changing and IK (inverse kinematics), which deals with the positioning of the character's arms when holding weapons. The `WeaponManager` class refers to the

WeaponControl class to determine which weapon the player is currently using. This WeaponControl class holds all of the properties of the gun, such as the amount of ammo it contains, its firing rate, as well as where the bullet spawns from. These properties are set up so that they can be easily modifiable which will tie in with the game's customization options.

A Day/Night cycle system will also be put into use. It can take effect while the player is traversing outside of missions and also in missions that take place outside, which will affect the player's visibility making it harder for them to spot enemies.

Classes that have yet to be developed that were not part of this diagram include customization for weapons, clothing as well as the Skill Tree.

6.3.10 System Evolution

The game could evolve overtime through downloadable content (DLC). This would essentially add brand new content to the game that would not have been present in the initial release. For example, brand new areas, weapons, enemy types and more could be added through DLC.

A multiplayer component could also be added at some point in the future after the game's release. There are plenty of API's that can be used with Unity which allow for multiplayer implementation such as the 'Photon Unity Networking' API. An idea for multiplayer for this game would be tradition deathmatch modes seen in many games today where a number of players are in one arena and have to get as many kills as possible. Once the timer is up, the player who has the most wins. A team-based deathmatch mode could also be implemented as well as 'Capture the Flag' where two teams have to capture each other's flags while preventing their own from being taken back to the opponent's base.

6.4 Product Design Specification

<Equilibrium>

Product Design Specification

Version <1.0>

<04/12/2015>

VERSION HISTORY

*[Provide information on how the development and distribution of the **Product Design Specification**, up to the final point of approval, was controlled and tracked. Use the table below to provide the version number, the author implementing the version, the date of the version, the name of the person approving the version, the date that particular version was approved, and a brief description of the reason for creating the revised version.]*

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	<Michael Kilfeather>	<04/12/2015 >	<name>	<mm/dd/yy>	Initial Design Definition draft

UP Template Version: 12/31/07

TABLE OF CONTENTS

1	Introduction.....
1.1	Purpose of The Product Design Specification Document.....
2	General Overview and Design Guidelines/Approach.....
2.1	Assumptions / Constraints / Standards.....
3	Architecture Design.....
3.1	Logical View
3.2	Hardware Architecture.....
3.3	Software Architecture
3.4	Security Architecture
3.5	Communication Architecture
3.6	Performance.....
4	System Design.....
4.1	Use-Cases.....
4.2	Database Design.....
4.3	Data Conversions.....
4.4	Application Program Interfaces
4.5	User Interface Design.....
4.6	Performance.....
4.7	Section 508 Compliance.....
	APPENDIX A: REFERENCES
	APPENDIX B: KEY TERMS

6.4.1 Introduction

6.4.1.1 *Purpose of the Product Design Specification Document*

The Product Design Specification document documents and tracks the necessary information required to effectively define architecture and system design in order to give the development team guidance on architecture of the system to be developed. The Product Design Specification document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and development team. Some portions of this document such as the user interface (UI) may on occasion be shared with the client/user, and other stakeholder whose input/approval into the UI is needed.

This Product Design Specification document is used to keep track of changes made during project development as well as defining the various architectures used as well as system design principles that will be incorporated into the game and to ensure that they meet the requirements of the user. This document is created during the Planning Phase of the project lifecycle and is a living document that will be updated consistently. The intended audience of this document is the project developer and supervisor.

6.4.2 General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

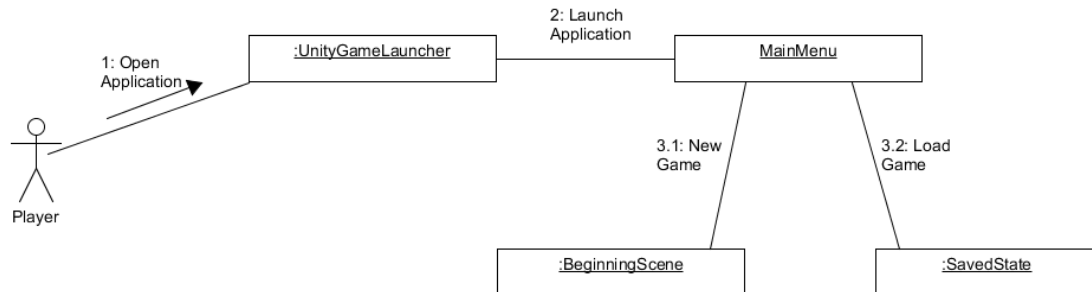
6.4.2.1 *Assumptions / Constraints / Standards*

Constraints: I have taken certain constraints into account, with the main constraints being time and 3D Modelling skills. In terms of time, I need to make sure that I can complete the core elements of the game in due time, so that I have enough time to focus on the less vital elements such as Day / Night Cycle or Weather system for example. Currently, AI, Player Control and Animation are my top priorities.

6.4.3 Architecture Design

This section outlines the system and hardware architecture design of the system that is being built.

6.4.3.1 Logical View



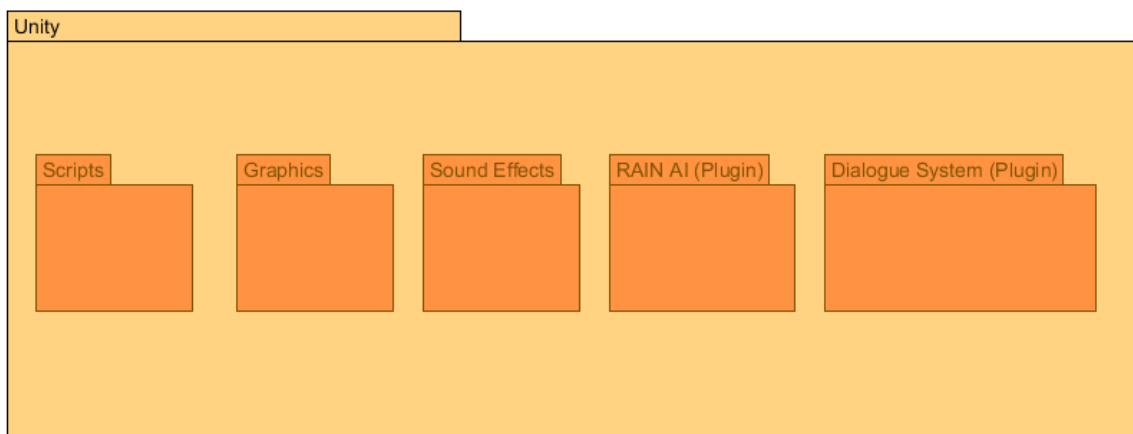
A simple logical view of how the user can launch the game and start a new game or load a previously saved game from the main menu. More Logical Views will be added as project development progresses.

6.4.3.2 Hardware Architecture

N/A

6.4.3.3 Software Architecture

A simple diagram detailing the software (Unity) and the primary assets (packages) it contains. This will be further fleshed out in future versions of this document.



6.4.3.4 *Security Architecture*

N/A

6.4.3.5 *Communication Architecture*

N/A

6.4.3.6 *Performance*

In terms of in-game performance, I'm currently aiming for a consistent 60 FPS. Assets will be added to the project overtime such as detailed geometry for levels, lighting effects e.g. light probes and reflection probes which can be taxing on the game's performance. Towards the end of the project's lifecycle, I will be optimizing the game to ensure that the FPS is smooth and consistent for the user. Graphical Settings can also be adjusted before the player starts the game, so they can adjust the performance of the game on their computer.

Save times and Load times should be fast so that there is not a long wait for the user to resume playing.

6.4.3.7 *Use-Cases*

All Use Cases related to this project can be viewed in the 'Requirements Specification' document.

6.4.3.8 *Database Design*

N/A

6.4.3.9 *Data Conversions*

N/A

6.4.3.10 *Application Program Interfaces*

Application Programming Interfaces that will be used during project development include the 'RAIN AI' plugin and the 'Dialogue System for Unity' plugin.

There are various API's that can be used for AI implementation into a Unity game such as NodeCanvas. I decided to go with RAIN AI due to its simplicity but more importantly, due to how powerful it is. RAIN AI is capable of physically displaying waypoints in a scene as well as visual and audio sensors for the AI rig, which is very useful for providing visual feedback on the responsiveness of AI agents.

Dialogue System for Unity is a powerful Unity API that allows for the implementation of Quests / Missions and Dialogue into your game. Dialogue can be created by using a visual node-based editor within the Unity editor itself. The Dialogue System for Unity also allows a developer to implement cutscenes, save and load functionality as well as quest tracking.

Quests / Missions, Dialogue as well as saving and loading all work through a Dialogue Database which is part of the plugin. The Dialogue Database stores everything into a Lua environment.

6.4.3.11 *User Interface Design*

The User Interface of the game is being designed primarily within the Unity 3D Game Engine using the Unity UI features. The player's HUD uses a 'Screen Space – Overlay' canvas, meaning that any UI elements inside this canvas are always in the same position as the screen. The player HUD will consist of UI elements such as the player's current weapon, ammo count, health etc.. Other UI elements will be displayed in a 'World Space' canvas. This allows UI elements to be displayed anywhere in the world and not fixed to the position of the screen.

Appendix A: References

[Insert the name, version number, description, and physical location of any documents referenced in this document. Add rows to the table as necessary.]

The following table summarizes the documents referenced in this document.

Document Name and Version	Description	Location
<i><Document Name and Version Number></i>	<i>[Provide description of the document]</i>	<i><URL or Network path where document is located></i>

Appendix B: Key Terms

[Insert terms and definitions used in this document. Add rows to the table as necessary. Follow the link below to for definitions of project management terms and acronyms used in this and other documents.

<http://www2.cdc.gov/cdcup/library/other/help.htm>

The following table provides definitions for terms relevant to this document.

Term	Definition
<i>FPS</i>	<i>Frames Per Second</i>
<i>HUD</i>	<i>Heads-Up Display</i>
<i>[Insert Term]</i>	<i>[Provide definition of the term used in this document.]</i>

6.5 Monthly Journals

6.5.1 Reflective Journal – September

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: September

My Achievements

This month, I was able to finalize my idea for the project, the mechanics that will be implemented into the game as well as all of the technologies that I will be using. I spent most of my time this month researching. Upon conclusion of my research I began putting together the project proposal document. I have also begun development of the project and am currently in the process of putting together a rough playable prototype.

My contributions to the projects included the required research of the project, the completion of my project proposal as well as commencing development on a prototype.

My Reflection

I felt, it worked well to do plenty of research before starting project development. I put as much time into researching as possible due to the scale of the game as I don't want to run into any major roadblocks along the way. I wanted to make sure the mechanics that I plan on implementing into the game can be achieved.

There are still certain parts of the project that I am considering such as mission structure and storyline. There may also be new ideas that I will implement during development that I may not have finalized at the moment.

Intended Changes

Next month, I will continue development of the project, implementing the basic mechanics of the game such as character controls, enemy AI as well as other additional features of the game such as customization and UI.

I realised that I need to put focus on time management, by planning out what mechanics of the game I will implement into the game and how long I should spend on each part as I don't want to drag out development on just one part of the game when I should also be focusing on others as I need to be aware of approaching deadlines.

6.5.2 Reflective Journal – October

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: October

My Achievements

This month, I was able to commence work on implementing Artificial Intelligence for enemies as well as complete the requirements specification document. I'm currently in the process of developing a simple test level of the game. This is so I will be able to test out the responsiveness of the enemy AI and if it interacts with the environment successfully e.g. If the enemy can recognize walls and walk around them instead of simply walking into them, as they're patrolling waypoints. I'm developing the AI using the powerful RAIN AI plugin for Unity. The enemy's behaviour is controlled using Behaviour Trees within RAIN AI. This allows the character to make decisions and set constraints for these decisions e.g. if the player is in the enemy's field of view, the enemy will then make the decision to chase and attack the player.

My contributions to the projects included the implementation and research of AI and learning how to use the RAIN AI plugin for Unity as well as completion of the requirements specification document.

My Reflection

I felt, it worked well to do research on Behaviour Trees when using RAIN AI. I followed plenty of video tutorials online and am starting to get a better understanding on how to use RAIN AI and all of its components.

However, I was not successful in fully developing the behaviour tree for enemies. There are still certain elements that I am unsure of and as of now, I'm currently researching on how to overcome these obstacles.

Intended Changes

Next month, I will try to have a fully functional behaviour tree so I can apply it to an enemy in the game as well as a complete level that can be used to test out the AI. I hope to begin developing customization features in the game as well as level designs.

6.5.3 Reflective Journal – November

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: November

My Achievements

This month, I was able to successfully develop a functioning behaviour tree for an enemy in my game. The enemy in the game is now able to patrol waypoints and navigate around walls and obstacles if they are obstructing the path of the enemy. If the player is in the enemy's line of sight, the enemy will now start moving towards the player. If the player has been seen by the enemy and the enemy follows the player, once the player is out of the enemy's line of sight e.g. hiding behind a wall, the enemy will switch to a 'searching' state. The enemy will wander around the scene, searching for the player.

I also spent some time polishing the player's animations. I had to rig the character again with a new skeleton, using Mixamo's Auto Rig software. Previously, the character's animations looked quite disjointed when it was aiming and walking. Now, after rigging the character again, the animations have been fixed.

My Reflection

I felt, it worked well to focus on one particular area of the project before moving on to others. My main focus for the month was AI implementation, and for the most part, it has been implemented successfully. I was not successful in fleshing out all of the abilities for the enemy character like aiming and shooting, as I hit a couple of roadblocks over the month. I spent a lot of time trying to fix the navigation of the AI, so they can move around obstacles, which has now been resolved.

Intended Changes

Next month, I will be adding an ability for the enemy to be able to aim and shoot at the player when the player is in its line of sight. I also want to be able to get the enemy to sense sound also (gunfire, footsteps), using Audio Sensors in RAIN AI. I am also currently in the process of developing a prototype level with mission objectives, so I can test out the responsiveness of the AI.

6.5.4 Reflective Journal – December

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: December

My Achievements

This month, I was able to expand upon the development of AI for enemy characters. Previously, I had it setup so that once the enemy loses track of the player, they'll randomly search the area for the player. Now, I have it so that once the player is out of the enemy's line of sight, the enemy will now move to where they last saw the player and then they will start looking around for the player. I have also implemented a ragdoll and location damage system. If the enemy is shot in the head for example, they will lose all of their health and the ragdoll system will activate so the enemy drops to the floor. The enemy also has the ability to shoot from their gun, but I need to improve this as the bullets don't seem to hit the player very often. The enemy shoots physical bullets for now, but I will need to change this so that the enemy uses 'raycasting' when firing instead of using physical objects as bullets. I have also implemented the ability for the player to be able to holster both weapons, however, it is not working entirely the way I want it to and this is still being worked on. I have also started implementing a UI for the player to be able to display their current health, ammo and weapon they are carrying.

My Reflection

I felt, it worked well to get advice from other Unity developers in the Unity Forums and also looking at various video tutorials when I encountered any issues. I had issues trying to get hit detection working and being able to keep track of enemy

health with the RAIN AI plugin. RAIN AI however, has a dedicated forum and I was able to overcome the problem by finding answers in the forum.

Intended Changes

Next month, I hope to improve the enemy's shooting ability. I also hope to be able to implement sound detection for the enemy too. If I succeed in implementing these, I will also try to get some levels / environments developed.

6.5.5 Reflective Journal – January

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: January

My Achievements

This month, I started development on the Skill-Tree system. This Skill Tree is displayed on the UI canvas and can be accessed through the pause menu. The player's skill points are stored in the 'CharacterStats' class. A text component in the Skill Tree UI refers to the 'CharacterStats' class to display the skill points. A 'MovieTexture' component was added to the Skill Tree so that when the user hovers over a skill node, it will display a preview video of the skill in use.

I have also managed to create one of the many skills from the skill tree. This skill is called 'Focus'. When a specific key is pressed, time slows down in the game for a limited time, giving the player more time to react during enemy encounters.

Persistent Objects have also been implemented, meaning that when the player moves from scene to scene, the player character, camera, weapons and other important components won't be destroyed / duplicated.

I also put some more time into the development of the main menu. I developed a graphics settings menu, allowing the player to change the quality of the graphics to suit the specifications of their computer.

In terms of Enemy AI, the enemy's ability to fire bullets has been fixed. Previously, the enemy shot physical objects. Now, I have implemented 'raycasting', which is far more accurate. With raycasting, the enemy shoots 'rays', which are invisible lines that start from a chosen start point and end at whatever distance the user specifies. This is helpful when modifying the range of various weapons.

Finally, I have begun development on quests / missions, using the 'Dialogue System for Unity' plugin. I developed a simple quest where the player has to kill an enemy. The quest status is tracked in the 'Quest Log' window and an alert is displayed when the quest has been completed.

My Reflection

I felt, it worked well to do research on Skill Trees and how they are implemented in modern games of today. As a result, I hope to be able to develop a skill tree that is similar, in terms of functionality and visual appeal, to what we see in modern games. In relation to Missions / Quests, I found it useful to take advantage of the forum support for the 'Dialogue System for Unity' plugin to overcome certain obstacles I encountered while trying to implement Quests.

Intended Changes

Next month, I plan on putting full focus towards the implementation of Quests into the game. I will also try and get a finalized Skill Tree developed. I also plan on implementing Weapon Customization / Purchasing and Clothing customization.

6.5.6 Reflective Journal – February

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: February

My Achievements

This month, I have achieved a number of objectives. I have implemented a basic saving and loading system, through the use of the 'Dialogue System for Unity' plugin. The user can save their progress and whenever the user loads their progress, it will bring them to the level they were on when they last saved and will keep track of weapons they were carrying. There is still some fine-tuning to be done with the save system such as keeping track of enemies in the scene, day or night and so on.

I have also began development on Weapon purchasing / customization. The player can visit a shop and when they approach the counter in the shop, by entering a trigger, a UI window will appear which shows weapons that the player can purchase and customize.

I am also in the process of developing an introductory scene for the game which will put the story of the game into perspective for the player.

Some minor adjustments have been made to the shooting mechanics for weapons, such as bullet spread and muzzle flash.

Sound detection has also been implemented for enemy AI in the game. Whenever the enemy's Audio Sensor picks up a sound from the player such as footsteps or gunfire, the enemy will move towards the position of where the sound was played.

Intended Changes

Next month, I will try to have the save system complete so everything in a scene is kept track of. I will also try and have the Weapon Shop functionality complete. For AI, I will try and get hit detection working so that the enemy will respond whenever they are hit by a bullet from the player. I will also continue to design levels for the game.

6.5.7 Reflective Journal – March

Reflective Journal

Student name: Michael Kilfeather

Programme: BSc (Hons) in Computing

Month: March

My Achievements

This month, I spent most of my time on developing the user interface and gamepad controls. Prior to this, I was only able to get the UI to work with the mouse. As of now, I have managed to successfully implement gamepad controls with the UI, such as the pause menu and main menu, by use of the InControl plugin, as well as some custom C# scripts.

I also spent some time working on level design. There will be 2 main areas that the player can visit, outside of the hub world where they can partake in missions. I hope to have these 2 levels fully designed within the next few weeks.

I am currently facing some issues with the 'Load Game' requirement of the game, whereby the user should be able to load their game state from the main menu, so I am currently in the middle of getting this fixed.

I am also currently in the middle of developing quests for the game, using the Dialogue System for Unity.

Intended Changes

Next month, I hope to have the two levels fully designed, the load game system working as intended and quests complete. I also plan on conducting some usability and customer tests towards the end of the month.

6.6 Other Material Used

6.6.1 Evaluation Surveys

<https://www.surveymonkey.com/r/7WJGVK8>

Q1: What type of a gamer would you consider yourself as?	
Casual Gamer - Experience with mobile games / arcade games / facebook games / flash games (e.g. Candy Crush, Clash of Clans)	
Q2: On a scale of 0 - 5, how would you rate the User Interface of the game overall? (Elements of the User Interface include: Quest Log Window, Skill Tree Panel, HUD, Weapon Shop, Dialogue, etc..)	
(no label)	5
Q3: On a scale of 0 - 5, how would you rate the weapon mechanics of the game? (Weapon mechanics include factors such as firing rate, recoil, bullet spread and more, for each weapon)	
(no label)	3
Q4: On a scale of 0 - 5, how would you rate the flow of the game? (Flow refers to the pacing of the game e.g. setting up context of story through dialogue / cutscenes)	
(no label)	2
Q5: On a scale of 0 - 5, how would you rate the Immersion of the game? (e.g. Sound effects, setting of the game, environments)	
(no label)	5
Q6: Please give explanation for any aspect of the game you liked / disliked.	
Story needs more exposition, for the flow of the game.	

Q1: What type of a gamer would you consider yourself as?

Avid Gamer - Plenty of Experience with games on various platforms such as Consoles (PS4, Xbox One) and PC (e.g. Metal Gear Solid V, Fallout 4, Grand Theft Auto, Battlefield 4)

Q2: On a scale of 0 - 5, how would you rate the User Interface of the game overall? (Elements of the User Interface include: Quest Log Window, Skill Tree Panel, HUD, Weapon Shop, Dialogue, etc..)

(no label) 4

Q3: On a scale of 0 - 5, how would you rate the weapon mechanics of the game? (Weapon mechanics include factors such as firing rate, recoil, bullet spread and more, for each weapon)

(no label) 3

Q4: On a scale of 0 - 5, how would you rate the flow of the game? (Flow refers to the pacing of the game e.g. setting up context of story through dialogue / cutscenes)

(no label) 4

Q5: On a scale of 0 - 5, how would you rate the Immersion of the game? (e.g. Sound effects, setting of the game, environments)

(no label) 4

Q6: Please give explanation for any aspect of the game you liked / disliked.

Good immersion - Good sound design and environments fit with the theme

Q1: What type of a gamer would you consider yourself as?

Casual Gamer - Experience with mobile games / arcade games / facebook games / flash games (e.g. Candy Crush, Clash of Clans)

Q2: On a scale of 0 - 5, how would you rate the User Interface of the game overall? (Elements of the User Interface include: Quest Log Window, Skill Tree Panel, HUD, Weapon Shop, Dialogue, etc..)

(no label) 4

Q3: On a scale of 0 - 5, how would you rate the weapon mechanics of the game? (Weapon mechanics include factors such as firing rate, recoil, bullet spread and more, for each weapon)

(no label) 5

Q4: On a scale of 0 - 5, how would you rate the flow of the game? (Flow refers to the pacing of the game e.g. setting up context of story through dialogue / cutscenes)

(no label) 3

Q5: On a scale of 0 - 5, how would you rate the Immersion of the game? (e.g. Sound effects, setting of the game, environments)

(no label) 4

Q6: Please give explanation for any aspect of the game you liked / disliked.

Weapons feel great to use in the game. Have a realistic feel.

Nice, responsive user interface.

Q1: What type of a gamer would you consider yourself as?

Avid Gamer - Plenty of Experience with games on various platforms such as Consoles (PS4, Xbox One) and PC (e.g. Metal Gear Solid V, Fallout 4, Grand Theft Auto, Battlefield 4)

Q2: On a scale of 0 - 5, how would you rate the User Interface of the game overall? (Elements of the User Interface include: Quest Log Window, Skill Tree Panel, HUD, Weapon Shop, Dialogue, etc..)

(no label) 5

Q3: On a scale of 0 - 5, how would you rate the weapon mechanics of the game? (Weapon mechanics include factors such as firing rate, recoil, bullet spread and more, for each weapon)

(no label) 5

Q4: On a scale of 0 - 5, how would you rate the flow of the game? (Flow refers to the pacing of the game e.g. setting up context of story through dialogue / cutscenes)

(no label) 3

Q5: On a scale of 0 - 5, how would you rate the Immersion of the game? (e.g. Sound effects, setting of the game, environments)

(no label) 5

Q6: Please give explanation for any aspect of the game you liked / disliked.

User Interface is visually appealing. Fits well with the theme of the game.

Flow of the game feels a little bit rushed.