CO3015 Computer Science Project

Games With Smartphones

Dissertation

Department of Informatics,
University of Leicester

3rd May 2018

Sayim khan

DECLARATION

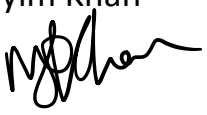
All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered

grounds for failure in this module and the degree examination as a whole.

Name: Sayim Khan

Signed: 

Date: 3rd May 2018

Abstract:

My problem is that I want to try to make my first 3D game for Android because of my love for games. However, I have no prior experience or knowledge of the process or tools behind making a game for a mobile platform. This posed several other issues in such a way I did not know where to start. What type of development environment I need, will the development use Java or a completely new language that I do not know, how to build and design the game world- and how do I even design basic game AI. These are a few problems and questions that I have to try and solve.

Since I had no prior experience or knowledge, my research mainly consisted of watching YouTube videos on common practices and techniques, world design and how to get started on Game Development. I used several Webpages for reference to things like building Game AIs and how to use or implement specific programming features/tools that I will need.

The project's conclusion was that, despite missing a few additional features and large amount of story content, it was a success which also met my goal which was being a 3D game using a modern game engine for the Android Mobile Platform. During the project, I was able to build my skills and experience and learnt a lot about process of Game Development.

Contents

Why a Unity-Based Android Game?

Why a Game?

When asked to pick and work on a yearlong project, I did not have an idea on where to start or what to do. I knew firstly that I wanted to do something that was interesting as a way to keep my motivation and attention on the project but did not know what. I knew that I was not going to do something ground breaking like solving a computational issue or designing a revolutionary system, as I know I don't have the skill or ambition to do something like that. Rather I felt that doing something that would push myself would make for a more interesting project, in that I would purposely put myself into a completely new area of software development. Thinking that it would help me learn about new development tools, processes, techniques and thought processes, all contributing to expand my skills and knowledge.

I am a really big 'gamer', started playing video games at four or five on the Gameboy Color, I currently own two consoles, three handheld gaming systems and a Gaming PC. I enjoy trying different types of games and experiences, other than technology.

I always thought about game development, but never did it because of my lack of experience, abysmal knowledge in typical game development processes or understanding of the required tools. I felt that if I wanted to push myself into doing something that I would feel uncomfortable about, however be really interested in, it would be Game Development.

Developing my own game would be a good project- because of my love and passion for games, it would keep me motivated. Additionally, it would be unfamiliar and uncomfortable enough to forcefully expand my knowledge and skills by researching them. In turn it leads to learning about a completely foreign area of Software Development.

Why did I choose Unity?

When researching where to start, I remembered that typically a large game would use a Game Engine. As an engine would handle the low-level processes, graphics rendering, hardware optimisation and provide a framework with game specific methods and classes to build upon. As a 'gamer', I knew about several Game Engines, there are some that are unavailable to the public like EA's Frostbite, Ubisoft's Anvil, etc. However, there are several paid engines such as Crytek's CryEngine. Thankfully they are popular and widely used free alternatives such as Epic's Unreal Engine 4 and the one I picked, Unity Technologies' Unity Engine 5.

As for why I chose Unity engine over Unreal, it was because the scripting language was C#. While researching C# because, I didn't know any C# at the time, I found out that it is VERY similar to Java and is also another higher-level language, C# learning curve would not be too hard and I was able to use my Java programming skills in C#. The other reason was that

when comparing Unreal to Unity, I found out that Unity performs a bit better on Smartphones than Unreal.

What are the Goals of my Project?

My personal goal is to broaden my skills as a programmer and introduce me to a new type of software development tools and procedures.

My main overall goal for the project is to create a fully 3D RPG with real-time combat. Breaking the overall goal down into smaller sub-goals are:

Firstly, to have fully functioning item and inventory system. What this meant is that I wanted an attempt at creating an item manager that would give the game player a facility to obtain new items from the game world then have them available to be used later on through a menu system. In this menu system, the player would be able to use a consumable item or equip an item what can be used in the game world, all while affecting the players stats in the form of decreasing or increasing.

Secondly, I wanted to create a simple but expandable NPC system that would allow the player to interact with the game world around them. The player could have a simple conversation with a person in the game world or listen to a request from an NPC and decide whether they wanted to accept the NPCs task. The System would be expandable in that it would be simple to add different types of NPCs that could do different things but still be interacted with the same way.

Thirdly, I wanted to a create a visual-novel style dialogue system with responses that would be accessible by all of the NPCs or sub characters if they wanted to converse with the player. NPCs would be able to start a conversation, not a one-sided dialogue with the player, in where the player would be able to listen to what the NPC had to say and be able to respond in a way that their Avatar would respond, this way the NPC would be able to react to what the player had said. All of this would be displayed in a clean and elegant way that would show the conversation in parts with the name of the speaker to make it easier to understand the conversation.

Next trying to figure out and implement a way of storing all of the conversation lines used by the NPCs and the dialogue system, that was simple to integrate with the dialogue system, but also allowed dialogue lines to be written in plain text in such a way that allowed decisions that would branch off into other dialogue, to be intertwined cleanly.

Next was to try to implement a simple real-time combat system with skills support. Every RPG will have some kind of fighting or combat mechanism to keep the game interesting, usually there are two types of combat systems, Turn-Based Combat and the one I wanted to try to implement in the game, Real-time Combat. The combat should also allow special attacks or abilities to be available to be used by the player and will either support the player to deal extra damage to the enemy, but at a cost of cooldown timer limiting the frequency of the skill usage.

Finally, the last goal of the game to have a basic but adaptive game AI. Nearly every game has some form of AI to compete against the player, usually it is an adaptive type of AI that uses data from the player, pre-determined algorithms, attack or behaviour patterns to help it compete against the player in situations when they would have to, for example predict and evade a player attack or decide on a move against the player. The AI usually also dictates the behaviour of the enemy outside of combat, for example whether they attack the player on-sight or after the player has done something, this is usually based on the specific role of the enemies.

Survey of Literature/ Information Sources:

Initial Research:

Before starting the game, I had only my gaming knowledge as a consumer. I knew the basic properties and structure of a few of the systems I was going to implement. However, I did not know how to implement these features or where to start.

During the first two or three weeks, I did research on several areas relating to the project. First, I researched and watched a tutorial on the language C# **[1]**, because before I could start with the project, I would have to become familiar with C# as I have never used or seen any C# code and didn't know the syntax or semantics of the language.

After I had finished researching C#, I decided to watch and read about the Unity and Game building **[2, 3, 4, 5, 6, 7, 8]**. I did this to understand to help me get started as well as point me in the right direction, because I had no idea or experience. I tried navigating the editor before starting this project and because I had no experience with Unity or a game editor in general, it was very daunting during the practice and during the first several weeks and there has a learning curve in understanding where everything in the editor was, but also trying to locate specific things like the performance optimiser and profiles.

Finally, I decided to watch a video on world building, more specifically level designing **[9]**. I wanted to see this so I could understand how to start designing and from where to start, because of this research I roughly understood how to start.

Research during the Development of the Project:

After creating the first floor called 'PrologueReal' (because the first prologue floor was a mess), I started to think about the game mechanics that I needed. The main things that I needed to research on how to do was moving the character and interacting with objects and characters. For the movement I watched a few videos **[1,2]** on how to register touch input from the screen on the device in engine and use that to get coordinates on the terrain. I found out about something called 'NavMeshAgent' when researching on google how ways to move a game object to a Vector3. I started my research on the NavMeshAgent by first read about it in the Unity Engine Manual **[10]**.

The next large part of research was on ideas on how to implement a Game AI. This is not a proper AI as it does not need to be self-aware or real 'Artificial Intelligence', but rather smart algorithms and code to understand and monitor the players behaviour, in some cases it uses the knowledge against the player in combat. I have never done anything relating to AI before, so I started a generic search on game AI, I found an interesting article on Intel's Developer page about game AIs **[11]**. It explained the basic ideas behind what a games AI should do and things to consider, it also explained about the basic state machine of an AIs state that it can be in. The article then explains the ideas behind an Adaptive game AI, this is where the certain behaviour is not hard-coded, but rather code than uses information and conditions to 'decide' on an action, such as trying to predict what the player will do based

on information such as past actions in similar situations, traits of the player and current viable options, this allows the Adaptive AI to ‘choice’ the best option.

Another Article which I found **[12]** was explaining the idea behind different types of enemy such as Emphasizers which encourage the use of a particular game mechanic, Enforcers that force the player to use specific game mechanics, etc. These different enemy type I was reading about can react differently from each other based on their roles which introduces similarities between their general behaviour, but difference unique traits to their specific types. This helped me understand that enemies are similar but they have to be coded and designed, in such a way, that they can have similar behaviours but must be allowed have different behaviours based on their roles. The article explained and introduced me to other important elements that must be considered when designing an enemy, this includes other sub-system like a tell system that helps inform the player of enemy actions with prompts, as well as, a system to handle the behaviour and coordinates the attacks of multiple enemies when the player engages with more than one enemy.

Requirements:

Functional Requirements:

World Navigation and Interactions:

As a Player, I …

- **Must be able to tap on a location in the game world and have the Avatar automatically move to that location**, so that I can navigate and move through the game world.
- **Must be able to click on an item chest and after the Avatar arrives at the chest, items in the chest will be acquired**, so that I can acquire new items or weapons to help me through the game.
- **Must be able to move the view camera in the same direction when moving the virtual analogue stick**, so that I can adjust my perspective and see more of the game world.
- **Must be able to tap on an enemy then after the Avatar moves into range, it will automatically start attacking the enemy**, so that I can engage in combat with an enemy blocking my path.
- **Must be able to tap on another location in the game world while in combat and the Avatar will disengage and move to the new location,** so that I can retreat from a combat situation with an enemy or attack them at a later time.
- **Must be able to initiate the story event when tapping on the story interaction icon**, so that I can view the event.
- **Must be able to see a blue interaction marker above an NPC Character who has a quest available for the player**, so that I know what a new quest is available for me and who to speak to receive the quest.
- **Must be able to initiate the quest dialogue scene when tapping on the Quest NPC interaction icon**, so that I can be told about the quest by the NPC and decide whether to accept to decline said quest.
- **Must be able to see the Dialogue Textbox appear when starting a conversation with an NPC or a Story dialogue scene with the Sub Characters**, so that I am able to see the text of what is being said in the dialogue.
- **Must be able to see the Dialogue Textbox disappear when finishing a conversation**, so that I can return back to the game world and continue playing.
- **Must be able to see the dialogue option appear on the UI when a Choice option is required during a dialogue scene**, so that I can see that a choice is required and be prepared to choose one.
- **Must be able to see the correct number of buttons for the total amount of choices available during the specific choice option**, so that there are no redundant buttons on the UI.
- **Must be able to tap on an option to select it as the chosen option**, so that I can easily pick my option with minimal hassle and keep the UI clutter free.
- **Must be able to see different dialogue text occur for each different dialogue option**, so that the dialogue has some significant meaning and variety.
- **Must be able to see the total amount of damage inflicted on an enemy after an attack above the enemy**, so that I know how much damage I am dealing to the opponent.

- **Must be able to see the 'Missed' text appear above the enemy during combat if the enemy avoids the attack**, so that I can be informed that my attack has been avoided.
- **Must be able to see a sign appear above the enemy when they notice my Avatar and try to engage in combat**, so that I can see that they have been alerted to my presence and are hostile.
- **Must be able to see a sign appear above the enemy when they are about to do an attack**, so that I can be informed that an attack is about to be performed, which lets me have enough time to prepare and perform actions such as an evade skill.
- **Must be able to see a sign appear above the enemy when they start to actively search for my Avatar after I have been able to successfully evade them and disengage in combat**, so that I can see that they are in a search state and they are trying to actively find me.
- **Must be able to see the enemy health bar where the red portion represents the amount of HP the enemy has left**, so that I have an idea of how much health the enemy has remaining until they are killed.

User Interface and Menu System:

As a Player, I …

- **Must be able to tap on the phone icon and the menu will appear**, so that I can have more option in relation to the Avatar's properties or perform other tasks relating to the game.
- **Must be able to tap on the 'Inventory' tab and the menu will display the Avatar's Inventory**, so that I can see what items the Avatar has obtained and change equipped weapons.
- **Must be able to Tap on an inventory item and see all the item's information in the Information tab**, so that I can read the item information.
- **Must be able to Tap on an inventory item and see a live item preview in the Information tab**, so that I can see what the item is physically.
- **Must be able to Tap on a Weapon item and see a 'Equip' button in the Information tab**, so that I can Equip a Weapon to my Avatar.
- **Must be able to Tap on a consumable item like a potion and see a 'Use' button in the Information tab**, so that I can Use an item on my Avatar.
- **Must be able to Tap on the 'Equip' button and have the Avatar's current weapon change to the Weapon displayed in the Live preview**, so that I can equip more powerful weapons acquired throughout the game.
- **Must be able to see the 'Equip' button become non-interactable after tapping on it**, so that I cannot equip a weapon that is already equipped.
- **Must be able to see the total amount of a consumable item decrease by one after clicking the 'Use' button on a consumable item**, so that I can see the correct amount of an item displayed after using one.
- **Must be able to see that the 'Use' button becomes non-interactable when the remaining item amount is zero**, so that I cannot use item's that I do not have.
- **Must be able to tap on the 'Stats' tab and have the current menu change to display the Stats menu**, so that I can see statistical and skill information on my Avatar.
- **Must be able to see a progress bar in the Stat display section showing how much Experience Points I have gained**, so that I know how much more until by Avatar will increase in a level.

- **Must be able to see the Attack label and the value in the Stat display section**, so that I can see what the Attack stat of my Avatar is and how hard their attacks are.
- **Must be able to see the Defence label and the value in the Stat display section**, so that I can see what the Defence stat of my Avatar is and how well they can take attacks.
- **Must be able to see the Agility label and the value in the Stat display section**, so that I can see what the Agility stat of my Avatar is and how quick they are in combat.
- **Must be able to see the HP label and the value in the Stat display section**, so that I can see how much HP they have and know how much damage they can take.
- **Must be able to see stat bonuses on the Stat boosts panel with their name and the boost value**, so that I can see what bonuses the Avatar has active because of the currently equipped weapon.
- **Must be able to see the 2 currently equipped skills icons in the skills section of the menu**, so that I can see what skills I have currently equipped.
- **Must be able to see a pop-up skill window when I tap on the 'Active Skill' button**, so that I can change my currently equipped skill and see what skills are available.
- **Must be able to see all available skills in a list in the left panel**, so that I can see all of the skills options that I have, which I can equip.
- **Must be able to see the skill information on the right information panel when I tap on a skill**, so that I know everything about the skill.
- **Must be able to see 2 slot buttons at the top of the Information panel that are interactable if the skill is unequipped**, so that I can equip a skill to any of the 2 slots if I like the skill.
- **Must be able to tap on one of the 2 skill slot buttons and have the skill be assigned to that slot**, so that during combat I can use the skill that I think is useful.
- **Must be able to see the 2 slot buttons become non-interactable after I equip the skill to one of the 2 slots**, so that I cannot equip a skill that is already equipped.
- **Must be able to see a lock icon in the Information panel when tapping on a locked skill**, so that I know that I cannot equip the skill.
- **Must be able to see the unlock requirement in the list next to a locked skill**, so that I can be informed of the requirement to unlock the particular skill.
- **Must be able to tap on the quest tab and the quest menu appears**, so that I can see what quests I have available that need to be completed.
- **Must be able to see a list containing all of the accepted quests**, so that I can see all of my quests that are available and need to be completed.
- **Must be able see the quest's title in the information panel when tapping on a quest from the list**, so that I can see what the quest is called.
- **Must be able to see an interactable 'Set as Active Quest' button in the information tab when tapping on a non-active quest**, so that I can set a quest to active and track the objective location in the game world.
- **Must be able to see a compass in the Game UI when specific types of quest are set as active**, so that I can be guided to the location of the quest objective.
- **Must be able to see the quest compass disappear when the active quest has been completed**, so that I am not misguided after the quest has been completed.
- **Must be able to see the Red bar of the HP Gauge shrink equivalent to the amount of HP lost during combat**, so that I have a correct indication of the remaining HP that the Avatar has.
- **Must be able to see the Red bar of the HP Gauge increase in size equivalent to the amount but always stay within the gauge restored when a HP potion is used from the Inventory**, so that I can visually see that the HP Potion has indeed restored the Avatar's HP.

- **Must be able to see a numerical value inside of the HP Gauge displaying the Avatar's current HP value and total HP**, so that I can see the exact amount of HP that the Avatar has remaining.
- **Must be able to additional skill buttons appear when I have equipped a skill**, so that I can activate the skills that I have decided to use.
- **Must be able to see the correct skill icon appear on the skill UI button respectively for the skill I have equipped via the skill's menu**, so that I have a visual aid as to what skill is equipped to which button.
- **Must be able to see the skill, light and heavy attack buttons become non-interactable after tapping on them**, so that I know that I did indeed use the skill.
- **Must be able to see the skill, light and heavy attack button cooldown animation where the button slowly fills up in a circle motion**, so that I can visually see how much longer until the skill is re-usable.
- **Must be able to see the skill, light and heavy attack button become interactable again after the button has been filled**, so that I can re-use a skill.
- **Must be able to see a notification at the top of the screen when I have open a chest with the names of the items that were acquired**, so that I know exactly what items I have just acquired.
- **Must be able to see a notification at the top of the screen when I have accepted a new quest that displays the new quest's title**, so that I can find and check the quest details in the quest menu via the quest's title.
- **Must be able to see a notification at the top of the screen when I have completed a quest**, so that I am informed when a quest is completed and need to return to the NPC who gave me the quest.

Combat + Skill System and AI Behaviour:

- **Must be able to Tap on the Quick Attack button and the Avatar will immediately perform a Quick attack**, so that that I have a way of executing a light but Quick attack to chain attacks together by using a UI Button in a real-time combat situation.
- **Must be able to Tap on the Heavy Attack button and the Avatar will immediately perform a slow but powerful attack**, so that I am able to execute a more powerful but slower attack by using a UI Button in a real-time combat situation.
- **Must be able to Tap on the UI Skill slot 1 button to perform the skill assigned to that button**, so that I can use the UI Skill 1 button to perform my assigned skill in a real-time combat situation when I need it.
- **Must be able to Tap on the UI Skill slot 2 button to perform the skill assigned to that button**, so that I can use the UI Skill 2 button to perform my assigned skill in a real-time combat situation when I need it.
- **Must be able to see the Avatar do a standard attack automatically every few seconds**, so that even when the other skills and attacks are on cooldown I can still attack.
- **Must be able to see that when I am in range of an idle enemy, they will automatically approach the Avatar and start attacking them**, so that it makes me aware of where the enemy is and my distance between us, as well as making the enemy AI adaptable in that it is able to notice the Avatars presence and react accordingly.
- **Must be able to see the enemy patrol a route when not in range of the Avatar**, so that the enemies AI is adaptable enough to act on their given task when idle.

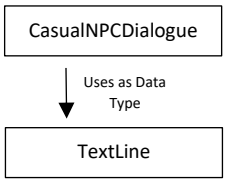
- **Must be able to see the enemy attack the player when on patrol, so that when the Avatar is in the area of their patrol route**, so the enemy AI is adaptable enough to see a threat to attack it on sight.
- **Must be able to see the enemy start to actively search for the Avatar's presence automatically after the Avatar has evaded the enemy**, so that the enemies AI is adaptable enough to actively seek out the opponent that they have lost 'sight' of.
- **Must be able to see the enemy automatically approach and attack the Avatar when it is actively seeking them out**, so that the enemy AI is adaptable enough to 'see' the target they were searching for and react accordingly.
- **Must be able to see the enemy 'give up' searching for the Avatar when they cannot find them**, so that the enemy AI is adaptable enough to not continuously search for a target they cannot find.
- **Must be able to see the enemy perform an offensive attack against the enemy**, so that the AI enemy has a way of damaging the Avatar.
- **Must be able to see the enemy AI 'predict' and try of avoid player non-skill attacks**, so that the enemy AI is able to use data learned from the player to 'predict' the probable chance of what attack they may do.
- **Must be able to see the enemy disappear after their HP bar has been completely depleted**, so the enemy can be killed and the combat situation can end so it does not go on forever.

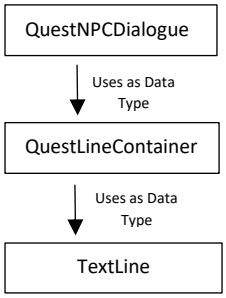
Software Details:

Design:

When designing the game, there were quite a few design decisions that I had to make when thinking about the overall structure of the sub-systems that make up the project. Here are a few reasons as to why I chose them:

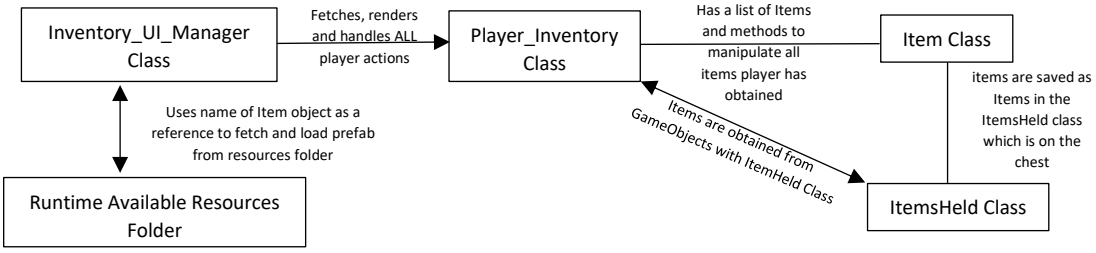
- **Handling NPC Quest Dialogue text –** When designing the Systems that handles the dialogue, I broke down what the overall system will do into two smaller systems, the CasualNPCDialogue and the QuestNPCDialogue:





The CasualNPCDialogue will manage the interesting but useless dialogue in the form of multiple TextLines for the NPC, while the QuestNPCDialogue class will handle all of the dialogue lines for the multiple quests that the NPC may have for the player as a QuestLineContainer, this in itself will contain multiple TextLines for all of the required dialogue for that specific Quest. This was the easiest way to differentiate the types of TextLines that can be used. By using this structure and Serialization, I can store the classes in JSON txt files, which allow me to add dialogue externally and not hardcoded **[11, 12, 13]**.

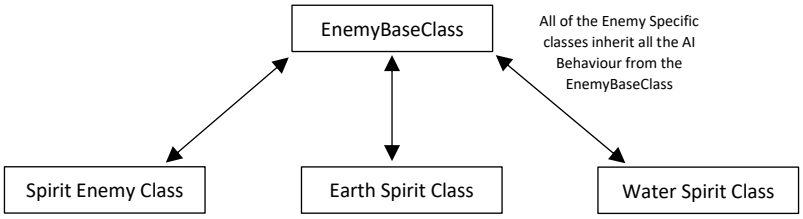
- **Player's Inventory System –** The Inventory system will manage all of the items that the player can obtain in the game.



The Inventory Systems main class is the 'Player_Inventory' as this is instantiated on the player's GameObject and acts like the player inventory and holds all of their items from potions to weapons. The Inventory system uses the Item Class to represent and hold item information such as total amount of the item available to the player and the name which is used as a reference to fetch the actual prefab and load it into the game. I needed a way for multiple items to be easily instantiated without having to hardcode it and had to decide where they would be instantiated. I decided on creating the ItemsHeld Class, this class contains all the item information such as name and amount. Since the variables are available publicly, I can see them in the Unity Engines inspector as property of the GameObject that I placed the script on and these public variables can be set here without hardcoding them, the reason for the list of primitive types is because unity's inspector can't handle custom data types, but a work around was that when the GameObject is created during runtime, the

Monobehaviour parent method Start() will create and store all of the items using data from the lists. By designing it this way, I can simply type in all the information for the items I want available to the player when they use a chest and these items will be auto-created with my information during runtime. I created the Player_UI_Manager to handle rendering and event handling the UI elements and this will display all of the items and information from the Player_Inventory class onto the game's UI, setting up a 3D Preview of an Item that the player has clicked on in the menu so they can see the object without equipping it as well as fetching data from Player_Inventory and using this data to fetch a prefab (Existing GameObject with classes attached to it) from the game's Resources folder and load it into the game world then set it in the correct hierarchy.

- **Enemy Types and basic Adaptive AI –** this set of classes will handle the different types of enemies and their basic AI functions.



In terms of the design for the enemy classes, it was designed in such a way that every enemy type will have their own dedicated class for their type specifics but also at the same time share the same basic behaviours that are common to the enemy type in general. I chose inheritance instead of interfaces because the basic AI behaviour of the enemy will be the same and inheritance means I don't have to repeat code and allows me to easily update all the enemies AI. The child classes in the hierarchy provide enemy specific information such as the enemies name and their type statistical data, this help with my goal of different enemy types. The parent class will provide the AI and combat components required to the enemies.

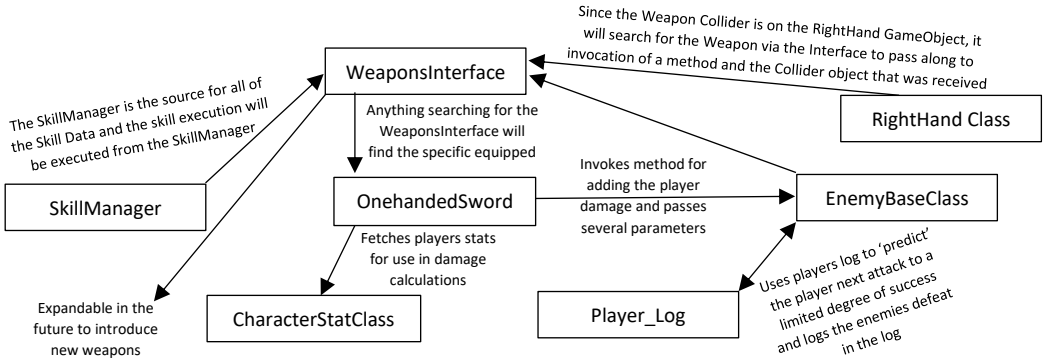
Initially, when designing the AI behaviour, I only had two states, 'Idle' for when the Enemy is doing nothing and is waiting, then I had an 'Alert' state for when the Enemy detects that the player is in range (Colliders enable this detection) and will automatically approach the player and attack them. After reading the article on the intel developer page **[11]**, I decided to try to implement several other different types of state to try and make the Enemies more 'Adaptive' in the sense that they would behave more 'human' like, e.g. looking around and trying to find something or someone.

As for the first addition state that I have to design was 'Pursing' state, this state was where if the player tried to escape from the enemy when they are in the 'Alert' state, the enemy will attempt to try and purse the player to reengage in combat, this is only while the player is in the enemies' range or 'vision', after that it will either resume its patrol or just wait.

Afterwards, for second state I decided to add was a simple 'Patrol' state, in which the enemy will patrol a set route and keep on patrolling until the player comes in range of it and then it will attack the player and initiate combat.

Lastly, for third state I added was an 'Aware' state, where the enemy had encountered the player and had lost sight of them, it will then generate several locations within the local area and go to these points and look around, trying to actively search for the player. When it does find the player, it will quickly approach them and try to attack them. If they do not find the player, the enemy will give up and resume what it was doing.

- **Real-time Combat System with skill support**



As for the first design issue I had to tackle, it was the player attacking the enemy and the registration of the damage. After doing a bit of research, I decided to use colliders in a similar way to how the enemy can detect the player. By placing a collider on the weapon hand then using the collider onTriggerEnter event **[8, 14]**, when the weapons hand collider hits and is triggered by the enemy's collider, it will automatically execute this method because Unity will look for a receiver method on any of the script on the GameObject with the key method name. In this case the receiver is on the RightHand script then using this I can find the weapon script via the WeaponsInterface and call the OneHandedSword Script method that I created for this purpose.

Now that I could have a method execute automatically, I needed to somehow send the damage from the player's weapon to the enemy. The easiest option to send the damage data to the object that the weapon hit was to use the Collider parameter that was passed when the method is executed because Unity will pass in the GameObject as the Collider Object, from this I can extract the GameObject reference and call the addDamage method which does the damage calculations.

The process for when the enemy inflicts damage on the player is done in a similar manner, but it will go through different classes. When the enemy attacks the player, they will calculate the total damage then call the add damage method on the player object from the CharacterStat class, which they obtained when the player entered their range. In the CharacterStat Class it will calculate the reduced damage and evasion chance, then deduct this amount from the player's HP.

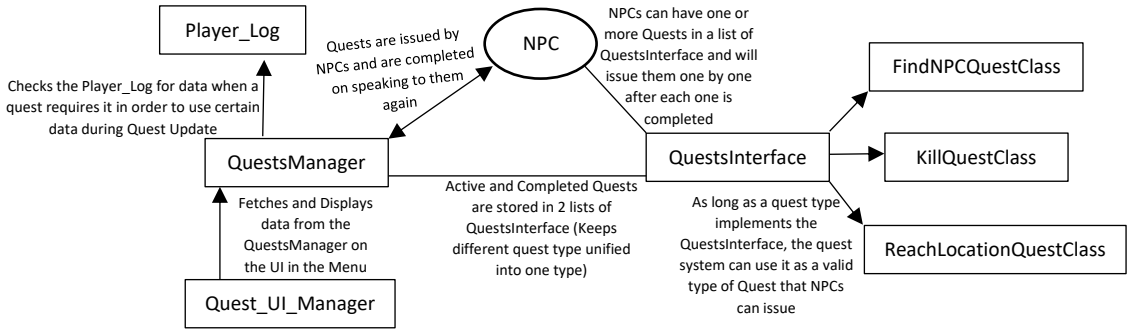
When thinking how to integrate the skill system, I wanted a core class that would use the WeaponsInterface to access the current equipped weapons script and set all normal attack parameters that were passed to the enemy to the correct parameters used by a skill such as skill name and multiplier. I decided to create the SkillManager to do this, as it is able to manage and store all of the players skills, it would also contain a list of skills that they player has available to them, track what skills are currently equipped and set on the two UI skill shortcut buttons, methods for adding and fetching skill data, calculating the damage done by the skill and act as an event handler for the UI Buttons that they player would use to execute the skills.

A skill can be broken down into two typical types, Offensive type where it would inflict more than normal damage to an enemy or a Defensive type where it would either protect the user from the enemy in some way or temporarily raise a certain stat of the player. I will need some way of distinguishing whether a skill is a Defensive or Offensive one, initially through

most of the development of the Skill System I had them as two different classes implementing a Skill_Interface, however I found that other than the different types, most of the other code and parameters were very similar or the same and I was just repeating code. I decided to change it by unifying the two classes under the SkillClass and used two constants called Offensive and Defensive then add a 'skillTyping' variable to the SkillClass to help to distinguish between the two types, this approach removed the duplicate code.

I decided to modify the design by adding a secondary effect, the reason being that specific skill like a 'Counter Attack' would have a Defensive Type in that it would allow the Player to evade an attack, but a less powerful secondary Offensive Type where the skill would cause attack damage to the enemy. By adding the optional that it configurable by the 'configureSkillSecondaryEffect' method it allowed the use of dual type skills as well as the single type skills.

- **Questing System –** The Questing System is where the player can talk to a specific type of NPC and they would have specific tasks that they would want the player to reach or complete.



When designing the QuestManager class, I wanted it to do several things, such as list of all the players current quests, completed quests, use data from the Player_Log to update and check all the quests and whether they are complete, be able to provide all the information required by the Quest UI script, and add new quests and set the player's waypoint marker to the quests location.

After implementing the main class, I had to implement the expandable quests where I could have the QuestsManager store and use all the different types of quest like they were one class, this would mean that only one implementation was needed to handle all the different and future types of quests. To achieve this, I used an interface to handle this and have all the quest types implement this interface, that would mean that all the quest types would be seen as the same type and can be stored in a unified list. This meant that if I was to add another type of quest in the future, theoretically the QuestsManager would not have to be adjust for this new addition, the interface just had to be implemented.

Implementation:

Here I will run though and explain specific and important pieces of code that is used within the game.

Enemy AI Behaviour (EnemyBaseClass):

This is the main class that contains the code for enemy behaviour, it has four specific states, Idle, Search, Patrol and Attack and these are all separated into separate methods, some use two or more methods due to them needing user interaction or collider events such as Attack.

The Attack code is broken down into several sections, first is the 'OnTriggerEnter(Collider col)' method, this is executed when anything enters the collider of the Enemy GameObject, the code from 158-165 is relevant to the Attack State behaviour. The Collider object has been checked before executing this code to see if it is the Player, then the collider assigns the reference to the Player class variable so that it is accessible outside of this method and to the whole class, next the 'Curious sign' which is used when searching was disabled to prevent TellSystem icon clashes.

The Enemies state is then changed to Alerted and the stopping distance of how far the NavMeshAgent will stop from the destination is also saved because it is modified throughout the rest of the code. Lines 162-165 are resetting engagement flags that are used to monitor the enemy state. This potion of code is important as it sets up the encounter and changes the enemies state.

The next part of the Attack behaviour is used to maintain combat and distance with the player and is handled by the 'OnTriggerStay(Collider col)' method. Lines 173-174 check and assign the player object. Line 176 will continuously check the distance between the Player and the enemy itself, if the distance is less than the specific amount then lines 178 to 182 will be executed which check to see if the enemy is in an attacking state and if it isn't then it will make the enemy attack. However, if the distance between the enemy and player is larger than the required amount then lines 184 -189 will execute. These lines will check if the enemy has stopped attacking (184), if it has not then the enemy will have switched into the stop attack state (185-187) by changing the animator trigger and the state flags. Finally on line 189, 'moveToPlayer()' will be executed.

The next method used during the Alert state is moveToPlayer that has previously mentioned. This is a small but important chunk of code from lines 363-371 and it will approach and follow the player as the name suggests. If the player is not null and is valid (365) then the stopping distance is saved (366) and the NavMeshAgent destination is changed to the players current position, so that the NavMeshAgent is always following the player (366-369).

The last method that is related to the Alert portion of the AI Behaviour is the 'OnTriggerExit(Collider col)' method (195-204). This method will automatically execute when a collider (Player GameObject) exits the enemies collider range and will first check to see if the collider is indeed the player object (196), if so then if will log the last position of the player by saving it to a variable called 'playerLastPosition' (197), then will null the player object because they exited the range and false the inCombat flag to indicate that the enemy is not in a combat situation. 'checkIfPlayerInRange()' will be executed and this other method will just send a virtual and invisible pulse outward from the centre of the enemy object collider to check if there are any other colliders in range that have the 'Player' tag. If there aren't any then the enemy state is changed to non-combat, this was done to fix a glitch where the enemy would auto-disengage combat occasionally. Finally lines 201 and 202 will check if the enemy is currently in a searching state and if the active search setting was enabled in the

Inspector, then will execute the 'initActiveSearch()' method which will switch the enemy into its other adaptive behaviour, such as actively searching for the player after they have been lost.

The next part of the AI Behaviour is the 'Search' state and is also like the 'Alert' state and is broken up into multiple different methods, primarily because it required an event and handler. The first of these methods is 'initActiveSearch()' (500-533) which is called by the 'OnExitTrigger' method and will initiate the search state. First it starts off by setting the 'currentlySearching' flag (514), so that the enemy is in a search state and then clears previous 'searchAreas' (518) to start fresh, then generate four different positions to search based on the player's last known position and the 'searchRadius' (521-525), which was set in the enemy settings in the Inspector. Finally, the method finishes after calling the animator on the enemy object and triggers the 'lostTarget' string trigger, which makes the enemy look around. At the end of this animation, another animation event is initiated which calls the 'resumeActiveSearch()' method.

This is the 2nd method of the Search state of the enemy, called 'resumeActiveSearch()' (537-594). It starts by first setting the animation trigger 'stopAreaSearch' which ends the current search animation (540) so the enemy can prepare to move to the new location, dependent on if the 'inCombat' flag has been set it will either stop searching or continue the search and will next check if there are any remaining areas left to search, if there is then it will get the next area index randomly (554) via the random generator instantiated earlier (550). After obtaining the next area to search is sets the enemies NavMeshAgent destination to the new location (558) and sets the 'movingToSearchArea' flag, so that the Update() (84-139) method will first log and reset the stop distance (96-99) so that it will try to stop exactly at the destination and then monitor the distance between the destination and the enemy to see when the enemy reaches the search destination (101). After the enemy has reached the destination, it will start the search animation (104) and remove the area from the remaining search areas (106) so that the current area is seen as finished and it can move on to another area later, finally it will unset the movingToSearchArea (107) to show that it is now not moving anymore.

The other option if there are not any more 'searchAreas' (552) left, it will disenable the 'CuriousSign' which is the search sign for the 'TellSystem' and set the animation trigger for the 'thinking' animation. Then the enemies next actions are dictated by the settable option (enablePatrolling) in the inspector (573), if true then the enemy will switch to the patrolling state (574), otherwise it will wait at the player's last known location (582-583) and idle or return to its starting position (579-580) and idle.

The last state to note of the AI's behaviour is the Patrol state, this has one primary method as it does not rely on other methods but is configurable via public variable accessible through the Editor's Inspector. The main method is 'Patrol()' (597-650), it starts by making sure that the 'currentlyPatrolling' flag is set (599), so the other method know that the enemy is currently in this state and also triggers the 'endPatrol' animation trigger (601) because this method is executed by a few other methods, as well as the Patrol animation, so it needs to end the animation to make sure there are no search animations player when the enemy is moving to reduce interferences. Next the enemy can only patrol when not in combat as it doesn't make sense to patrol when in combat and help reduce behaviour interference and the if statement check this condition (603). If the condition is not met then the method will finish execution because patrolling is not needed, however if the

condition is met then three things are initialized, first is the Random generator (605) for choosing a new location if the random flag was set, a ‘validNext’ flag (607) if the next patrol location is valid and the ‘temp’ variable for the index location in the list of positions called ‘newPosition’ (608). If random patrolling is enabled in the enemy inspector settings then the ‘if’ check (610) will execute lines 611-625, this block of code uses a while loop (612) to fetch a new index (614) for the list of positions set in the inspector and checks if the new index is the same as the old index (622), if it is then it will try again else it will set the ‘validNext’ flag to true and continue executing the method. If the ‘randomPatrolling’ flag was false (610) however, it will execute lines 628-631 which use an if statement to see if the last index is bigger than the list count (628), if no then the last index is just incremented (629), else it is reset to 0 (631).

The last block of code for the Patrol method will first fetch the ‘postion’ Vector3 from the list of patrol location using the new index position (636) and configure the NavMeshAgent (639-640) by resetting the stopping distance to 0 (639) so it stops at the exact location as well as setting the new destination (640) so the enemy will start to move to the new location in the game world. The method ends by setting the ‘movingToPatrolPos’ (645) which just monitors the current position of the enemy and the distance to the destination (120) and sets the trigger for the ‘waitAtPoint’ animation (126) so the enemy waits at the point, then it will re-call the Patrol method and just repeat the process until the player comes in range and the enemy will automatically switch states.

Critical Appraisal:

Now thinking about my original goals, I feel like the project in my eyes, personally is not finished in that the game mechanics are 80% which is because there are other systems such as party combat or engagement with a group of enemies that I would have liked to add in. Story and content wise the game is only about 20% done because only two floors are done.

The real-time system felt as if it well, however it is still rough and needs to be polished in areas such as the source code needing to be streamlined and more specific bugs need to be found and fixed. As my first game however, that I have ever made, I think it went well and just achieved my original impression of how a real-time combat system would be in an isometric view style mobile game would play out, not to mention the skill system was also added in and was more flexible than I had thought when I initial had the idea in that it allows skills to have two effects or one specific type.

I am very pleased with how the Dialogue System turned out and how the Dialogue is stored, in that the Dialogue System became larger than I thought. It can be used to display system wide notifications to the user by simply calling a method from anywhere. Additionally, it handles the display used to show Experience gains as well as finally is used to fetch and play though scripted dialogue sequences and allows simple branch dialogue based on the responses.

I like the way the scripted dialogue is handled because the scripted dialogue is written externally in a similar fashion to a play manuscript and when required by a story event is fetch by the Dialogue System, parsed and displayed line by line, additionally if I wanted to add a choice and side dialogue into a scripted scene I just use a '#' and the choice dialogue filename, then the Dialogue system will find that and add it in the dialogue seamlessly.

I finally feel that the Quest system and the way it handles quests has been done well, in that it is not fixed to specific number quest types, rather it is quick to add additional quest types, the interface just needs to be implemented.

On to a few things that I feel could have gone better, first is the Adaptive enemy AI where a few things that could be improved such as the attack prediction as it is only usable against standard, quick and heavy attacks and is basic in the way it 'predicts' attack in that it just looks at the past log and uses the pre-fixed player patterns to heavy influence the decision. It would be better if it looked at a bit more data such as player attack patterns such as after the same attack, what was the next attack and other types of data. Additionally, an improvement would be if it could also be used for 'predicting' skills as well as act based on several conditions such as player HP, enemy HP, active buffs and de-buffs, etc. It would then have to adapt to the changing situation, establish a type of flow in combat and achieve this by using different types of attacks and patterns, like the player.

Another thing I think did not go that well was the player movement and interaction that rely on raycasting and colliders. The way I implemented it is limited and clunky, as it sometimes did not want to register movement or registers because the raycast did not hit the collider, only the surrounding area. On a PC this kind of interaction system would not have these issues because a PC would use a mouse which is a much more precise pointing device against a finger which has a larger surface area than the tip of a mouse.

The last thing that I think did not go particularly well is the way the equipment is handled, due the only equippable weaponry being Swords/Katanas and there are no equip able armour. The weapon system approach was correct but could be implemented better because the weapon uses an interface to be accessed and used and all weapon specific animation or calculation was handled by

the implementation of the interface on the weapon specific classes. Because I put all the general weapon calculations and behaviour into the weapon implementation and not in a generic class on the Avatars hand object or has a central combat system, it would take some work to add in a weapon system as a decent amount of code will have to be repeated.

Thinking back, there are several things I would do differently, first being that I switch the movement style from point based to free-movement based. This means that in the game, currently to move the player you would tap on a location on the screen and the Avatar would automatically move to said location after finding a valid path using the NavMeshAgent. If I was to use a free-movement system, the player would have to move a virtual analogue stick and the Avatar would move in this direction and this would work for all 360° direction movement, this would help simplify the movement system as there would be less code to go wrong as the system is relying on the player to move around and path find, on contrast to now, where it is the other way around.

The other thing would be switching the combat system from real-time to turn based. A real-time combat system is where the combat participants freely attack each other without restriction, by contrast a turn-based system is a lot more controllable in that the participants decide on an action ahead of time and then based on a turn order which can be changed because of skills and agility stats, the participant will one by one perform their action and then repeat. Another key difference is that the real-time system takes place on the world or 'land' as the exploration, where as a turn-based one typically takes place on a separate and isolated area specifically created for these encounters, this means there is a clear separation of combat and exploration areas and would make it easier to manage because there would be two separate but more specialized UI for both situations.

During the development, I've been thinking what I would have to do to get it ready for a public release, more specifically as a free app on the Google Play Store. As a big 'gamer', on portable, console and PC gamer, I would look at the total amount of content available, this is in the form of the size of the levels, how big the core story (if it has one) is, how long it would take to finish and how many side things I can do such as side quests, collectable or exploration. I would have to finish the floors and ensure the story is finished, add more equip able weaponry, add equip able armour, add a larger variety of quest types and NPCs with dialogue, Google Play support with trophies and completely de-bug to the best of my ability, the last thing is to performance optimize the game to it runs smoothly at 30-40fps and set up several performance profiles or different hardware as not all phones will have a octa-core processor and a good GPU. The game would be feasible to deploy on the Play Store and hopefully would be received well by those who try it, provided these changed are implemented as there aren't many PC style RPGs on Android as there are casual mobile games.

The engine that I used was a good choice as Unity is one of the two most popular engines (The other being Unreal) that can work well on mobile platforms due to them being optimised for mobile devices and are developed to work well on ARM based processors just as well as x86 or x86-64 based processors. The modelling techniques that I am using such as mocking up drawings and then modelling using a 3D modelling such as Maya is one of the mainstream ways of modelling and creating 3D objects used in personal level as well as industry level, so this is an established technique. The way I am handling animation where I animate using an external animation software to animate the model, then export as an .fbx format and import that to Unity. This animation process is popular because of the fact it separates the model and the animation so that if the model

needs to be changed or edited and if the model uses the same ‘rig’ then the same animations can be used on the different model and it is a helpful paradigm in industry as it saves time.

My final goal for this project was to force myself out of my comfort zone and try something completely new, something that I have never done before to push myself. Now thinking back to what I had learnt and experienced throughout this past year, I have experience so many things and learnt a lot such as drawing humans, 3D modelling and how to use Maya, learned how to use Unity, learnt C#, learnt how to animate, how do design and build a very basic game AI, user interface design, how to use photoshop and thinking how to design complex game systems from a top-down style. These are all useful skills to have in creating my own game, learning C# and how to use and make games with Unity can be very useful on my CV and can be used to create a portfolio if I wanted to apply for position as a Game Developer as it shows my past experience and that I know the rough development process of a game, the drawing, 3D modelling and photoshop not sure at the moment how relevant they will be as I have beginner level skills in these areas, but if I were to continue developing these skills then they could prove useful if I pursue a career in Game Development, but the skills may be useful in other areas outside of Game Development such as Web Developer or Designer.

Conclusion:

From this project, I am very proud of what I have accomplished, but other people would not be impressed. As for myself it is very big in that I was able to improve my skills as well as experience a lot, the game can be used as part of my portfolio to demonstrate my skills.

With some more additional work and debugging the game can be put on the Play Store as a free app, so my goal of learning, experiencing and trying something new to expand my skills was a success. The other goal was to make a playable RPG with real-time combat that also had skills and in hindsight despite the improvement and being my first attempt at building a game, I think it did achieve my goal that I set because you can explore a 3D world, you can fight enemies in real-time and you can use skills during combat, you can partake in quests, talk to NPCs and make choices.

As a 'gamer', I can acknowledge this as an RPG game as it does what an RPG is expected to do, just needs to be improved and that thanks to this project I was able to grow, build and experience something related to my passion.

Bibliography:

[1] https://www.youtube.com/watch?v=lisiwUZJXqQ&list=PL-ptVPMXd-fbS8ebX9BGoYU0etgwnjiEK&index=3&t=0s, 'C# Tutorial', Derek Banas, 5 July 2015.

[2] https://www.youtube.com/watch?v=nM0h5pQYQxM&list=PL-ptVPMXd-fbS8ebX9BGoYU0etgwnjiEK&index=1&t=0s, 'How to Make an Android Game with Unity – Complete Tutorial 2017', Charger Games, 6 February 2017.

[3] https://www.youtube.com/watch?v=36ruPSD7FJk&list=PL-ptVPMXd-fbS8ebX9BGoYU0etgwnjiEK&index=2&t=0s, 'Level Design and Click to Move | Making a Simple RPG – Unity 5 Tutorial (Part 1)', GameGrind, 28 October 2016.

[4] https://www.youtube.com/watch?v=k-FVQ3jpRN8, 'Interactable Items and NPCs | Making a Simple RPG – Unity 5 Tutorial (Part 2)', GameGrind, 30 October 2016.

[5] https://www.youtube.com/watch?v=mXjRR1nnC5M, 'Dialogue System and Interaction | Making a Simple RPG – Unity 5 Tutorial (Part 3)', GameGrind, 4 November 2016.

[6] https://www.youtube.com/watch?v=wqEk5mzJB3M, 'Character Stats | Making a Simple RPG – Unity 5 Tutorial (Part 4)', GameGrind, 15 November 2016.

[7] https://www.youtube.com/watch?v=7T4dFqT62Js, 'Weapon Equipping | Making a Simple RPG – Unity 5 Tutorial (Part 5)', GameGrind, 29 November 2016.

[8] https://www.youtube.com/watch?v=HrNebvxSUsU, 'Sword Attack | Making a Simple RPG – Unity 5 Tutorial (Part 6)', GameGrind, 15 December 2016.

[9] https://www.youtube.com/watch?v=wRkXuTMWrG8, 'How to Design a Level in Unity 5', Sykoo, 21 February 2016.

[10] https://docs.unity3d.com/Manual/nav-MoveToDestination.html, 'Unity Documentation', 'Telling a NavMeshAgent to Move to a Destination', Unity Technologies.

[11] https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1, 'Designing Artificial Intelligence for Games (Part 1)', Intel Developer Zone, D. Kehoe, Published June 20, 2009, Updated January 1, 2015.

[12] https://www.gamasutra.com/blogs/BartVossen/20150504/242543/Enemy_design_and_enemy_AI_for_melee_combat_systems.php, 'Enemy design and enemy AI for melee combat systems' Gamasutra, B. Vossen, Published April 5, 2015.

[11] https://docs.unity3d.com/Manual/JSONSerialization.html, 'Unity Documentation', 'JSON Serialization', Unity Technologies.

[12] https://docs.unity3d.com/ScriptReference/JsonUtility.FromJsonOverwrite.html, 'Unity Documentation', 'JsonUtility.FromJsonOverwrite', Unity Technologies.

[13] https://docs.unity3d.com/ScriptReference/JsonUtility.ToJson.html, 'Unity Documentation', 'JsonUtility.ToJson', Unity Technologies.

[14] https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html, 'Unity Documentation', 'JsonUtility.ToJson', Unity Technologies.

Appendix:

EnemyBaseClass (1 - 60)

EnemyBaseClass.cs

Miscellaneous Files

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.AI;
6  using UnityEngine.UI;
7
8  public class EnemyBaseClass : MonoBehaviour {
9
10     //Spirits Initial Parameters
11     public int Level;
12     public int baseExpGain;
13     protected int MaxHP;
14     protected int currentHP;
15     protected int AttackStat;
16     protected int DefenceStat;
17     protected int AgilityStat;
18
19     //access to UI
20     public GameObject HealthDisplay;
21     public GameObject DamageDisplay;
22     public GameObject TellSystem;
23     public GameObject MainCamera;
24
25     //Settings for Patrolling (startPatrol and resumePatrol)
26     public bool enablePatrolling;
27     public bool randomPatrolling;
28     protected bool currentlyPatrolling;
29     public List<Vector3> patrolCoordinates = new List<Vector3>();
30     protected int lastPatrolPosition = 0;
31     protected Vector3 patrolPosVector;
32     protected bool movingToPatrolPos = false;
33
34
35     //holding player instance on contact
36     GameObject player;
37
38     //used for monitoring player position and following
39     protected bool inCombat = false;
40     protected bool attack = false;
41     protected bool stop = true;
42     protected Vector3 playerLastPosition;
43     protected Vector3 spiritsStartPosition;
44
45     //cross method variables for active search (initActiveSearch) and (resumeActiveSearch)
46     public bool enablePlayerActiveSearch;
47     public float searchRadius;
48     protected List<Vector3> searchAreas = new List<Vector3>();
49     protected bool currentlySearching = false;
50     protected bool movingToSearchArea = false;
51     protected int areaSelector = 0;
52     protected float savedStoppingDistance;
53
54     //used for attack type prediction percentages
55     protected float lastAttackTimeSeconds = 0;
56     protected Dictionary<string, int> percentages = new Dictionary<string, int>();
57
58     //Optional Action to perform on defeat
59     Action actionOnDefeat;
60
```

EnemyBaseClass (61 - 123)

```
61      // Use this for initialization
62      public virtual void Start() {
63
64          HealthDisplay.transform.Find("LevelDisplay").gameObject.GetComponent<Text>().text =  Level.ToString();
65          //storing the spirits initial position
66          spiritsStartPosition = transform.position;
67          savedStoppingDistance = gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance;
68          //disabling box collider to prevent interference between the two colliders
69          gameObject.GetComponent<BoxCollider>().enabled = false;
70
71          //setting values and keys for all possible attacks
72          percentages.Add(Player_Log.QuickAttack, 0);
73          percentages.Add(Player_Log.HeavyAttack, 0);
74          percentages.Add(Player_Log.StandardAttack, 0);
75          //----Add new Attacks here-----
76
77          //setting up patrolling settings
78          if (enablePatrolling && (patrolCoordinates.Count == 0))
79              enablePatrolling = false;
80
81          if (enablePatrolling) Patrol();
82      }
83
84      void Update() {
85          //using Update to change the position of the HealthDisplay in relation to the enemy position every frame
86          HealthDisplay.transform.LookAt(MainCamera.transform);
87          TellSystem.transform.LookAt(MainCamera.transform);
88          DamageDisplay.transform.LookAt(MainCamera.transform);
89
90
91          //monitoring navMeshAgent
92          if (movingToSearchArea && !inCombat) {
93
94
95              if (gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance != 0) {
96                  savedStoppingDistance = gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance;
97                  gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
98              }
99
100             if (Vector3.Distance(searchAreas[areaSelector], gameObject.transform.parent.transform.position) < 2) {
101                 //gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = savedStoppingDistance;
102                 gameObject.GetComponent<Animator>().SetTrigger("startAreaSearch");
103
104                 searchAreas.RemoveAt(areaSelector);
105                 movingToSearchArea = false;
106             }
107         }
108
109         //this will only check the distance between the player and a patrol position when not in combat and when moving
110         if (movingToPatrolPos && !inCombat) {
111
112
113             if (gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance != 0) {
114                 savedStoppingDistance = gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance;
115                 gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
116             }
117
118             if (Vector3.Distance(gameObject.transform.parent.transform.position, patrolPosVector) < 2) {
119
120
121                 movingToPatrolPos = false;
122                 gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = savedStoppingDistance;
123                 gameObject.GetComponent<Animator>().SetTrigger("waitAtPoint");
```

EnemyBaseClass (124 - 199)

```
124
125
126                 }
127             }
128
129
130             // only face the player continuously when they are in combat
131             if (inCombat && player != null)
132                 gameObject.transform.parent.transform.LookAt(new Vector3(player.transform.parent.position.x,
133                     player.transform.parent.position.y+.7f, player.transform.parent.position.z));
134
135         }
136
137
138         //when the player enters range, start attacking and make them the target
139         void OnTriggerEnter(Collider col) {
140             if ("Player".Equals(col.tag)) {
141
142                 if (currentlySearching) {
143                     gameObject.GetComponent<Animator>().SetTrigger("attack_Player");
144                     currentlySearching = false;
145                 }
146
147                 if (currentlyPatrolling) {
148                     gameObject.GetComponent<Animator>().SetTrigger("endPatrol");
149                     currentlyPatrolling = false;
150                 }
151
152                 player = col.gameObject;
153                 TellSystem.transform.Find("CuriousSign").gameObject.GetComponent<Text>().enabled = false;
154                 TellSystem.gameObject.GetComponent<Animator>().SetTrigger("alerted");
155                 gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = savedStoppingDistance;
156                 attack = false;
157                 stop = true;
158                 inCombat = true;
159                 movingToSearchArea = false;
160             }
161         }
162
163         /* this is the TriggerStay collider which is used to monitor the player when they are within range
164          * this method will also follow the player and attack, if they try to run away
165          */
166         void OnTriggerStay(Collider col) {
167             if ("Player".Equals(col.tag)) {
168                 player = col.gameObject;
169
170                 if (Vector3.Distance(gameObject.transform.parent.transform.position, player.transform.parent.transform.position) < 5) {
171
172                     if (!attack) {
173                         gameObject.GetComponent<Animator>().SetTrigger("attack_Player");
174                         attack = true;
175                         stop = false;
176                     }
177                 } else {
178                     if (!stop) {
179                         gameObject.GetComponent<Animator>().SetTrigger("stop_Attack");
180                         attack = false;
181                         stop = true;
182                     }
183                     moveToPlayer();
184                 }
185             }
186         }
187
188         //when the player leaves range stop targeting them
189         void OnTriggerExit(Collider col) {
190             if ("Player".Equals(col.tag)) {
191                 playerLastPosition = player.transform.position;
192                 player = null;
193                 inCombat = false;
194                 checkIfPlayerInRange();
195                 if ((!currentlySearching) && enablePlayerActiveSearch)
196                     initActiveSearch();
197             }
198         }
199
```

EnemyBaseClass (200 - 275)

```
200     /*Method for adding Damage to the Enemy, also checks to see if the gameObject has 0 health and destroys if true
201     *this method need the damage recieved in integer and the AttackName must be in the format of 'QuickAttack'
202     *
203     */
204     public int addEnemyDamage(int damageReceived, string AttackName, bool isSkill, int skillAccuracy) {
205
206         //this will log the players new attack
207
208         Debug.Log(isSkill);
209
210         if (!isSkill) {
211             player.GetComponent<Player_Log>().logPlayersAttack(AttackName);
212         } else {
213             //if the Attack is a skill then do some other type of logging
214         }
215         //this is whether or not the attack from the player will hit the enemy
216
217         float hpPercent = ((float)currentHP / (float)MaxHP) * 100;
218
219
220
221         //if the enemy is less than 40% health then enemy will become more defensive and the evasion chance will increase
222         if (hpPercent < 40) {
223
224             if (!isSkill) {
225                 int percentChange = (percentages[AttackName]);
226
227                 //Double Check this
228                 if (!willHitChance(percentChange + 20)) {
229                     damageReceived = 0;
230                 }
231             } else {
232
233                 if (!willHitChance(100 - (skillAccuracy - 10))) {
234                     damageReceived = 0;
235                 }
236
237
238             }
239
240         } else {
241
242             if (!isSkill) {
243
244                 if (!willHitChance((percentages[AttackName] - 30))) {
245
246                     damageReceived = 0;
247                 }
248             } else {
249                 if (!willHitChance(100 - skillAccuracy)) {
250                     damageReceived = 0;
251                 }
252             }
253         }
254
255
256
257         //this method is used to calculate the chances of which attacks will come next, these percentages will be used to
258
259         if (!isSkill)
260             getHitEvadePercentage();
261
262         currentHP = currentHP - damageReceived;
263
264         if (currentHP < 0)
265             currentHP = 0;
266
267         DamageDisplay.transform.Find("Text").GetComponent<Text>().text = "";
268         DamageDisplay.transform.Find("Text").GetComponent<Animator>().SetTrigger("display");
269
270         if (damageReceived != 0)
271             DamageDisplay.transform.Find("Text").GetComponent<Text>().text = damageReceived.ToString();
272         else {
273             DamageDisplay.transform.Find("Text").GetComponent<Text>().text = "Missed";
274         }
275         updateHPBar();
```

EnemyBaseClass (276 - 351)

```
276
277         if (currentHP == 0) {
278             if (player != null) {
279
280                 //Calculating the Exp Gain
281                 //baseExpGain
282
283                 if (player.GetComponent<CharacterStats>().CharacterLevel == Level)
284                     baseExpGain = baseExpGain/10;
285                 else if ((Level - player.GetComponent<CharacterStats>().CharacterLevel) <= 2)
286                     baseExpGain = baseExpGain / 2;
287
288                 player.GetComponent<CharacterStats>().addExperiencePoints(baseExpGain);
289                 player.GetComponent<Player_Log>().incrementTotalEnemyDefeated("Spirit");
290
291                 if (actionOnDefeat != null) {
292                     actionOnDefeat.DynamicInvoke();
293                     actionOnDefeat = null;
294                 }
295             }
296             Destroy(gameObject.transform.parent.gameObject);
297
298
299         }
300
301         return currentHP;
302     }
303
304
305
306     //this method will Add Enemey damage to the Player and decrease their health
307     public void addDamageToPlayer() {
308
309         if (player != null) {
310             player.GetComponent<CharacterStats>().addDamageToPlayer(AttackStat, AgilityStat);
311         }
312
313     }
314
315     //method for updating the HP Bar in World Space
316     void updateHPBar() {
317
318         float currentHPFloat = (float)currentHP;
319         float maxHPFloat = (float)MaxHP;
320         float percentageHP = (float)(currentHPFloat / maxHPFloat);
321
322         HealthDisplay.transform.Find("Bar").gameObject.GetComponent<Image>().fillAmount = percentageHP;
323     }
324
325     //this method will make the enemy approach the player
326     public void moveToPlayer() {
327
328         if (player != null) {
329             gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = savedStoppingDistance;
330             gameObject.transform.parent.GetComponent<NavMeshAgent>().SetDestination(
331                 new Vector3(player.transform.parent.position.x, player.transform.parent.position.y,
332                 player.transform.parent.position.z));
333         }
334     }
335
336     //this is a 'tell' method and will warn the player of an attack, in attempt to give the player time to prepare
337     public void warnPlayerOfStandardAttack() {
338         TellSystem.gameObject.GetComponent<Animator>().SetTrigger("standard_attack_warning");
339     }
340
341     //use this method to determine if the attack will hit
342     public bool willHitChance(int percentageChance) {
343
344         if (percentageChance > 100)
345             percentageChance = 100;
346
347         if (percentageChance < 0)
348             percentageChance = 0;
349
350         System.Random rnd = new System.Random();
351
```

EnemyBaseClass (351 - 426)

```
351
352         int genNo = rnd.Next(0, 101);
353
354         if (genNo > percentageChance)
355             return true;
356         else
357             return false;
358     }
359
360     //this method is used for calculating the percentage that the attack will hit,
361     //based on the current hit change and the previous attack
362     /*
363     *Attack cooldowns:
364     * - Heavy Attack = 10s, 15s with 5s offset
365     * - Quick Attack = 3s, 5s with 2s offset
366     */
367     public int getHitEvadePercentage() {
368
369         //Initializing required datatypes
370         Player_Log player_Log = player.GetComponent<Player_Log>();
371         Dictionary<string, int> temp = new Dictionary<string, int>(); //used for temperarily holding values
372
373
374         //this for each uses the player's combat history to calculate the base percentage changes
375         foreach (var attackType in percentages) {
376
377             float tempCount = player_Log.getLogDataForSpecificAttack(attackType.Key);
378             float tempTotal = player_Log.getTotalLoggedAttacks();
379             float initialPercentage = (tempCount / tempTotal) * 100;
380
381             temp.Add(attackType.Key, (int)initialPercentage);
382         }
383
384         //moving temp values into main Dictionary
385         foreach (var newVal in temp) {
386             percentages[newVal.Key] = newVal.Value;
387         }
388
389         //emptying temp holder
390         temp.Clear();
391
392         //these are percentage adjustments based on simple tactics and cooldown times
393
394         //Heavy attack checks and adjustments
395
396         //if it was the last attack
397         if (player_Log.getLastAttack().Equals(Player_Log.HeavyAttack))
398             if (percentages[Player_Log.HeavyAttack] > 0) {
399
400                 int half = percentages[Player_Log.HeavyAttack] / 2;
401                 percentages[Player_Log.HeavyAttack] = percentages[Player_Log.HeavyAttack] - half;
402
403                 if (percentages[Player_Log.QuickAttack] + half < 100)
404                     percentages[Player_Log.QuickAttack] = percentages[Player_Log.QuickAttack] + half;
405             }
406
407         //if the last attack is within 15s, normally would be 100 percent, but allows a lucky attack to the player, hence 15%
408         if (player_Log.getLastAttack().Equals(Player_Log.HeavyAttack) && (getAttackTimeDifference() < 10)) {
409             percentages[Player_Log.HeavyAttack] = 0;
410             percentages[Player_Log.QuickAttack] = 85;
411             percentages[Player_Log.StandardAttack] = 15;
412         }
413
414
415         //Quick attack checks and adjustments
416
417         //if the last attack is within 5s
418         if (player_Log.getLastAttack().Equals(Player_Log.QuickAttack) && (getAttackTimeDifference() <= 5)) {
419             percentages[Player_Log.HeavyAttack] = 85;
420             percentages[Player_Log.QuickAttack] = 0;
421             percentages[Player_Log.StandardAttack] = 25;
422         }
423
424         //      use method check 2nd last attack
425         if (player_Log.get2ndLastAttack().Equals(Player_Log.QuickAttack) && (getAttackTimeDifference() > 5)) {
426             percentages[Player_Log.HeavyAttack] = percentages[Player_Log.HeavyAttack] - 10;
```

EnemyBaseClass (427 - 502)

```
427             percentages[Player_Log.QuickAttack] = percentages[Player_Log.QuickAttack] + 30;
428             percentages[Player_Log.StandardAttack] = percentages[Player_Log.StandardAttack] - 10;
429         }
430
431         lastAttackTimeSeconds = Time.time;
432         return 0;
433     }
434
435
436     //this method is used to get the about of time elapsed since the last attack, to help calculate percentages for attack evasion
437     public float getAttackTimeDifference() {
438
439         return Time.time - lastAttackTimeSeconds;
440
441     }
442
443     //calling this method will start an active search to look for player within a certain range
444     public void initActiveSearch() {
445
446         //generate the search coordinates and save them in an list of vectors3
447         // use a random number gen to check pick a coordinate
448
449         //use animation rotateSearch to slowly spin around and looks for player, interrupt when found the player
450         //if completed search using a callback method from the anaimation, remove position from array and pick a new number from
451         //the new range and repeat.
452         //if finish randomly chose to either return to old position or wait at the last location
453
454         //changing passive colider to long range search collider
455
456         currentlySearching = true;
457
458         //setting the search locations
459
460         searchAreas.Clear();
461
462         //adding search areas
463         searchAreas.Add(new Vector3((playerLastPosition.x + searchRadius), playerLastPosition.y, playerLastPosition.z));
464         searchAreas.Add(new Vector3((playerLastPosition.x - searchRadius), playerLastPosition.y, playerLastPosition.z));
465
466         searchAreas.Add(new Vector3(playerLastPosition.x, playerLastPosition.y, (playerLastPosition.z + searchRadius)));
467         searchAreas.Add(new Vector3(playerLastPosition.x, playerLastPosition.y, (playerLastPosition.z - searchRadius)));
468         //can add more points later
469
470
471
472         gameObject.GetComponent<Animator>().SetTrigger("lostTarget");
473     }
474
475
476
477     public void resumeActiveSearch() {
478
479
480         gameObject.GetComponent<Animator>().SetTrigger("stopAreaSearch");
481
482         if (!inCombat) {
483             TellSystem.transform.Find("CuriousSign").gameObject.GetComponent<Text>().enabled = true;
484
485             System.Random rnd = new System.Random();
486
487             if (searchAreas.Count > 0) {
488
489                 areaSelector = rnd.Next(0, searchAreas.Count);
490
491
492
493                 gameObject.transform.parent.GetComponent<NavMeshAgent>().SetDestination(searchAreas[areaSelector]);
494                 //savedStoppingDistance = gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance;
495                 //gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
496
497                 movingToSearchArea = true;
498
499
500
501             } else {
502
```

EnemyBaseClass (503 - 577)

```
503                 TellSystem.transform.Find("CuriousSign").gameObject.GetComponent<Text>().enabled = false;
504                 TellSystem.GetComponent<Animator>().SetTrigger("thinking");
505
506                 if (enablePatrolling)
507                     Patrol();
508                 else {
509
510                     //change to continue patrolling if it is enabled
511                     if (rnd.Next(0, 2) == 0) {
512                         gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
513                         gameObject.transform.parent.GetComponent<NavMeshAgent>().SetDestination(spiritsStartPosition);
514                     } else {
515                         gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
516                         gameObject.GetComponent<NavMeshAgent>().SetDestination(playerLastPosition);
517                     }
518
519                     currentlySearching = false;
520                 }
521
522             }
523         } else {
524             gameObject.GetComponent<Animator>().SetTrigger("stopAreaSearch");
525             searchAreas.Clear();
526         }
527     }
528
529
530     public void Patrol() {
531
532         currentlyPatrolling = true;
533
534         gameObject.GetComponent<Animator>().SetTrigger("endPatrol");
535
536         if (!inCombat) {
537
538             System.Random rnd = new System.Random();
539
540             bool validNext = false;
541             int newPosition = 0;
542
543             if (randomPatrolling) {
544                 while (!validNext) {
545
546                     newPosition = rnd.Next(0, patrolCoordinates.Count);
547
548
549
550                     if (lastPatrolPosition != newPosition)
551                         validNext = true;
552
553                     lastPatrolPosition = newPosition;
554
555                 }
556
557             } else {
558                 if (lastPatrolPosition < (patrolCoordinates.Count - 1))
559                     lastPatrolPosition = lastPatrolPosition + 1;
560                 else
561                     lastPatrolPosition = 0;
562             }
563
564
565
566             patrolPosVector = patrolCoordinates[lastPatrolPosition];
567
568
569             gameObject.transform.parent.GetComponent<NavMeshAgent>().stoppingDistance = 0;
570             gameObject.transform.parent.GetComponent<NavMeshAgent>().SetDestination(patrolPosVector);
571
572
573             movingToPatrolPos = true;
574
575
576
577         }
```

EnemyBaseClass (578 - 628)

```
578     }
579
580
581     public void checkIfPlayerInRange() {
582
583         bool playerInRange = false;
584
585         Collider[] objects = Physics.OverlapSphere(gameObject.transform.parent.transform.position, gameObject.GetComponent<SphereCollider>().radius);
586
587         if (objects.Length > 0) {
588             foreach (var item in objects) {
589
590                 if (item.tag.Equals("Player"))
591                     playerInRange = true;
592
593             }
594         }
595
596
597         if (!playerInRange) {
598
599             inCombat = false;
600
601             gameObject.GetComponent<Animator>().SetTrigger("stop_Attack");
602         }
603
604     }
605
606     public void enableThinkingTell() {
607
608         TellSystem.GetComponent<Animator>().SetTrigger("thinking");
609
610     }
611
612     public int getDefenceStat() {
613         return DefenceStat;
614     }
615
616     public int getAttackStat() {
617         return AttackStat;
618     }
619
620     public int getAgility() {
621         return AgilityStat;
622     }
623
624     public void setActionToPerformOnDefeat(Action actionToPerform) {
625         actionOnDefeat = actionToPerform;
626     }
627 }
628
```