Final Year Development Project Main Report

Jack Moorin P17190172

Introduction

**Project Background:**

As part of my ‘Development Project’ module coursework, I have developed the 3D ‘Escape the Room’ style puzzle game ‘*The Eagle Hotel*’, a sly reference to the song ‘Hotel California’ by the Eagles, *you may check out but you can never leave*, for desktop. In the game players become trapped inside the fictious hotel and must solve a series of puzzles by interacting with objects throughout the hotel in order to escape. Each room the player reaches contains a new puzzle that must be completed in order to move onto the following room. The puzzles are built into the hotel and the player must interact with different objects in order to complete them. The fuse box puzzle for example, is made up of five different fuse boxes attached to the wall in the first room. Completing all the puzzles will allow the player to leave the room.

As mentioned above, I have developed this game as a part of the coursework of my ‘Development Project’ module. I am completing this module as part of my third and final year of study at De Montfort University for my three-year ‘Computer Games Programming’ degree. For the module I had to develop a project of my choosing, to a professional standard and document its planned development during the initial planning stages as well as my continued meetings with my project supervisor during the project’s development and my own critical analysis of the project once it’s finished.

At the start of the academic year we were given a list of possible projects that we could undertake for this module and the names of the faculty members who would supervise us throughout the project’s development. Initially I had intended on picking to develop the medieval merchant trading simulator game in Visual Studio, however I later changed my mind and picked the option to develop a 3D puzzle game instead as I realised that I would have a lot more freedom throughout the project’s development as the title of the project was a lot more generic than others. ‘Jethro Shell’ was the supervisor that was initially listed for this project, however due to its large popularity Jethro was unavailable to supervise my project and recommended my current supervisor ,’Conor Fahy’.

**Project Motivation:**

Before my first meeting with Conor I used a notebook to brainstorm ideas for the game. The most prevalent theme found in my notes was the idea to design my game around the basis of an ‘Escape the Room’ puzzle game. This idea was born out of the fact that I had recently at the start of the project completed a ‘Nuclear Apocalypse’ themed ‘Escape Room’ for my younger sister’s birthday and had been hugely intrigued by how well the room was able to convey the room’s complex narrative about the nuclear war between the major world powers that had led to the apocalypse through not just the room’s aesthetic and the flavour information given by the Escape Room companies employees as they explained the room to us but also through the puzzles we needed to complete in order to proceed through the room. In the first room we had to work out how to use different pieces of equipment to build a radio which we could then use to converse with a ‘rescue team’ that was coming to extract us and takes us to a safe zoom and whom we could also ask for a limited number of clues on how to solve other puzzles. This method of continuing to immerse the player/s in a created fictional world by the mechanics they use to interact with it is also often used in video games as well to assist with immersion, typically being referred to as Narrative Game Mechanics. The only difference between the way in which the Escape Room and my planned game would use these is that the Nuclear Apocalypse Escape Room’s puzzles were created to enforce the chosen narrative, whereas with my game the puzzles were devised first and I later crafted the narrative to explain them.

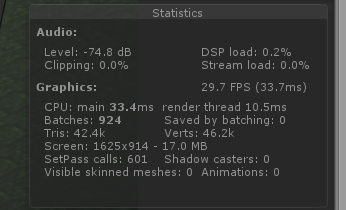
Being able to include a cohesive narrative in my game was itself also one of the main driving forces behind the ‘Escape Room’ idea, as when I had decided to create a puzzle game my initial worry was that players would quickly become bored with the game if they became stuck on a puzzle or just weren’t interested in puzzle games. I believed that by creating a thrilling narrative in which the player becomes trapped in an unknown environment and wants to escape, the game would be much more successful in captivating its audience.

**Aims and Objectives:**

After deciding to design the mechanics of my game around those of an ‘Escape Room’, the next step was to decide upon the game’s aesthetic. I decided to go for a spooky thriller theme as Conor and I agreed that if the player is playing slowly and cautiously because of a perceived, but non-existent, threat it would put players in the best state of mind to work out the game’s complex puzzles as they would be both thinking more deeply about what they are doing and not attempting to rush through the rooms. To that end I decided to base the game inside an old building, which later became *The Eagle Hotel*, that the player would have become trapped in at the start of the game. Continually developing the game’s environment to convey this sense of unease and thereby increase the player’s engagement with the game was one of my primary objectives.



Figure 1: The first room of the Eagle Hotel.

As well as developing the game’s aesthetic, another primary objective for this project is to create a series of complex puzzles that the player has to solve in order to escape the hotel. Each room should have one individual/set of puzzles that once the player’s completed allows them to progress into the next room and eventually leave the hotel. In order to work well with the game’s aesthetic, each of the puzzles must also be built into the game world as objects that the player can interact with. The first room’s puzzle for example is a series of pathfinding puzzle built around the idea of rewiring a set of fuse boxes. Similarly, there is a light puzzle that requires the player to turn on a series of lights by interacting with levers placed around the room. Every puzzle should also follow the design mantra ‘*Easy to learn, hard to master*’ in that the player’s interactions with and subsequently how they affect the puzzle should not be complicated however, actually completing a puzzle should be much harder and require the player to consider their actions.

My tertiary objective is to ensure that the game runs as efficiently as possible and that it works as intended. While this objective does seemingly ‘go without saying’ it is important however to make note of it as once the game is distributed it will be running on a myriad of different machines each with their own hardware specifications and the end users opinion of the game will be negatively impacted if the game runs slowly or they encounter any bugs in the system.

Figure 2: Unity’s Statistics panel, one of the things I used to measure the performance of my game

Main Body

**Interactable Objects:**

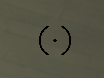
Before I could begin developing any of the game’s puzzles, I needed to program the player’s ability to interact with the various interactable objects in the game world. This was because one of the main requirements of the game is that the player solves each room’s puzzles by interacting with the objects in the game. During this initial phase of the projects development, I was still planning to develop the game for mobile as well as PC I decided the best way for the player to interact with objects would be for them to click on them as this would translate well into mobile where players can similarly tap on the objects on their screen. As when I began programming this I’d hidden the mouse cursor from view as part of getting the game’s camera to pan with the movement of the player’s mouse, I decided to add a static crosshair icon in the game’s GUI at the centre of the screen that the player would use as their cursor in game.

Figure 3: The crosshair icon.

One of the considerations I had to make with my design was because I knew that more and more objects would be added to the game throughout its development, one of the requirements of whatever system I developed was that it would need to include some way for the player to easily determine whether a game object is interactable or not. As I’d decided to base object interaction around clicking on objects with the players mouse, this fortunately meant that I had the option for interactable objects to react somehow when hovered over. This idea was born out of the fact that most modern software programs also do this to help indicate to users that they’ve hovered over a button on their screen, either by making them change in colour, highlighting them, making them bigger or even with audio cues. I liked the idea of having the objects change in colour because it would be a particularly obvious visual change that would definitely catch player’s eyes however, since I also knew that the objects would be made themselves be made up of lots of different colours and I also didn’t want it to become difficult to tell what the object was I decided that instead of setting the entire object to be one single colour, I would instead slightly lighten all of its materials. That way the player still notices the change and can also still tell what the object is. I also decided to include a sound effect for when the player first hovers over the object just to make it even more noticeable.

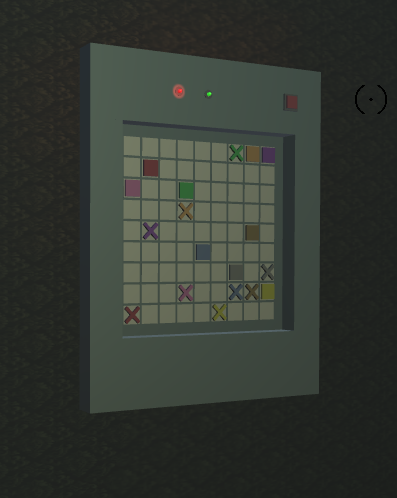
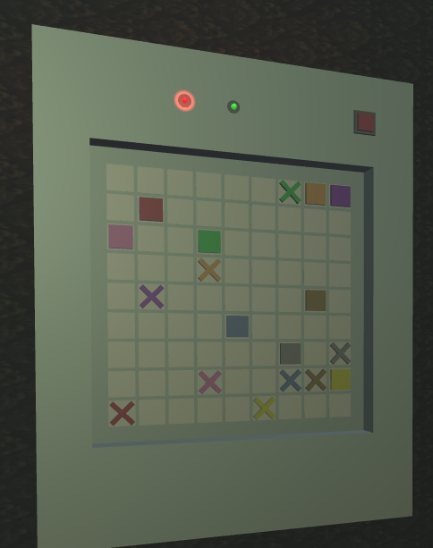


Figure 5: The fifth fuse box puzzle while it is being hovered on.

Figure 4: The fifth fuse box puzzle while it’s not being hovered on.

My initial version of this system used a raycast from the centre of the screen that returns the object it collides which I could then check if it is a child of an interactable parent object. For example, if it returned a fuse boxes grid cell’s transform, it would find that it’s a child of a fuse box puzzle and would that the fuse box puzzle must therefore be being hovered over. If the mouse button was being clicked, it would then call the object’s ‘On Click’ associated function, which in this case would make the player focus on the fuse box, and if not it would instead call its ‘On Hover’ function, which for the fuse box puzzles lights them up and plays the hovered sound effect. Next it calls the ‘On Not Hover’ functions for the interactable objects that aren’t being hovered over. For the fuse boxes this will darken it down to its original colour.

This initial version of the system was fairly easy to implement and did work as intended however, over time it became obvious that it didn’t scale very well as it was using Unity’s expensive ‘GetComponent’ function for all of the interactable objects to call either ‘On Hover’, ‘On Not Hover’ or ‘On Click’, every frame the player’s camera moved. To stop the functions being called so often I used an instance of the enum for indexing the interactable objects array to hold the object that’s currently being hovered over. Whenever the raycast happens now it checks if the object its currently hovering over is the same as the last time the functions were called and will only call them if the hovered object has changed, thus limiting the number of function calls to only when it’s necessary.

Another design consideration I made was that fact that I knew the player would need to be unable to move whilst they were focusing on an object, so that they can focus on and potentially use the movement keys to interact with the focused object. For example, while the player is completing one of the fuse box puzzles they shouldn’t be able to move using the usual movement keys as it will interrupt the player’s immersion. To stop this from happening, I decided to build the players current movement controls into a state within a Finite State Machine, that I would use to easily switch between different control methods. I copied the player’s current movement controls into a state called ‘Player Move’ and I created a second state, ‘Player Focused’ that would be used to control the player’s input while they are focusing on an object.

A consideration I made about the player’s ‘Player Focused’ state was that even though I did not have any future plans for objects that would use mouse input, I decided to make the mouse cursor that’s usually at the centre of the screen, instead follow the player’s mouse so that they would be able to click on parts of the object if necessary. Adding this did later allow me to add a reset button to the top right of each of the fuse boxes that the player could click on both while they were and weren’t focused on the puzzle. Thankfully as this method of more intricate object interaction does not require the player to know what things they can and cannot click on, the interaction raycast only needs to be performed when the mouse button is clicked.

**Fuse Box Puzzle:**

The fuse boxes were the first idea I had for a puzzle based around interaction with real world objects. They tasked the player to connect all of the fuse boxes wires on the grid to their relevant end nodes without any of them crossing over. I got the idea for the puzzle from the app ‘Flow Free’, that was recommended to me on the Google Play store and realised that I could recreate the puzzle in ‘The Eagle Hotel’ by basing on the premise of electrical wiring. I also decided that a single puzzle wouldn’t provide a significant challenge to the player, so I instead opted to have five increasingly difficult fuse box puzzles, that the player must solve in order to leave the room.



Figure 6: The fuse box puzzles before their grids are added.

After I’d created the five fuse boxes within the game, as seen in the above figure, I decided to create a prefab object of a single white square that each fuse box would be able to replicate multiple times in order to procedurally create the grid the puzzle would be on. The reason I chose to have the fuse boxes create their grid procedurally is because I knew that it would ultimately be much faster than if I had to place them all manually and also allow for easy editing if I decided to make any changes such as altering the spacing between each cell. Next I added a script to the cell prefab object, that allowed me to define a wire segment to be attached to it of a specified type (wire start node, wire end node, wire straight piece and wire corner piece) and colour. Once I’d decided on a puzzle for each fuse box and subsequently programmed each to create their grid and attach to correct start and end nodes to the relevant cells to resemble that puzzle, I then needed to get them to work with user input.

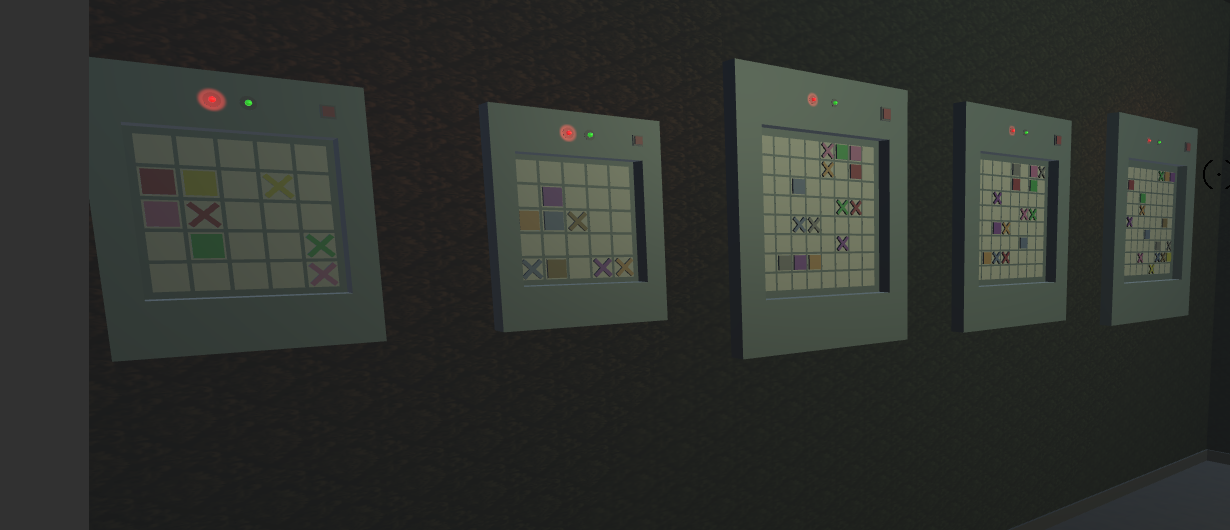


Figure 7: The fuse box puzzles after their grids have been added.

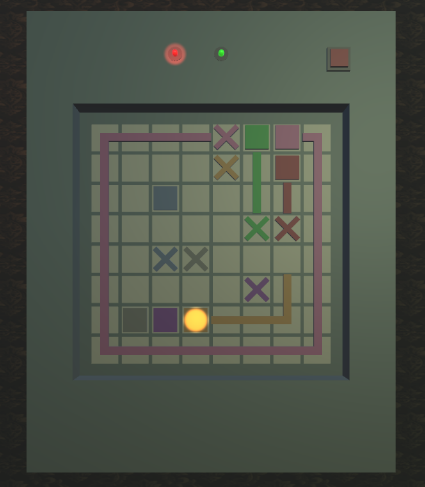
The first consideration I made for how user input was going to work was that the simplest method to have the users affect the puzzle was for them to use their computer’s arrow keys to move the wires from one grid cell to another, as this four directional movement input would fit in well with the grid and there wouldn’t be any issues with the player trying to move and accidentally affecting the puzzle as the players movement is linked to the WASD keys. In conjunction with this. I also made another consideration that I could both add another layer of difficulty to the puzzles and make the control scheme simpler if I made the fuse boxes cycle through controlling each of their wires in a given sequence. This would mean that I could make players have to think more deeply about how they should connect each wire if I made them connect the longest ones first thus making them have to also consider how they’re going to connect the shorter ones later (reviews and general research about the ‘Flow Free’ app and other puzzles like it showed that this type of forward thinking was one of the aspects of the game lots of players struggled with) and also meant that I wouldn’t need to develop another control to allow the player to choose which wire they’re controlling.

Figure 8: A partially completed fuse box puzzle where the orange wire is currently being controlled.

The first requirement of getting user input to work with the puzzle was getting the basic mechanics of the puzzle working. The wires needed to be able to be moved around the grid and once they’ve been connected to their correct end node, have them switch control over to the next wire or, if all the wires have then been connected, mark the puzzle as complete. First I gave each puzzle an array that held the grid indexes of each start node, another array that held each start nodes colours, an integer that counted the number of wires that had been completed, an current node index that held the grid square which the wire was currently in and a value from the grid cells colour enum that held the colour of the current wire. Then I got the wire from the first start node to move across the grid using arrow key input from the user. The first two problems I had to deal with here were the wire being able to ‘overwrite’ the wires in other cells by moving into their cells and it would also cause an error whenever I attempted to move the wire outside of the grid. Fortunately these were easily fixable however, by simply adding in a check to make sure that the grid cell its attempting to move into is empty and another that made sure that the index of the new cell was always within the boundaries of the grid. Once I’d gotten this working I then added in another check for if the new grid cell contains the wires end node and would then if the wire was not the final one in the array, set the current index of the wire to the start node of the next coloured wire, set its current colour to match it and turn on its light to show the player which one they’re now controlling. If it was the final wire however, the player would then simply stop focusing on the box and the puzzle would then be completed.

After I’d gotten the puzzle’s mechanics working, the next requirement of the puzzle, and what proved to be most difficult, was getting the wire segments on each of the grid cells to arrange in such a way that together they all looked like one single wire. At this stage in the development, when the wire moved into a new cell, it would simply leave a new straight section of the relevant colour in the cell, no matter which direction the wire is moving in. Fortunately, getting these straight sections to have a certain rotation when placed, so that they were always facing in the correct direction was relatively easy however, it did then led to a more complex problem. Whenever the wire changed direction the ends of the straight pieces weren’t connected to each other. To solve this problem, I decided to create a corner piece that would replace the wire segment in the previous cell whenever the wires direction changes. As each of the straight pieces now had a set rotation based on the direction they’re moving in, I could easily compare the direction of the wire in the current cell against the player’s input and thereby tell if a corner piece was needed. Implementing the corners replacing the straight pieces was easy however, what proved to be considerably difficult was figuring out the logic for what the corner’s rotation should be, so that it matches up with the wires. To make the matter simpler, I decided to refer to each of the four possible corners by always referring to their vertical movement first. So for example, a corner that moves up and right would be referred to as that even if it was being used to move the wire left and down. This meant that when the player went from moving horizontally to vertically I would need to reverse the type of corner I wanted to make in my head but, made it a lot easier to refer to each of them. Once I had this logic and I knew the necessary rotation of the corner piece to make each corner I was able to tell which corner should be created a lot more easily.



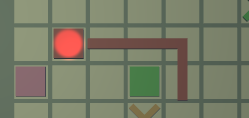
Figure 9b: An Up and Right corner

Figure 9d: A Down and Left corner

Figure 9c: A Down and Right corner

Figure 9a: An Up and Left corner

The final requirement for the mechanics of the fuse box puzzles was for the player to be able to backtrack over wire segments they’ve created, in case they’ve made a mistake. As I could now easily tell the direction of each straight or corner wire segment from their rotation as I could then tell that if the colour of the wire segment is the same as the current wire’s colour and it is moving into the current wire’s cell it must therefore have been placed just before the current piece and the player should therefore be able to go back over it. If the grid cell contains either a straight piece or the current wire colours start node, then the puzzle will simply delete the wire segment on the current grid cell and move the current wires cell back to that cell. If it contains a corner piece and the player is moving the wire horizontally, then the puzzle deletes the wire in the current cell and replaces the corner in the previous cell with a straight piece facing in the direction of the corner’s vertical movement. If the player is moving backwards vertically however, then the puzzle will replace the corner with the inverse of the corner’s horizontal movement. For example, if the player moved backwards down onto a down and right corner (also known as a left and up corner) it would be replaced with a left straight piece.



Figures 10a and 10b: A red wire before and after the player went back over a corner. As the player moved upwards onto an Up and Left corner (remember we go by the vertical movement first) it was replaced with the inverse of its horizontal movement; a Right straight piece.

Refer to Figure 14 in appendix A for a detailed UML diagram of the fuse box puzzle.

**Camera Panning:**

Once the player has finished all of the fuse box puzzles a door to the second room will open and allow the player to leave however, as the door was both behind the player and obscured from their position because it was down a corridor the players I got to test the game mentioned that if I had not been there to tell them to progress to the next room, they would have been troubled to figure out what to do next. My first idea to combat this problem was simply to move the door to be next to the final fuse box puzzle however, after musing on this idea for a while I decided that by placing the door right next to where the player will finish I would just be giving myself the opposite of my current problem as it would appear that I am ‘hand-holding’ the player through the game; which would severely counter the games spooky aesthetic. I eventually decided to put the problem on the backburner while I focused on developing the second room as I believed it to be the more pressing problem and it was while I was working on the second room that I decided on a feature I could implement that would both show the player where to go without making the game seem too simple and be reusable throughout the rest of the game; a camera panning function that I would use to move the player’s camera to show the door opening before snapping back to the player.

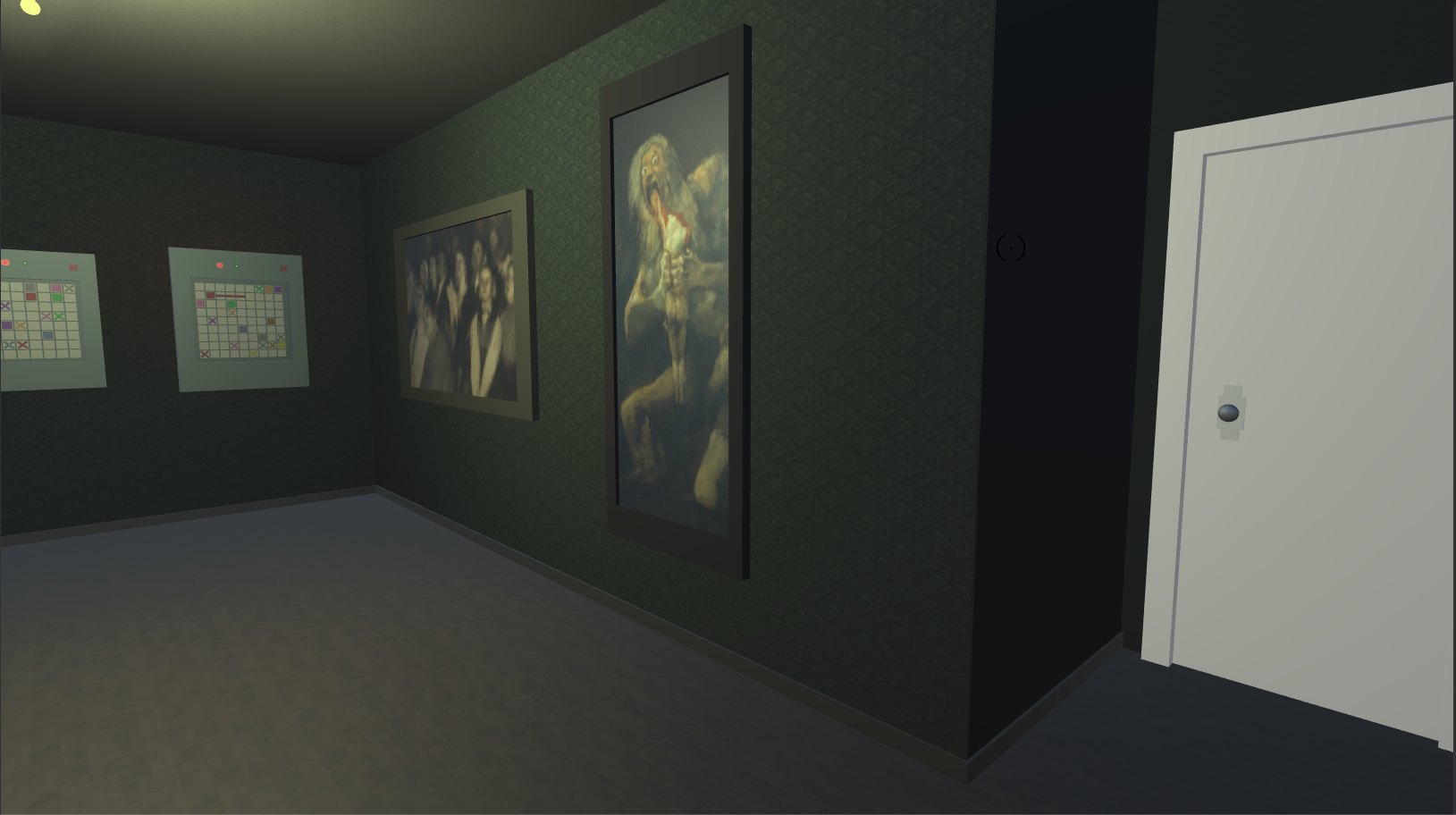


Figure 11: The door to the second room is hidden in a small corridor away from the view of players completing the fuse box puzzles.

Since I was already using a finite state machine to control the player’s current control scheme, I decided it would make sense to build the camera snap function into this as well. I gave it two new states; a ‘Camera Snap’ state and a ‘Camera Wait’ state. The ‘Camera Snap’ state would be used when the camera is panning, or ‘snapping, to another position and rotation while the ‘Camera Wait’ state is used once the camera has reached its target position and rotation to keep the camera in place for a given number of frames before using the ‘Camera Snap’ state to return the camera to its original position.

Getting the camera to move from one position to another using the ‘Camera Snap’ state was easily done. When the camera snap was first initialised, I would simply divide the difference between each of the camera’s current x, y and z value and its target values by the number of frames I wanted the camera to spend panning. Then simply in the ‘Camera Snap’ state apply these values to the camera’s position to get it to move to the target position. I attempted to use the same method for the camera’s rotation however it didn’t work properly due to various issues that came up because unity would always give rotation values between the range zero to three-hundred and sixty range which would screw up a lot of calculations when a value went outside of this range and was instantly changed so as to still be in the range. I tried a lot of different methods to control this change in the value however, after a lot of googling around the problem I found that Unity actually already had a method for dealing with this; ‘Rotate Towards’, that will rotate one quaternion towards another by a given step value every frame. By using the ‘Rotate Towards’ function we are also fortunately avoiding one of the other biggest problems that can occurs when trying to rotate something towards a target rotation over time using a calculated step; half of the time the object will rotate the long way around because since all rotation values must be in the range zero to three-hundred and sixty, calculating the step by dividing the difference between the start and the target by the number of frames, like with the position, will never give a step value that causes the object to rotate over the zero/three-hundred and sixty point. ‘Rotate Towards’ however, will always rotate in the direction that is closest to the target and as an extra catch will never rotate ‘past’ the target point if it is closer than the step value as it will just set the rotation to the target value.

Calculating the necessary step value to move the rotation from the start to the target seemed to provide the same problems as before, until some poking around on the Unity forums showed me how to calculate it. First the camera’s original rotation is multiplied by the inverse of the target rotation to get a quaternion difference between them, get the magnitude of its Euler angles and then divide by the length of the snap in frames. After I added both of these to the ‘Camera Snap’ state, I added a counter that would count down each frame from the number of frames the snap is supposed to last and when it depletes set the player’s current state to the ‘Camera Wait’ state so the camera will pause for the specified number of frames to give the player time to look at what the camera’s panned to.

To set up each camera snap, I created a function; ‘Initialize Camera Snap’ that takes in the necessary information about the snap; the target position, the target rotation, the length of the snap in frames, the length of the pause in frames and the state to switch the player into once the camera panning is finished. It saves the camera’s current position for the later camera snap back to its original position and calculates the necessary step every frame for the camera’s position and rotation to get it to its target. The reason it takes in a player state to go into once the camera has finished panning is to control the camera snapping back later. When the snap is first initialized we pass in the ‘Camera Wait’ state so that once the camera’s reached its target position it will begin waiting however, once the ‘Camera Wait’ state is over it will call the ‘Initialize Camera Snap’ function again and initialize a camera snap back to the camera’s original position and rotation and pass in the ‘Player Move’ state, the player’s default state, as the state to switch to which will set the player back to normal.

**Light/Lever Puzzle:**

Originally the room two puzzle was supposed to be based around a chessboard however, as I mentioned in the first deliverable documents, due to various constraints, more on that later, this was later replaced with the lever puzzle. The player must turn on all five of the wall lights by using a series of levers in the room. Each lever affects a different set of lights and will always switch them to the opposite of their current state.

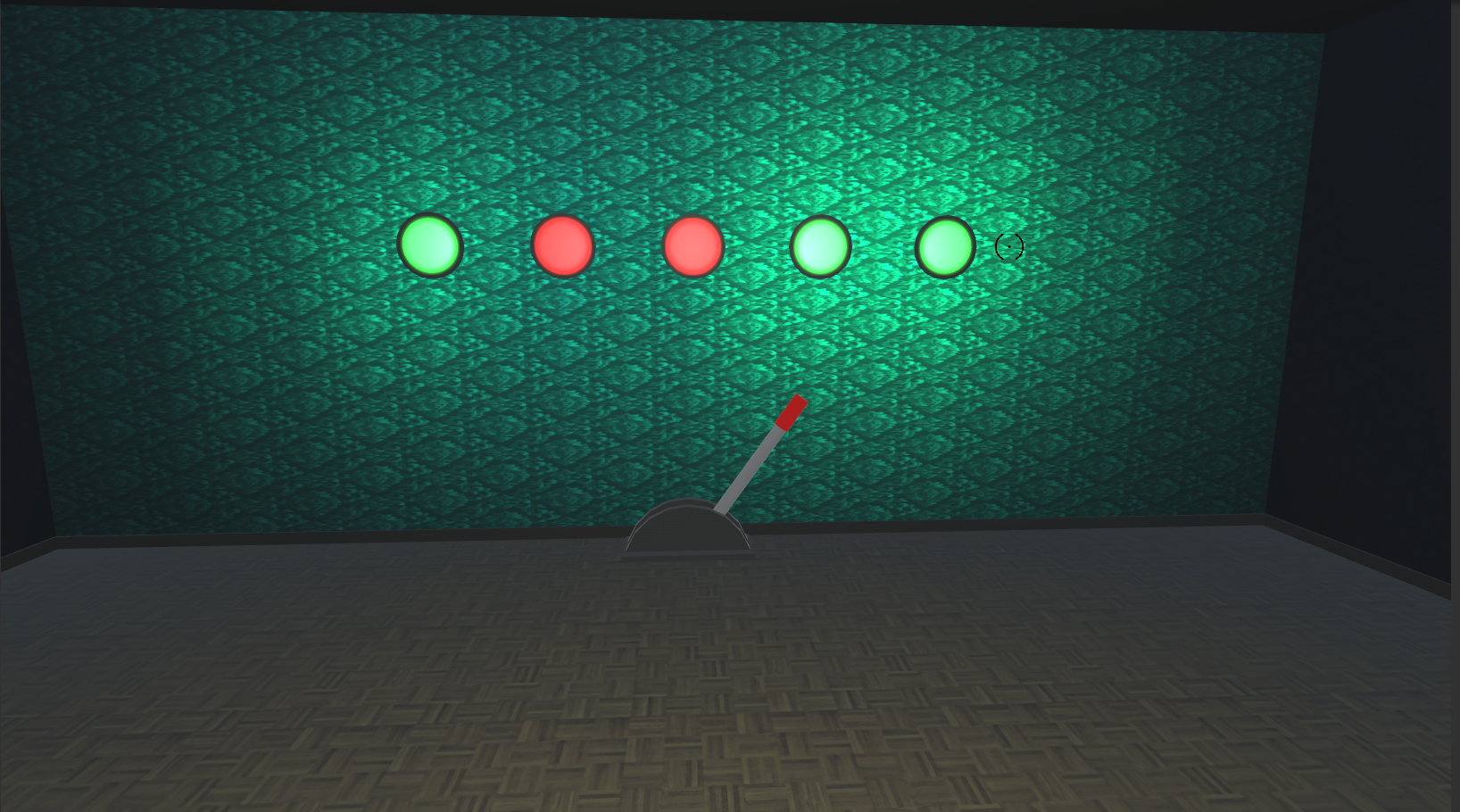


Figure 12: Screenshot of the wall in the second room with the five puzzle lights and one of the puzzle levers. The colour green means the light is on, while the colour red means its off. To complete the puzzle the player must get every light to green.

One of the first design considerations I made when I began developing the puzzle was the decision to make the game objects for room two out of a combination of the default meshes provided by Unity. The levers for example, are made up of two thin cylinders and a flattened cube that act as either sides and the bottom of the lever’s base respectively and two long but thin cylinders that act as the lever and its red handle respectively. The reason I chose to do this instead of developing a simpler mesh with a lot less polygons, in either Blender or Maya that would have less of an impact on the game’s performance is because I decided that due to my own fairly limited familiarity with both systems and the limited development time left that it would be better to get a working version of the room and the puzzle up and running that showed off the puzzle’s mechanics which I could potentially improve upon later.

The player’s aim is to switch on all of the five wall lights, that would then allow the player to move on the next room however, as this is unfortunately the last puzzle in the prototype of the game there isn’t a door to another room yet. Whilst I was developing the puzzle, I did decide that if I ever was going to develop a third room I wouldn’t use another physical door like the ones connecting the first and second puzzle rooms but instead when the player correctly switches on all of the wall lights, the section of the wall with the lights will slowly and dramatically lift up into the ceiling revealing a secret corridor behind it that leads to the third room.

One of the problems I faced when developing this puzzle was because as I spent less time designing it than the fuse boxes or the planned chess puzzle, I hadn’t thought of a complicated structure for how the puzzle would work. Currently all the lights can be turned on by simply clicking on the left and right levers. Fortunately however, as this is just a prototype of the finished game and I’ve got its mechanics working completely, in a future iteration of the game I could easily set the levers to affect different sets of lights or even potentially add more. Also I did manage to work in one of my considerations when designing the puzzle into it, however. The middle lever is completely unnecessary to solve the puzzle and serves as a misdirection to the player. This is so that if the player tries to rush through the puzzle and turn all the lights on by simply pulling every lever they’ll actually just be making completing it harder for themselves thereby further reinforcing the very first design consideration I had about the game that they should be playing through the game slowly and won’t be able to solve the puzzles without thinking them through properly.

Refer to Figure 15 in appendix A for a detailed UML diagram of the fuse box puzzle.

**Decision to Replace Room Two’s Planned Chess Puzzle with Lever Puzzle:**

One of the main problems I faced during the development of the game was the limited development time, and as I mentioned before I had previously intended to develop another pathfinding-based puzzle in the second room based around the idea of moving chess pieces on a board that light up the squares they move over and the player has to light up the entire board however, I ended up developing the above lever puzzle in the second room instead. This was primarily due to time constraints and also wanting to avoid too much similarity with the fuse box puzzle and issues with the 3D modelling.



Figure 13: A rosewood chessboard, that I wanted to base the design for the chessboard that would be used in the puzzle on. In the puzzle there would be multiple pieces on the board. Clicking on one and then the cell where you want it to go would cause it to move there and light up every cell it moves over. If a piece moved over a cell that was already lit up it would turn off. Some of the cells on the board would be darkened out to show that they aren’t traversable and not part of the puzzle. The harder the puzzle the more space the puzzle would take up and the less cells would be darkened out. If the player could light up every square on the board then they’ve completed it.

The first reason I chose to reconsider developing the planned chess puzzle was because, while I did have a fairly good amount of time left to develop the puzzle and get it working, I did not believe that I would get it to a professional standard or even a standard that I would be happy with presenting to my supervisor as part of my best work. This was ultimately due to the large complexity of the desired puzzle. For example, one of my design considerations was to have the board on the table spin one-hundred and eighty degrees in the table to reveal another puzzle on an identical board underneath, whenever the puzzle was reset or one was completed. I was also having trouble deciding how I wanted the puzzles control scheme to work. I had previously decided that when the player clicked on a chess piece the squares on the board where the piece could be moved to would light up which the player could then click on to show where they wanted it to move however, I later realised that this would method wouldn’t work with the Knight pieces in some of the puzzles because, they can move two different ways into a traversable cell; either horizontally then vertically or vertically and then horizontally.

The second reason I chose not to develop the chess puzzle was because I was worried about it being too similar to the fuse box puzzle because both are pathfinding based games except one is based around moving wires across a board and the other is about moving chess pieces across a board in a certain way in a certain order to get the board to completely light up and I didn’t want the player to become frustrated from having to complete multiple similar puzzles too quickly. I did decide, while I was working on the lever puzzle however, that hypothetically if I had more time to work on the game, then I would put it in either the third or fourth room as by then the player should have had enough of a break from a pathfinding puzzle that they wouldn’t mind doing another similar one.

Finally the third reason that I decided not to develop the planned chess puzzle was because I had concerns about both my own ability to model or find premade assets for the chessboard and the individual chess pieces that I was going to use; knights, rooks, pawns and bishops both in black and in white. Previously I had made use of the multiple default meshes provided by Unity to make more complex models by combining them together however, given the overall complexity of each of the individual chess pieces I knew that I wouldn’t be able to make them this way and if I did want to use realistic looking pieces in the game, then I would have to get them from Unity’s asset store. As the game is only intended to be a prototype I could have feasibly created a series of simple varied models that I could explain to the player’s trying the game, were stand ins for more detailed versions that more closely resembled the finished pieces which would be used in the finished game however, I felt that even given though the game was only intended to be a prototype, using lower poly stand in models would make lots of player look negatively on both the quality of the game and its development.

**Decision to Cancel Planned Development for the Mobile Platform:**

Another aspect of the prototypes intended development that I had to cut was porting the game to mobile devices. As I mentioned earlier and indicated in my first deliverable documents, I had originally intended to develop to develop ‘*The Eagle Hotel*’ for mobile devices as well as PC because of their general simplicity which works well with less powerful mobile devices and also the fact that they’re so incredibly portable most people often have one on them at all times. This itself works well with the fact that puzzle game players often daydream about puzzles they were unable to solve and can easily start the game up again once they’ve realised the solution.

Unfortunately, I did ultimately make the decision to not develop for the mobile platform because after careful consideration on how to best spend the short time I had to develop the prototype of the game, I decided it would be best spent to continue developing the game to a professional standard whilst continuing to make considerations throughout its development that would make developing a possible mobile port easier.

The most important of these considerations was developing the game’s object interaction system around the basis of clicking on objects using the player’s mouse as this particular kind of input translate very well into the very similar touchscreen input from a mobile smart phone. This is not only due to the fact that physically the inputs of clicking on and tapping on a screen are similar enough that the interaction system itself would not need to be changed much to work for a mobile phone port but also because since the player would not be able to hover over any of the in game objects, the system would only need to perform the raycast when the player actually taps o the screen, which unfortunately means that I would need to find another way to single out interactable objects to the player however, the interactable object system would only need to perform the raycast to check for the interactable objects when the tap happens as opposed to what the PC version does currently which is perform this raycast whenever the player or the camera moves.

**Project Development Lifecyle:**

Throughout the ‘*Eagle Hotel*’s development I used the Agile Development Methodology to manage my time and workload. Using a software development methodology when working on a piece of software is important because it allows the project team’s manager, in this case me, to split the overall lifetime of the project into different phases which helps the team members, just me in this case, prioritise their time and become more efficient. Agile Development specifically, refers to the development practice of a team repeatedly iterating on the finished product, making small increments to its overall functionality.

I opted to use Agile Development for the ‘*Eagle Hotel*’s development because due to the project’s limited development time, making small and consistent increments to its functionality means that whenever the development reaches a deadline, there will always be a recent working build of the game that I can show to Conor at our periodic meetings. Even if a planned aspect of the game hasn’t quite been completed yet, I can still show him what I’ve done towards completing it so far so he knows that I have been making use of the time between meetings.

Another reason why I chose to opt for Agile Development is because it allows to me to prototype new elements of the game quickly by developing their core functionality first and iteratively improving on and polishing it later, whereas with a more linear development method such as Waterfall, the new mechanic would take longer to implement and test because I would need to fully design it before testing whether it’s actually fun to play. Using Agile Development means that I can quickly play test every major new mechanic that’s being added to the game without having to fully develop it first and make quick decisions about whether it’s good enough to stay in the game and continue developing it or scrap it and try something else, allowing me to make better use of the projects limited development time.

Critical Evaluation and Conclusions

**Product Evaluation:**

Overall, I believe that while my project has changed a little from my original design due to time and other constraints, the game is at its core, very much still the one I planned to make because it retains the essence of its design. This is because even though I did not manage to implement the chess puzzle I had planned, I still managed to implement the fuse box puzzle to a professional standard, create the desired spooky and eyrie aesthetic throughout the game world and develop a completely new interactive puzzle for the game’s second room to replace the chess puzzle. Furthermore, I believe that as a prototype for the intended finished version of the game, what I have developed shows considerable promise both due to the professional standard to which it’s developed as well as the challenging nature of its puzzles. Due to the nature of the game’s linear progression through a series of contained rooms, when presenting the prototype to someone they would find it easy to picture the finished version of the game as it would simply have more rooms containing more puzzles similar to those in the prototype.

The best part of the protype in my own opinion is the fuse box puzzle in the first room. My reasoning for this is that developing it and getting it to work how I wanted was one of the biggest tasks I undertook as part of the game’s development and I’m still very proud of the fact that it works to such a professional standard. As I mentioned in the main body of the report, I had based the design for the puzzle off of the puzzle app ‘Flow Free’ and even though in the app the player can move back over wires and create corners by changing direction and in my level two functional requirements in the first deliverable I had explicitly mentioned that I intended to implement both of these features I believed that I either wouldn’t be able or considerably struggle to get them both working fully as intended. Fortunately, after I decided to always start referring to each of the four possible corners by their vertical movement first it became easy to understand which corners should be appearing in the wire’s previously vacated grid cell and this logic also later helped me when programming going back over the wires as I was able to easily reverse my previous logic to figure out which wire I should be replacing the corner with.

**Evaluation of my Approach:**

In terms of my own approach to the development of the ‘*Eagle Hotel*’, I believe that opting to make use of the Agile Development Methodology was Definity the best decision for the game. Given both its limited development time and the fact that I was also busy with university work for my other subjects for the majority of that time, making small and incremental improvements to the ‘*Eagle Hotel*’s mechanics and overall functionality was the option the definitely fit best around my schedule and the repeated prototyping meant that if I got caught out and wasn’t able to completely finish my planned amount of work in-between my bi-weekly meetings with Conor, I would still have a working prototype game that I could demonstrate to him to show him the progress I did manage to make.

Another aspect of Agile Development that I used in my approach to the game’s development was the linearly sequential nature of the improvements I made to the game over time because, it matched very well with the linear and sequential nature of the game. As I mentioned before, I unfortunately end up deciding that there wasn’t enough development time left to create a working version of the planned chess puzzle for the second room so I instead chose to create a new simpler puzzle for that room instead. As I was using the Agile Development methodology this fortunately was as simple as just developing the light puzzle instead. If I had been using say the Waterfall Methodology instead, not only would I have already spent a considerable amount of time designing the puzzle at the start of the game’s development. This would have taken time away from the rest of the game’s development and meant that when I decided I did not have enough development time left to create it that the time I’d spent designing it would have been wasted.

**Evaluation of Tools Used:**

The Unity game engine was the primary tool used throughout the ‘*Eagle Hotel*’s development. I used this particular game engine both because of my own familiarity with it, as I had used it before in previous game design modules to develop other 3D games, and also because thanks to that familiarity I knew how powerful the engine is and it was therefore the best choice of engine for me to use. Unity also works very well with cross-platform development, allowing its users to port games to different devices by simply choosing from a list the device they want to build the game for.

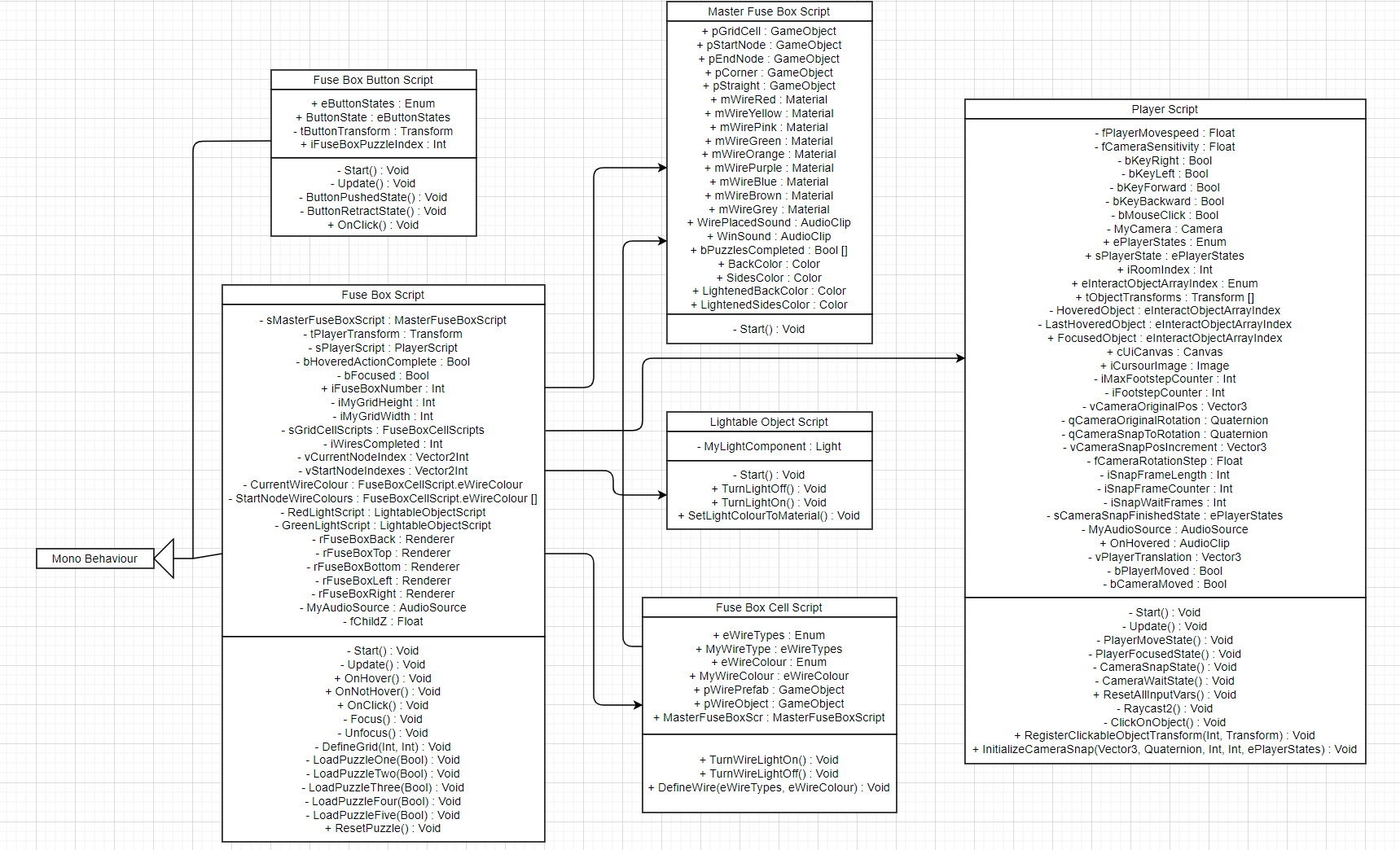
In tandem with the Unity game engine, I also used the C# programming language to implement the game’s mechanics. The reason I used the C# language as opposed to any of the other available languages is, C# is the default language used by Unity and is supported by it natively, which is why it’s the most commonly used language by Unity users by far. Using the most commonly used language also means that there will be lots of help available on the Unity forums and other websites, as other people using the language will have had similar problems to me.

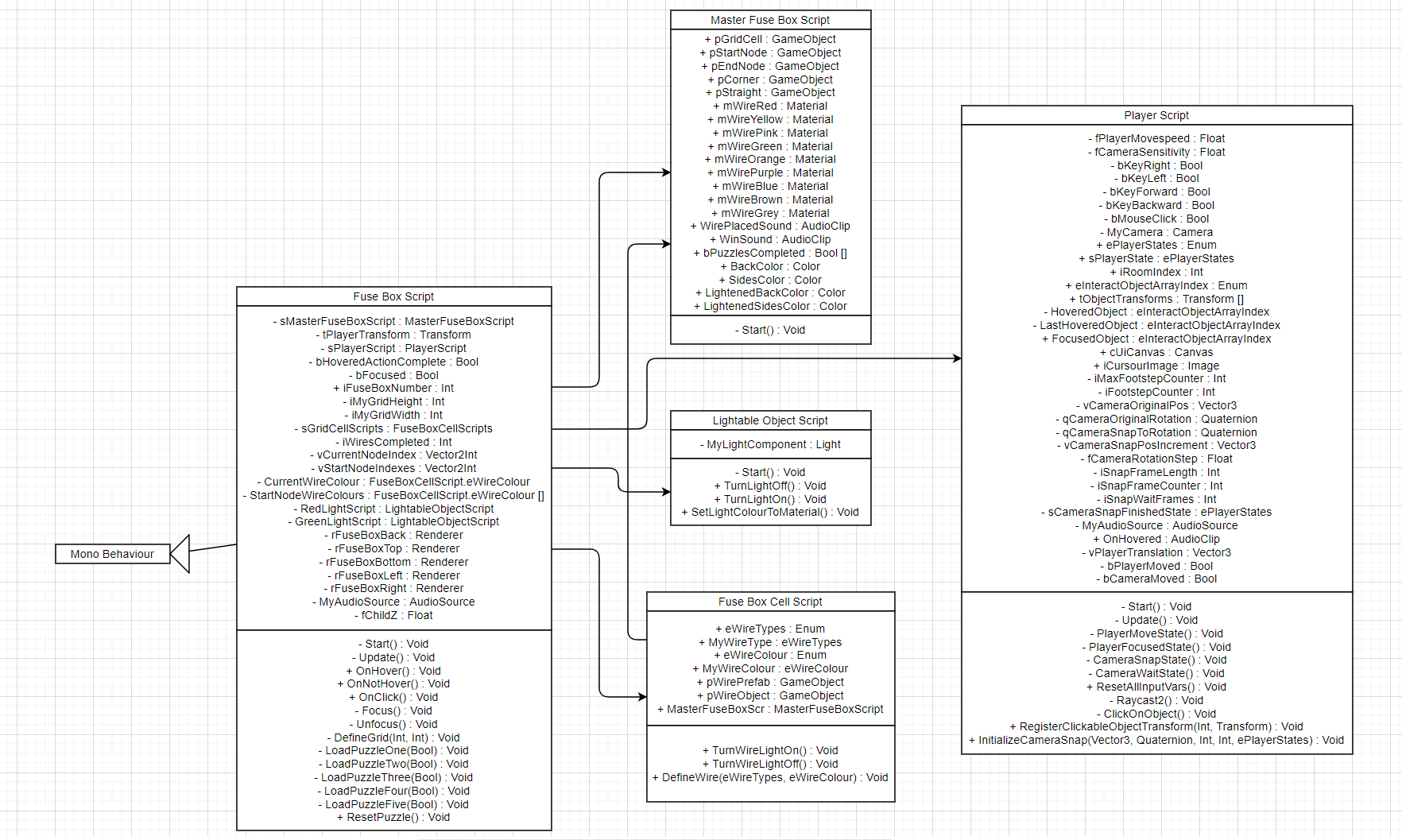
**Final Conclusion:**

As I mentioned in my project evaluation, the final ‘*Eagle Hotel*’ prototype has deviated slightly from its original design because of the various constraints, mainly time, that I faced throughout its development. I am still however, proud of what I managed to create during the prototype’s limited development lifetime as not only do I believe that I stayed true to themes of the initial design, but I also surprised myself throughout as I was able to develop parts of the game, specifically the mechanics of the fuse box puzzles, faster than I’d predicted I would. I am also proud of the spooky aesthetic I was able to cultivate throughout the game. During the initial design phase, making sure that players were thoroughly unsettled by the ‘*Eagle Hotel*’s interior and subsequently played through the game more slowly and cautiously, thus being in a better position to undertake the game’s complex puzzles, was one of the primary goals of the completed prototype. If I had had more time to work on the game, I would have wanted to develop the planned chess puzzle in another of the hotel’s rooms. I really liked the idea for the chess puzzle during the design phase because, I knew that it would both present a complex challenge for the players and the rosewood chess board and chess pieces would also fit in really well with the aesthetic of the spooky old hotel. Overall though, I am happy with my finished prototype of the ‘Eagle Hotel’ and I would have loved to do even more work on it had I had more time.

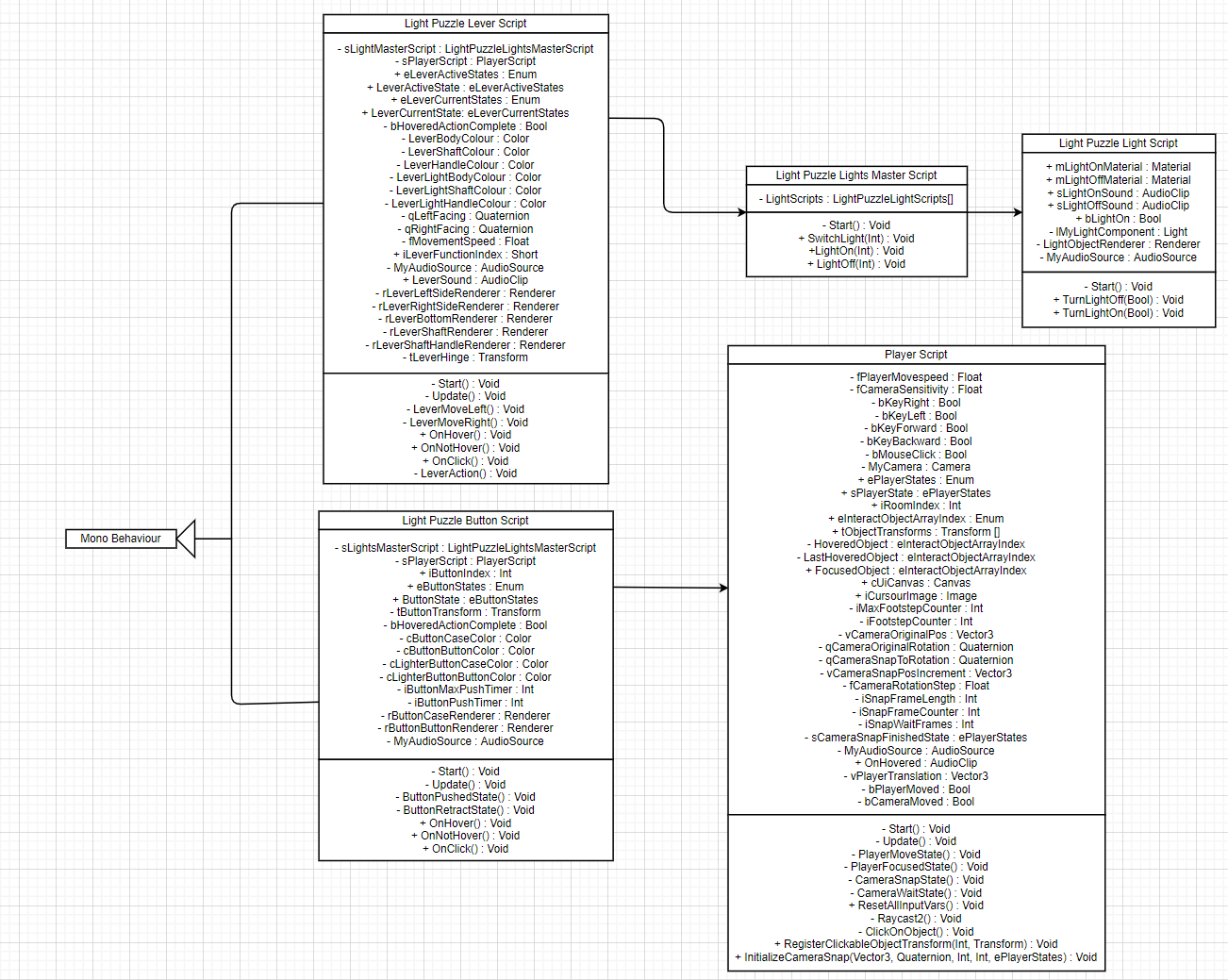
**Appendices:**

**A)** System UML Diagrams

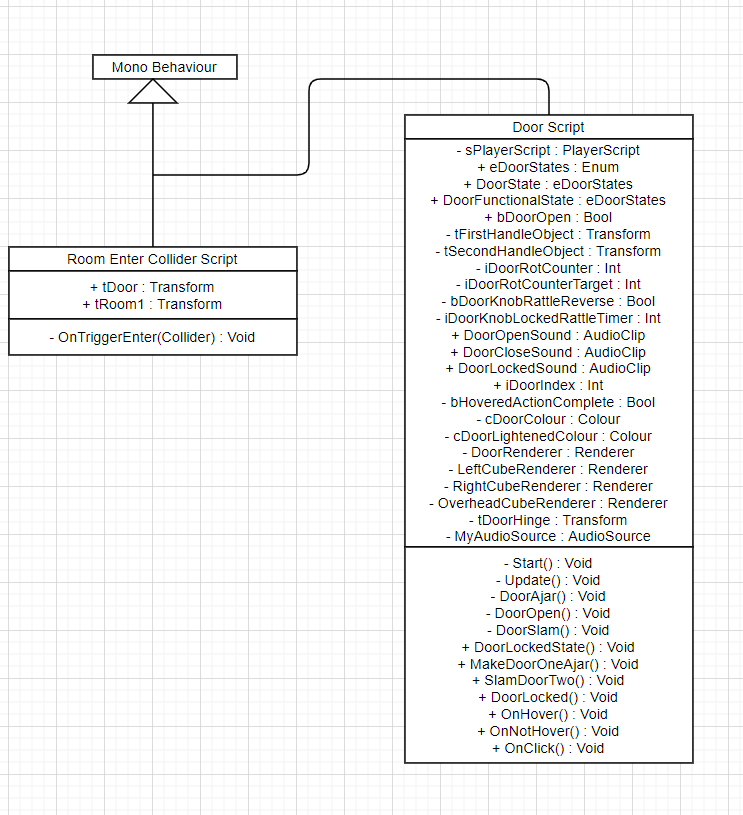




**Figure 14:** UML Diagram of the Fuse Box Puzzles.



**Figure 15:** UML Diagram of the Light/Lever Puzzles.



**Figure 16:** UML Diagram of the Room Enter Collider and Door Scripts.

**B1)** Blackbox Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Expt Result** | **Result** | **Pass?** |
| 1 | Testing if the PC can move to the right | The right movement key is pressed | The PC moves to the right | The PC moves to the right in line with the camera | Pass |
| 2 | Testing if the PC can move to the left | The left movement key is pressed | The PC moves to the left | The PC moves to the left in line with the camera | Pass |
| 3 | Testing if the PC can move forward | The forwards movement key is pressed | The PC moves forwards | The PC moves forwards in line with the camera | Pass |
| 4 | Testing if the PC can move backwards | The backwards movement key is pressed | The PC moves backwards | The PC moves backwards in line with the camera | Pass |
| 5 | Testing if the camera can be panned to the left | Moving the mouse to the left pans the camera to the left | The camera pans to the left | The camera pans to the left successfully | Pass |
| 6 | Testing if the camera can be panned to the right | Moving the mouse to the right pans the camera to the right | The camera pans to the right | The camera pans to the right | Pass |
| 7 | Testing if the camera can be panned upwards | Moving the mouse upwards pans the camera to the right | The camera pans upwards | The camera pans upwards | Pass |
| 8 | Testing if the camera can be panned downwards | Moving the mouse downwards pans the camera to the right | The camera pans downwards | The camera pans downwards | Pass |
| 9 | Clicking on a hovered over fuse box causes the player to focus on it | Pan the camera over a fuse box and then click on it. Test this with all fuse boxes | Each fuse box when clicked on becomes focused on | Each fuse box when clicked on becomes focused on | Pass |
| 10 | Testing if the player can move wires on the fuse boxes to the right | The right movement input is given, while a movement to the right is possible when the player is interacting with a fuse box | The wires can be moved to the right when possible | The wires are correctly moved to the right when possible | Pass |
| 11 | Testing if the player can move wires on the fuse boxes to the left | The left movement input is given, while a movement to the left is possible when the player is interacting with a fuse box | The wires can be moved to the left when possible | The wires are correctly moved to the right when possible | Pass |
| 12 | Testing if the player can move wires on the fuse boxes upwards | The upwards movement input is given, while a movement to the left is possible when the player is interacting with a fuse box | The wires can be moved upwards when possible | The wires are correctly moved upwards when possible | Pass |
| 13 | Testing if the player can move wires on the fuse boxes downwards | The downwards movement input is given, while a downwards movement is possible | The wires can be moved to the downwards when possible | The wires can be moved to the downwards when possible | Pass |
| 14 | Moving a wire left after just moving right will move the player back over themselves | Move the wire back over the previous segment | The player moves back over the wire | The wire segment is deleted properly and the position moves correctly | Pass |
| 15 | Moving a wire right after just moving left will move the player back over themselves | Move the wire back over the previous segment | The player moves back over the wire | The wire segment is deleted properly and the position moves correctly | Pass |
| 16 | Moving a wire up after just moving down will move the player back over themselves | Move the wire back over the previous segment | The player moves back over the wire | The wire segment is deleted properly and the position moves correctly | Pass |
| 17 | Moving a wire down after just up right will move the player back over themselves | Move the wire back over the previous segment | The player moves back over the wire | The wire segment is deleted properly and the position moves correctly | Pass |
| 18 | Moving a wire upwards and then right will create a corner | Move the wire up and right where possible | The proper corner is created | The corner is created | Pass |
| 19 | Moving a wire upwards and then left will create a corner | Move the wire up and left where possible | The proper corner is created | The corner is created | Pass |
| 20 | Moving a wire downwards and then right will create a corner | Move the wire down and right where possible | The proper corner is created | The corner is created | Pass |
| 21 | Moving a wire downwards and then left will create a corner | Move the wire down and left where possible | The proper corner is created | The corner is created | Pass |
| 22 | Moving a wire left and then up will create a corner | Move the wire left and up where possible | The proper corner is created | The corner is created | Pass |
| 23 | Moving a wire left and then down will create a corner | Move the wire up and left where possible | The proper corner is created | The corner is created | Pass |
| 24 | Moving a wire right and then up will create a corner | Move the wire right and up where possible | The proper corner is created | The corner is created | Pass |
| 25 | Moving a wire right and then down will create a corner | Move the wire right and up where possible | The proper corner is created | The corner is created | Pass |
| 26 | Moving the wire left over a corner it just created will replace it with a straight piece | Move the wire left back over a corner | A straight piece is created | A straight piece replaces the corner | Pass |
| 27 | Moving the wire right over a corner it just created will replace it with a straight piece | Move the wire right back over a corner | A straight piece is created | A straight piece replaces the corner | Pass |
| 28 | Moving the wire upwards over a corner it just created will replace it with a straight piece | Move the wire upwards back over a corner | A straight piece is created | A straight piece replaces the corner | Pass |
| 29 | Moving the downwards left over a corner it just created will replace it with a straight piece | Move the wire downwards back over a corner | A straight piece is created | A straight piece replaces the corner | Pass |
| 30 | Wires cannot be moved right when they are already at the right edge of the grid | Move the wire right until it reaches the right edge of the grid and then try to move right again | The wire won’t move anymore | The wire does not move | Pass |
| 31 | Wires cannot be moved left when they are already at the left edge of the grid | Move the wire left until it reaches the left edge of the grid and then try to move left again | The wire won’t move anymore | The wire does not move | Pass |
| 32 | Wires cannot be moved down when they are already at the bottom of the grid | Move the wire down until it reaches the bottom of the grid and then try to move down again | The wire won’t move anymore | The wire does not move | Pass |
| 33 | Wires cannot be moved upwards when they are already at the top of the grid | Move the wire upwards until it reaches the top of the grid and then try to move up again | The wire won’t move anymore | The wire does not move | Pass |
| 34 | Wires cannot be moved right into another wire segment | Attempt to move the wire right onto a cell containing a segment from another wire or a segment from the current wire that is not the previous one | The wire won’t move onto the cell | The wire does not move | Pass |
| 35 | Wires cannot be moved left into another wire segment | Attempt to move the wire left onto a cell containing a segment from another wire or a segment from the current wire that is not the previous one | The wire won’t move onto the cell | The wire does not move | Pass |
| 36 | Wires cannot be moved down into another wire segment | Attempt to move the wire down onto a cell containing a segment from another wire or a segment from the current wire that is not the previous one | The wire won’t move onto the cell | The wire does not move | Pass |
| 37 | Wires cannot be moved up into another wire segment | Attempt to move the wire up onto a cell containing a segment from another wire or a segment from the current wire that is not the previous one | The wire won’t move onto the cell | The wire does not move | Pass |
| 38 | When a wire is attached to its end node, the puzzle moves onto the next wire | Play through each fuse box puzzle and check what happens after completing each wire that is not the last wire in the puzzle | Completing these wires cause the puzzle to move onto the next | The puzzles moves onto the next wire properly | Pass |
| 39 | When the final wire in the puzzle is attached to its end node the player stops focusing on the fuse box | Play through each puzzle and check what happens when the player finishes them. | Completing the puzzle causes the player to be able to move correctly | The player moves back into moving correctly | Pass |
| 40 | Completing every fuse box puzzle causes the door to the corridor to room two to slowly partially open | Complete every fuse box puzzle | The door opens after finishing all the puzzles | The door opens as expected | Pass |
| 41 | Clicking on the door out of room one once all the fuse box puzzles have been completed, causes it to open | Click on the door after completing all the fuse box puzzles. | The door opens slowly | The door opens slowly | Pass |
| 42 | Clicking on the door to room two the first time causes it to open slowly | Click on the door from inside the corridor | The door opens slowly | The door opens as intended | Pass |
| 43 | Clicking on the levers in room two turns some of the lights on the wall on and off. Each lever affects a different set of lights | Click on the levers | The levers turn some lights on/off | Clicking on a lever turns on/off a set of lights | Pass |
| 44 | When the button in room two is pushed the lights puzzle is reset | Turn some of the lights on using the levers, then click on the button and look back at the lights | Clicking the button causes the lights to reset | The button resets the puzzle | Pass |

**Table 1:** The Blackbox tests performed on the finished prototype of ‘*The Eagle Hotel*’. Blackbox tests are tests performed by someone who is not familiar with the programs code and is thus treating it like a black box.

**B2)** Unit Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Result** | **Expt Result** | **Pass?** |
| 1 | Hovering the cursor in the middle of the GUI over a fuse box on the wall in the first room causes it to light up | Pan the camera over each of the fuse boxes | Each fuse box lights up correctly | Each fuse box lights up correctly | Pass |
| 2 | Hovering the cursor in the middle of the GUI over a fuse box on the wall causes the ‘On Hovered’ sound to be player | Pan the camera over each of the fuse boxes | The sound is played when the cursor moves over the fuse box | The sound is played when the cursor moves over the fuse box | Pass |
| 3 | Clicking on a hovered over fuse box causes the mouse cursor image to move to the location of the player’s mouse on screen | Click on a fuse box and watch the mouse cursor | The cursor moves | The cursor moves to the players mouse | Pass |
| 4 | Moving a wire upwards and then right will create an ‘Up and Right’ corner piece | Move the wire up and right where possible | The proper corner is created | The corner is created | Pass |
| 5 | Moving a wire upwards and then left will create an ‘Up and Left’ corner piece | Move the wire up and left where possible | The proper corner is created | The corner is created | Pass |
| 6 | Moving a wire downwards and then right will create a ‘Down and Right’ corner piece | Move the wire down and right where possible | The proper corner is created | The corner is created | Pass |
| 7 | Moving a wire downwards and then left will create an ‘Down and Left’ corner piece | Move the wire down and left where possible | The proper corner is created | The corner is created | Pass |
| 8 | Moving a wire left and then up will create an ‘Down and Right’ corner piece | Move the wire left and up where possible | The proper corner is created | The corner is created | Pass |
| 9 | Moving a wire left and then down will create an ‘Up and Right’ corner piece | Move the wire up and left where possible | The proper corner is created | The corner is created | Pass |
| 10 | Moving a wire right and then up will create a ‘Down and Left’ corner piece | Move the wire right and up where possible | The proper corner is created | The corner is created | Pass |
| 11 | Moving a wire right and then down will create an ‘Up and Left’ corner piece | Move the wire right and up where possible | The proper corner is created | The corner is created | Pass |
| 12 | Moving a wire left onto an ‘Up and Right’, corner turns it into an upwards straight piece | Move the wire up, right and then left. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 13 | Moving a wire left onto an ‘Down and Right’, corner turns it into a downwards straight piece | Move the wire down, right and then left. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 14 | Moving a wire right onto an ‘Up and Left’, corner turns it into an upwards straight piece | Move the wire up, left and then right. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 15 | Moving a wire right onto an ‘Down and Left’, corner turns it into a downwards straight piece | Move the wire down, left and then right. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 16 | Moving a wire down onto a ‘Down and Right’, corner turns it into a left straight piece | Move the wire left, up and then down. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 17 | Moving a wire down onto a ‘Down and Left’, corner turns it into a right straight piece | Move the wire right, up and then down. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 18 | Moving a wire up onto a ‘Up and Left’, corner turns it into a right straight piece | Move the wire right, down and then up. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 19 | Moving a wire up onto an ‘Up and Right’, corner turns it into a left straight piece | Move the wire left, down and then up. | The corner is changed back to the original piece | The correct straight piece replaces the corner | Pass |
| 20 | When the player completes a puzzle the cursor picture, goes back to the centre of the screen | Complete each puzzle and check what happens | The cursor image is reset to the screen’s centre | The image is reset correctly | Pass |
| 21 | When the player completes a puzzle the lights on the fuse box switch from red on and green off to red off and green on. | Complete each puzzle and check the lights when the final wire is connected | The red light is turned off and the green light is turned on | Red light turns off and green light turns on | Pass |
| 22 | Completing every fuse box puzzle causes the camera to snap around to show the door opening | Complete every fuse box puzzle and watch what the camera does | The camera pans to the door opening | The camera pans as intended | Pass |
| 23 | Clicking on the door to the corridor to room two without completing all the fuse box puzzles causes it the doorknob to rattle, indicating that it’s currently locked | Click on the door before completing all of the fuse box puzzles. | The doorknob will rattle as intended | The doorknob rattles properly | Pass |
| 24 | Clicking on the door to the corridor to room two without completing all the fuse box puzzles plays a ‘Door Locked’ sound effect. | Click on the door before completing any of the fuse box puzzles. | The sound effect is played as intended | The sound effect plays properly | Pass |
| 25 | The door slams after the player has entered the room | Walk into room two and check if the door behaves as intended | The door slams behind the player | The door slams properly | Pass |
| 26 | When moving into room two the camera snaps around to show the door being slammed | Walk into room two and check for the camera snap | The camera snaps around | The camera snaps properly | Pass |
| 27 | When the door has fully slammed a slamming sound effect is played | Walk into room two, triggering the slam, and see if the sound effect is played | The sound effect is played | The slam sound plays when the door closes | Pass |
| 28 | When the door has fully slammed, the entire first room is deleted, as since it’s no longer needed it can be deleted to optimise efficiency | Walk into room two, trigger the slam, then pause the game’s execution in unity and move the Scene camera to check if room one is still there | Room one is deleted on the slam | The first room is properly deleted | Pass |
| 29 | Hovering over a lever in room two causes it to light up | Hover the cursor over a lever in room two | The lever lights up | The lever lights up as intended | Pass |
| 30 | Hovering over a lever in room two plays the ‘hovered over’ sound effect | Hover the cursor over a lever in room two | The sound effect is played | The correct sound effect is played on the hover | Pass |
| 31 | Clicking on a hovered lever causes it to be pulled to the opposite side it’s currently in | Click on one of the levers | The lever’s shaft moves to the other side | The shaft moves properly | Pass |
| 32 | After the left lever is clicked on and its shaft reaches the opposite side, it switches the first two lights | Click on the lever in the left of the room | When the shaft reaches the opposite side the first two lights turn on | The lever switches the correct lights | Pass |
| 33 | When the middle lever is clicked it switches the second and third lights | Click on the lever in the middle of the room, under the lights | Clicking the lever switches the second and third lights | The lever switches the correct lights | Pass |
| 34 | When the lever on the right side of the room is clicked the third, fourth and fifth lights are switched | Click on the lever on the right side of the room. | Clicking the lever switches the last three lights | The lever switches the correct lights | Pass |
| 35 | The lights on the wall always shine a red light when they are ‘turned off’ | Turn the lights on the wall on and off again to check the light colour is always red when off | The lights are always red when the light is off | The lights are always red when turned off | Pass |
| 36 | The lights on the wall always shine a green light when they are ‘turned on’ | Turn the lights on the wall on and off again to check the light colour is always green when on | The lights are always green when the light is turned on | The lights are always green when turned on | Pass |
| 37 | When the button on the back wall of room two is hovered over it lights up | Hover over the button | The button lights up when hovered | The button lights up | Pass |
| 38 | When the button is hovered the ‘On Hovered’ sound effect is played | Hover over the button | The sound effect is played | When the button is hovered over the sound effect is played | Pass |
| 39 | When the button is clicked on the button moves inside of its case and then back out, resembling a button being pushed | Click on the button in room two | The button moves in the case | The button moves into the case like its being pressed | Pass |

**Table 2:** The unit tests performed on the finished prototype of ‘*The Eagle Hotel*’. Unit tests are tests that are, in contrast to Blackbox testing, carried out by someone who is familiar with the programs code.

**B3)** Performance Profiling

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Mean FPS** | | | |
| **1** | **2** | **3** | Mean (x-bar) |
| 1 | Testing the speed of the first frame when the game loads | Measure the FPS at the very start of the game | 1.86  (536.72ms) | 1.86  (536.37ms) | 1.86  (536.37ms) | 1.86  (536.37ms) |
| 2 | Testing the FPS when the player is moving about the game world WITHOUT moving the camera | Move around the game world WITHOUT moving the camera and measure the average FPS over one minute | 22.95 | 26.53 | 22.13 | **23.87** |
| 3 | Testing the FPS when the player is looking around the game space WITHOUT moving | Look around the game world WITHOUT moving and measure the average FPS over one minute | 21.41 | 21.33 | 21.03 | **21.26** |
| 4 | Testing the FPS when the player is moving around the game world AND moving | Move around the game world whilst also moving the camera and measure the average FPS over one minute | 17.95 | 22.87 | 34.22 | **25.01** |
| 5 | Idle FPS when focused on a fuse box puzzle | Click on a fuse box and measure the average FPS over one minute whilst doing nothing | 31.27 | 26.58 | 28.60 | **28.82** |
| 6 | Average FPS whilst moving wire in the fuse box around | Click on a fuse box and measure the average FPS whilst manipulating the wires for a minute | 18.57 | 13.43 | 12.37 | **14.79** |

**Table 3:** The results of the performance profiling tests, conducted on the finished prototype of ‘The Eagle Hotel’. This tests how well the game runs under different scenarios. Each FPS value is tested three times to ensure accuracy. It should also be noted that during these tests the game was being run on my own personal, not very powerful, laptop while multiple other applications were also running, that was being displayed on my 4K monitor.

**C)** First Deliverable Document

**Jack Moorin P17190172 Final Year Project First Deliverable Documents**

**Literature Review**

For my final year project, I have chosen to develop a three-dimensional ‘Escape the Room’ style puzzle game. A puzzle video game is a digital game that present players with a logical or conceptual puzzle that they have to solve. Most puzzle games offer the player a series of puzzles based around single theme involving pattern recognition or logic. For example, the ‘Portal’ series gives players a series of logical puzzles that they have to solve using their player character’s ‘Portal Gun’ that allows them to create portals that they can use to transport themselves and objects across the game world in order to solve puzzles. In an ‘Escape the Room’ puzzle players must find their way out of a room, or potentially a series of rooms, by solving a series of puzzles that when completed will allow them to escape. They have found a niche hardcore following among certain puzzle enthusiasts who, according to Escape Room fan David Spira in a 2019 interview with Vox, “*[have] a maybe-not-entirely mainstream sense of what is fun. …The people who are looking for mentally stimulating things to do in their free time, where they’re the player, not watching someone else have an adventure. That’s the kind of people it attracts, and those are the types of people we just loving having in our lives.*” (2019, Vox).

The ‘Escape the Room’ puzzle game genre first became popular in the 2010s, that exists both in and outside of video games. Real world ‘Escape the Room’ puzzle games were invented by Takao Kato, a thirty-four-year-old publishing employee, who stated in a 2009 interview, “*I wondered why interesting things didn’t happen in my life, like they did in books. I thought I could create my own adventure, a story, and then invite people to be a part of it*” (Japan Times, 2009), showing how Kato wanted to bring more of the adventures he’d encountered in fiction into the real world. A manga lover, Kato grew up reading about characters having adventures fighting mythical beasts or looking for treasure and like most fantasy fans was disappointed with the comparative dull reality of life causing him to want to bring more adventure into his real life as an adult.

Despite the popularity of earlier puzzle rooms, real life ‘Escape the Room’ puzzles took a while to become accepted by the public. The fairly recent rise in popularity of these puzzles can in part be attributed to multiple franchises creating their own versions for their promotional value. The BBC Television series ‘Sherlock’ themed Escape Room was created during a hiatus between seasons after Steve Moffat and Mark Gattis (the creators of the series) tried Nick Moran’s ‘Time Run’ Escape Room in London in 2015 and later agreed to develop the ‘Sherlock’ themed room together. Moran would later describe the process that led to the design of the game in a 2019 interview with ‘The Guardian’ “*They didn’t have plans for a new season any time soon and had wanted to make a live experience. So, we sat in a room and talked.*” (The Guardian, 2009). He would also later, in the same interview, voice his surprise at the slow adoption of Escape the Room games into the public eye saying, “*[I’m] surprised that they have taken this long to become fully mainstream*”.

Video game ‘Escape the Room’ puzzles predate the invention of the real-life versions and got their start from point-and-click adventure games, first being seen as a game mechanic in the 1988 text adventure game ‘Behind Closed Doors” and gaining further popularity thanks to their inclusion in the ‘Myst’ series. Popular ‘Escape-the-Room’ puzzle video games include 2012’s ‘The Room’ by Fireproof Games and ‘Nine Hours, Nine Persons, Nine Doors’ by Chunsoft. In an article for video gaming magazine ‘Vox’ in 2016, Alex Abad-Santos stated that "*while adventure games with more open exploration may be too aimless, the appeal of escape room games is in their immediacy and constricted world*” (Vox, 2016), illustrating how they had become such a popular, and staple, element of point-and-click adventure games.

Escape the Room style games are even starting to be used in education with museums and schools even starting to use games in education as much more interactive alternatives to historical re-enactments or waxwork exhibits. Scott Nicholson, a professor of gaming in Ontario Canada, created an Escape the Room style game for Canadian schools to use to help teach children about Canada’s electoral system. In a 2017 interview with ‘The Guardian’ Nicholson voiced his enthusiasm about Escape the Room games being adopted into education saying, “*Escape rooms have something other group activities don’t have because they are not about competition but collaboration. …We now want to design games that can make a real difference*” (The Guardian, 2017). The educational escape room has gained much renown in Canada and lots of Canadian schools have begun taking regular field trips to visit the room as they’ve found that it achieves a much higher level of engagement from children than their own lessons about the electoral system.

The Escape the Room video game that I have chosen to develop for my project, is going to have an eerie and creepy aesthetic. I have chosen this aesthetic because I believe that if the player is playing slowly and cautiously, they will be in the best frame of mind for dealing with the game’s puzzles. If I had say fast paced guitar music and the player was in a particularly energetic mood, then they would be more inclined to try and rush through and might find the game’s slow puzzles annoying instead. My Escape the Room game is in fact one of very few Escape the Room games that have a scary aesthetic. Escape the Room enthusiast Sarah Dodd recounted how she was disappointed in the Hollywood horror film ‘Escape Room’, which featured six players who were stuck in a deadly booby-trapped escape room, since “[*the film] was released in the same month as the Polish Room Fire Incident*” (The Times, 2019). The incident Dodd was referring to being an incident in Poland where five girls were killed because in a fire in an Escape Room as they couldn’t leave due to the doors to the room being genuinely locked. This incident has prompted the majority of Escape Room companies to build in ‘Panic Buttons’ which players can press if they need to leave the room without solving it’s puzzle due to an emergency.

Another way in which my projects Escape Room video game will differ from more traditional versions is the fact that my game will not time the players attempt to escape the room and will instead allow player to play indefinitely. I decided not to include a timer after discovering in a 2015 article in the independent discussing the recent increase in Escape the Room games in the UK, that “*only twenty percent of all Escape the Room players actually manage to escape the room in time, while eighty percent find the room too difficult to solve before their time runs out*” (The Independent, 2015). Given the complexity of the puzzles I wanted to include I thought that instead of creating a faster paced experience by using a timer that forced players to restart if they run out of time, which according to the article would happen to approximately eighty percent of them, I could allow the players to take their time with the difficult puzzles and ultimately have a better experience over all.

I have chosen to develop my game for mobile phones as well as PC. This is because from my own experience with ‘Escape the Room’ style games on mobiles, I have found them to be a hugely enjoyable and easy to play through. This is in part down to the fact that ‘Escape the Room’ games, and indeed other puzzle games, want the player to think about what they are doing and how they are going to solve the game’s puzzles. Therefore, since this large stage of the gameplay occurs in the players head, the game itself does not need to contain things such as complex effects or high-fidelity graphics, which in turn makes it perfect for platform of mobile phones, since they are restricted in the amount if processing power and memory they can use up. It is also the reason mobile games designer ‘Gavin Hughes’ attributed to the large and ongoing success of the puzzle game genre in a 2013 interview with ‘Rock Paper Shotgun’, “*I think really the simplicity of puzzle games is what makes them so great for use on mobile. Obviously with the mobile platform you do have restrictions with processing speed and memory capacity, which means that they often struggle to render complex graphics and the like so that’s why they match so well with the puzzles I think*” (Rock Paper Shotgun, 2013). He would also later go on to give the easy portability and access to our mobile phones as another factor for puzzle game success saying, “*It’s also great for when gets to a puzzle that they don’t get immediately. It sticks around in they’re heads all day and then they realize how to do it and they can just take their phone out of their pocket and do it straight away*”.

Another reason that I have chosen to develop my game for mobile as well as PC is that the popularity of and constant access to mobile phones, created thanks to our societies predilection with having one on your person at all times, has not just fed into the critical success of mobile puzzle games but also their success commercially. In a 2018 report, video game marketing intelligence website ‘Newzoo’, predicted the value of the 2019 video games market to be worth around $68.5 ‘Billion (Newzoo, 2018), a number which their statistician ‘Tom Wijman’ attributed to “*the amount of people spending so much collective time on their phones has a clear knock-on effect on how much these games make. Smaller purchases have proven to be the most popular, particularly those that result in the removal of adverts from the game, however the sheer number of people who do end up spending the smaller amounts of money is what makes for the huge success of these games*”. He later went on to mention the importance of ‘whales’ in the market, ‘whales’ being the term often used by mobile game developers to refer to those people who spend considerably more the average on a mobile game, “*While the median and the range of the money people spent on mobile games does show a leaning towards revenue being mostly made up by lots of people spending smaller amounts, it is worth mentioning the increase in the number of Whales in the past few years. That 0.15% of people who make up almost 50% of the mobile game revenue. The identifiable number of these people has also increased dramatically over the last five years, in part thanks to the slow inclusion of mobile games into the e-sports and online streaming worlds*”.

**Literature Review Bibliography:**

* *Vox 2019, [The great escape], Vox, viewed 24th December 2019, <* <https://www.vox.com/the-goods/2019/8/7/20749177/escape-room-game>>
* *Japan Times 2009, [Interview with ‘Escape the Room’ game creator Takao Kato], Japan Times, viewed 17th December 2019, <*[*https://www.japantimes.org/articles/esapce-the-room-creator-interview/*](https://www.japantimes.org/articles/esapce-the-room-creator-interview/)*>*
* *The Guardian 2009, [Get me out of here! Why escape rooms have become a global craze], The Guardian, viewed 18th December 2019, <*<https://www.theguardian.com/games/2019/apr/01/get-out-how-escape-rooms-became-a-global-craze>*>*
* *Vox 2016, [The strange appeal of escape the room games, explained], Vox, viewed 18th December 2019, <* <https://www.vox.com/culture/2016/10/26/13311456/escape-the-room-games-explained>>
* *The Guardian 2017, [Escape the Room Games in Education], The Guardian, viewed 19th December 2019, <*<https://www.theguardian.com/games/2019/apr/01/escape-the-room-games-in-education>>
* *The Times 2019, [Film Causes Controversy Among Escape Room Enthusiasts], The Times, viewed December 19th 2019, <*[*https://www.thetimes.com./ar/escape-room-film-controvesy*](https://www.thetimes.com./ar/escape-room-film-controvesy)*>*
* *The Independent 2015, [You Can’t Escape the Escape Room Craze], The Independent, viewed December 19th 2019, <*[*https://www.theindependent.co.uk/2015-articles/you-cant-escape-the-escape-room-craze*](https://www.theindependent.co.uk/2015-articles/you-cant-escape-the-escape-room-craze)*>*
* *Rock Paper Shotgun 2013, [Why are Mobile Puzzle Games so Popular These Days], Rock Paper Shotgun, viewed January 2nd 2020, <*<https://www.rockpapershotgun.com/2013/5/13/why-are-mobile-games-so-popular-these-days/>>
* *Newzoo 2018, [The Global Games Market Will Generate $152.1 Billion in 2019 as the U.S. Overtakes China as the Biggest Market], Newzoo, viewed January 2nd 2020, <*<https://newzoo.com/insights/articles/the-global-games-market-will-generate-152-1-billion-in-2019-as-the-u-s-overtakes-china-as-the-biggest-market/>>

**Functional Requirements**

**Introduction:**

The functional requirements of a piece of software are the minimum standard that a piece of software must achieve for it to be able to be classified as it’s intended type of program. For example, the functional requirements of an antivirus program such as Norton, is to identify, detect and protect a device from computer viruses. In this document, I will be identify and outline the functional requirements of the game that I am developing for my final year project.

**Game In-Depth Overview**

‘The Eagle Hotel’ is going to be a 3D ‘Escape the Room’ style puzzle game for PC and mobile devices. I am developing this game as part of my Final Year Development Project under the supervision of Connor Fahy. The premise of the game is that the player is trapped in a room within the fictitious ‘Eagle Hotel’, so named after the song Hotel California by the Eagles, *you may check out but you can never leave*. In order to leave the player must solve a series of puzzles in order to unlock the door and escape. To solve the games puzzles the player has to move around and interact with the puzzles which are built into objects within the room. For example, the fuse boxes on the wall and the chess board on the table both contain path finding puzzles.

**Level 1 Functional Requirements**

The level one functional requirements of my game are the things it needs to be able to do in their most simplistic form. These are:

* Use inputs from the player to control the movement and view direction of the PC (player character).
* Use inputs from the player to interact with objects within the game space.
* Have interactable puzzles within the room.
* Allow the player to leave the room once the puzzles have been completed.

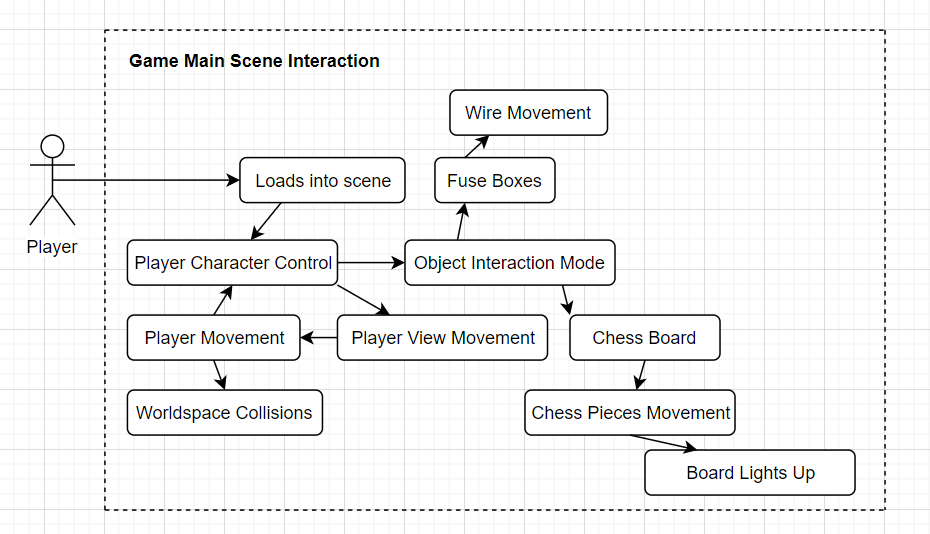
**Level 2 Functional Requirements**

The level two functional requirements of my game are more specific versions of the more abstract level one requirements. Defining these more specific versions of the level one rules can result in one rule being split up into multiple rules in order to ensure there is no ambiguity.

* Allow the player to move the PC forwards
* Allow the player to move the PC backwards
* Allow the player to move the PC to the left
* Allow the player to move the PC to the right
* The PC cannot move through in-game objects
* The player’s movement of the PC is relative to the direction of the PC current field of vision
* Allow the player to move the PC’s field of vision upwards
* Allow the player to move the PC’s field of vision downwards
* Allow the player to move the PC’s field of vision to the left
* Allow the player to move the PC’s field of vision to the right
* Allow the player to go into an ‘interactive state’ with certain objects in the room
* Allow the player a higher level of interactivity with objects when in an ‘interactive state’ with them
* There are fuse boxes on the wall that can be interacted with in the PC’s ‘interactive mode’
* The fuse boxes each contain an interactable route-finding puzzle using wires
* Allow the player to move the wires within the fuse boxes across a grid, unless they would then go over another wire
* Allow the player to ‘go back’ on a wire they’ve previously put down
* Allow the player to complete each fuse box puzzles once they’ve successfully navigated all the wires into the correct positions
* Allow the player to complete the fuse box puzzle section once they’ve completed every fuse box puzzle
* There is a chess board on the table can be interacted with in the PC’s ‘interactive mode’
* The chess board contains chess pieces which can be moved across the board in that piece’s movement style onto any available cells
* Board cells that have been moved across become lit up, unless they’re already lit up in which case they go back to normal
* Lighting up all the board cells causes the board to switch to the next puzzle
* Completing the final chess board puzzle allows the player to complete the chess board section
* Completing both the fuse box section and the chess board section allows the player to leave the room
* The player cannot leave the game room until the fuse box and chess board puzzles have been completed

**Software Use Cases**

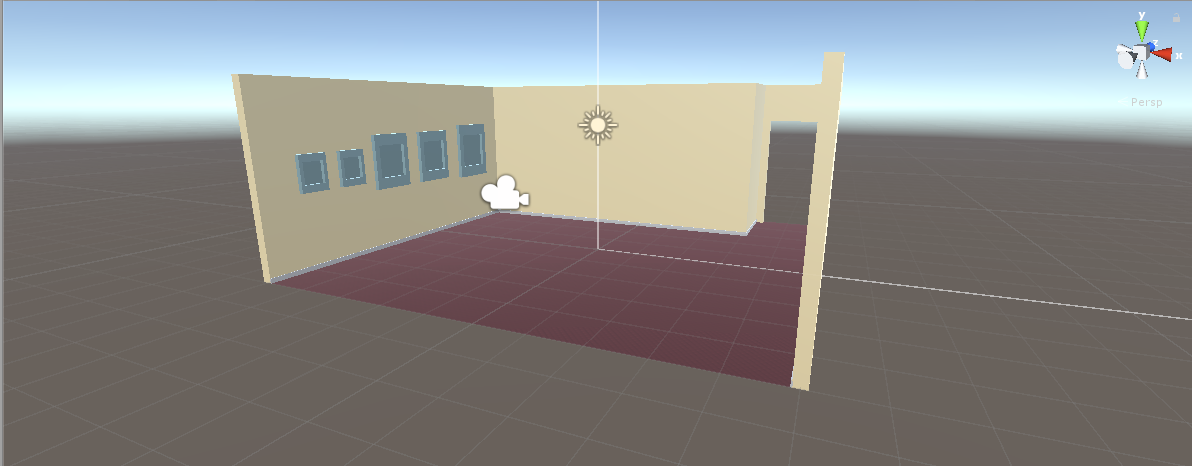
A software use case describes how a user interacts with a piece of software. They show us how users can use the software to do different things and what the software needs to be able to do to achieve them. For my project these use cases will describe how the player will play ‘The Eagle Hotel’ and make use of its functional requirements.



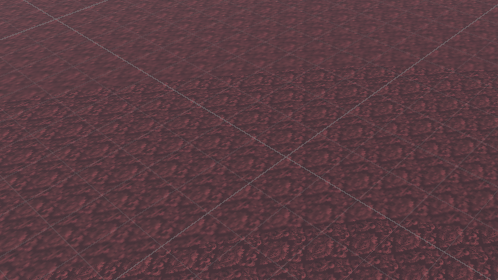
**Current Software Implementation (October to December)**

**Game World:**

At this current stage of my programs development I have been working on my game for the last three months. I have created the game start scene which the player will be loaded into upon the start of the game.



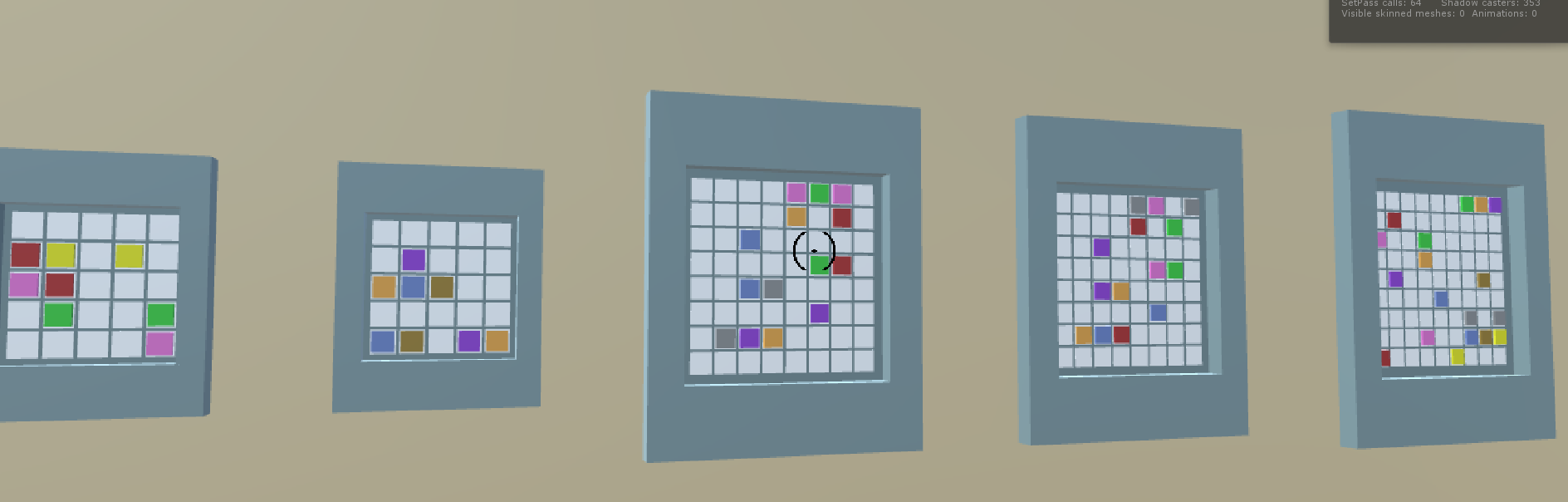
I have begun to develop the aesthetic for the envisioned environment. It is still in development; however, I have settled on numerous elements for the final game. For example, as you can see in the above image I have created the room’s geometry, modelling a room with five walls (there will be six when the rooms finished) and space for a door. I have also added textures to the walls and floor. While I am not yet sure about the wall texture I have decided that I like the floor texture because I believe that it looks like a particularly fancy carpet.



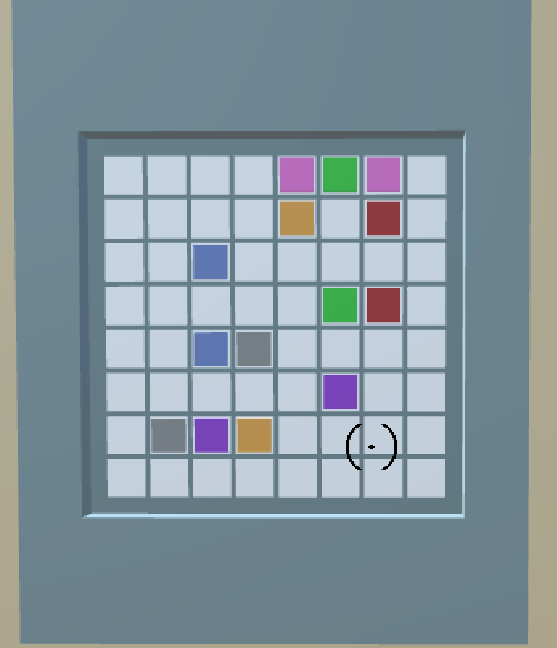
I have also added skirting boards to the walls to add some more detail. These do not have an assigned texture however since they’re supposed to be white. In the future I may decided to give them a texture if I want them to be a certain shade and not just pure white.



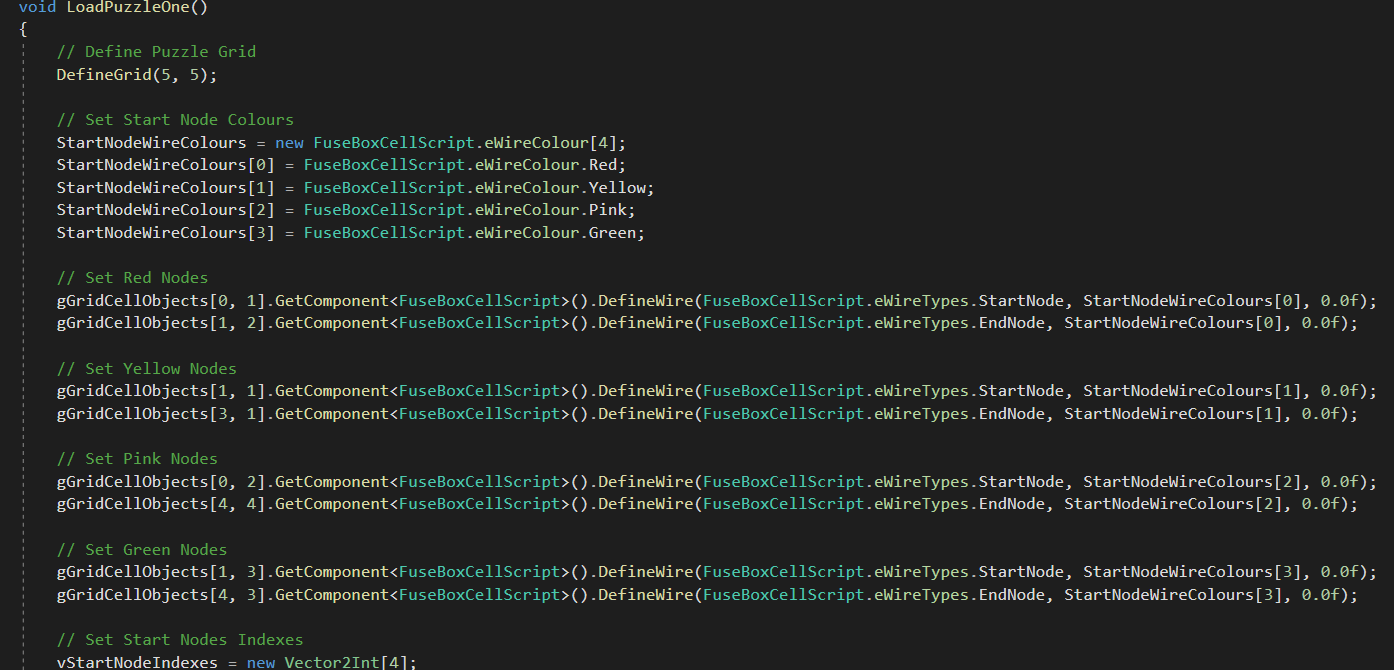
There are five fuse boxes on the left-hand wall. Again, I have not yet settled on their final design but currently they look like simple light blue boxes affixed to a wall.



They each contain a grid which their puzzles are based off of. The grid size gets bigger across the boxes and the boxes themselves have also been scale up to accommodate this. Each gird contains a series of nodes that represent the start and intended end points of the wires in the puzzles. At the moment they both use the same shape but in the final version this will be changed.



Each fuse boxes grid and the wires it contains are dynamically created at the start of the game. I have decided to create them this way as it will make for easier debugging of the puzzle’s mechanics, they are all defined by the same code and therefore fixing one thing will affect all the puzzles, and at this point in the game’s development it also makes it a lot easier to alter how they work and look.

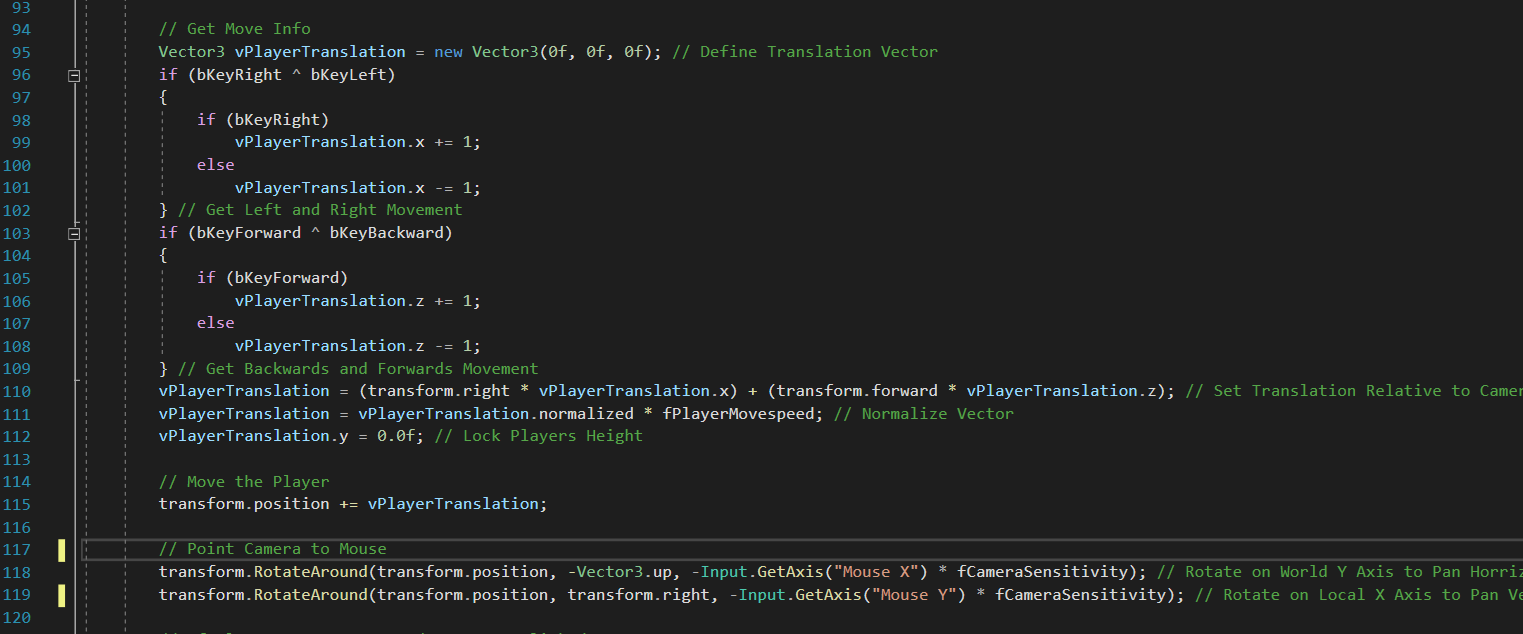


**Player Character:**

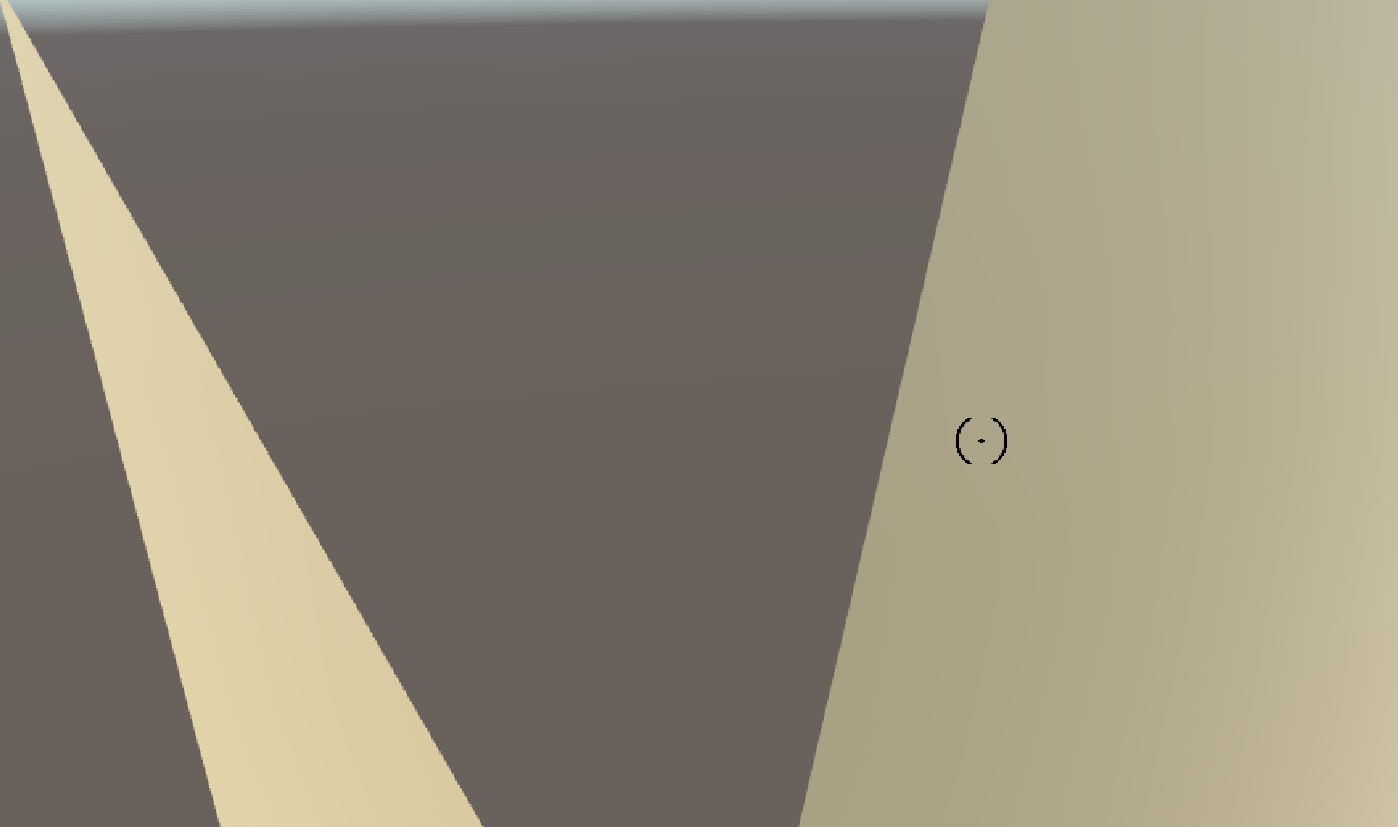
I have fully implemented movement and panning on a camera object that will serve as the player. I have decided that I do not need to create an actual player object since the game view is viewed from the PC’s (Player Characters) point of view meaning that the PC will never be visible, and the player therefore will not know that it’s just a camera.



I also made sure to implement the player’s movement in such a way that it is relative to the position and rotation the camera. This means that the if the player moves the PC forwards the PC will move forwards in the direction that the camera is facing.



I have not yet implemented movement collisions, so the player can walk through the objects in the game world.

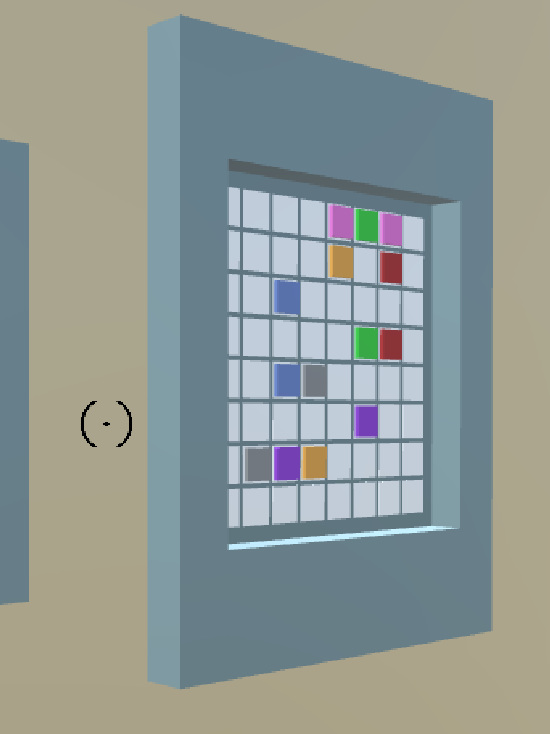
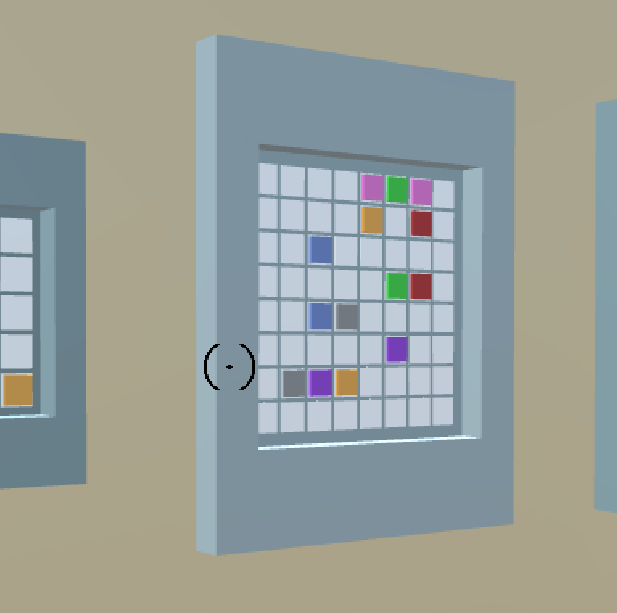


Finally, I have added a cursor image to the game’s UI. In the PC’s base movement state, it is shown at the centre of the screen. The PC’s camera is set to move as the player’s mouse moves which in turn makes the cursor image itself appear to move with the mouse.

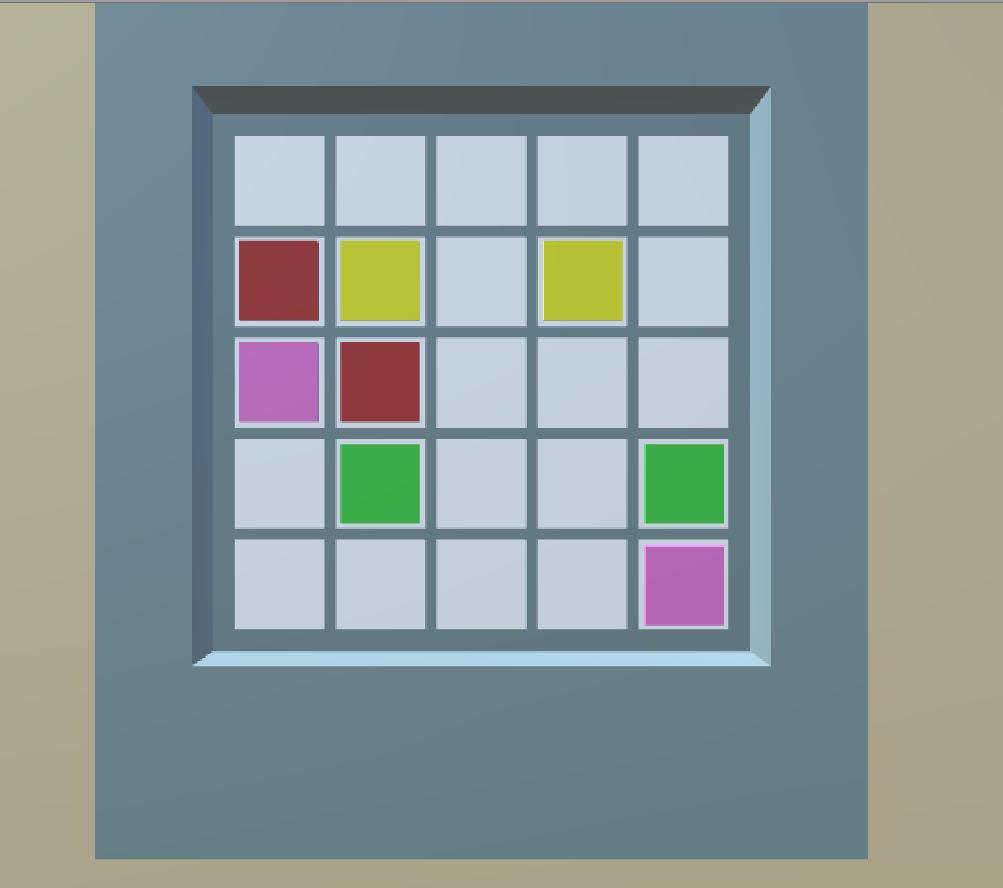


**Interaction:**

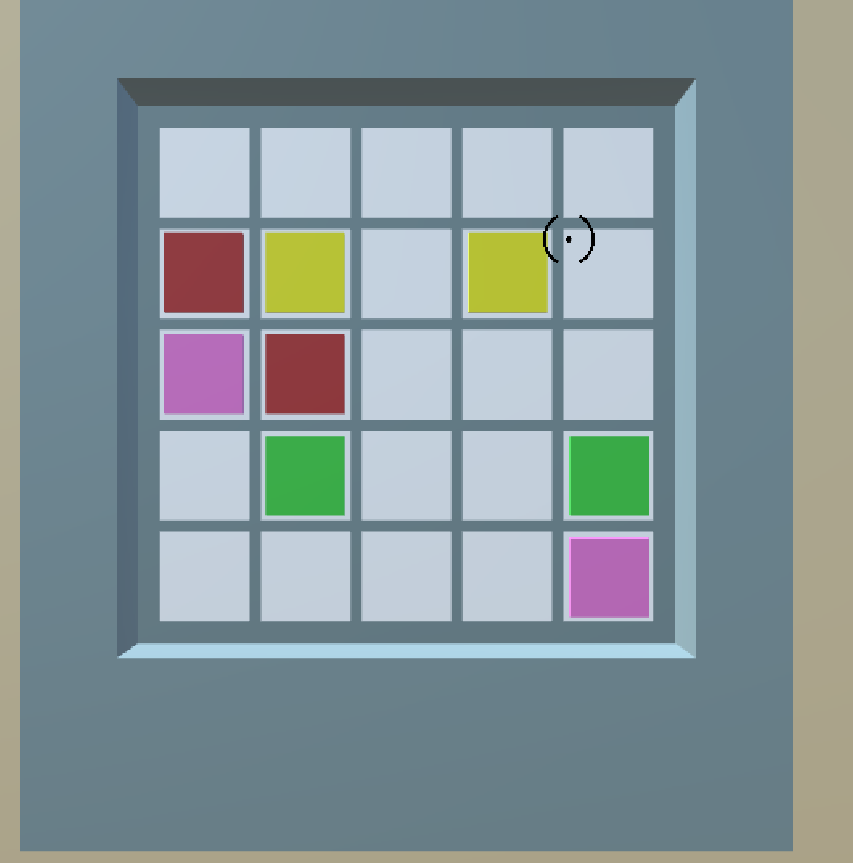
When the UI cursor is positioned over an interactable object, the object becomes lighter to indicate that it can be clicked on and interacted with. Clicking on it allows the player to enter an interaction state with the object.

The functionality of the interaction state will vary for different objects, however the fuse boxes which I have implemented this state for will position the player in front of their puzzles and switch the movement inputs from moving the player to moving the puzzles wires.



When the PC is in an interaction state with an object the camera does not move, and the cursor is set to the player’s exact mouse position on the screen. This is because the player interacts with objects by clicking on parts of them and since the device’s mouse is usually hidden the game needs to have its own one to show the player where on the object it’s currently hovering.



**Testing**

When developing a piece of software, it’s important to submit it to rigorous testing before release to ensure that it does not contain any bugs and to make sure that our program has it’s intended functionality. For my own project, an ‘Escape the Room’ puzzle game, I will want to make sure that it has the base functionality that someone might consider it to be a program of this type. The purpose of this document is to identify and outline the most effective testing strategy for my program at this current state in its development.

**Test Objectives**

The objectives of the testing process are to ensure that the game has reached a certain level of polish and overall quality, to make sure it that functions reliably, how easy the game is to use and how well it performs on different devices. The testing methodologies that I am going to make use of are unit testing, Blackbox testing and performance profiling. These tests will conform to the intended functional requirements of the game to account for the rest of it being currently undeveloped.

When testing the game, I am going to assume that it is currently bug free. Any aspects of the game that are still under development will be excluded from testing since we can obviously tell that they’re not going to be completely functional as they’re not yet finished. The tests will be focused predominately on the game mechanics that I have currently managed to implement, such as movement and object interaction. I will also be testing the game’s performance on my home laptop, comparing how fast it runs while doing different things. This is so I can identify any game mechanics that are visibly slowing the game down and I should therefore go back over and attempt to make more efficient.

**Testing Strategy**

For the development of my program, I am making use of the agile software development methodology, or SCRUM, because it allows for gradual but constant improvements to the software in the form of multiple iterations of it created during ‘sprints’. This ultimately fits my own schedule as well since I am also working on multiple other projects as part of my final year at university.

I will also be making use of the agile methodology during the testing process. This means that I will be able to develop and test my program as I go, making the process ultimately much easier since I may not have yet fully developed certain aspects of my game by the time I want to start the first round of testing. This is incredibly useful, especially towards the end of the game’s development when if I had adopted another testing methodology I might have to do all of the testing then. This method of continues testing also allows for easy identification of code blocks that need to be made more efficient, since I can tell from my previous round of testing that the blocks included then worked fine when tested and the code slowing the game down must have been implemented since then.

Another testing methodology that I will be making use of is exploratory testing. This is because exploratory testing is an extremely useful methodology for use when testing while a project is still being developed. The test cases for exploratory testing do not test functionality that is yet to be implemented but rather once it is implemented. This does however have the unfortunate drawback of the fact that I will have wasted time testing any parts of the system that get replaced during development and do not make it into the final product anyway.

**Blackbox Testing**

Blackbox testing is a type of testing that focuses purely on how the behaviour and performance of a software application when it is given certain inputs. The person doing the tests does not have access to the programs code and thus looks at the program as if it were a simple black box, with inputs going in and outputs coming out and no idea as to how the inputs are being used or how the outputs are calculated. They simply want to know just if certain inputs give the desired output. This also helpfully decreases the number of necessary test cases since we are only going to be testing the final output while if we were testing the programs code we might have had to test multiple functions or blocks of code to see if they give the correct output that ultimately results in the output to the user.

In relation to my program, black box testing will enable the identification of inefficient or broken parts of the game application at its surface level only, since black box testing does not encompass looking through the programs code. The fact that we can create test cases so quickly and keep their number to a minimum hugely aids in the speed of the programs development and it why I have chosen to use it for the testing of this project.

**Blackbox Testing Cases for Current Program State**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Result** | **Expt Result** | **Pass?** |
| 1 | The PC moves to the right correctly | The right movement input is used | The PC moves to the right | The PC moves to the right | Pass |
| 2 | The PC moves to the left correctly | The left movement input is used | The PC moves to the left | The PC moves to the left | Pass |
| 3 | The PC moves forwards correctly | The forwards movement input is used | The PC moves forwards | The PC moves forwards | Pass |
| 4 | The PC moves backwards correctly | The backwards movement input is used | The PC moves backwards | The PC moves backwards | Pass |
| 5 | The PC cannot move through game objects | Attempt to move the player through a game object | The player moved through the walls | The attempt is unsuccessful | Fail |
| 6 | The player can pan the camera to the left | The camera left movement input is used | The camera pans to the left | The camera pans to the left | Pass |
| 7 | The player can pan the camera to the right | The camera tight movement input is used | The camera pans to the right | The camera pans to the right | Pass |
| 8 | The player can pan the camera upwards | The camera upwards movement input is used | The camera pans upwards | The camera pans upwards | Pass |
| 9 | The player can pan the camera downward | The camera downwards movement input is used | The camera pans downwards | The camera pans downwards | Pass |
| 10 | Hovering over a fuse box on the wall causes it to light up | Pan the camera over each of the fuse boxes | Each fuse box lights up correctly | Each fuse box lights up correctly | Pass |
| 11 | Clicking on a hovered over fuse box causes the player to go into an ‘interaction mode’ with the box | Pan the camera over a fuse box and then click on it, test with all fuse boxes | Each fuse box can be interacted with | Each fuse box can be interacted with | Pass |
| 12 | The player can move wires to the right when possible | The right movement input is used, while a movement to the right is possible | The wires can be moved to the right when possible | The wires can be moved to the right when possible | Pass |
| 13 | The player can move wires to the left when possible | The left movement input is used, while a movement to the left is possible | The wires can be moved to the left when possible | The wires can be moved to the left when possible | Pass |
| 14 | The player can move wires upwards when possible | The upwards movement input is used, while an upwards movement is possible | The wires can be moved upwards when possible | The wires can be moved upwards when possible | Pass |
| 15 | The player can move wires downwards when possible | The downwards movement input is used, while a downwards movement is possible | The wires can be moved to the downwards when possible | The wires can be moved to the downwards when possible | Pass |
| 16 | Moving a wire backwards onto a wire its just made will delete that wire | Move a wire in a direction and then move it back in the opposite direction, test all possible cases | The wires cannot be moved like this | Moving the wires in this way successfully deletes wires | Fail |

**Unit Testing**

In contrast to backbox testing, unit testing is a method of testing that focuses specifically on testing parts of code. The person carrying out the tests has knowledge of the programs code and as such can then make predictions about which parts of the game’s functionality may not work as intended. It also allows the tester to isolate certain units of the programs code from others for testing

With relation to my own program as a piece of software for testing, unit testing will allow me to identify which parts of my code are working as fully intended and others that need to be debugged as well as other units that may work correctly but use up enough processing power or memory to have a visible effect on the game’s speed and I should therefore attempt to make more efficient. This is the reason why I have chosen to use unit testing for my software programs development.

**Unit Tests**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Result** | **Expt Result** | **Pass?** |
| 1 | Moving a wire horizontally following a vertical movement creates a corner | Move a wire upwards or downwards and then left or right, test all possible cases | Moving the wires in this way does not create a corner | Moving the wires in this way successfully creates a corner | Fail |
| 2 | Moving a wire vertically following a horizontal movement creates a corner | Move a wire left or right and then upwards or downwards, test all possible cases | Moving the wires in this way successfully creates a corner | Moving the wires in this way successfully creates a corner | Pass |
| 3 | The player cannot tilt the camera upwards so that it goes behind the player | Tilt the camera upwards and see it the game limits how much it can be tilted | The camera does not stop being tilted and does go behind the player | The camera stops going upwards once it’s facing directly upwards | Fail |
| 4 | The player cannot tilt the camera downwards so that it goes behind the player | Tilt the camera upwards and see it the game limits how much it can be tilted | The camera does not stop being tilted and does go behind the player | The stops going downwards once it’s facing directly downwards | Fail |

**Performance Profiling**

Performance profiling is a method of testing which monitors the performance of a given system whilst it is running the subject program. Unlike unit testing the tester has no knowledge of the state of the program’s code base and unlike Blackbox testing we do not care about the result of the testers inputs into the software. Performance profiling only tests how well the program runs during certain different scenarios. This is to test if there are any parts of the code base that use large amounts of processing power or memory and will henceforth have a visible impact on the speed of the subject program.

With respect to my game performance profiling will allow me to determine the speed of the game at different points by using the frames per second, or FPS, metric. The frames per second that my game is running at affects how smooth the gameplay will appear to a user and is especially important when attempting to create a fully immersive experience. This value will be higher when running on devices with better hardware specifications than others, and as such the tested game speed on my home laptop may be slower than the speed a user will experience on a gaming PC however I should still endeavour to keep this value as high as possible to keep the gameplay smooth. Each result is tested three times to ensure accuracy.

**Performance Profiling Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **No#** | **Summary** | **Process** | **Mean Results** |
| 1 | The game’s first frame when the scene is first loaded | Begin the game and then exit as we are only interested in the very start | FPS 55.2, this indicates that the start frame took 0.018 seconds |
| 2 | When moving the player around the game space | In game move the player around the game space | Average FPS 55.6 |
| 3 | When looking around the game space | In game look around the game space | Average FPS 47.2 |
| 4 | When moving and looking around the game space | In game move around the game space whilst also looking around | Average FPS 60.2 |
| 5 | When interacting with the fuse boxes | In game click on a fuse box to interact with it | Average FPS 55.3 |
| 6 | When manipulating the wires inside the fuse boxes | In game interact with a fuse box and move the wires inside | Average FPS 51.3 |

**System Design**

As part of my final year at university I am developing a 3D first person ‘Escape the Room’ puzzle game. I have chosen to call my game ‘The Eagle Hotel’ in reference to the Eagle’s song ‘Hotel California’, *you can check out but you may never leave*, and the game’s Escape Room nature. The aim of the game is to escape from the eponymous hotel by completing the puzzles it presents. In order to solve the game’s puzzles the player will have to interact with objects in the room. Each puzzle is based around an object that the player will have to interact with to use. For example, there are fuse boxes on the wall that the player will have to interact with in order to wire their circuits correctly and complete their puzzles. Once all of the puzzles have been completed the player will be able to leave the room.

**System Design Document**

**Overview**

The purpose of an SDD is to describe the requirements of the subject system, the architecture of the system, the format of the inputs given to the system and the design of the user interface. This information is important because over the course of the program’s development my perception of the kind of program I want to develop will change and having a document like this to refer back to will allow me to stay on track with developing the correct kind of game.

**System Design**

With my game ‘The Eagle Hotel’ I wanted to fully immerse players into the game world and create a need in them to complete all the games puzzles and leave successfully. The aesthetic of the intended game world and the background audio help to create the sense that the player’s environment is unsafe and that they need to leave. This creepy theme also feeds into their experience with the puzzles as I want the player to spend time on trying to complete them as opposed to trying to rush through them or just looking up a walkthrough.

**Objectives of the Design**

- Camera movement and rotation control

- Interaction with game objects

- Smooth gameplay

- Simple UI when needed

Since the game is going to have a creepy aesthetic that some users might find uncomfortable it is important that my game to both a sufficient age rating, as younger users will be more likely to be too scared to play, and also contain content warnings so that others who would be too scared are also warned. I have also built this consideration into the games design, as the creepy aesthetic of the game will be present right from the start so that anyone does find the game too scary and decides that they don’t want to continue will realise this as soon as the game starts and not risk becoming more scared later into the gameplay.

**Design Assumptions**

This game is going to be developed primarily for PC devices and secondarily for mobile devices if it is deemed possible towards the end of development. This is because as the game is being developed on a PC, the process of both developing and testing the game becomes a lot easier since I do not have to copy the game over to another device every time I want to test the game’s mechanics. I could also end up spending much less time making the game’s code more efficient as if no phone version gets made the game will only ever run on more powerful systems which won’t experience drops in frame rate that a less powerful phone might. I have however chosen to develop a mobile version if there is time towards the end of the game’s development as the mobile phone gaming platform has proved hugely successful for puzzle-based games. This is believed to be due to their simplistic nature being suited to their restrictive processing power and memory and also their incredible constant accessibility suits players who work on puzzles in their minds and will want to complete one that they’ve been stuck on once they’ve realised how to solve them. Another reason for choosing to also develop for mobile phones is the ever-increasing mobile gaming industry, which is now bigger than that of the console gaming industry because while the average user spends very little or even nothing on mobile games their ginormous player base ensures that the industry is still making more money overall.

**Constraints**

The game that I am developing will run in 3D. This is has a huge impact on the overall processing power needed to run the game and the rendering of the game world will most likely be the biggest factor in the game’s overall CPU usage. In order to account for this, I will endeavour to make the rest of the game’s code as efficient as possible, so that none of the host device’s processing power is used that does not need to be. This in turn will have a knock-on effect on the amount of game content that I will be able to create as I will have to make sure that anything new I add is able to run at a suitable CPU usage level instead of just carrying on adding things.

The fact that the game is being developed as an element of my university coursework will also have a negative effect on the development process. This is because I will have a constraint on the amount of time that I’m able to spend working on the game, since I have to make sure that I’ve completed the game in time for me to hand it in on the day the coursework is due in. Furthermore, another aspect of this that will add to the time constraint is the fact that I have other coursework that I’m going to be working on at the same time as this. This means that I won’t be able to spend as much of the limited time I have to work on this as I want and the overall quality of the game will be negatively impacted because of this.

**System Architecture**

**PC:**

- Keyboard Input

- W for forward movement

- D for right movement

- S for backwards movement

- A for left movement

- Up Arrow Key for upwards movement when interacting with objects

- Right Arrow Key for right movement when interacting with objects

- Down Arrow Key for downwards movement when interacting with objects

- Left Arrow Key for left movement when interacting with objects

- Mouse Input

- Upwards Panning tilts the game camera upwards

- Right Panning tilts the game camera to the right

- Downwards Panning tilts, the game camera downwards

- Left Panning tilts the game camera to the left

**Phone (Currently Hypothetical as Mobile Prototype Not Developed)**

- Left Joystick

- Horizontal Axis for horizontal movement, right leaning causes right movement and left leaning causes left movement

- Vertical Axis for lateral movement, upwards leaning causes forwards movement and downwards leaning causes backwards movement

- Right Joystick

- Horizontal Axis for horizontal camera rotation, right leaning causes right panning and left leaning causes left panning

- Vertical Axis for vertical camera rotation, upwards leaning causes upwards panning and downwards leaning causes downwards panning

- Touch Screen Input

- Tap and Drag for moving the wires on the fuse box grids

**User Interface**

The user interface of a game refers to how the user playing the game interacts with the game’s software and reversely how the software outputs back to the user. For my game outside of the game’s input and the visual output of the game world I am going to include multiple other UI elements to ensure that interaction between the user and the software is a smooth as possible. The most important of these added elements is the UI cursor. This is an image that when the player is moving around is fixed to the centre of the game screen and serves to indicate to the user exactly where on the screen objects have to be for the player to be hovering over them. When interacting with objects the position of the cursor follows the player’s mouse around the screen. This is so that the player can click on parts of the object, provided that they can be interacted with in this way, and to click on any UI buttons that allow for further interaction with the current object.

The next UI element that will be added to my game is buttons that will allow for further interaction with the game’s objects. For example, when interacting with one of the games puzzles I will include the following buttons:

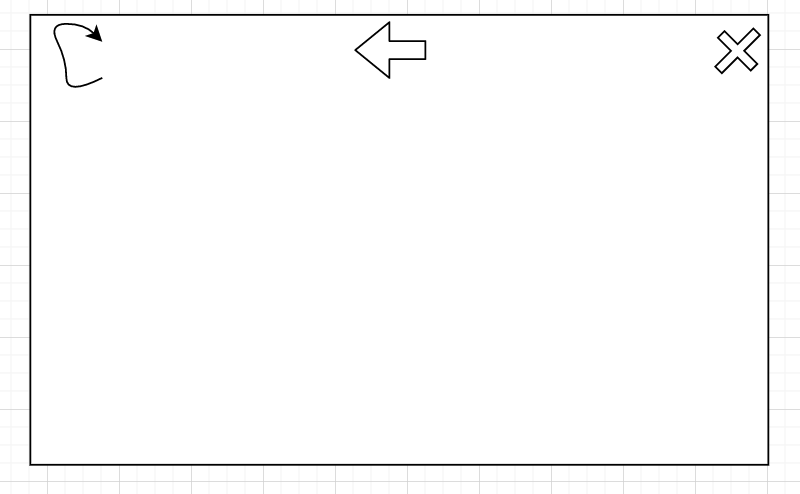
* Reset: A button that resets the puzzle into its original state
* Exit: A button that exits the players interaction mode
* Back: A button that allows the player to undo an action

Outside of object interaction the main game will also feature these buttons

* Pause: A button that pauses the game so that the player can leave and come back or just take a break
* Exit: A button that closes the game application

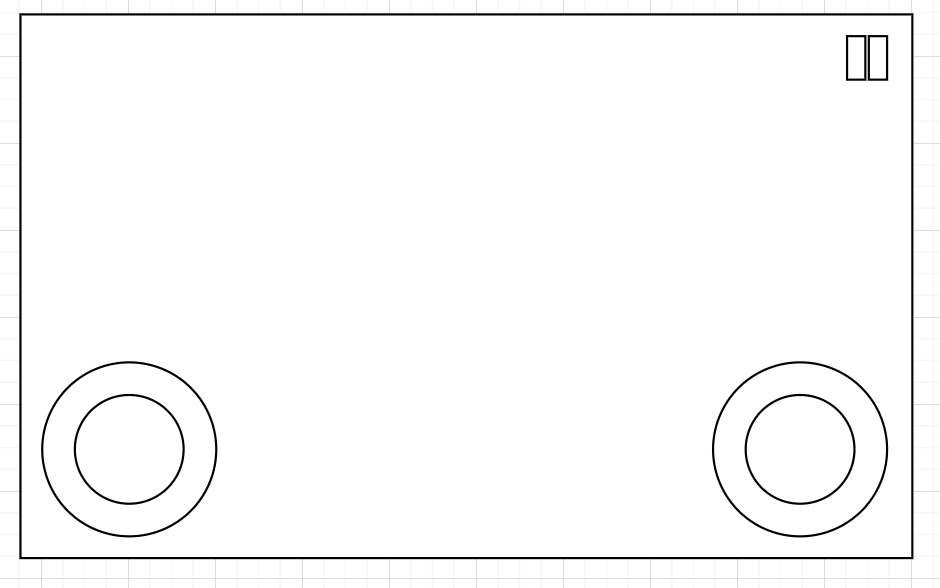
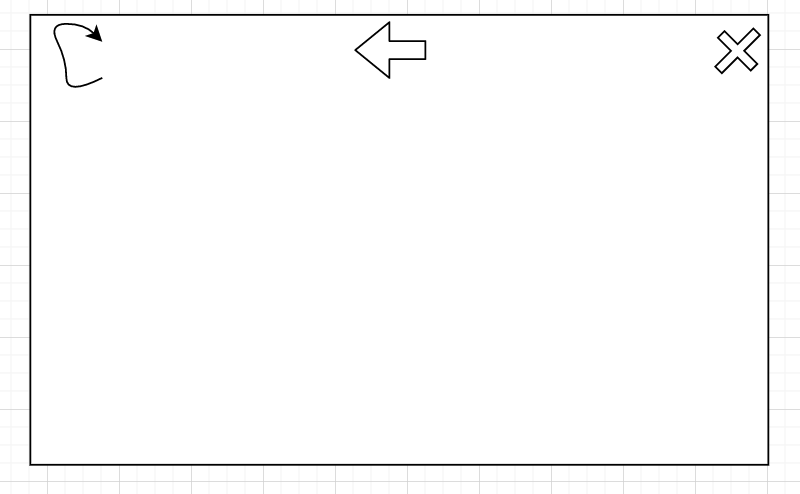
**UI Mock-ups**

**PC:**



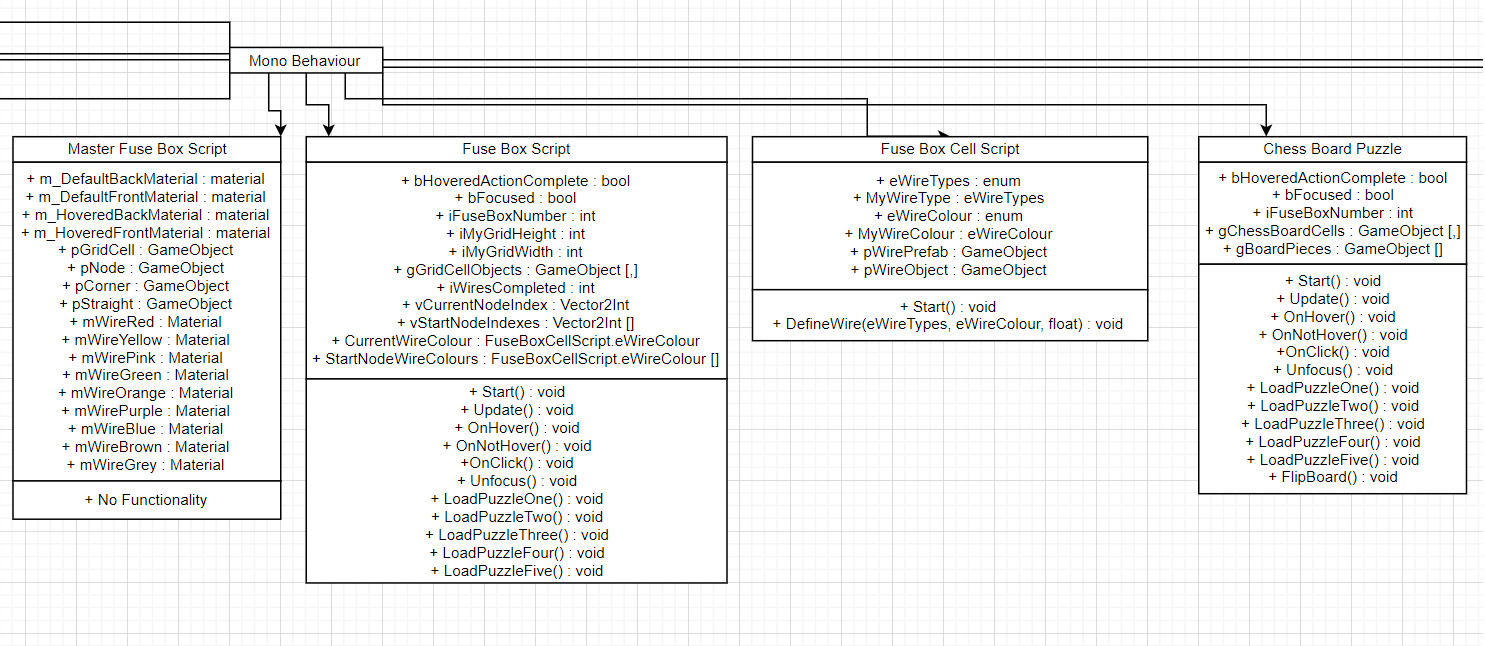
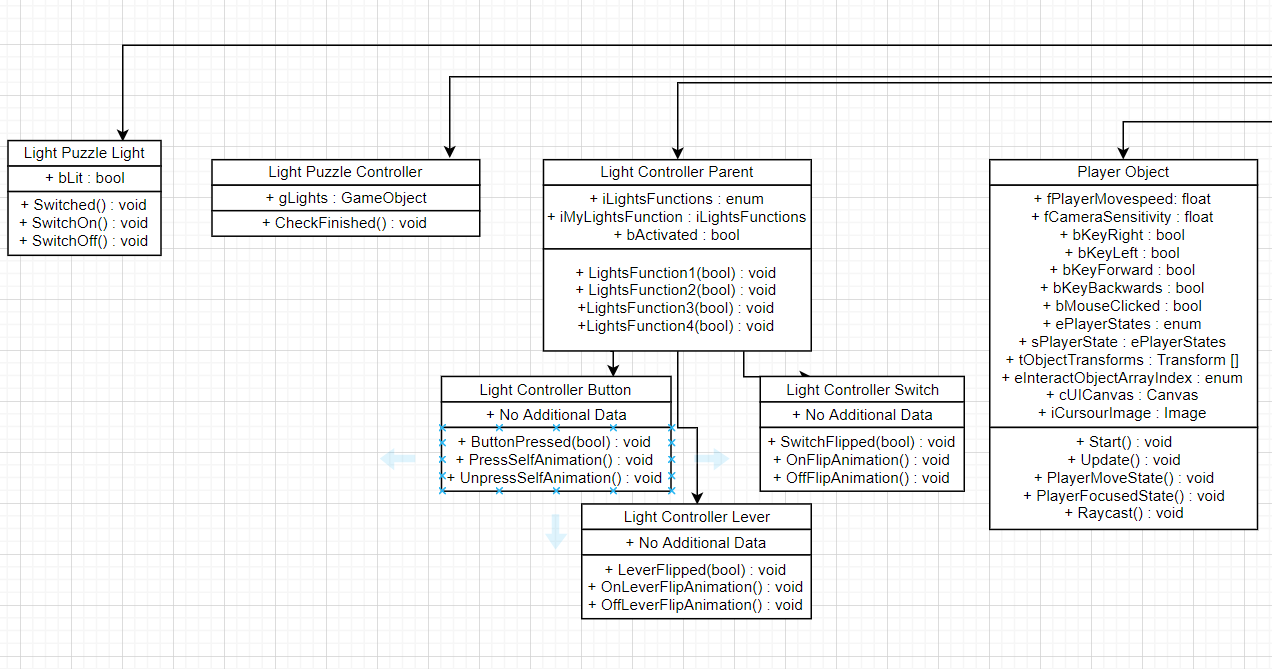
This is a mock-up of the intended UI for the PC version of the game. The diagram on the left shows the UI mock-up for when the player is moving and the diagram on the right shows the mock-up for when the player is interacting with an object. The blank space will show the rendering of the game world. On the general movement mock-up, the Pause and Exit buttons are shown in the top right corner. On the object interaction mock-up at the top of the screen form left to right we can see the Reset button, the Back button and the Exit button.

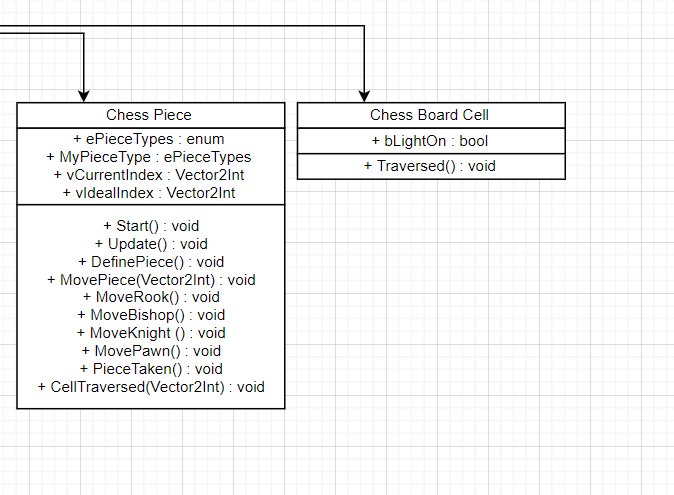
**Mobile:**

****

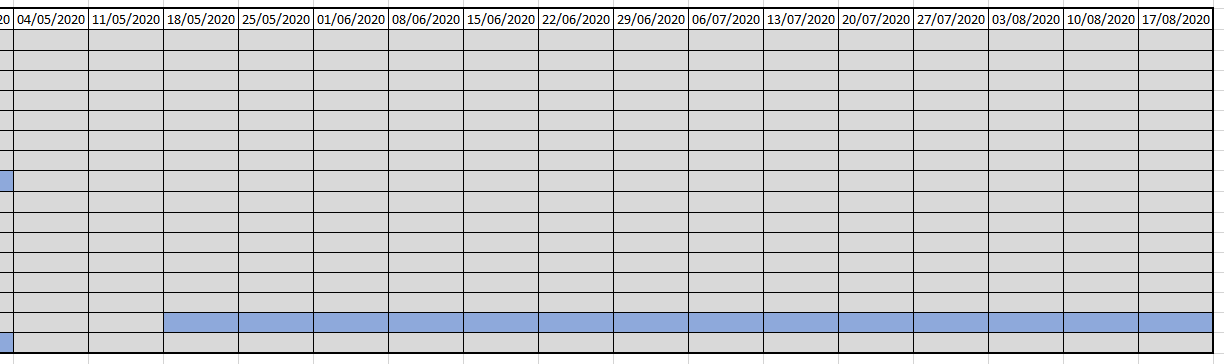
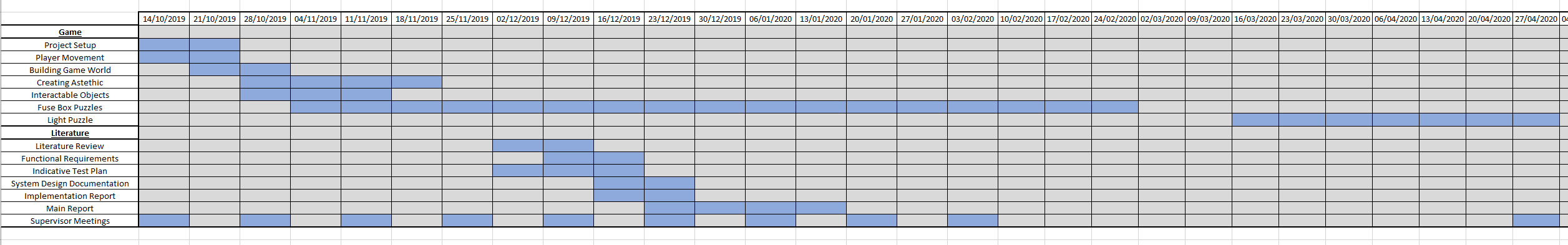
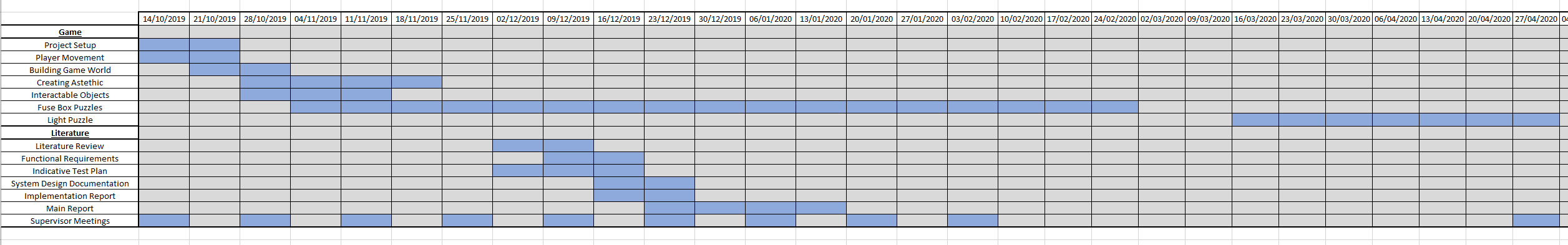
This is a mock-up of the intended UI for the mobile version of the game. The diagram on the left shows the UI for when the player is moving and the diagram on the right shows the mock-up for when the player is interacting with an object. The UI mock-up when the player is moving around has two joysticks on the bottom of the screen and the Pause button in the top right. The left joystick is for moving the player and the right joystick is for moving the player camera. On the UI mock-up for when the player is interacting with objects on the top of the screen from left to right we can see the Reset button, the Back button and the Exit button.

**System UML Diagram**





**D)** System Gantt Chart



**Figure 17:** The updated Gantt chart for the project that shows its development over time