City, University of London

BSc Computer Science with Games Technology

Final Year Project Report

First-Person Stealth Game Prototype

By

Tayyab Hussain

Consultant: Martin Walter

Table of Contents

1. Abstract

Third-Person Stealth Games have been a staple in the Games industry for over a decade. Games such as Hitman 3(IO Interactive, 2021) and Splinter Cell: Blacklist(Ubisoft Toronto, 2013) are regarded by many as some of the best games ever made, not just in terms of stealth. Naturally, game studios attempted to replicate this success in the form of First-Person Stealth Games/Levels. This was largely unsuccessful. A market gap exists for an intriguing and tense First-Person Stealth Game/Level.

This project aimed to develop a First-Person Stealth Game Prototype that would implement some of the features which make blockbuster Third-Person Stealth Games great. To do this, the project was divided into five key areas to explore, Artificial Intelligence, User Interface, Game Mechanics, Movement/Traversal and Game Balance.

2. Introduction

2.1. Description of the Problem

Stealth games have been highly prevalent in the last decade. Games like Hitman 3(IO Interactive, 2021), Batman Arkham Knight(Rocksteady Studios, 2015), Metal Gear Solid V: The Phantom Pain(Kojima Productions, 2015) and Splinter Cell: Blacklist(Ubisoft Toronto, 2013) are all exceptional examples of good stealth games. However, these are all Third-Person games. There is a severe lack of First-Person Stealth games. Dishonoured 2(Arkane Studios, 2016) is the only relatively recent game that meets the First-Person Stealth criteria.

There have been many attempts by games such as Call of Duty: Modern Warfare(Infinity Ward, 2019) and Battlefield 1(DICE, 2016) to include stealth levels within their FPS games. However, these often feel shoehorned in. A few areas in which these games need to improve compared to their Third-Person counterparts include a poor stealth AI, an unintuitive UI design for stealth, a lack of stealth-specific mechanics, limited movement/traversal options and a lack of balance between the player and the enemies.

This project aimed to create a prototype First-Person Stealth game that would incorporate a few of the features that make Third-Person Stealth games great. The specific areas the project looked at were the Stealth AI, Stealth UI, Stealth Mechanics, Player Movement/Traversal and Balancing the player and enemies.

2.2. Project Objectives

The project's main objective was to create a First-Person Stealth level using Unity. This included five main aspects, AI, UI, Mechanics, Movement and Balance. The exact objectives were defined as follows:

2.2.1. Stealth AI

a. The stealth AI shall be designed with a behaviour tree with at least five states for the enemy.
b. The AI shall use a pathfinding algorithm to find the player when in the searching state.
c. The AI shall not have a binary detection of the player. The AI should slowly detect the player over time and not instantly go into a state of alert when they spot the player.

2.2.2. Stealth UI

a. There shall be a small icon to allow the player to determine the current state of an enemy.
b. There shall be a UI element allowing the player to determine whether they are about to be spotted.

2.2.3. Stealth Mechanics

a. There shall be two different stealth mechanics/gadgets that the player can use in the level.
b. The player shall only be able to use the mechanic a finite number of times within the level.

2.2.4. Movement/Traversal

a. The player shall have a unique and original way to traverse the level that differs from walking, running, crouching, and crawling prone.

2.2.5. Balance

a. Enemies shall be much stronger than the player in terms of health and damage they can do.
b. The weapons the player can use shall be very weak when taking on multiple enemies.
c. The player shall not regen health or have any way to replenish health in the level.

2.2.6. Other Functionality

a. Main menu and pause menu.
b. Audio and Visual FX.
c. Saving and Loading.

2.3. Project Beneficiaries

The primary beneficiary of this project was other developers specifically working on FPS games who wanted to implement a stealth level into their game. The project should give them a working prototype of a stealth level better implemented using some prominent features used successfully in Third-Person Stealth games.

2.4. Work Performed

The project was developed using an Agile Development methodology. The development was split into stages based on the primary objectives, AI, UI, Mechanics, Traversal, Balance and Other Functionality. The stages were then sorted by their importance to the project's success. Therefore, AI and UI were developed first, and Other Functionality was left until the end of development. This was done to ensure that if there were time overruns, the essential features of the prototype had been completed.

The features were designed, implemented and tested for each stage of the project. If the feature did not meet the objective, it was redesigned, implemented and tested again. Use Case Requirements and Use Case Testing were utilised to design and test the features. This worked effectively when paired with the Agile methodology as it allowed for a feature to be easily redesigned and retested if it did not meet the objectives or if the design was not up to scratch and the feature had to be improved.

2.5. Limited Scope

The scope had to be limited in development due to time constraints. It was underestimated how much time it would take to implement parts of the prototype that were not vital to the project. Things such as a moderately comprehensive player controller and a much larger level design to incorporate some ziplines for the Traversal objective took time. Before starting the project, this time was not initially accounted for in the work plan. However, thanks to how the work plan was assembled, with the most vital stages of the prototype being completed at the beginning, the only stages removed from the development were the Balance and Other Functionality.

The scope of Objective 1A regarding the Guard AI`s Behavior Tree was also limited. The objective entailed having a minimum of five key states for the Guard to be in. However, it soon became apparent that having four key states was comprehensive enough and that having another state for the Guard to be in would be unnecessary and likely be of a lesser quality than the previous four.

3. Output Summary

3.1. Guard Behaviour Tree

The first output is the Behaviour Tree which controls the actions and states of each Guard in the prototype. It is software code coded in C# consisting of 15 classes and 904 lines of code(not including comments), of which I wrote 698 and 206 were re-used as detailed in the Results section. The intended recipients of the output are future developers. They can adapt the basic framework of the current Behaviour Tree for use in their games, or they can use this Guard Behaviour Tree as a base and build upon it.

3.2. Group Searching and Attacking Algorithm

Objective 1B entailed developing a system for the AI to use pathfinding to search for the player. For this, a searching algorithm was required to tell the AI guards where to pathfind to, to make it look like the Guards were actively searching for the player as a collective. Later this developed to also house functionality that would allow the Guards to all organise an attack together if one of the Guards spotted the player. This output was a single class I wrote in C#, consisting of 184 lines. The intended recipients of this output include future developers who can take this simple yet effective searching algorithm and make it more complex and more efficient for use in their games.

3.3. Detection System

The AI required functionality to spot the player slowly over time. To do this, a system was created to implement a vision cone for a Guard and then split that vision cone into five different vision zones. These vision zones would detect the player at varying speeds based on how close the player was to the Guard. The output was two classes I wrote in C# consisting of 280 lines of code.

Another aspect of the detection system was implemented in the UI stage of development. There was a UI element that would depict how close the player was to being detected based on the previously mentioned detection speed. The output consisted of two main parts. The first part was C# code consisting of 2 classes and 60 lines of code, of which I wrote 41 and 19 were reused as detailed in the Results section. The second part of the output was the sprites that would be used to display the detection amount. Two sprites were initially created as .png and then converted in Unity to sprites. I made one sprite using Piskel, and another was found online, and both were a combined size of 177KB. The sprites were detailed further in Appendix B.

The intended recipients of this output are the same as the previously mentioned future developers. With this output, they can easily modify the detection area and how quickly the player can be detected to suit their need for their specific games. The sprites can also be treated as placeholders and replaced by a more visually appealing design. The complete output is detailed in Appendix B.

3.4. State Depiction System

To inform the player of what state a Guard is in, a UI element was designed to depict the current state of a Guard. This output was simple and only depicted three of the four AI states. These included a UI element for the Searching state and a UI element for both the Chasing and Attacking states, called the Alerted state. The output was split into two parts. The first was C# code to enable and disable the UI elements. These were divided into two separate classes, along with a third class which handled turning the UI elements into a billboard. This last class was re-used, as mentioned in the Results section. From the three classes, 60 lines of code were written, 41 by me and 19 were re-used.

The second part of this output came in the form of sprites. These were used as the actual visual representation of a Guard's state. As detailed in Appendix B, I designed the Searching and Alerted sprites using Piskel, and both sprites came to a total size of 201KB.

Future developers are the intended recipients of this output, and they can look to build upon this output by implementing more AI states and then complementing those states with more UI elements to depict those states. Future developers may also want to keep the current implementation of the UI but alter the graphic design of the sprites to make them more visually appealing. The complete output is detailed in Results.

3.5. Hiding Mechanics

A simple yet efficient and easy method of implementing hiding mechanics was implemented into the prototype. The output consists of two simple parts. The first is a layer mask called Obstacle. I added this to the layers section in the Unity Inspector Window. This allows future developers to choose an element they want the Guard not to see through, and it will enable the developers to quickly create a vast array of places the player can use to hide. The second component of this output consists of a single line of code embedded into a much larger class, coded in C#. The total amount of code in the class comprises 127 lines, and only one line is required for this output. I wrote the entire class. Future developers can also make changes here, they only have to change one line, and their AI will not be able to see through any other layer mask they choose. The complete output is detailed in the Results chapter.

3.6. Traversal

For objective 4A, a zipline was implemented into the prototype to allow the player to traverse the level more dynamically. This implementation consisted of 2 C# classes, both of which I wrote. The code amounted to a total of 98 lines. The intended recipients of this output are future developers who can build upon this feature and implement more complex zipline physics, animations, and a more polished output than the one in this prototype. The complete output is detailed in the Results chapter.

4. Literature Review

4.1. Introduction

Stealth games have been one of the most popular genres of video games since the turn of the millennium. Games such as Metal Gear Solid(Hideo Kojima, 1998), Tom Clancy`s Splinter Cell: Blacklist(Ubisoft Toronto, 2013) and Hitman(IO Interactive, 2016) defined the genre. An overwhelming majority of these games are in the Third-Person. The only notable recent exception is Arkane Studios` Dishonoured 2(Arcane Studios, 2016). There have been many attempts by FPS studios to include a stealth level within their action-packed games, but these often feel shoehorned in. For my project, I made a prototype First-Person Stealth game that sets out to achieve what most FPS Stealth games/levels are missing. In this literature review, I will compare and contrast five themes: AI, UI, Mechanics, Traversal and Balance in Third-Person Stealth and First-Person Stealth games/levels. This allowed me to find gaps in current FPS stealth levels that my prototype aimed to fill.

4.2. Artificial Intelligence

A stealth-based AI is a critical component of a stealth game. An AI that the user perceives to be unintelligent will make the game less intense and provide less of a challenge for the player. One essential part of a comprehensive AI is how the AI detects the player. In Splinter Cell: Blacklist(Ubisoft Toronto, 2013), the team used vision cones and vision zones(Walsh, M. 2014) to allow the player to be slowly detected over time based on the player`s location within the enemy's vision cone. This is an example of great architecture for stealth AI and provides a realistic interpretation of how someone would spot a foreign entity in real life. In contrast, Metal Gear Solid(Hideo Kojima, 1998) has a binary detection system. This means that whenever the player is within the enemy's vision cone, they are instantly detected. This could be very frustrating to the player as even if the player`s arm was barely visible for a split second, the enemy would go into an attack state. This makes AI seem unrealistic and unfair. However, unlike some games, Metal gear solid(Hideo Kojima, 1998) does allow the player to hide after being spotted and return to the stealth aspect of the game. This is evidence of an in-depth implementation of a behaviour tree or finite-state machine(FSM) for AI(Millington, I. 2019). To sum up, the AI needs a non-binary detection system and a behaviour tree or FSM that allows the player to escape after being spotted.

4.3. User Interface

Stealth games must have an intuitive UI design. A good UI can help the player understand what state the enemies are in, if the player is hidden and how close the enemies are to spot the player. Dishonoured 2(Arcane Studios, 2016) has an excellent UI that uses markers above the enemy's head to depict the enemy`s current state. If we compare this to Call of Duty: Modern Warfare(Infinity Ward, 2019), we find that during the mission 'Going Dark', the UI gives the player no indication of what state the enemy is currently in. This does not give the player confidence and can make the player play safer and not explore the level to its fullest. Overall, it is crucial to have an intuitive UI that helps the player understand what state the enemy is in.

4.4. Mechanics

A game will not be fun if its mechanics are boring and unbalanced(Adams, E. and Joris Dormans, 2012). Games are made by how good their mechanics are, and stealth games are no different. This is most clearly seen during the Battlefield 1 mission ‘Fall from Grace’(DICE, 2016). During this mission, there are only two stealth mechanics; both are overused and unbalanced. The first is throwing an item to take the guard’s attention, and the second is destroying a communication box to stop reinforcements. Not only are they overused, but they are also unbalanced. The player can find things to throw to distract the guards all over the level, and disabling the communications boxes provides no real challenge. Therefore, mechanics in stealth games must be original and balanced to prevent the player from abusing the mechanic.

4.5. Traversal

Traversal is a vital component of any good game. The ability to traverse the level uniquely provides more replayability for a level. Stealth games have been using unique forms of traversal for a long time. Examples include Sam Fisher`s split jump in Splinter Cell: Chaos Theory(Ubisoft, 2005) and Batman`s grapnel in Batman: Arkham Asylum(Rocksteady Studios, 2009), which the player can use to climb above the guard’s line of sight or use ventilation shafts/grates to traverse the level below the guards’ feet. This kind of traversal is largely missing in First-Person Stealth games/levels. An example is Battlefield 1`s level ‘Fog of War’(DICE, 2016). In this level, the player has no unique ways to traverse the map. There is no way to change the verticality or manoeuvre around enemies. This leaves this level feeling a little flat. Ultimately, traversal is necessary for all games, but the lack of unique traversal in stealth games can leave the player with no opportunities to tackle a level in an original way.

4.6. Balance

In stealth games, balance between the player and enemies is crucial. The Last of Us made a single enemy lethal to encourage stealth (Naughty Dog, 2013). However, this alone would not be balanced when the player was fighting a large group of enemies in a non-stealthy environment. When facing a large group in The Last of Us, only one or two enemies shoot at the player simultaneously for better balance (McIntosh, T. 2014). This meant the player was still cautious of the enemy, but at least it gave them a fighting chance. Getting the balance right is tedious but results in a more enjoyable experience.

5. Method

This project was developed using an Agile methodology. Specifically, each primary objective was run in an iterative design, implementation and testing loop. This ensured that each objective was completed to a good quality before moving on.

The methodology chapter is divided into four parts, one for each primary objective. These chapters are then divided into sub-chapters, including the design and implementation required to complete that objective. They may also include the requirements, evaluation and testing of the objective/implementation.

The build plan was initially split into six major sections and was structured so that the most vital parts of the project were completed first. However, the build plan was changed during development, and only four of the six objectives were completed. This meant the build plan was split into four major sections. This will be detailed further in the Results section.

To track the progress of the project, a work plan was designed in the form of a Gantt chart, which was updated regularly to reflect the progress made on the project.

Requirements and testing were done using use cases. Before development, use case requirements for an objective were laid out. The implementation then sought to achieve what the requirements had specified. At the end of the implementation, use case testing was performed to ensure the objective was met.

5.1. Stealth AI

The following three chapters detail the methods to design, implement and test the work for project objective 1, Stealth AI. The three objectives were completed back to front. The reasons for this will be detailed in the Results chapter.

5.1.1. Guard Detection (Objective 1C)

The first objective was to implement the Guard`s ability to slowly detect the player over time. This was done by first implementing some essential detection. This was achieved by performing a raycast from the Guard to the player and checking that the player was within a specific view angle relative to the Guard`s forward vector(ensuring the player was in front of the Guard). The raycast would also check for any obstacles between the player and Guard by utilising a layer mask check. The next step was to check that the player was within a specific range of the Guard and that there were no obstacles in the way. A more in-depth system involved the Guard slowly detecting the player over time. To implement this, the Guard`s view cone was split into different zones. This was done by splitting the view cone into three angles, wide, medium and close. The smaller the angle, the faster the player should be detected. The next step is to add three distinct ranges within the view cone. This allows the player to be detected slower when further out from the Guard. This created 15 distinct zones categorised into 5 zone types within the Guard`s view cone, as shown in Figure 1. The last step is to set a timer for each zone type to detect the player at a different speed. To help with testing at this stage, a gizmo or spotlight to check which zone the player is currently in and if they have been spotted would be beneficial.

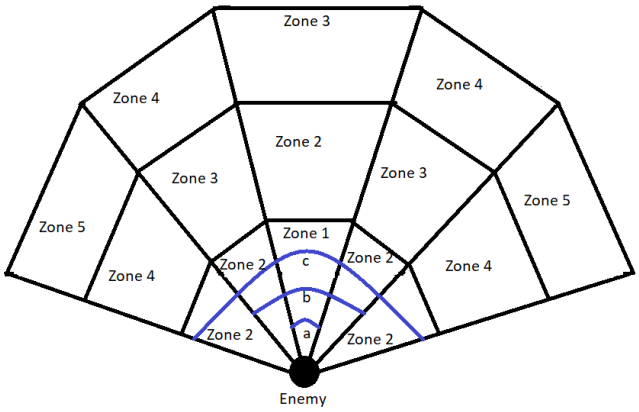


Figure 1

An example of a Guard`s Vision Cone. Fifteen vision zones are split into five zone types. The closer a zone is to zone five, the slower the detection. 'a', 'b' and 'c' are the close, medium and wide angles displayed with the blue lines.

5.1.2. Guard Pathfinding Search (Objective 1B)

Objective 1B entailed a Guard being able to independently pathfind around the map to search for the player. The first component of this was to implement Unity`s NavMesh system. An online tutorial was used to obtain the necessary Unity package for the NavMesh and instructions on how to bake a NavMesh (Brackeys, 2018). Once a Nvamesh was baked onto the level, a Guard could be specified a location to travel to, and the Navmesh would help the Guard pathfind to that location. This Nav Mesh system implements an A* pathfinding algorithm to find the shortest route to a given target. To implement a coherent searching algorithm using the pathfinding algorithm, all the Guards must work together to search the map. A separate search method is used, containing all Guards in a group to coordinate their search. This method is housed within the BTGuardGroup class. If one Guard spots the player, the BTGuardGroup class assigns all the other Guards in the group to convene at the player's last known location. This makes it seem like the AI has communicated the player's whereabouts. The search class randomly assigns each Guard to a location on the map for them to search. A timer controls the search. Each Guard has two search locations. The first search begins 5 seconds after losing the player. This was done to allow time for the Guards to convene on the player's last known location. After searching for 20 seconds, the Guards will be assigned a new location on the map to search, and they pathfind to the new location on their own. After 15 seconds spent on the second search, the search class calls an end to the search. The Guards also use the NavMesh pathfinding system to chase the player if they spot them. The Guards will enter a patrol path if they are not chasing or searching. The patrol path utilises the NavMesh system to have a Guard constantly patrol around a fixed set of locations around the map. Each Guard has their own set of locations/patrol routes.

5.1.3. Guard AI Behaviour Tree (Objective 1A)

Objective 1A required the Guards to use a behaviour tree to dictate their actions. The specific behaviours include Patrol, Search, Chase and Attack. First, a generic behaviour tree architecture was implemented from a tutorial (Mina Pêcheux, 2021). The behaviour tree had four nodes. The first was a root node at the top of a behaviour tree. The second and third were Selector and Sequence nodes, respectively. A selector node works like an OR logic gate (If any child node returns success, then return success). A sequence node works like an AND logic gate (if all child nodes return success, then return success). The final node type is a leaf node where all classes/actions will be housed. A behaviour tree works in order from left to right. This is important as it allowed priority to be given to the left-most nodes since they would be the first to return. The Guard Behaviour Tree used in this project implemented four key states. These included Patrolling, Chasing, Attacking and Searching. The exact formation of this Behaviour Tree can be seen in Figure 2. The entire detailed workings of the Guard Behaviour Tree are in the Appendix.

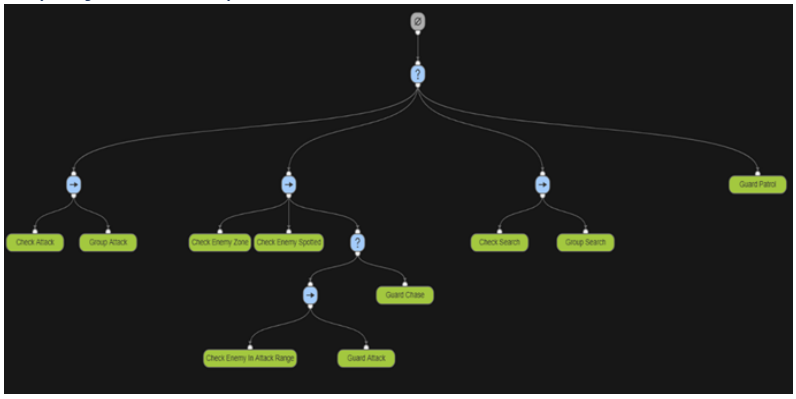


Figure 2

Behaviour Tree Diagram showing the Guard Behaviour Tree layout and classes. (The topmost node is the root node. The '?' nodes are selector nodes. The '->' nodes are sequence nodes. The green nodes are leaf nodes.) This diagram was designed using Adobe Behaviour Tree Visual Editor (opensource.adobe.com, n.d.)

5.2. Stealth UI

The following two chapters cover the methods used to design, implement and test the work for project objective 2, Stealth AI.

5.2.1. UI Depicting Guard State (Objective 2A)

Objective 2A involved implementing UI elements above the Guards` heads to tell the player what state the Guard was in. The game had four main Gurard states: Patrolling, Searching, Chasing and Attacking. The Patrolling state would have no UI element. The Searching state would have a question mark UI element, and the Chasing and Attacking states both had an exclamation mark UI element to depict an 'Alerted' enemy. The implementation involved binding these symbols to the Guards` heads and enabling and disabling them based on that enemy`s particular state. The symbols were simple sprites that could be designed in 2d design software. The specific software chosen for this project was Piskel(Descottes, 2019). The sprites were imported into Unity as assets and rendered as images under a Canvas UI game object. A tutorial was followed to add a Canvas game object and assign sprites to the Canvas (Brackeys, 2020). From there, whenever a Guard was in a state other than Patrolling, the corresponding UI element would be enabled and the others disabled.

5.2.2. UI Depicting Guard Detection (Objective 2B)

Objective 2B required the development of a UI element which would help the player understand how close a Guard was to spot them. The first part of the implementation involved designing and importing a detection bar into Unity and having that be another child of the Canvas UI game object. This was partly done in Piskel(Descottes, 2019), and the other part of the design was found online(creazilla.com, n.d.). Then, the detection UI element was set up to be enabled when the player was in the process of being detected and disabled if the enemies were alerted by the player or had not seen the player. A simple UI Image handled the detection level in Unity; the method to implement this was re-used from an online tutorial(Brackeys, 2020). Its position was set relative to the Canvas object, and its size was dictated by a built-in Unity component called a Slider. This component had built-in functionality, allowing the blank image to be scaled based on a pre-determined amount, in this case, the detection level. The detection level was set in the CheckGuardSpotted class by taking the time the player had been currently visible to the guard and dividing it by the total time it would take to spot the player in that particular zone.

5.3. Stealth Mechanics

The following two chapters cover the methods used to design, implement and test the work for project objective 3, Stealth Mechanics.

5.3.1. Two Stealth Mechanics (Objective 3A)

The first mechanic to implement as part of Objective 3A was a smoke bomb. The mechanic had two components. The first was a smoke bomb canister, which re-used a prefab found online (assetstore.unity.com, n.d.). Upon the player's button press, the canister would be initialised into the level, falling just in front of the player as if they had let it go from their hand. This component was re-used from an online tutorial(Brackeys, 2017). The second component was the smoke, which was an effect found on the Unity Asset Store (assetstore.unity.com, n.d.) and had been manipulated using the Unity Particle System component to make the smoke look more appealing and do a better job of hiding the player. Once the canister had been dropped, it had a delay before the canister was culled from the game, and the smoke was instantiated. The smoke object had a box collider and the same layer mask as any obstacles in the scene. This was done so Guards could not see the player through the smoke. This implementation worked well but did not fully cover what was required in the Use Case Requirements Specification. The Results section will discuss the shortcomings and the reasons for those shortcomings.

The second mechanic to implement as part of Objective 3A was a hiding mechanic. This works predominantly the same way as the smoke bomb mechanic. The mechanic can utilise any mesh that seems realistic for a player to hide in, such as a bush or tall grass. A bush found online was used for the implementation at this stage (assetstore.unity.com, n.d.). The bush was rendered as a mesh and had a box collider component. The box collider component had the 'Is Trigger' boolean value set to true. This meant the physics engine ignored the box collider allowing the player and Guards to walk through the bush. The last thing to add was a layer mask. The layer mask was set to 'Obstacle', the same for the smoke bomb and other buildings around the scene. This ensures the Guards cannot see through the bushes and spot the player.

5.3.2. Limitng the Use of the Stealth Mechanics (Objective 3B)

Objective 3B involved limiting the use of the previously implemented mechanics so that they were not overpowered and the player could not solely rely on them when completing the level.

The first part of this was limiting the use of the smoke bomb mechanic. This was done in 2 aspects. The first was to implement a cooldown after deploying a smoke bomb to stop the player from repeatedly using the smoke bomb in quick succession. This was done by implementing a cooldown timer after a smoke bomb was dropped and not allowing the player to drop another smoke bomb until the timer had run down. The second aspect of limiting the use of the smoke bomb was to prevent the player from having an infinite number of smoke bombs at their disposal. This was done simply by adding a variable that would store the total number of smoke bombs and decrementing it by one every time a smoke bomb was deployed. Once the counter reached 0, the player could not drop any more smoke bombs.

The hiding mechanic cannot be limited to a finite number of uses or implement a cooldown such as the smoke bomb mechanic. In order to limit the effectiveness of this mechanic, the developer must place the hiding locations thoughtfully in the level. For this implementation, this was done by having the hiding areas at the edge of the map be large and plentiful, allowing the player to easily and safely recon the area. However, once the player moved closer to the main part of the level, the hiding areas would be smaller and fewer and far between. Finding a balance between where and how to place hiding locations will differ on a map-to-map/level-to-level basis. Each level must have its own interpretation of this implementation to ensure the player cannot simply hide in the hiding areas for most of the level.

5.4. Traversal (Objective 4A)

Objective 4 entailed the design of a unique and dynamic way for the player to traverse the level. This was chosen to be a zipline. To implement a zipline, first, two locations had to be chosen in the level, one being a starting location for the zipline and one being the landing zone. In this prototype, they were stored as game objects with a rudimentary mesh attached to them designed bespoke in Unity`s ProBuilder. After this, linear interpolation was used to implement a zipline imitation of the player going between the two points. Linear interpolation in Unity can be used as “Vector3.Lerp” to transform the position of an object smoothly. It does this by taking three parameters, the start location, the end location and a number between 0 and 1. The number relates to how far along between the two points an object is. For example, if the number is closer to one, the object will be closer to the end location; if it is closer to 0, it will be closer to the start location. By setting the number of time elapsed while on the zipline over how much time the developer wants that zipline to take, a smooth interpolation can be achieved. Using this interpolation as the player’s position if they press a button can create a nice zipline effect. To ensure the zipline works accordingly, disabling the player’s movement functionality and gravity is advised, as it prevents any weird juddering or glitches. After the linear interpolation has finished, these can be reset when the player exits the zipline.

6. Results

The Results chapter will detail the outputs from each objective completed in the project. The chapter will be split into four main parts related to the four significant objectives completed during development.

The outputs will be categorised into two main groups, code outputs and Unity outputs. Code outputs relate to the specific code and algorithms used to implement a feature. Unity outputs relate to things such as game objects and prefabs contained within the Unity project. Prefabs are templates that a developer can use to create new instances of that prefab quickly.

The Agile methodology adopted for the project worked well. The iterative loop for objectives allowed for features to be continually improved upon until the tests were successful. The requirements and design for each project objective did not need to be changed during development.

Due to the strict timeframe for this project, the last two objectives could not be completed. Initially, the build plan was split into six major sections, but due to the timeframe, the build plan was split into four major sections. Thankfully, this did not significantly impact the project as a whole, thanks to how the build plan was laid out. Since the most vital parts of the prototype were completed first, Objective 5 and Objective 6 played a minor role in how well the project would turn out.

The use of a Gantt chart did help tremendously, and it allowed the project to better adapt to the strict timeframe. A week or so before development would begin on Objective 4, it could be seen on the Gantt chart that plans to complete all six objectives were not entirely possible. With this information, the build plan was re-organised to put less emphasis on trying to complete all six objectives and more effort was put in to ensure the first four objectives were completed to a high degree.

Use case requirements, and testing worked well in conjunction with Agile. If an implementation did not meet the criteria of the use case tests, then the implementation could be quickly reworked. It also allowed for requirements to be redone if the output from the objective was not up to scratch in terms of design, but this was never required in this project.

6.1. Stealth AI

The following three chapters detail the results from objective 1, Stealth AI. The three objectives were completed back to front. This was because the objective was designed with the behaviour tree in mind. Thus during the Project Definition stage, it was assumed that the best course of action would be to develop a behaviour tree first and then add functionality to that with classes from other objectives. However, before development began, this decision was reversed, and instead, it was decided that the classes should be made first and then implemented as part of the behaviour tree. The benefit of doing this was that if the states were correctly implemented beforehand, the behaviour tree could be tested quickly, as the expected results from the classes within the behaviour tree would be known. However, what was wrongly assumed at this stage was that there would be little or no changes to the classes when they would go from being implemented as part of a stand-alone class to one that had to work as part of a behaviour tree and work together with many classes. Refactoring these classes to fit within the behaviour tree architecture took considerable development time.

6.1.1. Guard Detection (Objective 1C)

The Guard Detection objective created outputs used by the AI to spot the player slowly over time.

The requirements of this objective entailed a Guard not having a binary detection of the player but instead slowly detecting the player over a few seconds. The design for the solution to this problem was simple and elegant. The Guard would have a view cone, which was split into different sections, and each of these sections would detect the player slower or faster based on how far the player was from the Guard.

The first code output produced by this objective was the algorithm used to detect the player. The algorithm created vision zones within a Guard`s vision cone, and the player would be spotted at different rates depending on the zone they were occupying.

Part of the algorithm related to checking if the player is within close proximity to the player(nearViewingDist) can be seen below. The algorithm can be modified to include more zones by adding more angles in the Guard`s vision cone and adding more viewing distances for the Guard.

```
// Checks if there is an obstacle between the guard and player
if (!Physics.Linecast(_transform.position, _player.position, _obstacleMask))
{
	// Checks to see the distance between the guard and player
	if (Vector3.Distance(_transform.position, _player.position) < nearViewingDist)
	{
		// Checks to see the angle between the guard and player
		if (playerGuardAngle < angleA / 2f)
		{
			_guard.zone = GuardBehaviourTree.ZoneState.zone1;
		}
		else if (playerGuardAngle < angleC / 2f)
		{
			_guard.zone = GuardBehaviourTree.ZoneState.zone2;
		}
		else
		{
			_guard.zone = GuardBehaviourTree.ZoneState.emptyZone;
		}
	}

}
```

Another part of this algorithm is that a timer is started once a player is found in a zone. Once the timer ends, the player will be spotted by the Guard. The timers can be modified depending on the zone to create a specific detection timer per zone for any future implementation of this algorithm.

The implementation also has a raycast check. This stops the Guard from being able to see the player through specific layers. An output in Unity was a layer mask related to this called Obstacle Mask. This was placed on any objects, such as buildings or trees, to prevent the Gurad from being able to see through the game object. With the Unity IDE, developers can easily set a new object as part of this Obstacle layer, which will easily prevent the Guard from being able to see the player through that object. Within the Guard Behaviour Tree script for the Guard prefab, there is an option to change the Obstacle layer mask to any other layer mask in the game. This is an efficient way to stop the Guard from spotting the player through other layers, not just the Obstacle mask.

The use case tests for this implementation passed. The Guard could successfully detect the player at different rates based on what zone they were in. The use case testing is further detailed in the Appendix.

6.1.2. Guard Pathfinding Search (Objective 1B)

The Guard Pathfinding Search objective created outputs to allow a Guard to navigate the game world independently and search for the player semi-realistically.

The requirements for this objective entailed a Guard pathfinding to the player's last known location, coordinating with their team to pathfind to a location on the map to look for the player and then return to their patrol paths.

The design for this implementation required the Guards to stay patrolling until a Guard spotted the player. This Guard would then communicate to the others about the player's whereabouts, and they would all converge on that location. If they could not find the player after having spotted them, they would pathfind to random locations on the map, and then after their search had ended, they would return to their patrols. The design was changed during development; the locations to search would be selected randomly, but the actual search locations would be pre-determined by the developer. This allows future developers to have some say in where the Guards search for the player. Also, it can make the AI seem more intelligent as they could be made to look in places the player is likely to hide in instead of random locations.

The first output of this objective is the NavMesh system. This project package was imported, and a tutorial was followed to allow a NavMesh Agent to be created on the level(Brackeys, 2018). The output from this was the ability to add a NavMesh script onto the Guard prefab and set the Guard`s pathfinding parameters. These parameters can be modified to support different types of Enemies in the game. For example, a new NavMesh agent type can be quickly created for a larger Enemy. This Enemy could be slower than a regular Guard and not be able to fit through tight spaces. These parameters can be easily set and adjusted with the NavMesh script.

The NavMesh system requires code to tell a NavMesh agent, in this prototype, this was the Guard, where to go. The second output, therefore, naturally followed on from Objective 1C and told the Guard to set its destination to the player`s location upon spotting the player. This modified the previous algorithm from just spotting the player and was now a Chase class. Of course, when the Guard lost sight of the player at this stage, they would not go anywhere. The following output changed this.

The BTGuardGroup script encapsulates all of the Guards in a group. At this stage, it was only responsible for checking if one of the Guards had spotted the player and then telling the other Guard to go to the player's location. This was used as a rudimentary attempt at communication between the Guards, making the AI look more intelligent. However, once all of the Guards had lost sight of the player, the BTGuardGroup script would then organise a search for the player. It did this by randomly assigning the Guard to go to a specific location on the map and stay there for a few seconds before checking another location and returning. This could be manipulated easily to fit any other search pattern. The search locations can be stored as game objects in the scene, and the position of these objects can be used to set the search locations. This allows easy changes to the search locations to adapt to new environments. The search locations for a Guard group in the prototype can be seen in Figure 3.

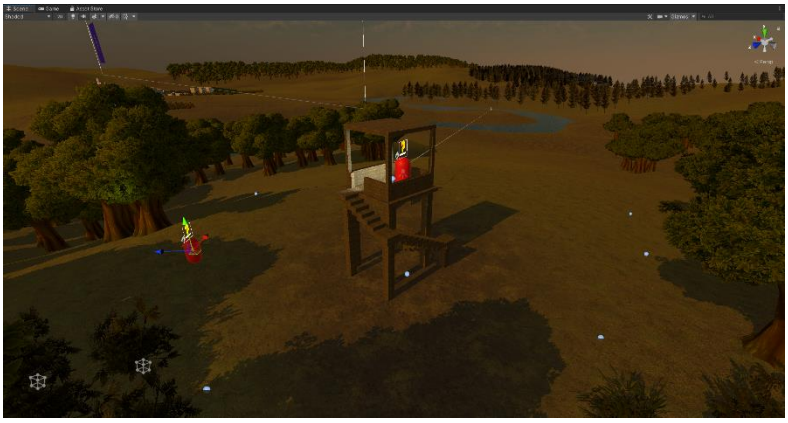


Figure 3

Scene view of prototype. Small blue dots on the ground show the search locations for the two Guards in this area.

The final output of the Guard Pathfinding Search answered what the Guards would do when they are not Chasing or Searching for the player. The answer for this was Patrolling. The same NavMesh system works with game objects in the scene to create a Patrol Path for the Guard. The Guard prefab comes with a child game object called Patrol Paths, which then has three empty child objects that can be used to store the Patrol Path. The Guard Behaviour Tree script under the Guard prefab has built-in components to allow developers to select Patrol Points for a particular Guard. The developer simply has to drag and drop the Parol Path game objects into the Patrol Points component to create a Guard`s Patrol Path.

During testing, the expected results of the test were observed. However, if there had been more time to develop this project, the requirements and design could have been improved to accommodate a better search pattern. Currently, the Guards stand stationary when they are at a search location. A better implementation would have a more realistic interpretation of what the Guards would do when they have reached their search location, such as moving around the search location or maybe turning a few degrees to make it look like they are looking over their shoulder to look for the player. For this project, the current implementation did the job. It created an efficient pathfinding system and a searching pattern with Guards who look like they are communicating. However, there is room for improvement.

6.1.3. Guard AI Behaviour Tree (Objective 1A)

The requirements for the behaviour tree involve having four main states that can control the Guard`s behaviour. These would primarily be based on the previously implemented classes from objectives 1C and 1B.

The design of the behaviour tree proved to be very complicated. This was due to two main reasons. The first was detailed at the start of the Results chapter. Due to the order of operations for this objective, the classes previously developed for the AI had to be heavily adapted to accommodate the Guard Behaviour Tree. The second reason was due to the implementation of the BTGuardGroup class having to have access to the Guard components such as its NavMesh Agent; often, the behaviour tree and the BTGuardGroup class would clash, and both try to do different things at the same time. To solve this, four new classes were added, Check Attack, Group Attack, Check Search and Group Search. These classes were essentially used to stop the behaviour tree from doing anything when the BTGuardGroup class coordinated a search or an attack.

The architecture for the Behaviour Tree was re-used from a tutorial(Mina Pêcheux, 2021). The architecture included four node types, as detailed in the Method chapter. Therefore, the first output for this objective is the ability for any developer to quickly and easily modify and change the structure of the Behaviour Tree to suit their requirements. They can efficiently add or remove states and design a new Behaviour Tree with their classes with the architecture provided.

The Guard AI Behaviour Tree implements an efficient method of selecting the correct state for a Guard to be in based on their current situation in the game. In this implementation, four main states were created. These include Patrolling, Chasing, Searching and Attacking. The first three were implemented as part of the last two objectives and modified to fit within the architecture of the Guard Behaviour Tree. On the other hand, the Attacking state had to be developed by splitting the Chasing state into two parts. This was done by changing the Chasing state to only chase the player if they were in zones 2-5. If the player were in zone 1, the Attacking state would instead be run. The four different states are one of the outputs of this objective. However, the Attacking state is currently just a placeholder. The actual attacking functionality from the Guards was not in this project’s scope as it was deemed that First-Person games already do this exceptionally well. A Guard can correctly execute the Attacking state, but in its current form, the Guard performs no attacks when in the state.

Alongside the placeholder attacking functionality, the BTGuardGroup class was modified. This was done so that if one Guard attacked the player, all other Guards would be sent to the player’s location. This was done to mimic how a team of operatives would communicate in real life and converge on an opponent. The BTGuardGroup class is another output in the prototype. By assigning the script to a game object, a developer can assign Guards to it. This will, in turn, ensure that these Guards work together in the game.

The Guard Behaviour Tree script not only housed functionality for when a Guard should be in a specific state, but it also had additional parameters that could be used to modify the Guard`s behaviour slightly or to debug the Guard`s current state. For example, a developer could change the Obstacle mask the Guard has to a different layer in the scene. The script has many boolean values that the developer can use to check specific parameters and understand what the guard is currently doing. The specific script for the Behaviour tree is the final output for this objective.

Despite the long-winded development of the behaviour tree, the use case testing passed without a hitch.

6.2. Stealth UI

The first output from this objective was a billboard script. This was re-used from an online tutorial (Brackeys, 2020). The billboard script would orient the UI elements above the head of the Guard to always face the player. The billboard script could be placed onto any Canvas (an object which holds UI elements).

6.2.1. UI Depicting Guard State (Objective 2A)

The requirements for this objective involved multiple UI elements above the Guard`s heads depicting what state they were currently in. Three of the four states within the Guard Behaviour Tree had UI elements attached to them. The search state had one UI element attached to it, and the Chasing and Attacking states were combined to have an Alerted UI element attached to them.

The design for the UI elements depicting the Guard`s state was simple. It required each UI element to be displayed or hidden based on the current state of the Guard. Since this is a prototype, the visual for the UI elements did not have to be magnificent, they just had to be clear in what they represented, which was achieved.

The first output for this objective was the scripts to load and unload the sprites. The Guard prefab has a child object called a Canvas. The Canvas object is used to implement 2d sprites into a scene. By binding it to the Guard and utilising the afore-mentioned billboard script, the Canvas object would float above the Guard`s head and always face the player. The sprites were stored as Image components under this Canvas game object. To load and unload them, a script was implemented for each sprite. The script had the functionality to enable and disable the Image component. The Guard prefab already had the Searching and Alerted sprite stored in the Guard Behaviour Tree script and, therefore, could enable and disable them by simply calling one of two functions, “display” or “disable”. This is a straightforward output, but it is easy to reproduce for any new sprites added to the game.

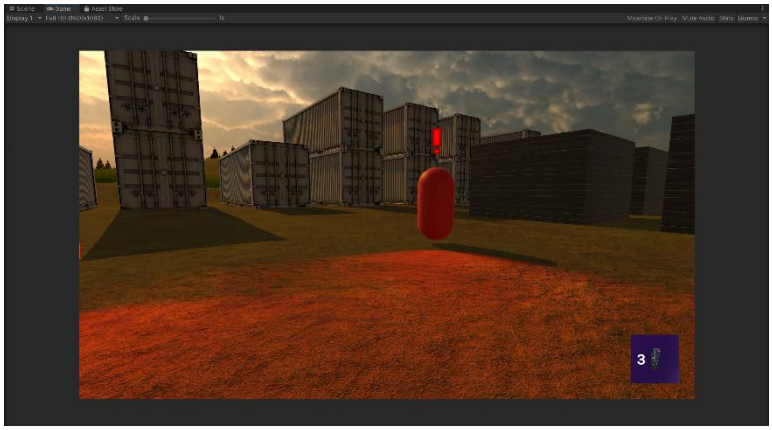


Figure 4

Screenshot of game view showing a Guard`s Alerted UI element.

The second output for this objective was the actual sprites. Two sprites were designed in the free online sprite editor, Piskel (Descottes, 2019). There was one for the Searching sprite, resembling a question mark symbol and one for the Alerted sprite resembling an exclamation mark. These can be seen in Figure 4 and Figure 5, respectively.

Figure 5

Screenshot of game view showing a Guard`s Searching UI element.

6.2.2. UI Depicting Guard Detection (Objective 2B)

This objective required implementing a detection bar UI element that would tell the player how close a Guard was to spotting them.

The design for this objective was split into two parts. The first part entailed displaying the detection bar at the desired time. This was done by enabling the bar when the player was in the process of getting spotted and when the player was actually spotted. The second part of the design involved how to depict the detection level. How this could be done in Unity during the design stage was unclear. However, the general principle for how to work out the detection level was easy to design. It would be based on the time the player had currently been visible for over the time it would take to fully spot the player.

Similarly to objective 2A, this objective required the design of a class that would load and unload the Detection Bar UI element. In this case, it was done when the Guard could see the player, but the player had not been visible long enough for the Guard to have spotted them. The Detection Bar script re-used functionality from an online tutorial to smoothly increase or decrease the bar(Brackeys, 2020). This was done by having a blank image scaled using a Unity Slider component. The Detection Bar also had a Gradient component. This allowed the blank image to be coloured differently based on its size (a larger image would mean the player was closer to being detected). The Slider and Gradient components and the associated code were also re-used from the online tutorial(Brackeys, 2020).

The Guard Behaviour Tree script sets the Detection Bar level for the Guard. It sets the level of detection based on the timer used to spot the player. For example, if the player had been seen for one second and the Guard would fully spot the player after five seconds, then the Detection Bar level would be at 1/5th capacity. This was done by passing a value to the Detection Bar script from the Guard Behaviour Tree script based on the detection level.

The final output of this objective was the sprites associated with the Detection Bar. There were three sprites used. The first was the blank Image component, scaled up and down to show the detection level. The second was a border for this Image component, designed in Piskel(Descottes, 2019). The last sprite was a re-used asset found online(creazilla.com, n.d.). It was used to give the player some indication of what the bar was showing. The visual for the Detection Bar can be seen in Figure 6.

Figure 6

Screenshot showing the Guard`s Detection Bar UI element.

During the use case testing of this feature, two problems occurred. The first was fixed relatively quickly. It was to do with when the detection bar would be enabled and disabled. The test found that the detection bar would disappear when the player was out of the Guard`s line of sight. This was because the design of the detection bar stated that the UI element must only be displayed when the player is in the process of being spotted. Since the Guard is not currently spotting the player, the detection bar will disappear when the player moves out of the Guard`s line of sight. A quick fix was implemented, ensuring that the detection bar would only be disabled if the detection level was at 0. This fixed this issue.

The second issue that arose after this one had been fixed was that the detection bar would jump down when the player left the Guard`s vision zone. This occurred because of the different zone timers in the Guard's vision cone and how the detection bar lowered after the player left the Guard`s vision cone. The detection level is set relative to every zones timer. For example, it was set to five seconds for zone five and one second for zone one. If the player jumped between zones or jumped out of a zone entirely, then the timer needed to fully spot the player would change. This led to the detection level jumping up and down based on where the player moved to in the detection zone. To explain this clearly, imagine if the player was in zone 1 and had 1 second before being spotted. If the player was halfway to being spotted, their visible timer would be 0.5 seconds, and the Guards bar would be half full. If the player instantly teleported to zone 5, where the timer is 5 seconds, then the time the player has been spotted for has not changed, it is still 0.5 seconds, but now crucially, the time for the player to be spotted has gone up. So now the bar would not be half full, but a tenth full. These kinds of situations were causing the jumping in the detection bar.

An elegant redesign for this solution may involve storing the current detection level based on the percentage of the bar that has been filled and not solely on the timer. That way, if the player changes zones, the percentage would remain the same, and the timer can decrease steadily.

6.3. Stealth Mechanics

6.3.1 Two Stealth Mechanics (Objective 3A)

Objective 3A required the development of two mechanics that would aid the player in playing stealthily.

The smoke bomb mechanic specifically required the smoke bomb effect not to be seen through by Guards and that if the Guards were caught inside the smoke bomb radius, they would lose sight of the player. Lastly, if a Guard spots the smoke but does not spot the player, they will communicate with the other Guards in their group and convene at the location of the smoke and then enter a search pattern to look for the player.

The design for this implementation seemed simple at first. The design involved having a smoke bomb canister drop and then be culled from the game world while simultaneously the smoke effect would be instantiated. The smoke effect would have the obstacle layer mask to prevent Guards from being able to see through it. Then a simple collision check will be used whenever a smoke effect has been instantiated to check whether a Gaurd was within the smoke effect. If they were, their movement would be disabled temporarily. Lastly, to get the Guard to spot the smoke, the Guard Behaviour Tree would have another class implemented into it, returning success if the smoke was within the Guard`s vision cone. This would, in turn, set a boolean variable for the Guard to true, and the BTGuardGroup script would read this variable to be true and then set all of the other Guards in the group to converge to this location.

The smoke bomb mechanic was implemented relatively unsuccessfully. The drop smoke script comes as a component on the Player prefab. This script handles the deployment of a smoke bomb canister. The mesh of the canister was found online (assetstore.unity.com, n.d.). The script also has built-in parameters that allow a developer to alter things such as the throwing force of the canister, the exact mesh thrown or the angle it is thrown. This script was re-used from an online tutorial(Brackeys, 2017). After a canister was instantiated, another script would begin running. The smoke bomb script was attached to the smoke bomb prefab. Upon deployment of the canister and the running of the smoke bomb script, there would be a short delay before the canister was culled. For example, this delay is adjustable by future developers who may want a longer fuse time. After the canister was culled, the smoke particle effect would be instantiated. Again, this effect can be altered in whatever way a developer sees fit due to its accessibility in the inspector window under the smoke bomb prefab. The smoke bomb script was also re-used from an online tutorial (Brackeys, 2017). The smoke bomb particle effect originated from a Unity Store Asset (assetstore.unity.com, n.d.). However, this effect did not suit the requirements in its original form. The effect had to be modified using the built-in particle system component to make the smoke look more realistic. The change from the original

Figure 7

The screenshot shows the original smoke effect on the left and the modified smoke effect on the right.

smoke effect to the modified one can be seen in Figure 7. The smoke effect particle prefab had a box collider and a layer mask to make the Guards unable to see through the smoke. The Guards could not see through the smoke by applying the Obstacle layer mask, providing a realistic interpretation of a smoke bomb. A final script was also created. This one was placed directly on the modified smoke particle effect prefab. It was called destroy smoke and culled the game object from the world after the smoke effect had run out. These can be categorised into five main outputs at this stage. The first three are the scripts, including the smoke bomb script, the drop smoke script and the destroy smoke script. The other two outputs are the smoke bomb canister prefab and the smoke particle effect prefab.

Use case testing for this mechanic failed, and there should have been more outputs from this mechanic. The smoke bomb requirements detailed that the smoke bomb should be able to be spotted by a Guard who would then go over to investigate the smoke. They should have also communicated with the other Guards in the group to tell them to go over to where the smoke was spotted. Lastly, if a Guard were caught inside the smoke bomb, they would stop moving until the smoke had cleared, which was meant to represent the Guard's confusion and increase the mechanic's power. These features could not be implemented. Due to a lack of understanding of Unity, it was not possible within the time frame to make the Guards able to spot the player and the smoke. The attempted solution involved modifying how the Guard Detection algorithm worked to use it to allow a Guard to spot the smoke, but this was not possible.

The second mechanic to implement was a hiding mechanic. The requirements for this objective entailed allowing the player to take cover within a game object, such as a bush or tall grass. In this prototype, this was in the form of some bushes which were found online (assetstore.unity.com, n.d.). Due to the implementation in Objective 1C relating to creating a layer mask called an Obstacle mask, the implementation for this mechanic was simple. By placing a box collider around a mesh with the IsTrigger component selected and applying the obstacle mask to the object, any mesh could be used as a hiding place for the player. This mechanic in its design is only limited by the developer’s creativity. The obstacle mask has been previously discussed as an output. The other output specific to this objective is the creation of three bush prefabs which can be dragged and dropped into the map to provide an easy hiding spot. They come with the box collider and Obstacle mask, as detailed. The use case testing for this objective passed.

6.3.2. Limiting the use of Stealth Mechanics (Objective 3B)

Objective 3b required that the player could not overly rely on the game's mechanics.

The design for limiting the effectiveness of the smoke bomb mechanic was simple. The smoke bomb would have to be limited in its number of uses and how much time the player would have to wait between deploying smoke bombs so that they could not drop multiple smoke bombs repeatedly in quick succession.

Figure 8

The screenshot shows the HUD graphic for the smoke bomb counter.

The drop smoke script was modified to include a parameter for the developer to define how many smoke bombs the player should start with. This would then be decremented as the player used them, and once the player had run out, they could not drop any more smoke bombs. Another output, a simple HUD, accompanied this. The HUD was only in place to inform the player how many smoke bombs they had left. Again the drop smoke script was modified to access the number of remaining smoke bombs and print that number onto the screen. An online tutorial was used to implement this (GameDevTraum in English, 2022). Alongside this counter, a small graphic was designed to show the player exactly what this number meant at the corner of the screen. The graphic also re-uses the smoke bomb canister asset (assetstore.unity.com, n.d.). The full graphic can be seen in Figure 8. The HUD graphic is contained within the Player prefab.

The smoke bomb mechanic was also limited in another way. The drop smoke script was modified again to stop the player from dropping the smoke bombs in quick succession. This was done by a timer which would set the cooldown time between each deployment of a smoke bomb.

The limitation of the hiding mechanic was not easy to design. The outputs from the hiding mechanic offer nothing to limit the uses or prevent the mechanic from being used in quick succession. However, due to the nature of the outputs from the hiding mechanic and how easy they are to drag and drop into the level, the developer can easily manipulate where the player can hide. Thus, they can regulate the effectiveness of the mechanic by how frequently the hiding objects are placed in a level.

Both mechanics passed their use case tests.

6.4. Traversal (Objective 4A)

The project's Traversal component requires the player to have a unique and dynamic method of moving through the level. In the prototype, a zipline was chosen as the solution.

The zipline design entailed having two points between which the player would travel. At this stage, it was assumed that a simple change to the player's transform component over time would be the simplest solution. However, upon further research in the Unity and C# documentation, a more straightforward solution using linear interpolation was found.

The first output from this objective was the zipline script. This was stored under the player prefab. The script implemented linear interpolation to move the player down the zipline. It also disabled the player's move script and their gravity to ensure a smooth ride down before re-enabling these when the player dismounted at the bottom of the zipline. The script could be easily modified with the parameters in the inspector window to increase or decrease the time the player spends on the zipline.

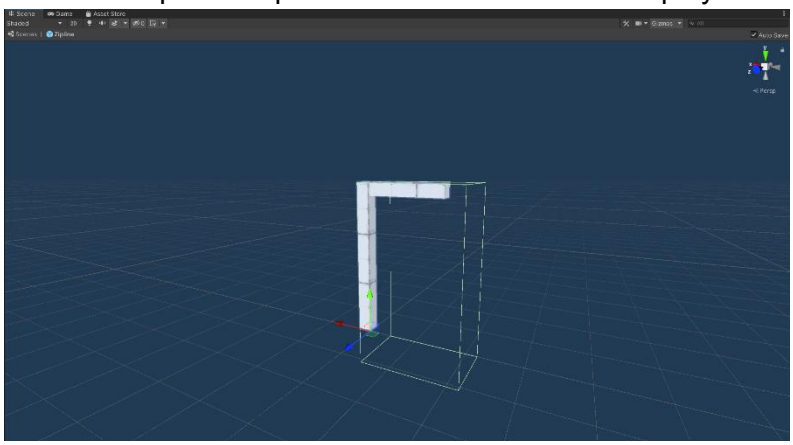


Figure 9

A screenshot showing the zipline hook point prefab, built using Unity ProBuilder.

A zipline prefab was also created as an output for this objective. The zipline prefab had the zipline script built in. Developers could use this to set the zipline's starting location dn ending location. This was vital as it prevented the zipline from being used backwards. Of course, the option is there for the developers to change this and have the zipline be usable in both directions if their implementation requires it.

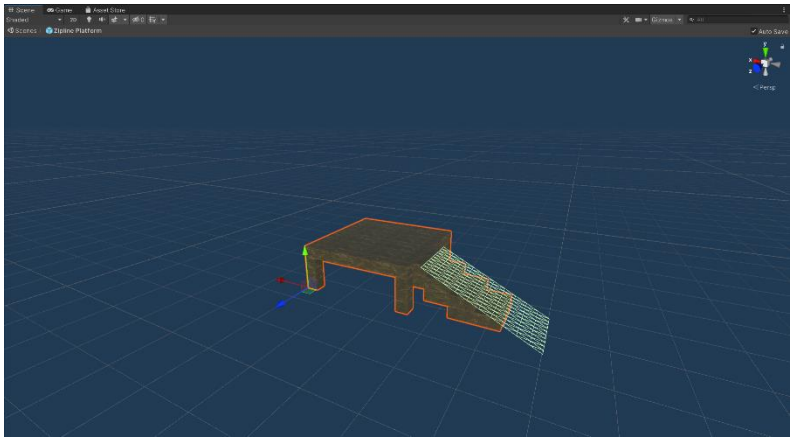


Figure 10

A screenshot showing the zipline platform prefab, built using Unity ProBuilder.

The zipline prefab also has a simple zipline hook point object attached to it. This was designed from scratch in Unity using the built-in ProBuilder tool, as shown in Figure 9. Also, a simple zipline platform was designed from scratch again in ProBuilder, which can be seen in Figure 10. The two components combined could be used to have a comprehensive zipline area in-game, as seen in Figure 11.

Figure 11

A screenshot showing the zipline hook point prefab and the zipline platform prefab together in the scene.

Use case did pass for this feature. However, it was determined that the feature, in general, needed to be more polished. Further details can be found in the use case testing in the Appendix.

7. Conclusions and Discussion

Of the six proposed objectives, only four of them were completed. Objective six was not vital to the project. It was not a part of the main project objectives, and it only sought to add a finishing touch to the prototype at the end of development. On the other hand, objective five was a part of the main project objectives. It required implementing a good game balance between the player and the Guards. A failure to meet this objective due to the lack of time did hinder the project slightly, as it could not meet one of the five criteria. That being said, objective five was the last in order of development due to its relative unimportance compared to the other objectives. The build plan accounted for this, allowing for the most important objectives to be answered first. As a whole, if any of the major objectives had to be left out, objective five would have had the most negligible impact on the project.

The other four objectives were largely a success. Objective 1 was met largely comprehensively. The Stealth AI was a neat implementation and passed the use case testing with flying colours. The only slight part of the Stealth AI objectives that were not met was the requirement for the behaviour tree to have at least five main states. The prototype only had four. This was because, during the design stage, it was believed that having four was more than enough for this implementation and that a fifth would either be of lesser quality or not fit within the game's theme. Overall, it did not make a huge difference, especially as the behaviour tree architecture allows for quick additions to the number of states in the behaviour tree. The Stealth AI and the Guard Behaviour Tree, in particular, have been designed to allow future developers to build upon the implementations easily, and these implementations could also be dragged and dropped into future games. The AIs discussed in the literature review had two main drawbacks. The first was a binary detection system that would spot the player instantly. The second was a lack of depth in their behaviour trees that would not allow the player to return to a stealthy playstyle after being spotted once. These concerns were addressed, and this project`s Stealth AI does both well and implements a pathfinding element and Guard communication and patrolling.

Objective 2 was also met successfully. The UI elements communicated well to the player what state the Guard was currently in and how close a Guard was to spotting the player. The use case testing reflected this. There was a minor glitch involving the detection bar jumping up and down when the player would change zones. However, this was not deemed a significant concern at this stage. The project only created a prototype, and the feature still works as intended but requires a slightly more complex implementation. Future developers can leverage the work done on the Staelth UI and develop the feature by implementing better graphics for the UI elements. They can also quickly add more UI elements whenever they add a new state. When looking back at the Literature Review for the UI, the only required part was the UI element that would help the player understand what state the enemy was in. This was met comprehensively. Alongside this, however, the detection bar element provided more depth to the UI and gave the player even more information about the Guards. This helped solve the main problem with UI in other First-Person Stealth games/levels: they do not give the player enough information.

The implementation of objective 3 could definitely be improved upon in the future. The main objectives were met, there were two mechanics to aid the player in being stealthy, and they were limited in their use. However, more specifically, the smoke bomb mechanic could have been better implemented. The implementation could not be seen by Guards, and Guards would also not be distracted by it, which was a vital part of the requirements for this mechanic. This caused the smoke bomb to be slightly unrealistic in the game. It also took away from the perceived intelligence of the Guard AI as they could not react to it in the desired manner. The hiding mechanic, however, worked well and would be extremely easy to

implement for future developers. It allows future developers to easily apply the mechanic to any mesh they have in the game, and the mechanic is only limited by the developer's creativity. Overall, despite the smoke bomb mechanic not working as intended, the objective was met. The Literature Review detailed that the mechanics must be original and balanced. The mechanics were undoubtedly balanced, and at the very least, the design of the mechanics was somewhat original.

Objective 4 required the player to have a unique and original method of traversing through the level. In this prototype, this was achieved with the implementation of a zipline. The zipline was relatively simple to code and easily adaptable by future developers. However, this feature does suffer from a lack of polish. Since it was the last objective in this project to be implemented, it was slightly rushed. This led to the zipline rope not being appropriately aligned and the zipline hook point not having a suitable texture applied to it. However, it is a prototype. The proof of concept is definitely there to see for future developers, and they can also apply some minor finishing touches themselves should they wish. For the purpose of the prototype and meeting the objective, the implementation does the job. However, when looking at the Literature Review, there is a minor inconsistency with the project objectives. The Literature Review detailed the need for the player to be able to tackle the level uniquely. However, the objective required the player to be able to traverse the level uniquely. The Literature Review requires the player using the traversal dynamically in order to play the game in a unique manner. This was not possible with the current implementation because the two ziplines in the prototype are both primarily just for moving around the map. Neither zipline allows the player to play the level differently than any other player since they are both just used as a means of getting to different parts of the level. It is a slight discrepancy from the objective, which only requires a unique traversal method, not for that traversal to open up gameplay opportunities. To conclude, the project objective for this feature was met, but the Literature Review did not analyse the same kind of traversal. Therefore it is challenging to make a solid conclusion about the feature as a whole.

The project as a whole was a success. Although it was not entirely finished in the time frame, the four main completed features satisfied the relevant project objectives. Overall, it can be concluded that there are lots of things that First-Person Stealth games/levels can learn from their Third-Person counterparts, and this project successfully implemented some areas in which First-Person Stealth games could do that.

My progress through the project was steady. The actual work for the project was challenging, but I managed to problem-solve my way through that and end up with a prototype I am content with. The biggest thing I struggled with was time management and estimating how long each stage of development would take. I spent too much time implementing the Guard Behaviour Tree and trying to implement the Smoke Bomb mechanic. Although the Guard Behaviour Tree was a staple of the project, the smoke bomb mechanic could not be implemented. If it had been implemented successfully, it would not have drastically improved the prototype. I should have cut my losses on the smoke bomb mechanic earlier for these reasons. I also struggled with estimating how long each objective would take. There were severe delays in the project when I underestimated how long it would take to implement a simple character controller and design the terrain for the level so that it could realistically house some ziplines. If I were to do the project again, I would implement more reuse to solve this issue. Re-use of code already helped a lot during this project, so leveraging it more for future projects should help minimise time loss on non-essential parts of the project. Next time, I would also be slightly more strict with deadlines for the objectives so that if an implementation took longer than expected, I would halt work on it sooner and focus my

efforts on the other parts of the project. If this had been done for this project, I might have finished the last two objectives.

8. Reference List

Hideo Kojima (1998). *Metal Gear Solid* [Computer Game]. Available at https://store.steampowered.com/sale/metal-gear/ (Downloaded: 10/03/2023)

Ubisoft Toronto (2013). *Tom Clancy's Splinter Cell: Blacklist* [Computer Game]. Available at https://store.steampowered.com/app/235600/Tom_Clancys_Splinter_Cell_Blacklist/ (Downloaded: 10/03/2023)

IO Interactive (2016). *Hitman* [Computer Game]. Available at https://store.steampowered.com/app/236870/HITMAN/ (Downloaded: 10/03/2023)

Arcane Studios (2016). *Dishonoured 2* [Computer Game]. Available at https://store.steampowered.com/app/403640/Dishonored_2/ (Downloaded: 10/03/2023)

Walsh, M. (2014) 'Modelling AI Perception and Awareness in Splinter Cell: Blacklist', *2014 Game Developers Conference*, San Francisco, USA, 17-21 March. Available at: https://gdcvault.com/play/1020436/Modeling-AI-Perception-and-Awareness (Accessed: 10/03/2023)

Millington, I. (2019). *AI for games*. Boca Raton, Fl: CRC Press, Taylor & Francis Group.

Infinity Ward (2019). *Call of Duty: Modern Warfare* [Computer Game]. Available at https://us.shop.battle.net/en-us/product/call-of-duty-modern-warfare (Downloaded: 10/03/2023)

Adams, E. & Joris Dormans (2012). *Game mechanics : advanced game design*. Berkeley, Calif.: New Riders.

DICE (2016). *Battlefield 1* [Computer Game]. Available at https://store.steampowered.com/app/1238840/Battlefield_1/ (Downloaded: 10/03/2023)

Ubisoft (2005). *Tom Clancy`s Splinter Cell: Chaos Theory* [Computer Game]. Available at https://store.steampowered.com/app/13570/Tom_Clancys_Splinter_Cell_Chaos_Theory/ (Downloaded: 10/03/2023)

Rocksteady Studios (2009). *Batman: Arkham Asylum* [Computer Game]. Available at https://store.steampowered.com/app/35140/Batman_Arkham_Asylum_Game_of_the_Year_Edition/ (Downloaded: 10/03/2023)

Naughty Dog (2013). *The Last of Us* [Computer Game]. Available at https://www.playstation.com/en-gb/games/the-last-of-us-part-i/?emcid=pa-co-449058&gclid=Cj0KCQjwn9CgBhDjARIsAD15h0DNj95ThyV3s08zZpITGPKn2hYHl2xZ7g2lTVpcvc6wZnaFaqJEzugaArYbEALw_wcB&gclsrc=aw.ds (Downloaded: 10/03/2023)

McIntosh, T. (2014) 'The Last of Us: Human Enemy AI, *2014 Game Developers Conference*, San Francisco, USA, 17-21 March. Available at: https://gdcvault.com/play/1020338/The-Last-of-Us-Human (Accessed: 10/03/2023)

Brackeys (2018). *Unity NavMesh Tutorial - Basics*. Available at: https://www.youtube.com/watch?v=CHV1ymlw-P8&t=302s&ab_channel=Brackeys [Accessed 4 Apr. 2023].

Mina Pêcheux (2021). *Create an AI with behaviour trees [Unity/C# tutorial]*. Available at: https://www.youtube.com/watch?v=aR6wt5BlE-E&t=1083s&ab_channel=MinaP%C3%AAcheux [Accessed 22 Apr. 2023].

opensource.adobe.com. (n.d.). *Behavior Tree Visual Editor*. [online] Available at: https://opensource.adobe.com/behavior_tree_editor/#/dash/home [Accessed 06 May 2023].

Descottes, J. (2019). *Piskel - Free online sprite editor*. [online] Piskelapp.com. Available at: https://www.piskelapp.com/.

Brackeys (2020). How to make a HEALTH BAR in Unity! Available at: https://www.youtube.com/watch?v=BLfNP4Sc_iA&t=882s&ab_channel=Brackeys [Accessed 7 May 2023].

creazilla.com. (n.d.). *Binoculars clipart. Free download transparent .PNG | Creazilla.* [online] Available at: https://creazilla.com/nodes/25373-binoculars-clipart [Accessed 09 May 2023].

assetstore.unity.com. (n.d.). *Grenade M18 Smoke | 3D Weapons | Unity Asset Store.* [online] Available at: https://assetstore.unity.com/packages/3d/props/weapons/grenade-m18-smoke-66223 [Accessed 13 May 2023].

assetstore.unity.com. (n.d.). *Particle Pack | VFX Particles | Unity Asset Store.* [online] Available at: https://assetstore.unity.com/packages/vfx/particles/particle-pack-127325.

assetstore.unity.com. (n.d.). Yughues Free Bushes | 3D Plants | Unity Asset Store. [online] Available at: https://assetstore.unity.com/packages/3d/vegetation/plants/yughues-free-bushes-13168.

Brackeys (2017). *GRENADE / BOMB in Unity (Tutorial)*. Available at: https://www.youtube.com/watch?v=BYL6JtUdEY0&ab_channel=Brackeys [Accessed 13 May 2023].

assetstore.unity.com. (n.d.). *Particle Pack | VFX Particles | Unity Asset Store.* [online] Available at: https://assetstore.unity.com/packages/vfx/particles/particle-pack-127325.

GameDevTraum in English (2022). *How to CHANGE TEXT FROM A SCRIPT in Unity using TextMesh Pro*. Available at: https://www.youtube.com/watch?v=Xw506Rfd9Q4&ab_channel=GameDevTrauminEnglish [Accessed 29 May 2023].

assetstore.unity.com. (n.d.). *Grenade M18 Smoke | 3D Weapons | Unity Asset Store*. [online] Available at: https://assetstore.unity.com/packages/3d/props/weapons/grenade-m18-smoke-66223.

9. Appendix

9.1. Appendix A: PDD

Problem to be solved:

Stealth games have been very popular in the last decade. Games like Hitman 3(IO Interactive, 2021), Batman Arkham Knight(Rocksteady Studios, 2015), Metal Gear Solid V: The Phantom Pain(Kojima Productions, 2015) and Splinter Cell: Blacklist(Ubisoft Toronto, 2013) are all exceptional examples of good stealth games. However, these are all third-person games. There is a severe lack of first-person stealth games. Dishonoured 2(Arkane Studios, 2016) is the only relatively recent game that meets the first-person stealth criteria.

There have been many attempts by games such as Call of Duty Modern Warfare(Infinity Ward, 2019) and Battlefield 1(DICE, 2016) to include stealth levels within their FPS games, however, these often feel shoehorned in. A few areas in which these games lack compared to their third-person counterpart include a poor stealth AI, an unintuitive UI design for stealth, a lack of stealth-specific mechanics, limited movement/traversal options and a lack of balance between the player and the enemies.

I am going to create a prototype First-Person Stealth game that will incorporate a few of the features that make third-person stealth games great. The specific areas I want to look at are the stealth AI, stealth UI, stealth mechanics, player movement/traversal and balancing the player and enemies.

The prototype will be one level of the game created using Unity and the scripts will be written in C#. To assist me with the technical side of development I will utilise the Unity Learn website(Unity Learn, 2020), Game Coding Complete(Mcshaffry, 2013) and Artificial Intelligence for Games(Millington, 2019).

Project Objectives:

The project's main objective is to create a First-Person Stealth level using Unity. This will have 5 main aspects, AI, UI, Mechanics, Movement and Balance.

1) Stealth AI
    1. The stealth AI will be designed with a behaviour tree that will have at least 5 different states for the enemy. E.g., Cautious, Search, Attack, etc
    2. The AI will use a pathfinding algorithm to find the player when in the searching state.
    3. The AI will not have a binary detection of the player. The AI should slowly detect the player over time and not instantly go into a state of alert when they spot the player.
2) Stealth UI
    1. There should be a small icon to allow the player to determine the current state of an enemy. E.g., Red for an attack state and amber for a search state.

    2. There should be a UI element that allows the player to determine whether they are about to be spotted or not
3) Stealth Mechanics
    1. There should be 2 different stealth mechanics/gadgets that the player can use in the level. E.g., Binoculars to mark targets, Agent 47`s piano wire(IO Interactive, 2021), and Sam Fisher`s fibre optic cable(Ubisoft Toronto, 2013).
    2. The player should only be able to use the mechanic a finite number of times within the level.
4) Movement/Traversal
    1. The player should have a unique and original way to traverse the level that differs from walking, running, crouch walking and crawling prone. E.g., Batman`s Grapnel (Rocksteady, 2015) and Dishonoured`s Blink ability (Arkane Studios, 2016)
5) Balance
    1. Enemies should be much stronger than the player in terms of health and damage they can do.
    2. The weapons the player can use should be very weak when taking on multiple enemies
    3. The player should not regen health or have any way to replenish health in the level
6) Other Functionality
    1. Main menu and pause menu
    2. Audio and Visual fx
    3. Saving and loading

Project Beneficiaries:

The main beneficiary of this project is other developers specifically working on FPS games who want to implement a stealth level into their game. My project should give them a working prototype of how to better implement a stealth level using some of the prominent features used successfully in third-person stealth games.

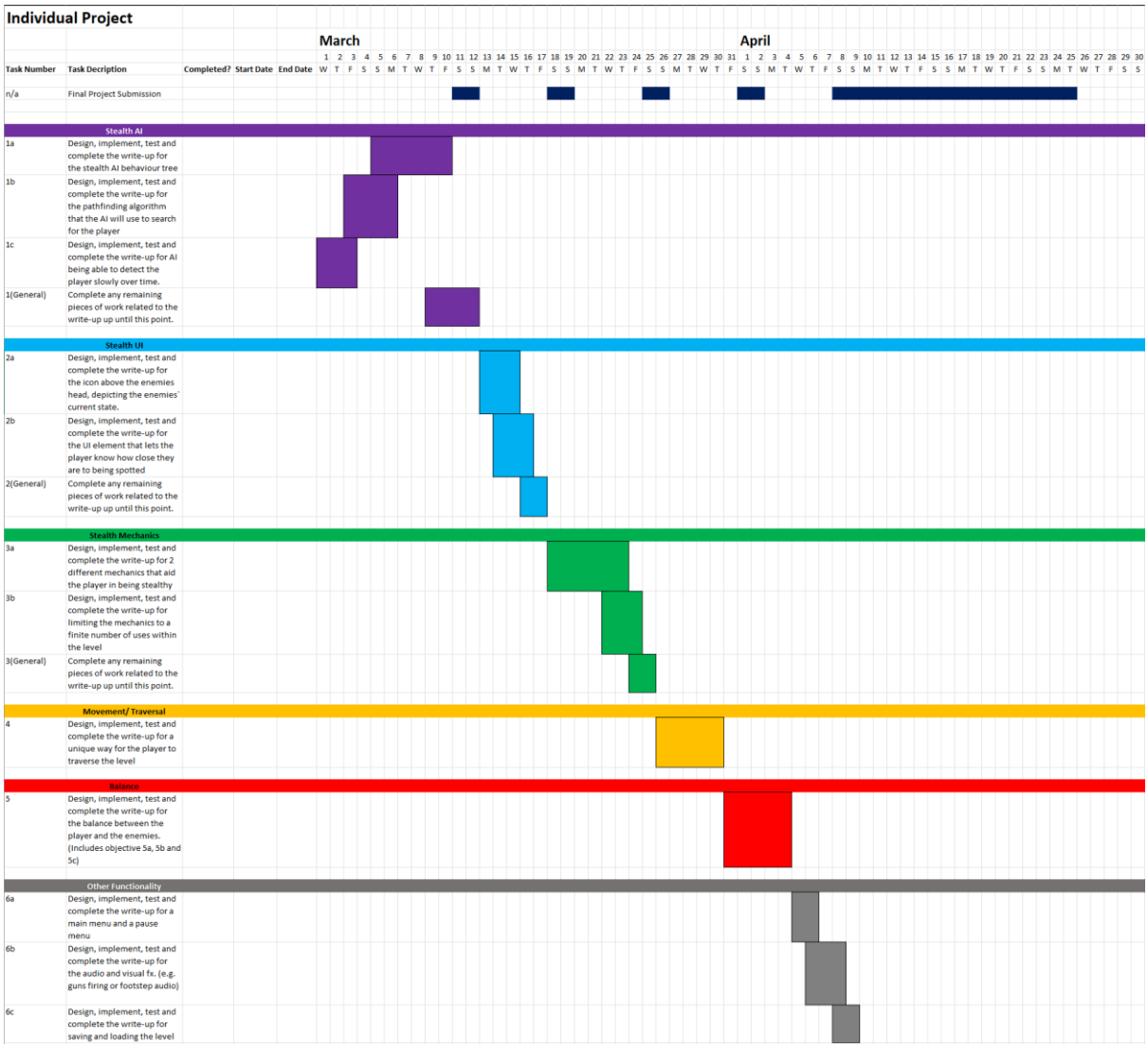
Another beneficiary of my project is developers who may want to build on top of my prototype to produce a full first-person stealth game.

Work Plan:

I will be using an Agile development methodology to develop my project. For each stage of my project, I will design, implement and test the feature. If the feature does not meet the objective, I will redesign, implement and test it again. For each stage, I will also include some time towards the end of the total time dedicated to that feature to complete the write-up of any methods, tests or anything reacted to that feature that needs to go into my Final Project Submission.

I have included a Gantt chart below. It should be noted that where I have put ‘complete the write-up’ below a specific objective involves things such as the methods, code and testing related specifically to that objective. Whereas at the top with the Final Project Submission, the work here is more related to the project as a

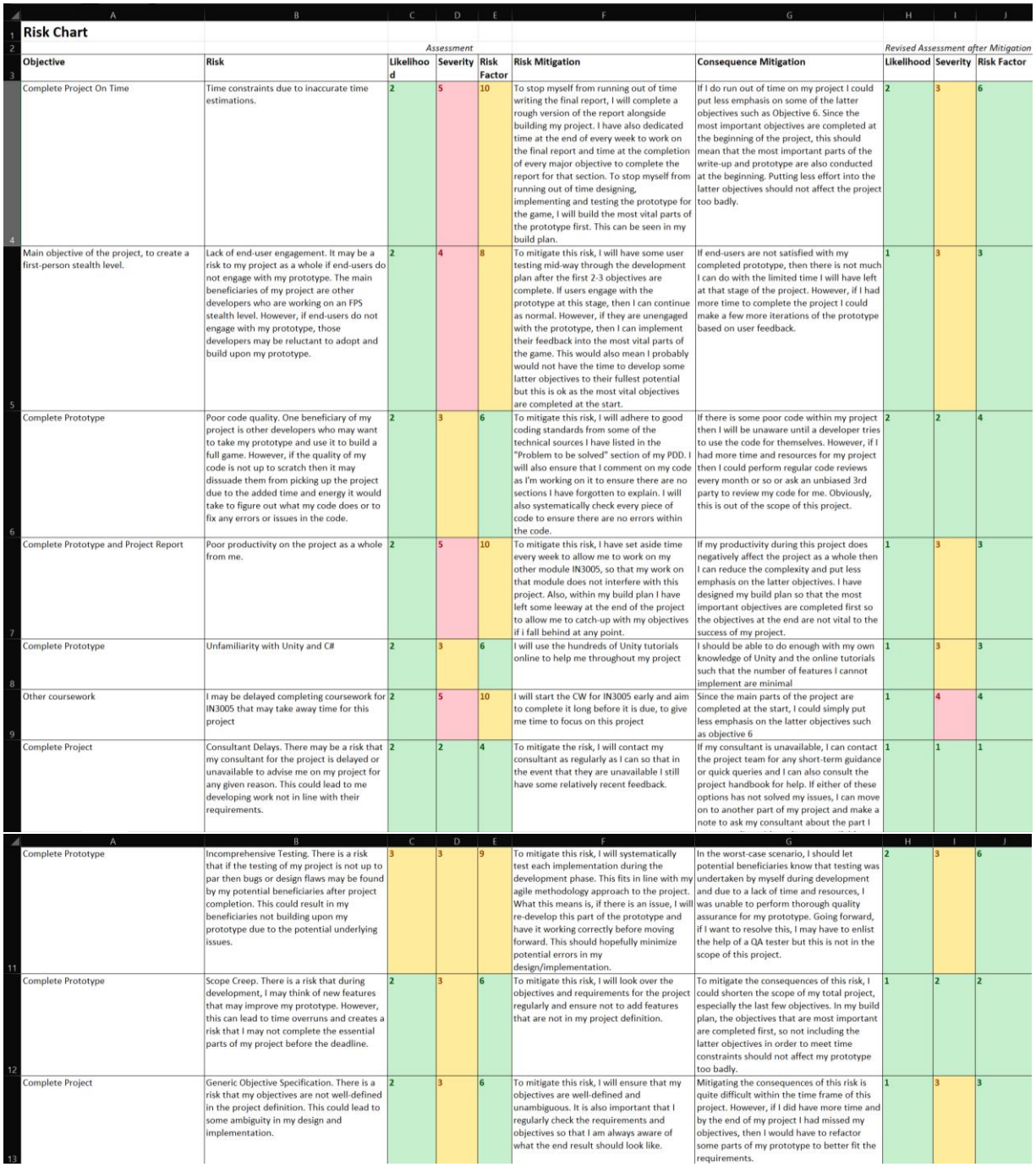
whole for things such as the literature review or the output summary for the final submission.



Risks to my project:

I have included a simple risk chart that looks at what the risks to my project may be. How likely they are to happen, how severe the consequences would be if they were to happen, how can I mitigate the risks, how can I mitigate the consequences if the risk occurred and what impact the mitigations would have on my project.

- **Likelihood:** 1(Unlikely) - 5(Likely)
- **Severity:** 1(Insignificant) – 5(Significant)
- **Risk Factor** = Likelihood x Severity
    - Risk Factor 0 - 7: **Low risk**
    - Risk Factor 8 - 17: **Medium Risk**
    - Risk Factor 18 - 25: **High Risk**



	A	B	C	D	E	F	G	H	I	J
1	**Risk Chart**									
2			*Assessment*					*Revised Assessment after Mitigation*		
3	**Objective**	**Risk**	**Likelihood**	**Severity**	**Risk Factor**	**Risk Mitigation**	**Consequence Mitigation**	**Likelihood**	**Severity**	**Risk Factor**
4	Complete Project On Time	Time constraints due to inaccurate time estimations.	2	5	10	To stop myself from running out of time writing the final report, I will complete a rough version of the report alongside building my project. I have also dedicated time at the end of every week to work on the final report and time at the completion of every major objective to complete the report for that section. To stop myself from running out of time designing, implementing and testing the prototype for the game, I will build the most vital parts of the prototype first. This can be seen in my build plan.	If I do run out of time on my project I could put less emphasis on some of the latter objectives such as Objective 6. Since the most important objectives are completed at the beginning of the project, this should mean that the most important parts of the write-up and prototype are also conducted at the beginning. Putting less effort into the latter objectives should not affect the project too badly.	2	3	6
5	Main objective of the project, to create a first-person stealth level.	Lack of end-user engagement. It may be a risk to my project as a whole if end-users do not engage with my prototype. The main beneficiaries of my project are other developers who are working on an FPS stealth level. However, if end-users do not engage with my prototype, those developers may be reluctant to adopt and build upon my prototype.	2	4	8	To mitigate this risk, I will have some user testing mid-way through the development plan after the first 2-3 objectives are complete. If users engage with the prototype at this stage, then I can continue as normal. However, if they are unengaged with the prototype, then I can implement their feedback into the most vital parts of the game. This would also mean I probably would not have the time to develop some latter objectives to their fullest potential but this is ok as the most vital objectives are completed at the start.	If end-users are not satisfied with my completed prototype, then there is not much I can do with the limited time I will have left at that stage of the project. However, if I had more time to complete the project I could make a few more iterations of the prototype based on user feedback.	1	3	3
6	Complete Prototype	Poor code quality. One beneficiary of my project is other developers who may want to take my prototype and use it to build a full game. However, if the quality of my code is not up to scratch then it may dissuade them from picking up the project due to the added time and energy it would take to figure out what my code does or to fix any errors or issues in the code.	2	3	6	To mitigate this risk, I will adhere to good coding standards from some of the technical sources I have listed in the "Problem to be solved" section of my PDD. I will also ensure that I comment on my code as I'm working on it to ensure there are no sections I have forgotten to explain. I will also systematically check every piece of code to ensure there are no errors within the code.	If there is some poor code within my project then I will be unaware until a developer tries to use the code for themselves. However, if I had more time and resources for my project then I could perform regular code reviews every month or so or ask an unbiased 3rd party to review my code for me. Obviously, this is out of the scope of this project.	2	2	4
7	Complete Prototype and Project Report	Poor productivity on the project as a whole from me.	2	5	10	To mitigate this risk, I have set aside time every week to allow me to work on my other module IN3005, so that my work on that module does not interfere with this project. Also, within my build plan I have left some leeway at the end of the project to allow me to catch-up with my objectives if i fall behind at any point.	If my productivity during this project does negatively affect the project as a whole then I can reduce the complexity and put less emphasis on the latter objectives. I have designed my build plan so that the most important objectives are completed first so the objectives at the end are not vital to the success of my project.	1	3	3
8	Complete Prototype	Unfamiliarity with Unity and C#	2	3	6	I will use the hundreds of Unity tutorials online to help me throughout my project	I should be able to do enough with my own knowledge of Unity and the online tutorials such that the number of features I cannot implement are minimal	1	3	3
9	Other coursework	I may be delayed completing coursework for IN3005 that may take away time for this project	2	5	10	I will start the CW for IN3005 early and aim to complete it long before it is due, to give me time to focus on this project	Since the main parts of the project are completed at the start, I could simply put less emphasis on the latter objectives such as objective 6	1	4	4
	Complete Project	Consultant Delays. There may be a risk that my consultant for the project is delayed or unavailable to advise me on my project for any given reason. This could lead to me developing work not in line with their requirements.	2	2	4	To mitigate the risk, I will contact my consultant as regularly as I can so that in the event that they are unavailable I still have some relatively recent feedback.	If my consultant is unavailable, I can contact the project team for any short-term guidance or quick queries and I can also consult the project handbook for help. If either of these options has not solved my issues, I can move on to another part of my project and make a note to ask my consultant about the part I	1	1	1

	A	B	C	D	E	F	G	H	I	J
11	Complete Prototype	Incomprehensive Testing. There is a risk that if the testing of my project is not up to par then bugs or design flaws may be found by my potential beneficiaries after project completion. This could result in my beneficiaries not building upon my prototype due to the potential underlying issues.	3	3	9	To mitigate this risk, I will systematically test each implementation during the development phase. This fits in line with my agile methodology approach to the project. What this means is, if there is an issue, I will re-develop this part of the prototype and have it working correctly before moving forward. This should hopefully minimize potential errors in my design/implementation.	In the worst-case scenario, I should let potential beneficiaries know that testing was undertaken by myself during development and due to a lack of time and resources, I was unable to perform thorough quality assurance for my prototype. Going forward, if I want to resolve this, I may have to enlist the help of a QA tester but this is not in the scope of this project.	2	3	6
12	Complete Prototype	Scope Creep. There is a risk that during development, I may think of new features that may improve my prototype. However, this can lead to time overruns and creates a risk that I may not complete the essential parts of my project before the deadline.	2	3	6	To mitigate this risk, I will look over the objectives and requirements for the project regularly and ensure not to add features that are not in my project definition.	To mitigate the consequences of this risk, I could shorten the scope of my total project, especially the last few objectives. In my build plan, the objectives that are most important are completed first, so not including the latter objectives in order to meet time constraints should not affect my prototype too badly.	1	2	2
13	Complete Project	Generic Objective Specification. There is a risk that my objectives are not well-defined in the project definition. This could lead to some ambiguity in my design and implementation.	2	3	6	To mitigate this risk, I will ensure that my objectives are well-defined and unambiguous. It is also important that I regularly check the requirements and objectives so that I am always aware of what the end result should look like.	Mitigating the consequences of this risk is quite difficult within the time frame of this project. However, if I did have more time and by the end of my project I had missed my objectives, then I would have to refactor some parts of my prototype to better fit the requirements.	1	3	3

Risks that my project can cause to others:

I do not believe my project can cause harm to any end-user. It is a simple prototype level of a single-player offline game and does not pose any obvious threats to any end-user.

Ethics Checklist:

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**

2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/*	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**

A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**NO**

9.2. Appendix B: Reuse Summary

The source code can be found by first opening the project in Unity. Then navigating in the Project tab to Assets -> Scripts. Then open a folder and then a subsequent script. A Visual Studio IDE should open. All source code throughout the entire project will be displayed in the Solution Explorer of Visual Studio.

9.2.1. Entirely Reused Source Code Files

This will include all code that is entirely re-used and has not been altered by me.

- "NavMeshComponents" folder
  - https://github.com/Brackeys/NavMesh-Tutorial/tree/master/NavMesh%20Example%20Project/Assets/NavMeshComponents
  - https://www.youtube.com/watch?v=CHV1ymlw-P8&ab_channel=Brackeys
- "Behaviour Tree" folder
  - https://github.com/MinaPecheux/UnityTutorials-BehaviourTrees/tree/master/Assets/Scripts/BehaviorTree
  - https://www.youtube.com/watch?v=aR6wt5BlE-E&ab_channel=MinaP%C3%AAcheux
- "Player Movement and Camera" folder
  - https://www.youtube.com/watch?v=f473C43s8nE&t=379s&ab_channel=Dave%2FGameDevelopment
  - https://www.youtube.com/watch?v=xCxSjgYTw9c&list=PLh9SS5jRVLAleXEcDTWxBF39UjyrFc6Nb&index=8&ab_channel=Dave%2FGameDevelopment
- Within the "Smoke Bomb Scripts" folder, the "Smoke Bomb" script
  - https://www.youtube.com/watch?v=BYL6JtUdEY0&ab_channel=Brackeys
- Within the "UI Scripts" folder, the "Billboard" script
  - https://www.youtube.com/watch?v=BLfNP4Sc_iA&t=836s&ab_channel=Brackeys
- "Unity Asset Store" folder
  - https://assetstore.unity.com/packages/2d/textures-materials/floors/20-man-made-ground-materials-12835
  - https://assetstore.unity.com/packages/2d/textures-materials/door-texture-pack-223425
  - https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361#content
  - https://assetstore.unity.com/packages/3d/vegetation/trees/mobile-tree-package-18866#content
  - https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633
  - "TerrainSampleAssets" is a built-in downloadable package within Unity.
- "Unity Technologies" folder
  - https://assetstore.unity.com/packages/vfx/particles/particle-pack-127325
  - https://assetstore.unity.com/packages/vfx/particles/legacy-particle-pack-73777

9.2.2. Reused Modified Source Code

This includes source code that I reused but then adapted or changed in some way. Each file will be listed, and my changes will be highlighted in RED.

“DropSmoke” script within the “Smoke Bomb Scripts” folder under “Scripts”

https://www.youtube.com/watch?v=BYL6JtUdEY0&ab_channel=Brackeys

https://www.youtube.com/watch?v=Xw506Rfd9Q4&ab_channel=GameDevTrauminEnglish

```
using UnityEngine;
using TMPro;

// This file was largely NOT written by me
// I only wrote the parts related to the cooldown, and limiting the smoke bombs
uses

// To make it clear, the parts i wrote will have //[ME] at the end of the line

// Main Tutorial from:
https://www.youtube.com/watch?v=BYL6JtUdEY0&ab_channel=Brackeys
// Tutorial fro TextMeshPro/TMP from:
https://www.youtube.com/watch?v=Xw506Rfd9Q4&ab_channel=GameDevTrauminEnglish

// Class to throw a smoke bomb

public class DropSmoke : MonoBehaviour
{
    [Header("Variables")]
    public float throwForce;
    public float throwCooldown; //[ME]
    public int bombCount; //[ME]
    bool readyToThrow = true; //[ME]
    Vector3 offset = new Vector3();

    [Header("Objects")]
    public GameObject grenadePrefab;
    public Transform Orienatation;
    public TMP_Text bombCountText;

    [Header("Grenade Key")]
    public KeyCode grenadekey = KeyCode.G;


    void Start()
    {
        // Set current smoke bomb count in HUD
        bombCountText.text = bombCount.ToString();
    }

    // Update is called once per frame
    void Update()
    {
        // only throw the grenade if ready to throw and the player has grenades
to throw. Once thrown only reset the throw after 'throwCooldown' time
        if (Input.GetKey(grenadekey) && readyToThrow == true && bombCount > 0)
        {
            readyToThrow = false; //[ME]
```

```
            ThrowGrenade();

            bombCount = bombCount - 1; //[ME]

            // Set new smoke bomb count in HUD
            bombCountText.text = bombCount.ToString();

            Invoke(nameof(ResetThrow), throwCooldown); //[ME]
        }
    }

    // throw grenade in direction of player orientation object. Found under
Player in hierachy
    void ThrowGrenade()
    {
        offset = (Orienatation.forward * 1.5f);
        GameObject grenade = Instantiate(grenadePrefab, transform.position +
offset, Orienatation.rotation);
        Rigidbody rb = grenade.GetComponent<Rigidbody>();
        rb.AddForce(Orienatation.forward * throwForce, ForceMode.VelocityChange);
    }

    void ResetThrow()
    {
        readyToThrow = true; //[ME]
    }
}
```

9.2.3. Source Code Written Entirely by Me

These files are entirely written by me and can be found under the “Scripts” folder

- “Guard AI” folder
- “Destroy Smoke” script within the “Smoke Bomb Scripts” folder
- “Alerted Sprite” script within the “UI Scripts” folder
- “Searching Sprite” script within the “UI Scripts” folder
- “DetectionBarSprite” script within the “UI Scripts” folder
- “Zipline” folder

9.2.4. Reused Software

- Unity 2020.3.29f1
- Visual Studio 2022
- MS Paint (used to design guard vision cone and zones graphic)
- Adobe Behaviour Tree Visual Editor
    - https://opensource.adobe.com/behavior_tree_editor/#/dash/home
    - Used to design Guard Behaviour Tree Graphic
- Piskel
    - https://www.piskelapp.com/
    - Used to design UI sprites

9.2.5. Reused Tutorials for Unity

These are specific tutorials I used that had no outputs in code form but had outputs in Unity. E.g. creating a game object.

- Setting the level of Detection in the Detection Bar
    - https://www.youtube.com/watch?v=BLfNP4Sc_iA&ab_channel=Brackeys
- TextMeshPro tutorial. Used to set smoke bomb number on the screen.
    - https://www.youtube.com/watch?v=Xw506Rfd9Q4&ab_channel=GameDevTrauminEnglish

9.2.6. Reused Unity Assets

These include Unity assets I used in the level, but that have no code attached to them and therefore do not show up under the “UnityAssetStore” folder

- https://creazilla.com/nodes/25373-binoculars-clipart
- https://assetstore.unity.com/packages/3d/props/weapons/grenade-m18-smoke-66223
- https://assetstore.unity.com/packages/3d/vegetation/plants/yughues-free-bushes-13168
- https://assetstore.unity.com/packages/2d/textures-materials/wood/plank-textures-pbr-72318
- https://assetstore.unity.com/packages/3d/vehicles/land/zil-130-military-truck-208991
- https://assetstore.unity.com/packages/3d/props/weapons/free-missile-72692
- https://assetstore.unity.com/packages/3d/props/industrial/cargo-container-45175

9.3. Appendix C: Use Case Requirements

Use Case: Enemy Detection	**ID:** 1C
Description: An enemy will be able to detect the player slowly over time.	
Primary Actors: Enemy	**Secondary Actors:**
Preconditions: 1) Player is in an undetected state	
Main Flow: 1) The use case will begin when the player enters the enemy`s field of view 2) A timer will start to check how long the player is in the enemy`s field of view 3) If the timer reaches a pre-determined endpoint, the player will be spotted 4) The timer should have a shorter pre-determined endpoint if the player is closer and more central in the enemy`s line of sight	
Postconditions: 1) The enemy spots the player	
Alternative Flows: The player escapes the enemy`s field of view before the timer ends	
Preconditions: 1) The player is in the enemy`s field of view, and the timer has not ended	
Alternative Flow: 1) The player escapes the enemy`s field of view 2) The timer starts counting back down until 0 3) The player is not spotted, and the timer reaches 0	
Postconditions: 1) The enemy does not spot the player	

Use Case: Enemy Pathfinding	**ID:** 1B
Description: The enemies will use a pathfinding algorithm to reach the player's last known location. The enemies will then pathfind from that point outwards to attempt to locate the player.	
Primary Actors: Enemy	**Secondary Actors:**
Preconditions: 1) The player has been spotted by an enemy and has since escaped	
Main Flow: 1) All of the enemies nearby convene on the player`s last known location 2) The enemies use a pathfinding algorithm to traverse different parts and corners of the map	
Postconditions: 1) The enemies end their search after a short search and return to their pre-determined patrol paths	
Alternative Flows: N/A	

Use Case: Enemy Behaviour Tree	**ID:** 1A
Description: A behaviour tree containing four states will dictate the enemy's behaviour. The four states include Patrolling, Chasing, Attacking, and Searching.	
Primary Actors: Enemy	**Secondary Actors:**
Preconditions: 1) There are instances of guards using the behaviour tree within the game	
Main Flow:	

1) If the player has not been spotted, the guard will patrol a pre-determined route 2) When a guard spots the player, all of the guards will chase the player 3) When the guards are within range, they will attack the player 4) If the player escapes, the guards will search the area a) If the player is found again, they will chase and attack b) If the player is not found again, they will go back to their patrol paths
Postconditions: 1) The states will reset back to what they were before the player was spotted
Alternative Flows: N/A

Use Case: UI Depicting Enemy States	**ID:** 2A
Description: The enemies will have a small UI element/sprite above their heads, depicting their current state. The four states the enemy can be in include, Patrolling, Chasing, Attacking and Searching. These will be split into three groups. One for Patrolling, which will have no UI element. One for Searching will have a UI element. Finally, one for Chasing and Attacking, which will have the same UI element.	
Primary Actors: Enemy	**Secondary Actors:**
Preconditions: 1) The game will be running	
Main Flow: 1) The guards will have no UI element above their heads when patrolling 2) The guards will have a small exclamation mark icon above their heads when the player has been detected, and the guards are either chasing or attacking 3) The guards will have a small question mark icon above their heads when the player has been lost, and they are searching	
Postconditions: 1) The guards will return to having no icons above their heads if the player has not been found, and they will return to patrolling	
Alternative Flows: N/A	

Use Case: UI Depicting Enemy Detection Level	**ID:** 2B
Description: When the player is visible to the enemy, a small UI element will appear above the head of the enemy. This bar will fill up slowly based on how close the enemy is to detecting the player. The closer the enemy is to seeing the player, the further the bar will fill up. The quicker the enemy detects the player, the faster the bar will fill up.	
Primary Actors: Enemy	**Secondary Actors:**
Preconditions: 1) The game will be running, and the UI element for detection will not be above the enemy`s head.	
Main Flow: 1) The player will start in an undetected state. 2) The player will then enter the enemy's vision zone. The detection bar should take the same amount of time to fill up as it does for the enemy to detect the player in that zone. E.g. it should take 1 second for zone 1, 5 seconds for zone 5, etc. 3) Once detected, the detection bar should disappear and reveal the alerted UI symbol from objective 2A.	
Postconditions: 1) The detection bar will only reappear when the guard is not in an alerted state, and the guard is in the process of spotting the player.	
Alternative Flows:	

The player hides while the enemy`s detection bar is half full.
Preconditions: 1) The player is in an undetected state
Alternative Flow: 1) The player will go into the enemy`s vision cone. 2) The enemy will begin to detect the player, and the detection bar should start to fill up 3) Before the enemy fully detects the player, the player will leave the line of sight. 4) The detection bar should start returning to empty if the player is not visible to the enemy.
Postconditions: 1) The detection bar return to empty and subsequently disappears.

Use Case: Smoke Bomb Mechanic	**ID:** 3Ai
Description: The player will be able to use a smoke bomb to evade the enemies. The smoke bomb will temporarily blind any enemies within the smoke bomb radius, and all enemies will lose sight of the player while the player is within the smoke bomb radius.	
Primary Actors: Player	**Secondary Actors:** Enemies
Preconditions: 1) The player will have been spotted by the enemies and attempting to escape	
Main Flow: 1) The player will press a button on the keyboard to drop a smoke bomb at their feet 2) The smoke bomb will explode soon after it hits the ground 3) The smoke bomb will have a radius much larger than the player 4) The smoke will obstruct the vision of any enemy 5) Any enemy within the smoke will lose sight of the player and remain stationary until the smoke clears	
Postconditions: 1) After the smoke clears, the enemies will go into a search state if they cannot immediately see the player	
Alternative Flows: The smoke bomb explodes while the enemies have not spotted the player	
Preconditions: 1) The player will be hidden from the guards	
Main Flow: 1) The player will drop a smoke bomb 2) If the guards see the smoke bomb, they will investigate it 3) After the smoke clears and the guards have stopped investigating the smoke bomb, they will enter a search pattern	
Postconditions: 1) The guards will search for the player as normal	

Use Case: Hiding Mechanic	**ID:** 3Aii
Description: The player will be able to escape the enemy's line of sight by entering an object such as a bush or tall grass, etc.	
Primary Actors: Player	**Secondary Actors:** Enemy
Preconditions: 1) The player will be visible to the enemy	
Main Flow: 1) The player will then move into the bush/tall grass	

2) The enemy will lose sight of the player and move to the bush/tall grass to investigate
Postconditions: 1) The enemy will search for the player as normal if the player cannot be found inside the bush
Alternative Flows: N/A

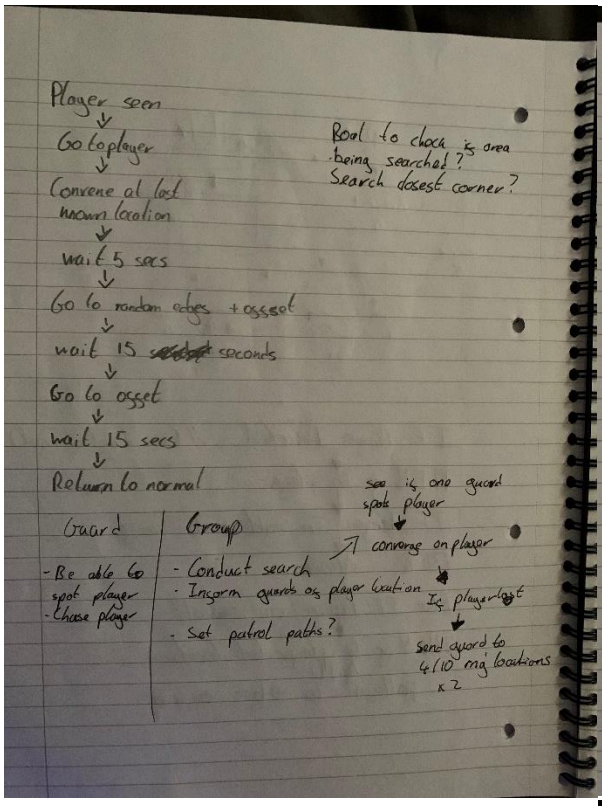
Use Case: Limiting Use of Smoke Bomb Mechanic	**ID:** 3Bi
Description: The player will only be able to use a set number of smoke bombs before they run out, and the player should not be able to use them repeatedly in quick succession.	
Primary Actors: Player	**Secondary Actors:**
Preconditions: 1) The game will begin as normal	
Main Flow: 1) The player will drop a smoke bomb and have to wait a specified cooldown period before dropping another 2) Once the cooldown timer has ended, the player can drop another smoke bomb 3) Once the player has dropped all of their specified smoke bombs, they cannot drop any more	
Postconditions: 1) The player has no remaining smoke bombs and cannot drop any more	
Alternative Flows: N/A	

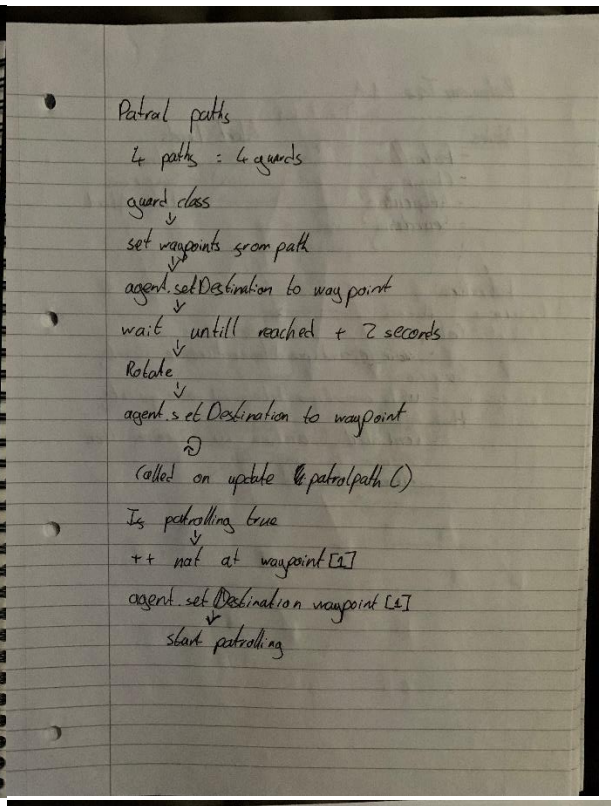
Use Case: Limiting Use of Hiding Mechanic	**ID:** 3Bii
Description: The player should not be able to rely solely on the game's hiding mechanic.	
Primary Actors: Player	**Secondary Actors:**
Preconditions: 1) The game will begin as normal	
Main Flow: 1) The player will have many hiding options when on the outskirts of the level, allowing the player to perform reconnaissance on the level safely. 2) As the player moves closer to the main part of the level, the number of hiding options will drastically reduce and be significantly smaller than those on the outskirts.	
Postconditions: 1) The player continues the game as normal	
Alternative Flows: N/A	

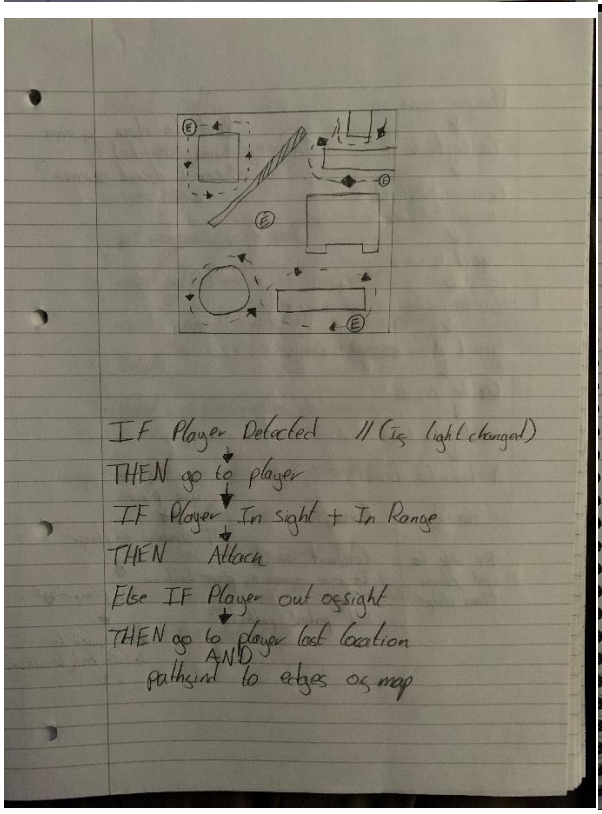
Use Case: Unique Traversal	**ID:** 4a
Description: The player should be able to traverse the level dynamically and uniquely.	
Primary Actors: Player	**Secondary Actors:**
Preconditions: 1) The player is playing level as normal	
Main Flow: 1) The player will approach a zipline and stand within 1 meter of it 2) The player will press a button on the keyboard 3) The player will be hooked onto the zipline and start moving to the end of the zipline 4) The player will not be able to use any of their controls at this stage	

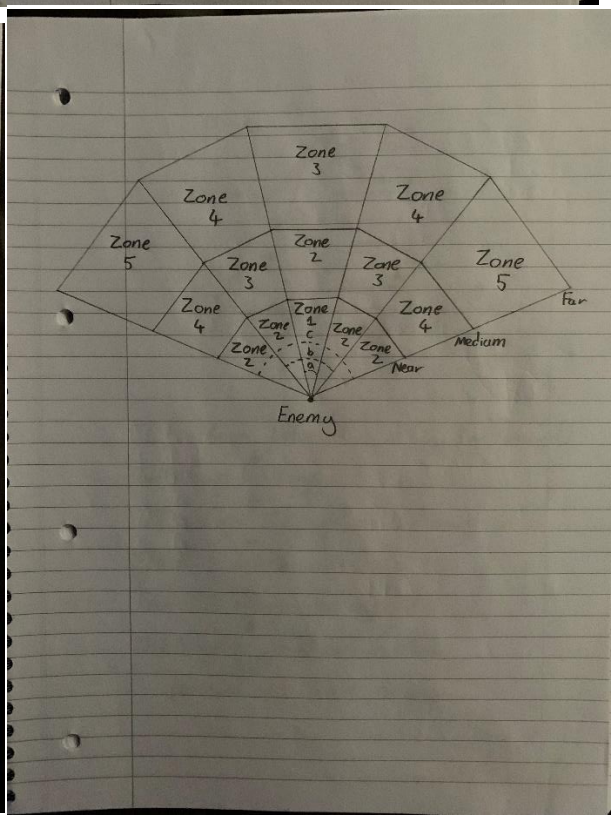
5) Once the player reaches the end of the zipline, they will be thrown off and can move around freely again
Postconditions: 1) The player can continue the level as normal
Alternative Flows: N/A

9.4. Appendix D: Design Documentation

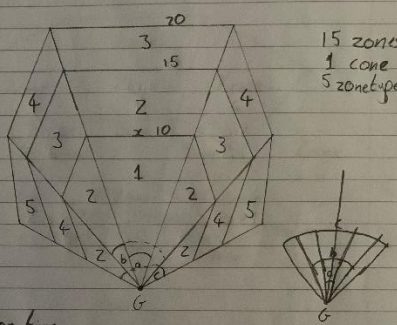








Guard AI detection system



15 zones
1 cone
5 zone types

Detection time

R	1: 1s	
M O	2: 1.5s	
Y Y	3: 2s	
G	4: 3s	
B	5: 6s	

Is player is between angle a
AND
Distance < x
THEN
Start timer
Is timer = 1s
THEN
state Attack

Is player is between angle c but not ab
AND
Distance < x
THEN
Zone 2/4/5 in c
start timer

Objective 2A UI for Enemy states

States: Patrolling, chasing + Attacking, Searching

Alerted

Alerted:

- [!]

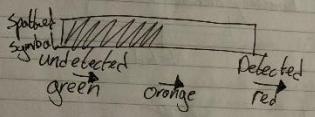
Searching:

- [?]

Patrolling:

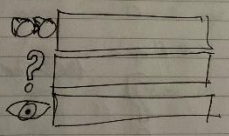
- Nothing

Objective 2B UI for enemy detection



Symbols:

- Binoculars?
- Small '?'?
- Eye?



Start a Timer

For loop through ALL Guards

Is the Timer is between 10s and 25s

Set the Guards to go to a random search location on the map

Else IF the Timer is between 25s and 40s

Set the Guards to go to another random search location on the map

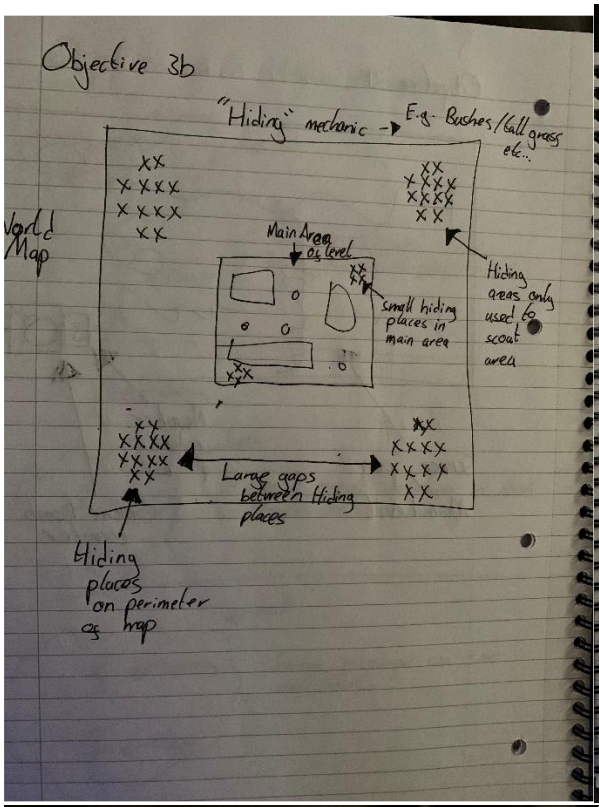
Else IF the Timer is above 40s

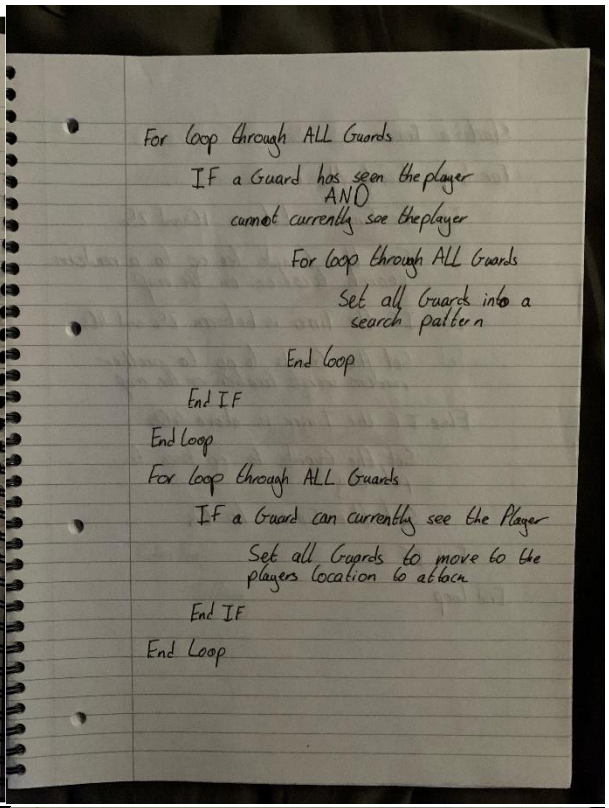
Set the Guards to go back to patrolling

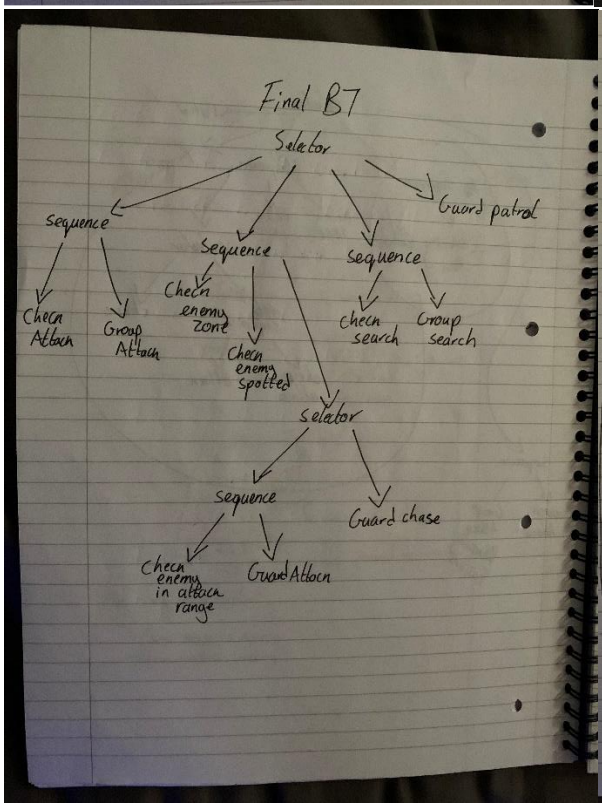
Set the Timer to 0s

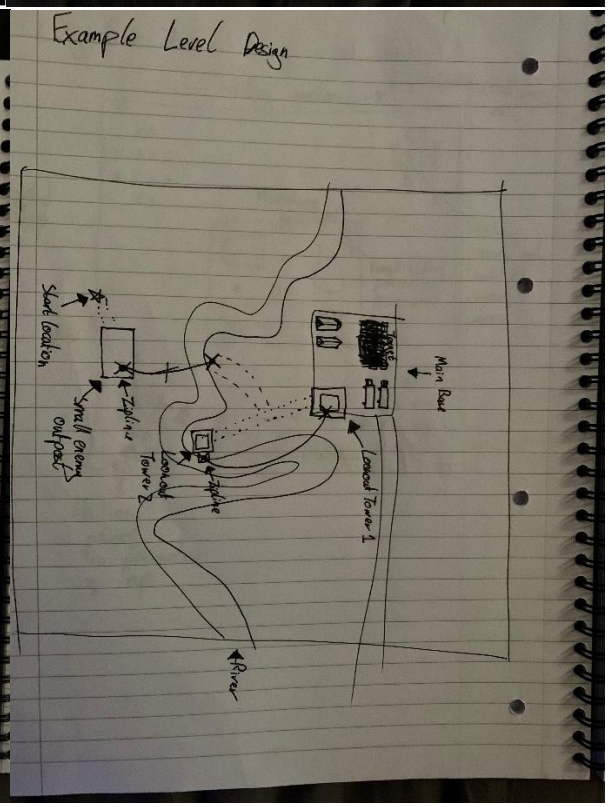
End IF

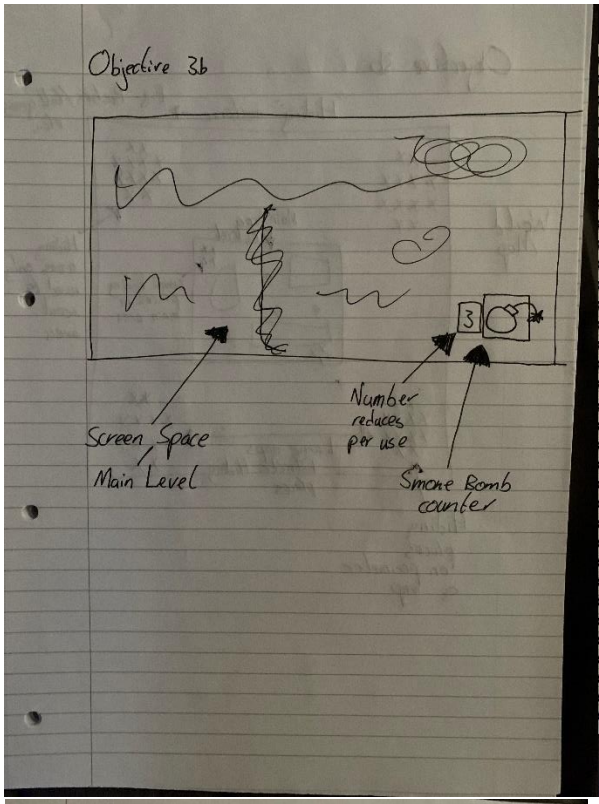
End loop

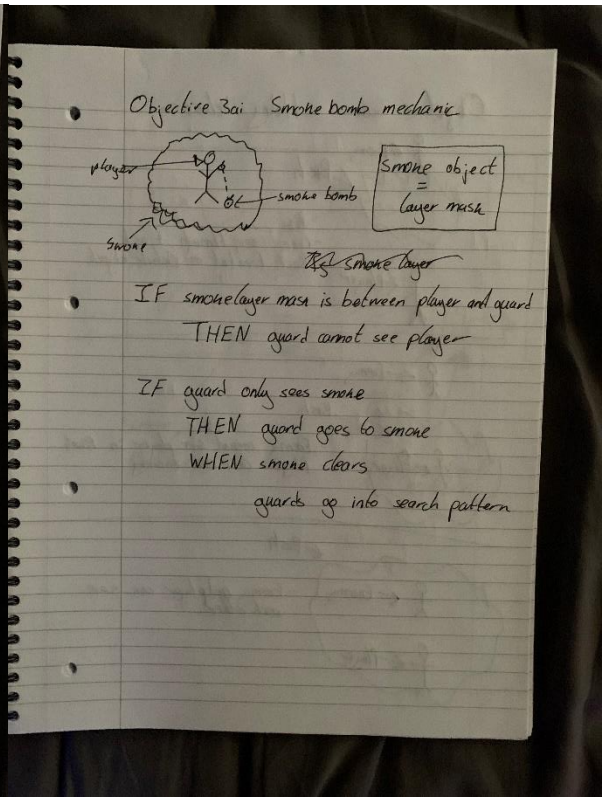

Objective 3b
"Hiding" mechanic → E.g. Bushes/tall grass etc...
World Map
Main Area of level
Small hiding places in main area
Hiding areas only used to scout area
Large gaps between Hiding places
Hiding places on perimeter of map

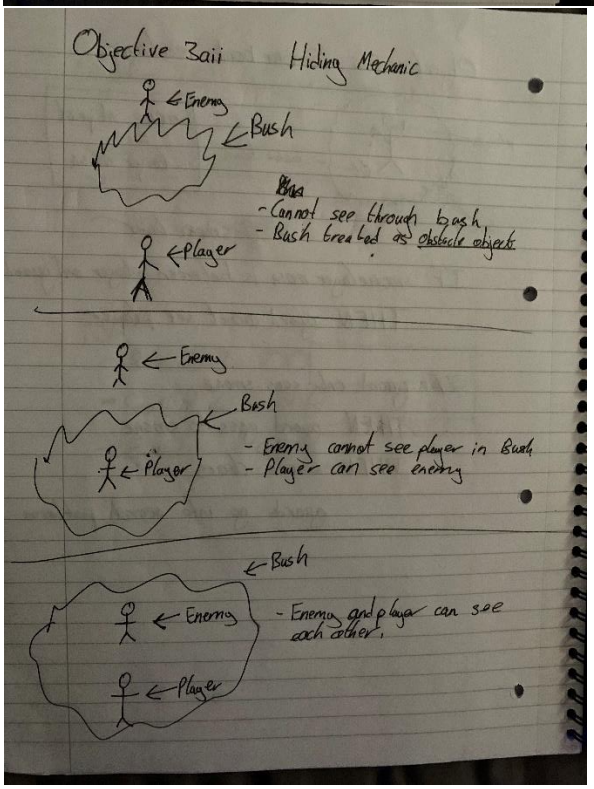

For Loop through ALL Guards
IF a Guard has seen the player
AND
cannot currently see the player
For Loop through ALL Guards
Set all Guards into a search pattern
End Loop
End IF
End Loop
For Loop through ALL Guards
IF a Guard can currently see the Player
Set all Guards to move to the players location to attack
End IF
End Loop


Final BT
Selector
Sequence
Guard patrol
Sequence
Sequence
Check Attack
Group Attack
Check enemy zone
Check enemy spotted
Check search
Group search
Selector
Sequence
Guard chase
Check enemy in attack range
Guard Attack


Example Level Design
Start location
Small enemy outpost
Zipline
Lookout Tower 2
Zipline
Lookout Tower 1
Main Base
River


Objective 3b
3
Screen Space
Main Level
Number reduces per use
Smoke Bomb counter


Objective 3ai Smoke bomb mechanic
player
smoke bomb
smoke
Smoke object = layer mask
Smoke layer
IF smokelayer mask is between player and guard
THEN guard cannot see player
IF guard only sees smoke
THEN guard goes to smoke
WHEN smoke clears
guards go into search pattern


Objective 3aii Hiding Mechanic
Enemy
Bush
- Cannot see through bush
- Bush treated as obstacle object
Player
Enemy
Bush
Player
- Enemy cannot see player in Bush
- Player can see enemy
Bush
Enemy
Player
- Enemy and player can see each other.

9.5. Use Case Testing

Use Case: Enemy Detection	**ID:** 1C
Test Number: 1	
Objective: To test whether a player will be detected by the enemy and be detected at different speeds based on where they are in the enemy's field of view	
Set up: The player will take turns standing in the enemy's five vision zones. The player will start outside the enemy's field of view AND ensure the timer has not started. The player will then move to a vision zone and test how long it takes for the player to be spotted. The enemy's spotlight will change to a different colour based on which zone the player has been spotted in, making it easy to visualise this test.	
Expected Results: After spending 1 second in zone 1, the player should be spotted, and the enemy`s spotlight should go red. After spending 1.5 seconds in zone 2, the player should be spotted, and the enemy`s spotlight should go magenta. After spending 2 seconds in zone 3, the player should be spotted, and the enemy`s spotlight should go yellow. After spending 3 seconds in zone 4, the player should be spotted, and the enemy`s spotlight should go green. After spending 5 seconds in zone 5, the player should be spotted, and the enemy`s spotlight should go blue.	
Test: The player will enter zone 1, check how long it takes for the enemy`s spotlight to go red, leave the enemy`s field of view, and wait for the timer to reset. The player will enter zone 2, check how long it takes for the enemy`s spotlight to go magenta, leave the enemy`s field of view, and wait for the timer to reset. The player will enter zone 3, check how long it takes for the enemy`s spotlight to go yellow, leave the enemy`s field of view, and wait for the timer to reset. The player will enter zone 4, check how long it takes for the enemy`s spotlight to go green, leave the enemy`s field of view, and wait for the timer to reset. The player will enter zone 5, check how long it takes for the enemy`s spotlight to go blue, leave the enemy`s field of view, and wait for the timer to reset.	
Test Record: Expected results observed	
Date: 23rd March 2023	**Tester:** Tayyab Hussain
Result: Passed	

Use Case: Enemy Pathfinding	**ID:** 1B
Test Number: 2	
Objective: To test whether the enemies succeed in independent pathfinding to search for the player.	
Set up: The enemies should be going along their patrol paths, as usual, to begin with. After they spot the player, they should path find to the player's location. Once the player has escaped and the enemies can no longer see the player, they should pathfind to search for the player.	
Expected Results: The enemies should all congregate at the player's last known location. They should then spend 20 seconds searching a pre-determined position on the map using the pathfinding algorithm to traverse to that location. After 20 seconds, they should then pathfind to another location on the map and search there for 15 seconds. After 40 total seconds of	

searching(accounting for 5 seconds at the start to get to the player's last known location), they should path find back to their patrol paths and continue patrolling.
Test: The player will start in an undetected state and check that the enemies are following their patrol paths. The player will then enter an enemy's vision zone and be spotted. The player will then run, hide, and be outside any enemy vision zone until the enemies have completed both searches and returned to their patrol paths.
Test Record: Expected Results Observed
Date: 22/04/23 **Tester:** Tayyab Hussain
Result: Passed

Use Case: Enemy Behaviour Tree	**ID:** 1A
Test Number: 3	
Objective: To test whether the behaviour tree can successfully implement the four relevant actions defined in the requirements specification and do so at the correct time based on the current game state.	
Set up: The player will begin the game outside the map to test the enemy patrol paths. The player will then be placed into the map and play in a way that will result in the enemy AI having to use all of the four different actions within the game and use them at the correct time.	
Expected Results: The enemies should patrol when the player has yet to be spotted. If a guard spots the player, all the other guards should be alerted and should pathfind to the player's location. If the guards are close enough to the player, they should attack. They will chase him if they have spotted him but are too far away. If they all lose sight of him after having previously seen him. They will enter a search pattern. They will search two randomly assigned locations on the map until they have been searching for 40 seconds. Then they should return to their patrol paths. If they find the player whilst searching, they should return to either chase or attack the player based on the distance between them and the player.	
Test: The player will start in an undetected state and ensure that the enemies all follow their patrol paths. The player will then appear in the vision cone of 1 enemy and be subsequently spotted by the enemy. The player will wait until all the guards have traversed to the player's location. The player will then slowly walk around the map. The tester will ensure that all guards attack the player when they are close and chase the player when they are far away. The player will then go into hiding. The tester will ensure that all guards enter their search pattern. During the searching, the player will then attempt o be spotted again by the guards to ensure they switch states from searching to attacking or chasing. The player will then go back into hiding. The tester will ensure once again that the guards enter their search pattern. The tester will then observe if the guards return to their patrol paths after 40 seconds.	
Test Record: Expected Results observed	
Date: 05/05/23	**Tester:** Tayyab Hussain
Result: Passed	

Use Case: UI Depicting Enemy States	**ID:** 2A
Test Number: 4	

Objective: To test whether the UI elements depicting the enemy`s state are working correctly based on the current state of the guard.	
Set up: The player will play the game normally, going through all four states for each enemy and checking if the UI correctly depicts each guard's state.	
Expected Results: The enemies should not have a sprite above their heads when patrolling. The enemies should have an exclamation mark sprite over their heads when the player has been spotted. The enemies should have a question mark sprite above their heads when in a search pattern. The enemies should return to having no UI element above their heads when they return to a patrolling state after the search is complete.	
Test: The player will start the game outside of the map. The tester will ensure the guards have no UI elements above their heads while patrolling. The player will then be put inside the map and try to be detected by a guard. The tester should only see an exclamation mark over the guard's head when the guard has completely spotted the player. The player will then hide, and the guards should enter a search pattern. During this search pattern, the tester should check that the guards no longer display an exclamation mark sprite over their heads and now display a question mark sprite. The player will attempt to be spotted again by a guard while the search is ongoing to check if the UI elements can change back from a question mark sprite to an exclamation mark sprite. The player will then leave the map entirely, and the tester will observe if the guards all change their UI elements to a question mark sprite while in a search pattern. Once the search pattern has concluded, the tester will observe if the guards have returned to patrolling and no longer have any UI element above their heads.	
Test Record: Expected Results observed	
Date: 08/05/23	**Tester:** Tayyab Hussain
Result: Passed	

Use Case: UI Depicting Enemy Detection Level	**ID:** 2B
Test Number: 5	
Objective: To test whether the UI correctly depicts how close the enemy is to detecting the player.	
Set up: The test version of the game will have only one enemy on the map, and the enemy will not be able to move or look around. This will make it easier to check the detection level.	
Expected Results: The enemy`s detection bar should not appear above their head until the player walks into the enemy`s vision zone. The enemy will then have a detection bar above their head, which should slowly start filling up at the same rate the enemy would detect the player. Once the enemy`s detection bar is full and the enemy detects the player, the detection bar should disappear, and the Alerted UI element should appear instead.	
Test: The player will start outside of the enemy`s vision cone. The player will then walk into the vision cone, and the tester should ensure the detection bar appears above the enemy and starts to fill up.	

The player will stay in the zone until the bar is half full and then step out of the zone. The tester should ensure that the bar starts to decrease slowly, and the bar should disappear once empty. The player will then re-enter the enemy`s vision zone and stay in the zone until the bar is full. The tester should observe that the bar disappears and the Alerted UI element appears above the enemy`s head.
Test Record: The test was largely successful. However, there was one minor pitfall. When the player leaves the enemy's vision zone, the bar will initially jump down before slowly decreasing. This is due to the way the bar`s detection amount is coded. This is discussed further in the Results section.
Date: 11/05/23 **Tester:** Tayyab Hussain
Result: Passed, but a slight adjustment is needed to make the feature more comprehensive

Use Case: Smoke Bomb Mechanic	**ID:** 3Ai
Test Number: 6	
Objective: To test whether the smoke bomb deploys correctly, obstructs the view of the Enemies, stops the enemies from moving when inside the smoke and is correctly detected by an enemy without the player's presence.	
Set up: To test the smoke bomb, a test version of the game will be set up with only the player and a test enemy who cannot move. This is so that the tester can verify whether the enemy can see the player through the smoke and verify the basics, such as the deployment of the smoke and the visuals. Then the normal game will be played to check the enemies` reaction to the smoke.	
Expected Results: When the player hits the 'G' key on the keyboard, the smoke bomb canister will spawn in front of them and drop to the ground. After a 1-second delay, the smoke should be instantiated, and the canister should be culled. The enemies will not be able to see the player if the smoke obstructs them. The enemies will not move if they are caught in the smoke. The enemies will search for the player if they cannot find them after the smoke clears. If a smoke bomb appears, but the player has yet to be spotted by the enemies, the enemies should move towards the smoke but not go in the smoke and search for the player after the smoke dissipates.	
Test: The player will start in the test version and be spotted by the test enemy. The player will then press 'G' and drop the smoke bomb. The tester will ensure the canister will behave as a physics object and drop to the ground and have a 1-second delay before being culled from the level. The tester will also observe the smoke being instantiated at the location of the now-culled canister. As the smoke billows, the tester will observe the enemy and ensure that the enemy can no longer see the player when the smoke obstructs the player. Once the smoke clears, the tester will ensure that the enemy can now see and spot the player. Then the normal version of the game will be played. The player will be spotted by all of the enemies and drop the smoke. The tester will ensure that the enemies do not move while in the smoke. The player will leave the map, and the tester will ensure that the enemies start searching for the player and complete a full search pattern before returning to their patrol paths.	

The player will then drop smoke in front of one of the enemies on the level without being spotted. The tester will ensure that the enemy spots the smoke even without seeing the player and moves all of the enemies to the location of the smoke. The tester will ensure that when the smoke dissipates that the enemies enter a searching pattern again.
Test Record: The tests related to the smoke itself passed. This includes the smoke being released from the canister after a delay and the canister being culled. The smoke also blocks the enemy's vision, and the enemies cannot spot the player through the smoke, which works as intended. However, the remaining parts of the test all failed. This was predicted, as they were not implemented successfully into the build. The enemies did not spot the smoke, they only saw it as an obstacle in their way, and therefore they did not stop when inside the smoke or move to the smoke when they spotted it without seeing the player. This will be explored further in the Results section.

Date: 20/05/23	**Tester:** Tayyab Hussain
Result: Failed	

Use Case: Hiding Mechanic	**ID:** 3Aii
Test Number: 7	
Objective: To test whether the player can be spotted when by the enemy when the player is inside a bush	
Set up: A test version of the game will be set up. This will include the player, the bush and a test enemy.	
Expected Results: When the player stands next to but not in the bush, the enemy should be able to spot the player, and the enemy`s UI symbols should reflect this. The enemy should lose sight of the player when the player enters the bush. The enemy should then move to where the player was last spotted. The enemy should enter a search pattern as usual if the player cannot be found.	
Test: The player will begin next to the bush and let the enemy spot them. Once the enemy has spotted the player, they will move into the bush. The tester will ensure that the UI elements above the enemy`s head indicate that the enemy has lost sight of the player. The tester should ensure the enemy moves towards the player's last known location but cannot spot the player through the bush. The player will remain in the bush, and when the enemy is also in the bush, the tester will ensure that the enemy has spotted the player.	
Test Record: Expected Results Observed	
Date: 21/05/23	**Tester:** Tayyab Hussain
Result: Passed	

Use Case: Limiting Use of Smoke Bomb Mechanic	**ID:** 3Bi
Test Number: 8	
Objective: To test whether the smoke bombs have a finite use and ensure they cannot be used repeatedly in quick succession.	
Set up:	

A test version of the game will be set up with just the player and no enemies.
Expected Results: When the player presses the button for the smoke bomb, a smoke bomb will deploy. There will then be a delay before the player can drop another smoke bomb. The player also should only be able to drop a specified number of smoke bombs.
Test: The player will drop the first smoke bomb. After the first smoke bomb is dropped, the player will continue to press the button to drop another smoke bomb. The tester should ensure that no more smoke bombs are dropped until the specified cooldown timer for the smoke bomb has ended. Once the timer has ended, the player will be able to drop another smoke bomb. They will continue to press the smoke bomb button continuously. The tester will ensure that when the specified number of smoke bombs have been dropped, no more smoke bombs are dropped by the player, despite them still pressing the smoke bomb button.
Test Record: Expected Results Observed

Date: 22/05/23	**Tester:** Tayyab Hussain
Result: Passed	

Use Case: Limiting Use of Hiding Mechanic	**ID:** 3Bii
Test Number: 9	
Objective: To test whether the player can rely solely on the game's hiding mechanic.	
Set up: The normal game will begin	
Expected Results: The player will be able to hide in several locations on the outskirts of the map. Once the player gets closer to the centre of the map, the number of hiding locations will reduce significantly.	
Test: The player will begin by going to each hiding area on the map's outskirts. The player will then go closer to the level's main part and utilise all the hiding areas. The tester will ensure that there are significantly fewer places for the player to hide in the main area of the level as opposed to the outskirts.	
Test Record: Expected Results Observed	
Date: 22/05/23	**Tester:** Tayyab Hussain
Result: Passed	

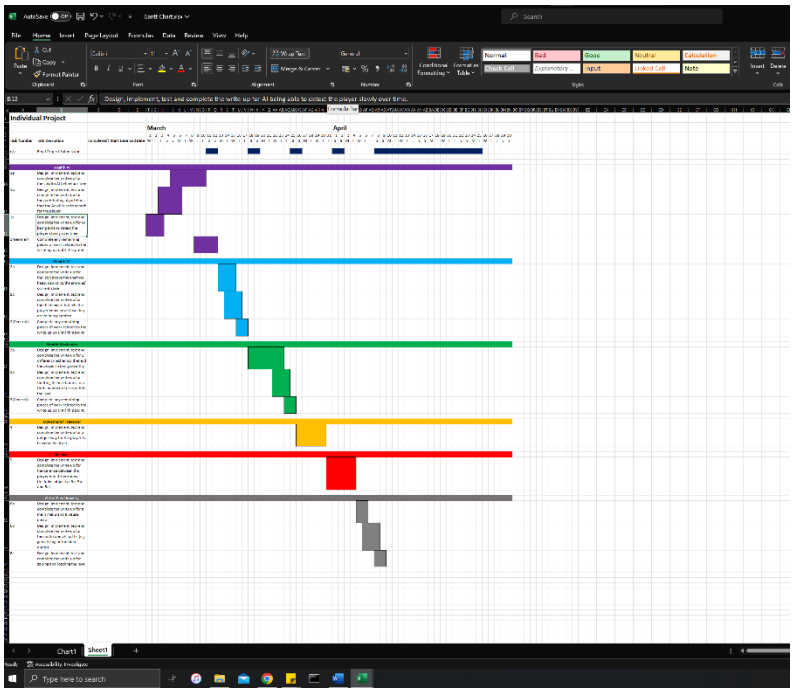
Use Case: Unique Traversal	**ID:** 4a
Test Number: 10	
Objective: To test whether the player can successfully use the zipline as intended	
Set up: The player will start positioned within 1 meter of the start of the zipline	
Expected Results: The player will be transported down the zipline at a constant speed when they press the correct button on the keyboard. The player will not be able to move around while on the zipline The player will, however, be able to look around when on the zipline When the player lands on the other side, they should be able to move freely again The player should not be able o use the zipline backwards(going uphill)	

Test: The player will first press the zipline button on the keyboard When travelling down the zipline, the player will then try to use every movement control in the game, such as walking, sprinting, jumping and crouching Before the player reaches the end of the zipline, they will check that moving the mouse still allows them to look around wherever they want while on the zipline When the player lands, they should recheck every control and ensure that they can now move freely Lastly, The player will ensure that they cannot use the zipline back to where they came from	
Test Record: Expected Results Observed	
Date: 06/06/23	**Tester:** Tayyab Hussain
Result: Passed	

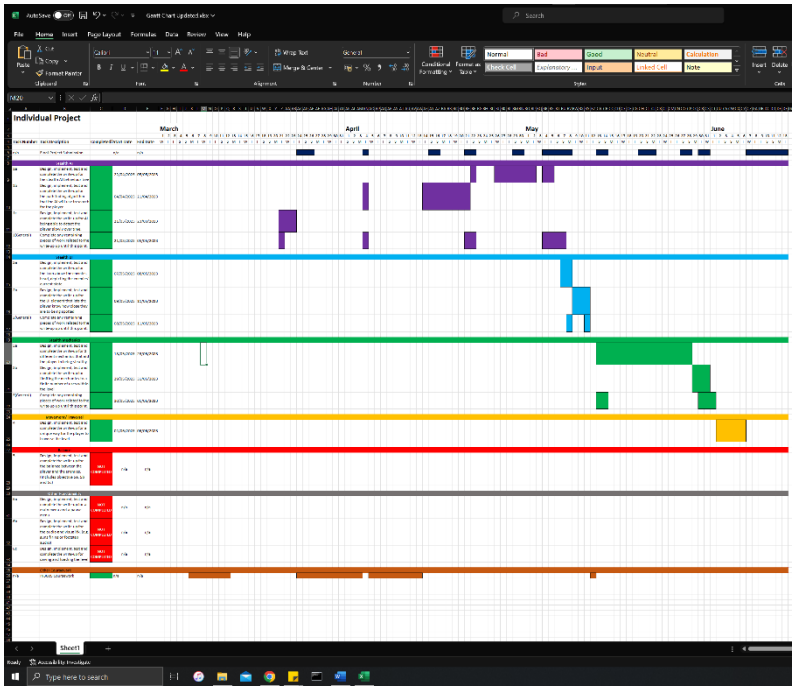
9.6. Appendix F: Gantt Chart

The Gantt chart was changed drastically during development due to delays out of my control before the project got underway.

Old Gantt Chart:

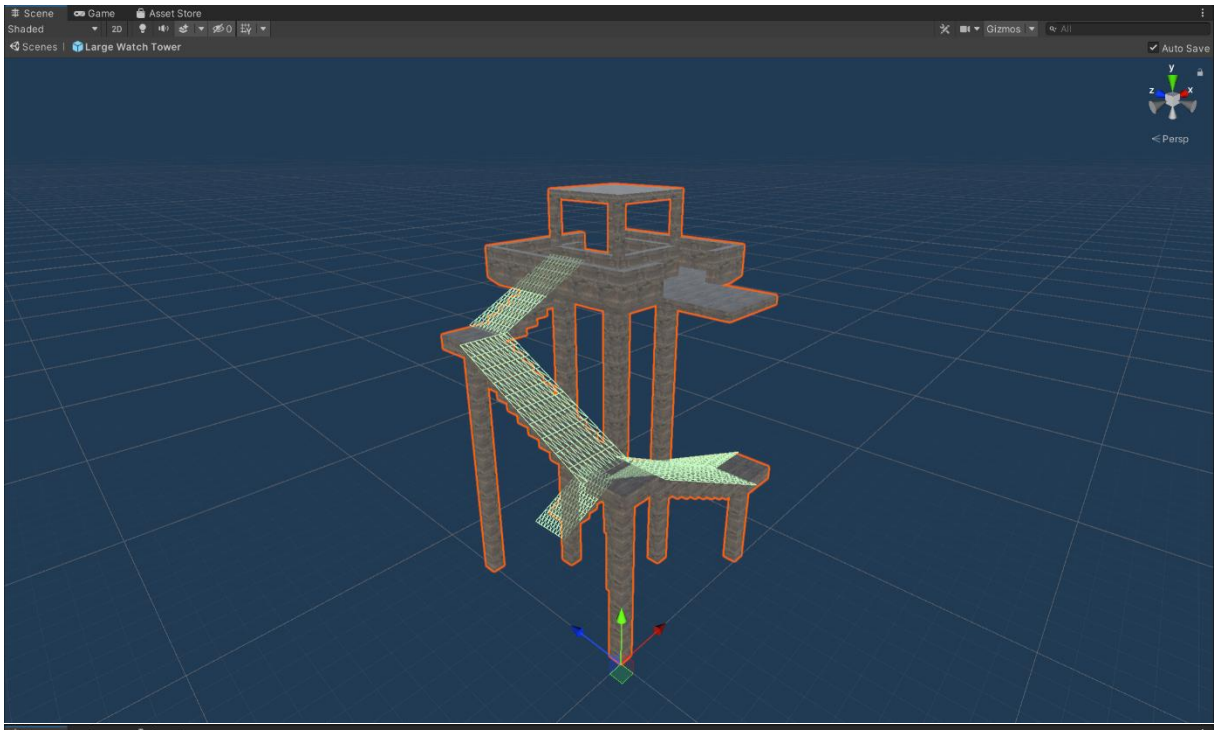


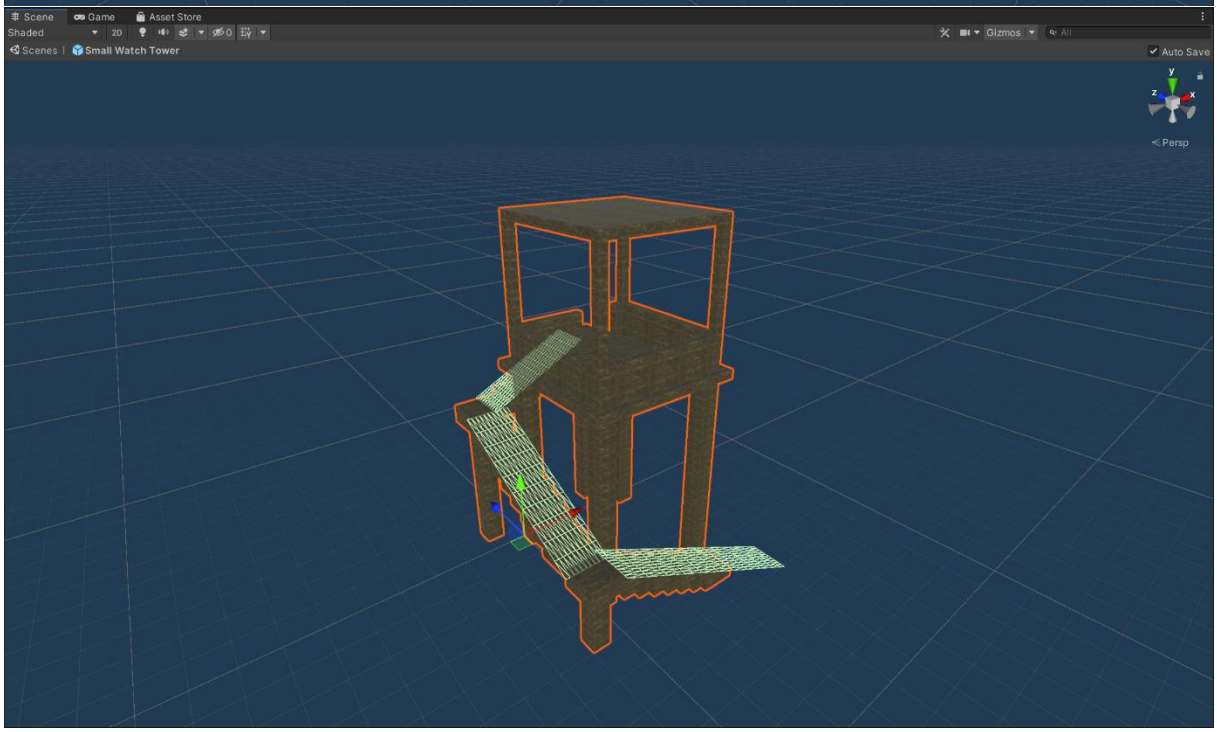
New Gantt Chart:

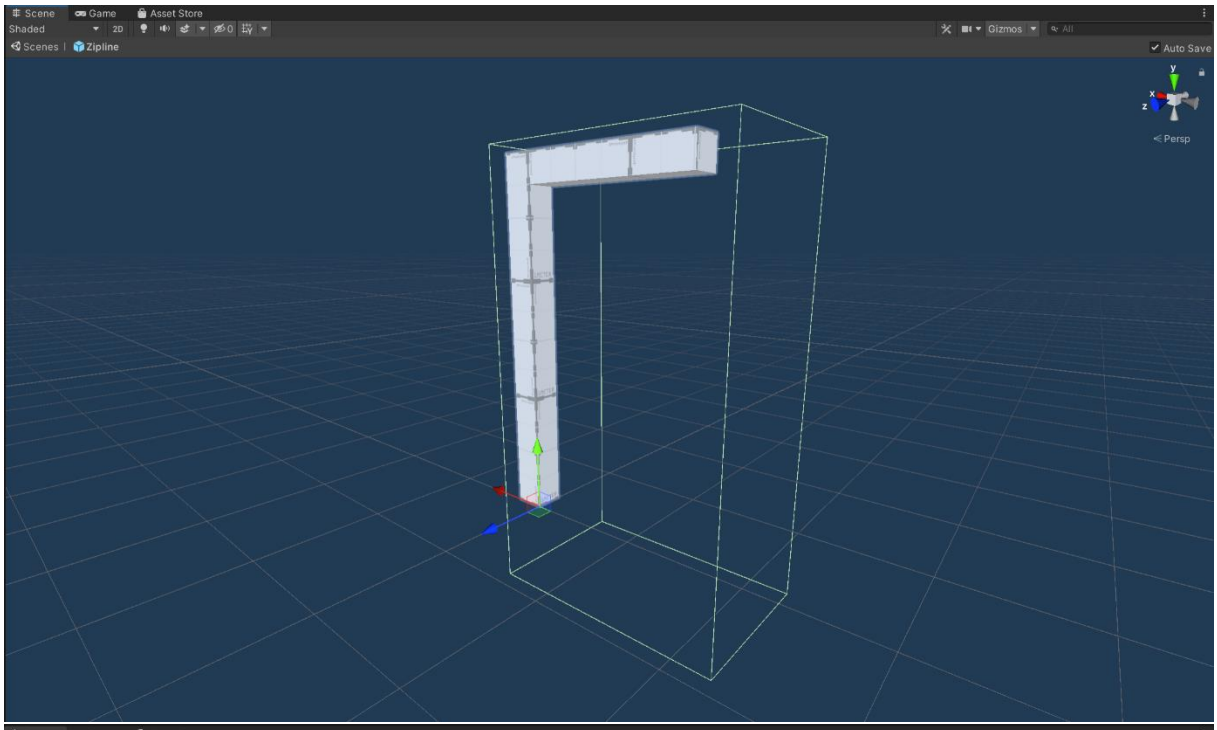


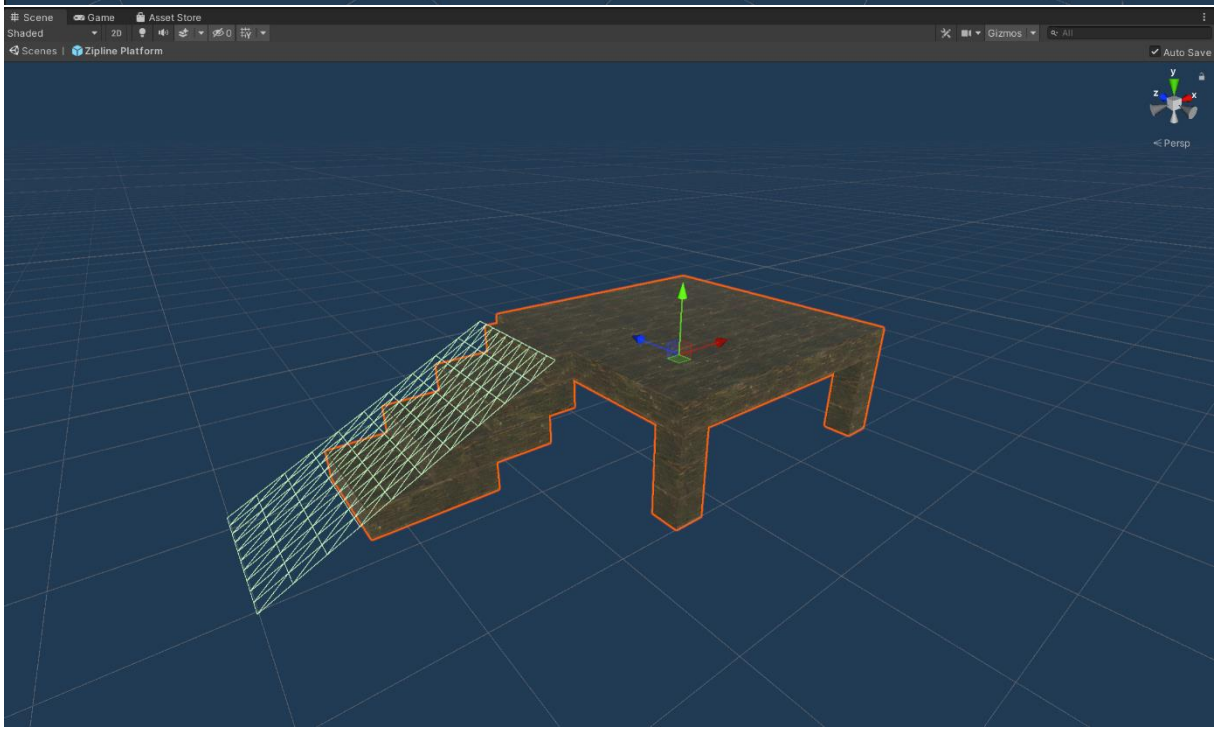
9.7. Appendix G: Probuilder Models

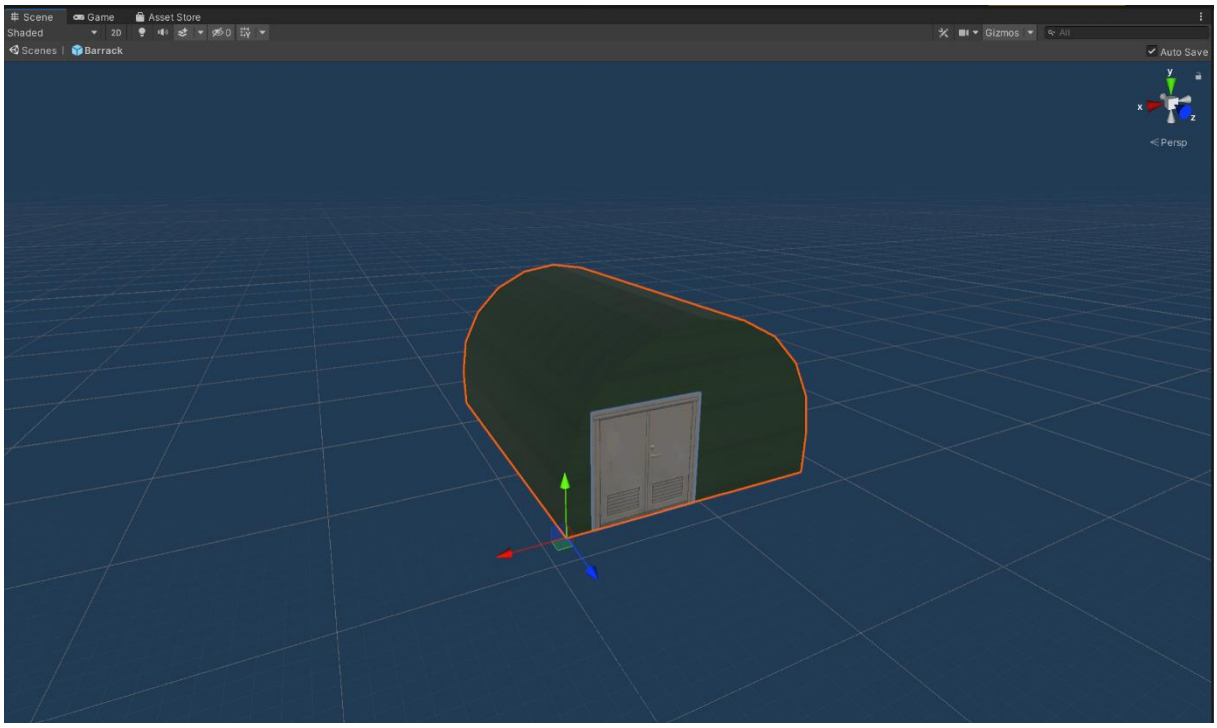
I had to make simple models using Unity`s ProBuilder tool of objects I could not find online. These were used to populate the level and are displayed here. Note that the texture on the models is not mine. They are re-used and are listed under Reuse Summary.



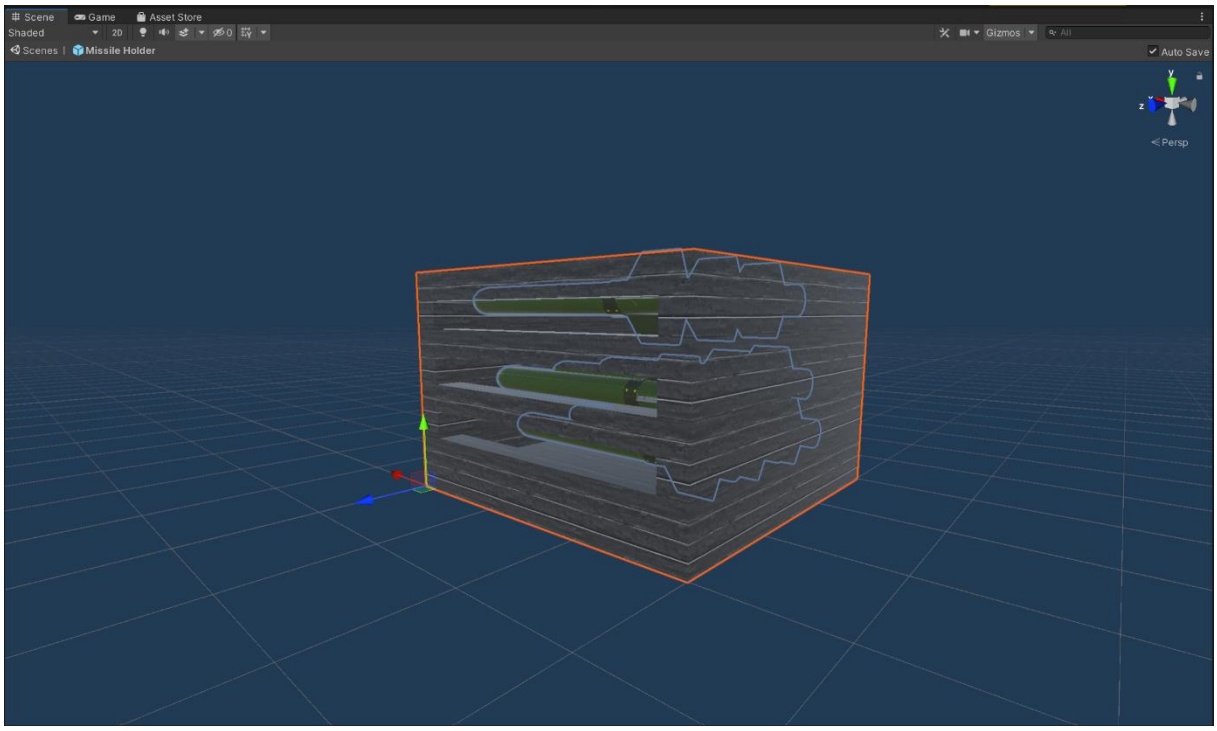



Scene
Game
Asset Store
Shaded
2D
Gizmos
Scenes | Zipline
Auto Save
Persp


Scene
Game
Asset Store
Shaded
2D
Gizmos
Scenes | Zipline Platform
Auto Save
Persp



Just to be clear, the door texture is re-used.



Just to be clear, the missiles are re-used.