University of Kent

THE ESCAPE ROOM
PROJECT REPORT

Tomas Valevicius (tv73)

BSc Multimedia Technology and Design (Honours)

EL636: Final Year Project

Word count: 5007

April 12, 2019

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped me during the development of this project, and the entire course, including but not limited to, professors, lecturers and tutors who taught me many new skills and helped me understand myself better. Thank you to my project supervisor and academic advisor for their willingness to help. Thank you to all participants who gave their time and honest feedback during the evaluation process.

ABSTRACT

This project aimed to explore the concept of escape room games and, combined with elements from traditional adventure puzzle-solving games, create an experimental version of a game that would contain a mix of these features.

The project consists of a research section where a few such video games have been analysed, and a review of the attributes of escape rooms. The development section lays out the process of the game's creation, including its design, asset creation and coding. Finally, a user evaluation was performed to understand which features worked and what could be improved. The results showed that visuals were important, but a few interaction and story elements could have been ameliorated.

Lastly, the conclusion summarises the key takeaways from this project, including the wins and shortfalls of the game and a newfound appreciation for the complexity of video games gained during the project.

CONTENTS

INTRODUCTION

The aim of this project was to explore the concept of the escape room game, as well as the genre of adventure and puzzle games, and to create one drawing inspiration from other well-known and successful games of a similar genre. Specifically, the goal was to familiarise with the creative and technical aspects that make a good puzzle game, and create an experimental game where ideas from both the real-life escape rooms and video games could be combined.

Given the lack of experience and limited ability possessed for this project, creating a polished, professional video game was not the first priority, although the aspiration for that did exist. The entire project was considered as a learning opportunity, a safe space to try out new ideas, be creative, and learn more about the Unity engine and scripting in C#.

Finally, a video game could provide useful experience in a wide range of fields including scripting, 3D modeling, animation, art and many others. A well-rounded learning experience can teach many useful skills during a project such as this.

RESEARCH

Several games of a similar genre have been analysed to get a better understanding of what makes a good puzzle game. In addition, real-life escape room concepts were explored, including their origins, what makes them popular, and how they work.

Myst

In 1993, a video game by the name of Myst was released for the Macintosh platform. It completely changed the way video games were perceived. Before the arrival of Myst, video games were viewed as low-class entertainment, filled with violence and gore. Myst was something different - it was a game that brought a new art form that was visually appealing and simplistic. It was enjoyed by many audiences, especially people who had never played video

games. In fact, it is widely believed that Myst was the game that brought the term “casual” in the gaming community (Andreadis, K., 2014).

Myst was admired as a game that would change the future of adventure games. Wired Magazine called it “[…] beautiful, complicated, emotional, dark, intelligent, absorbing. It was the only thing like itself; it had invented its own category,” (Carroll, J., 1994) while The New York Times wrote that the game “seems to reflect the condition of the video game itself, poised at the brink of something new even before it has finished mastering something old.” (Rothstein, 1994)

One of the reasons Myst may have been such a huge success is because the creators, Rand and Robyn Miller, were not actually big fans of video games. They did not even have a key demographic for the game they were making. In fact, Myst was not really a game in their minds but an interactive experience that could be enjoyed by non-gamers (Adkins, J. G., 2018). Perhaps the knowledge and creative talents of a few people outside the gaming industry was the key to making such a unique game that attracted massive audiences and challenged the gaming conventions of its time. They brought fresh ideas from their own respective fields that helped shape Myst into the game it ended up becoming. This could be an indication that having someone with knowledge in other areas can be incredibly useful for making a successful game.

One of the most stunning elements about this game was the visuals. When Myst was released, its graphics were state-of-the-art, even called photorealistic at the time. It was able to immerse the player into the virtual world, something no other game was able to achieve. The slideshow-style navigation became one of the game’s most recognisable features. Each of the thousands of computer-generated images were incredibly detailed that could hide hundreds of clues and items. One could argue that visuals are not everything in a video game. While that may be true, an image is a powerful tool, and the never-before-seen graphics and the attention to detail was one of the reasons that made Myst exceptional.

Myst consisted of puzzles that were notorious of being so difficult that many people never even finished the game. Many of them were obscure and unnecessarily long. One thing common in puzzle solving games and escape rooms is the concept of finding parts of a larger whole, collecting them, and using them in a bigger puzzle that would act as a final obstacle in the game. Myst was riddled with puzzles such as this, which made the game extremely challenging where only the most dedicated would be able to finish it.

Researching Myst provided some useful guidelines when developing The Escape Room. Using a game that is so revolutionary as a foundation for this game seemed like a good way to start off. These guidelines may not be the most ideal, but at least they help develop a sense of how any puzzle game should look and feel like.

First, the visuals had to be realistic enough to provide the same or a similar immersive experience, otherwise the game risked of being bland and not captivating, even if the level design was good. That is why the textures used for this game were of high-quality and detailed rather than simplistic. This does not mean that the same effect could not be achieved with simpler graphics, nor that visuals alone will make the game interesting. It simply means that the process becomes more difficult without changing the visuals, and then the challenge is to look for other ways to make the game immersive.

Second, the puzzles would be created in a way that would be short and relatively easy to solve. They would provide instructions that were clear but not obvious. This way the player would still have a challenge in front of them, but not something requiring note-taking or hours of guessing. The aim became to maintain the difficulty at a moderate level – not too difficult as seen in Myst, but not too easy where the game would become boring. A pleasantly engaging and mentally stimulating experience was the goal.

The Room 3

Released for mobile platforms in 2015 and for Microsoft Windows in 2018, The Room 3 is the third installment of The Room series produced by Fireproof Games.

The puzzles in the game are a lot simpler compared to Myst, requiring the player to perform rather simple actions such as clicking buttons, turning switches and solving easy puzzles that are very well animated. They are still very satisfying to solve - watching the environment change at the flick of a switch or by inserting an object in the right place gives the player a sense of power.

One interesting feature in this game is the eyepiece, which allows the player to zoom in the tiniest parts of a puzzle like inside a lock, only to find another puzzle inside. This little tool adds a whole new dimension to the game.

The multi-layered puzzles hidden in one object a big aspect in this game – the player can spend over half an hour analyzing something, often finding more than meets the eye, going deeper inside, and unlocking more pieces of the same item until, finally, the final piece is solved and the player can progress.

While it would be interesting to have a similar tool in The Escape Room, the idea seemed far too complicated and outside the abilities possessed at the time. In addition, copying these concepts was also not the intention, although it would have provided a good learning opportunity. But the concept of puzzles consisting multiple layers was considered, given that it was something that could be achieved within the given time-frame.

Escape rooms

The concept of Escape rooms originated in Japan when in 2007, an adventure game-inspired escape room called *Real Escape Game* opened its doors. But later the fad spread through other parts of the world in 2010s. They come in many different themes, ranging from mysterious labs and ancient tombs to cramped city apartments and modern offices (Sally, F., Marmor Shaw, J., 2015).

Escape rooms use many different methods of creating their puzzles and hiding clues. Obviously hiding objects is important, but some techniques involve using light cleverly in the room, such as having clues appear in UV light, substituting symbols, riddles, ciphers, sounds and basic calculations. Another way is to hide something obvious in a location easily accessible, the meaning of which is only revealed at a later point. Some of these tools are more common than others as using certain techniques can be risky – not everyone can solve a cipher. But there are a variety of ways that can make an escape room interesting (Pedersen, F., 2016).

One of the main differences between a video game and an escape room, is that inside the escape room everything occurs inside an enclosed space. All the puzzles and clues are more contained which makes the game more manageable. This is perhaps why they are so addicting – solving puzzles that are just at the right difficulty and are not too long can feel satisfying, especially when the players know that all the clues they need are nearby, they just have to look for them. In

addition, escape rooms heavily rely on teamwork, so these games provide a casual and entertaining way to spend one's time with their friends.

Some of the ideas used in escape rooms were considered for this game. These include the use of lights as parts of the puzzles, hiding obvious items around the room and identifying patterns. There are many more interesting concepts escape rooms have that could be explored in further detail to seek inspiration for a puzzle game.

DEVELOPMENT

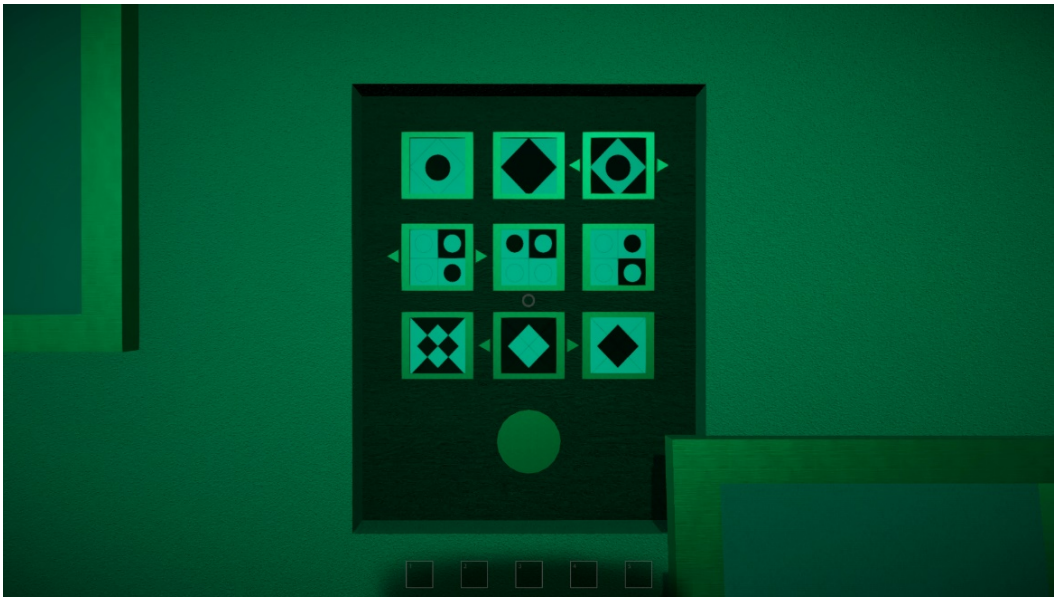
Design

The first stage of development was the design process, which was important for identifying what the initial project would look like. This involved making concept art, designing puzzles, the interaction system, and game progression. This section provides insight into this part of development and describes the thought process.

Initially, the idea for the game was not to make a set of puzzles for the player to solve, but to create a sequence of smaller obstacles the player would have to pass in order to progress and reach the end. This would mean using custom-made tools and thinking outside the box to pass through obstacles. For example, using tape and cutting tools to repair objects, or using fire and alcohol in order to burn something, perhaps even make a torch to help the player navigate (Fig. 3). While these ideas sounded interesting, they proved to be too ambitions and tedious as they would require much greater amounts of 3D modeling, texturing and scripting to make them work.

Therefore, it was decided that the game would involve a set of puzzles that would have to be solved in order to win. The idea was to make at least three puzzles that would allow the player to proceed through the game once solved, either by providing an item for a further puzzle, or unlocking a new section of the game.

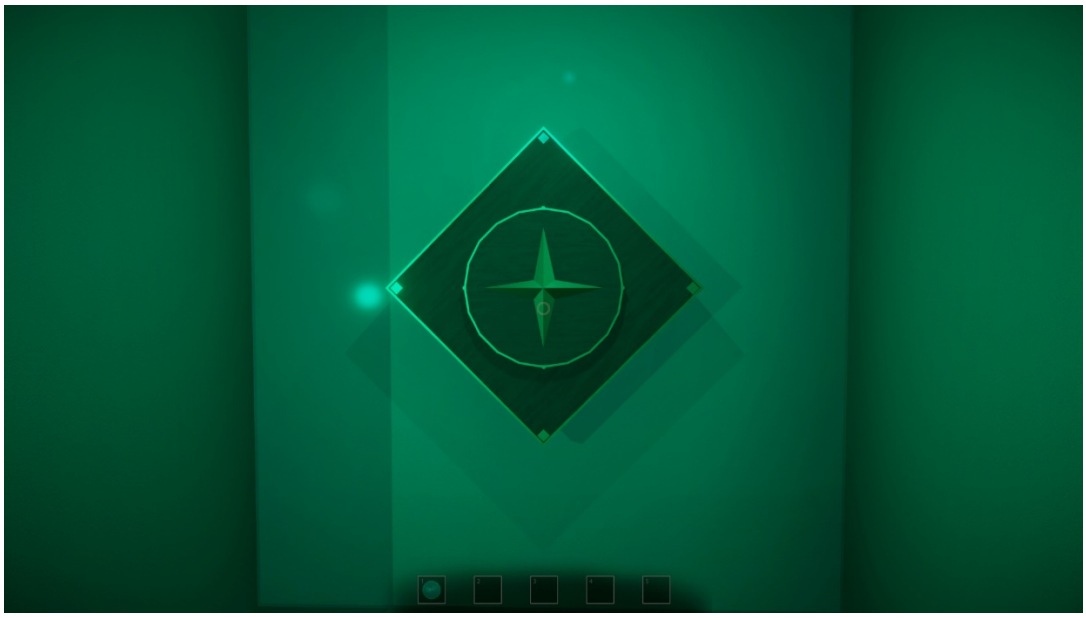
The puzzles took over a big part of the design process. Some of them also underwent multiple stages before being appropriate enough for the final game.



The Image Puzzle

The Image puzzle (Fig. 4) was influenced by The Room 3 that had been researched previously. One of the puzzles in that game contained a puzzle where the player had to find the missing emblems and place them on the wall. After that, the player needed to rotate the pieces in order to get the correct pattern. There were four sets that needed to be solved, each having their own pattern. If the player got all the patterns correctly, the puzzle would give them an object.

In a similar way, the Image puzzle followed that mechanic, but instead of using emblems, it would have images made up of shapes that would follow a certain logic. The puzzle would be split into three rows, and each row would have its own pattern. The player would have to identify this pattern in each row, and change images to the ones that followed the pattern. If done correctly, the puzzle would provide the player with an object required for the next puzzle.



The Colour Puzzle

The Colour Puzzle (Fig. 7) was the last puzzle to be made, and it followed a similar pattern of the Image Puzzle, but instead of using images, it used colours. It was made of multiple tasks or layers: the first was to find a way to open the door using the gemstones hidden throughout the room; the second to match the colours with the positions of the gemstones; and the third was to press the button that would release the platform in the ceiling. There was an idea to make a fourth layer that would conceal the centre button even further, but more layers meant physically smaller puzzles, so it became clear that there would be no more than three tasks.

The Platform Puzzle

The Platform Puzzle (Fig. 5) was the most difficult one to make, and it underwent the greatest number of changes until the final version was designed. Originally, it was going to be more complex when first created. It was supposed to be a circular shape made up of several different platforms that would have to be enabled by placing pieces of a disc on a table nearby. The disc was supposed to represent the same pieces on the circular shape. Once all the pieces were collected, the platforms would rise from the ground, but at different heights. The final task the player would have to do is to level out the height in order to press the button hidden in the centre that would unlock the door. This puzzle was problematic due to its complexity, and implementing it became too time-consuming. The decision was made to leave whatever had been completed and simplify the solution. Instead, three spheres would have to be brought to the table and would have to be touched a certain number of times in order to solve the puzzle.

One of the problems that occurred during this stage was that the puzzles were not being included in the design of the environment. The reason was that the puzzles themselves had not been fully designed at the time and it was important to have a space where they could be tested. In the end, the puzzles were simply integrated into that space later on with a few adjustments being made to accommodate them.

A key feature that distinguishes this game from most of the escape room games is the navigation. Games of this sort of genre normally have a point-and-click navigation system, meaning all movement and actions are performed with a mouse instead of a keyboard. This game aimed to challenge that idea and provide a simpler method of navigation that was less constrained. Instead of using the mouse, the player can move using the keys as in a traditional first-person game.

Once the navigation was made, it created an interaction problem. The player needed to be able to pick up and use objects, and that is easily achieved in a point-and-click interaction system. However, this becomes more difficult without a mouse pointer. To solve this issue, a custom pointer was designed that would sit in the middle of the screen and would act not only as a guide for aiming, but also as an indicator – it would light up whenever a player was looking at an interactable object. The downside of this solution is that the pointer may be a nuisance to some players as it could appear “in the way”. To mitigate this problem, the alpha of the sprite was reduced so that it would blend in with the environment somewhat, however it is worth noting that the problem may still remain.

Lighting was used not only as a basic source of illumination but also as a visual aide that would guide the player towards any points of interest. This is a very common tool in video game design, as light can show (or hide) any objects or locations that could potentially be useful, or if something needs to stay hidden.

The section of the game that ultimately became the ending underwent a few changes. Initially, the idea was to follow a more traditional style of an escape room by providing the player a sense of freedom at the end. After performing user evaluation, the concept of flipping the ending completely and turning it into a trap became more interesting. It was decided that instead of letting the player out, they would be met with a surprise ending where the moving walls would attempt to kill them. This way a story element was added to a game, something that potentially could be further developed in a future version.

3D modeling & texturing

The majority of the 3D models were created using Autodesk Maya, with only a few custom models downloaded from the Unity Asset Store (they can be found in the Appendix).

Most of the 3D assets were textured using PBR (Physically Based Rendering) materials. These textures are made in a way that react to light more accurately than standard textures. They are normally made up of several different layers, or maps, including but not limited to: albedo, normal (bump), roughness, displacement, and ambient occlusion. When combined together, these layers create a material that accurately responds to different lighting settings and the result is a much better-looking object. The downside to these materials is that they usually come in very high quality, often in 2K or even 4K resolution as they tend to be used in realistic 3D renderings. This might create unnecessary drag on the machine when they are excessively used in a game that may not require such high-quality textures.

Coding

Implementing most of these designs required heavy use of code. This section aims to explain how each feature was realised, and includes an explanation of several new coding techniques that were used to make these elements work. The code itself can be found in the Technical Documentation appendix.

1. Navigation & Interaction

To create a functional interaction system, there were several obstacles to overcome. First, the game had to identify objects that the player was looking at during gameplay. Then, it would have to check whether the current object being observed was interactable. Only then would the player be able to pick up or use said object.

The identification problem was solved using a ray cast. In Unity, as the name implies, a ray cast acts a beam (or ray) that is transmitted from one point and lands on another. For this particular problem, the ray would be generated inside the camera object and would beam into the 3D space of the game. This means that it would land on the object that the camera is centered on. Once the ray landed on an object, all the information about it could be retrieved. In order for this to work properly, the ray would have to be cast every frame so the information would be constantly up to date. That is why it had to be inserted into the Update function of the script.

If a ray landed on an interactable object, the cursor would turn white, indicating that an object had been located. At the same time, the object would be stored inside a variable called “interactable” so that it could be accessed by other scripts if necessary.

Once a ray landed on one of the interactable objects, a mouse click would be detected from the player and a Boolean value would be set to true, letting the script know that the mouse had been pressed. Storing this value inside a Boolean helps simplify the process as it does not require to write the input code every time.

2. Inventory

The inventory was perhaps the most challenging part of the whole development process. It became clear that it would require several scripts in order to function properly. The code used for this inventory was written by another developer who is credited in the Third-party assets section of the appendix.

3. Items

As part of a functioning inventory, there had to be a way to store items inside it. A new item object had to be created that would have its own properties such as item name, ID, icon, and the actual prefab that it referred to. This was achieved by creating a scriptable object named Item. It contained all the fields mentioned above, as well as a Use function which would utilise the item in some way. This allowed for the creation of individual Item objects inside Unity that had their own individual properties. So, when a player picks up an object, the inventory will display the icon of the item object that is linked to that object, and store the name and ID for later use.

4. Puzzles

The challenge with the Image Puzzle was to create code that would work across all the image frames, so that there would be no need to write code for each individual frame. That is why it was important to separate each frame from one another by using them as separate game objects

in the script. That script would then use those game objects individually while constantly checking if all three are displaying the correct images. If so, the puzzle would be solved.

The Colour Puzzle, due to its multiple layers, was split into two scripts - one for the door and the other for the colour task inside. The scripts had to be attached to different game objects in order for the entire puzzle to work. Most of the other mechanisms worked in the same way as in the Image Puzzle.

The Platform Puzzle was difficult not only in terms of design, but also in implementing it. As mentioned, it had undergone several stages of development, and was made up of many layers to work with. Enumerators were used to create the sphere blinking and light flickering. Creating the count mechanism was difficult and, just like in the Image Puzzle, it was important to separate the spheres by giving them their own objects. That way, the same script could work on all three of them.

5. Animations

Implementing the animations, however simple they may be, proved to be a difficult step. Some of them worked well by using a simple trigger that was called as soon as a condition was met, for example, if a player solved a puzzle. However, other animations, such as the ones in the final room with the moving walls, had to be called after a certain amount of time had passed. In order to achieve this effect, an Enumerator was used to create a sequence of actions to be performed combined with a wait function where the program would stop running for a few seconds before continuing. This provided an easy way to play the animations at a more desirable time and create pauses between them.

CRITICAL EVALUATION

Because the development took longer than anticipated, the testing could only begin at a later stage when significant portions of the game had been completed. It was assumed that certain aspects would be evaluated prematurely given that they were incomplete, so the plan was to seek

critique on what had already been fully implemented and ask for guidance on the soon-to-be-made elements.

The testing began when most of the environment had been build and the Image Puzzle was working. The other puzzles were explained in detail and their designs were shown.

The evaluation process was made up of a series of gameplay tests involving 5 participants. They were given the chance to play the game uninterrupted while explaining their thought process. Once they finished playing, the participants were allowed to provide any useful tips they had in mind. The results are shown below:

The positives:

- Good visuals, especially the depth of field and darkening around edges of the screen;
- Good quality 3D models and textures;
- Animations make the puzzles a lot more interesting;
- The environment has an interesting design, especially the lighting in the ceiling;
- Sounds make the game more immersive, ambient noise is a good addition;
- Easy navigation.

The negatives:

- Cursor too small;
- Puzzles can be too difficult to understand for some people;
- Ending not interesting/satisfying;
- Clicking on objects to interact with is difficult;
- Level feels too empty.

In addition, a few more general tips were given:

- More levels/rooms would be a good addition;
- A story would give the game more depth;
- When creating the inventory, make it intuitive;
- Create a menu or some way to begin the game;
- Add a hint system so the player can ask for help.

The findings showed that some issues had been expected such as puzzle difficulty and lack of detail in the environment. While some negative aspects were mitigated, others were not able to be completely dealt with as the game was already at an advanced stage and time did not allow to solve these issues.

The negative aspects were attempted to be solved by tackling each issue one by one:

- The cursor was enlarged so it was clearly visible.
- While the Image puzzle was not able to be simplified anymore after its completion, the other puzzles were, as described in the development section. In addition, a hint system was implemented that gave a 'nudge' to the player in case they got stuck.
- The ending was changed so that it had a more surprising finish, again as described in the development section.
- This was mitigated by increasing the range at which the player was able to detect interactable objects.
- A few extra decorations were added to the level, although creating new 3D models would have been ideal.

CONCLUSION

The aim of this project was to make an experimental game using ideas derived both from real-life escape room games and traditional puzzle-based video games to see how these worlds are similar to each other.

There were many new lessons learned while developing this project. The most important one is perhaps a new appreciation for the complexity of video games, and the time and effort it takes to make one. Video games require expertise in both artistic and technical fields in order for them to be successful, and it can be extremely challenging for one person to undertake all these tasks.

From an artistic point of view, it was difficult to create ideas that seemed interesting and were also feasible to achieve within the time given. Sometimes the lack of ideas was not the issue, but

the confidence in one's abilities was. Having made creative projects in the past, this comes as no surprise. Although sometimes it can be useful to rationally identify one's limits and not get overly ambitious.

That is why doing research proved to be a helpful way to mitigate that – not only to understand how previous games were made, but to see how the creators of those games also felt the same way, and how they overcame those doubts, sometimes in the presence of insurmountable pressure. In fact, it would have been better to have done a greater amount of research for this project.

Regarding the game itself, there were parts of it that worked well: the puzzles seemed challenging and interesting, based on user evaluations; the visuals were definitely appealing and had a certain charm; a sizeable amount of 3D models seemed to fit in the environment well.

One aspect that could have been improved were the interaction system, specifically the inventory which still felt a bit too difficult to understand how to use. Using a mouse for all navigation and interaction would have simplified matters, but the intention here was to not make this a traditional point-and-click game. A better way of interacting could have been found.

Another shortfall was the lack of any story in the game that could give it more meaning. The ending does provide an inkling of there being more to the game than what is on the surface, but that idea could have been explored further. More items could have been added in the room, such as notes or books that perhaps contained a dark secret the player could discover.

References

1. Cyan, Inc. (1993) *Myst.* Video Game. Mac OS. Broderbund Software, Inc.
2. Adkins, J. G. (2018) *Two Histories of Myst.* Medium. [online] Available at: https://medium.com/picking-up-the-pieces/two-histories-of-myst-8b37e1504f9e#eb59 (Accessed 12 Feb. 2018)
3. Andreadis, K. (2014) *Revisiting Myst: One of The Most Successful Games Ever.* IGN. [online] Available at:

https://web.archive.org/web/20150306205329/http://www.ign.com/articles/2014/04/28/revisiting-myst-one-of-the-most-successful-adventure-games-ever (Accessed 12 Feb. 2018)

4. Carroll, J. (1994) *Guerrillas in the Myst.* Wired Magazine. [online] Available at: https://www.wired.com/1994/08/myst/ (Accessed 12 Feb. 2018)
5. Rothstein, E. (1994) *A New Artform May Arise from the 'Myst'.* The New York Times. [online] Available at: https://www.nytimes.com/1994/12/04/arts/a-new-art-form-may-arise-from-the-myst.html?pagewanted=1 (Accessed 12 Feb. 2018)
6. Fireproof Games. (2018) *The Room Three.* Video Game. Microsoft Windows. Fireproof Games.
7. Sally, F., Marmor Shaw, J. (2015) *The Unbelievably Lucrative Business of Escape Rooms.* Market Watch. [online] Available at: https://www.marketwatch.com/story/the-weird-new-world-of-escape-room-businesses-2015-07-20 (Accessed 4 Feb. 2018)
8. Pedersen, F. (2016) *101 Best Escape Room Puzzle Ideas.* [online] Available at: https://nowescape.com/blog/101-best-puzzle-ideas-for-escape-rooms/ (Accessed 4 Feb. 2018)

Appendices

Appendix 1: Concept Art

Artwork was made to get a better visual grasp of the environment and what would be the theme of the game. The decision had already been made to have a realistic style, at this point it was a matter of determining how the room would look like, and how it would make the game more appealing.

Fig. 1: One of the first ideas was inspired by The Room 3, where the environment is mostly dark and moody. In this environment, lighting would play a more important role in guiding the player around the room, and the atmosphere seemed right for a puzzle game such as this. The layout seemed adequate – it was not too big, although the dark colours may have made the room feel smaller, which is why the final design became lighter. The window on the left was supposed to provide some natural light, but due to problems implementing it the idea had to be scrapped.

Fig. 2: The second design was more akin to the real-life escape rooms that often look like everyday environments, but a bit older and washed out. The idea was to hide puzzles behind ordinary objects and let the player find them. More important objects would be well-lit and placed in easily reachable areas of the room. In the end, the design seemed too boring on its own, so then it was decided to combine the two artworks and have something that would have elements from both concepts. The result became similar to what ended up in the game.

Appendix 2: Notes

Below is a collection of some of the notes that were taken during the design and development stages. Please note that these notes may be difficult to read as they were only intended to be as a personal aid. They were included here to provide more details about the creative process. The notes might be easier to see on a digital version of the report.

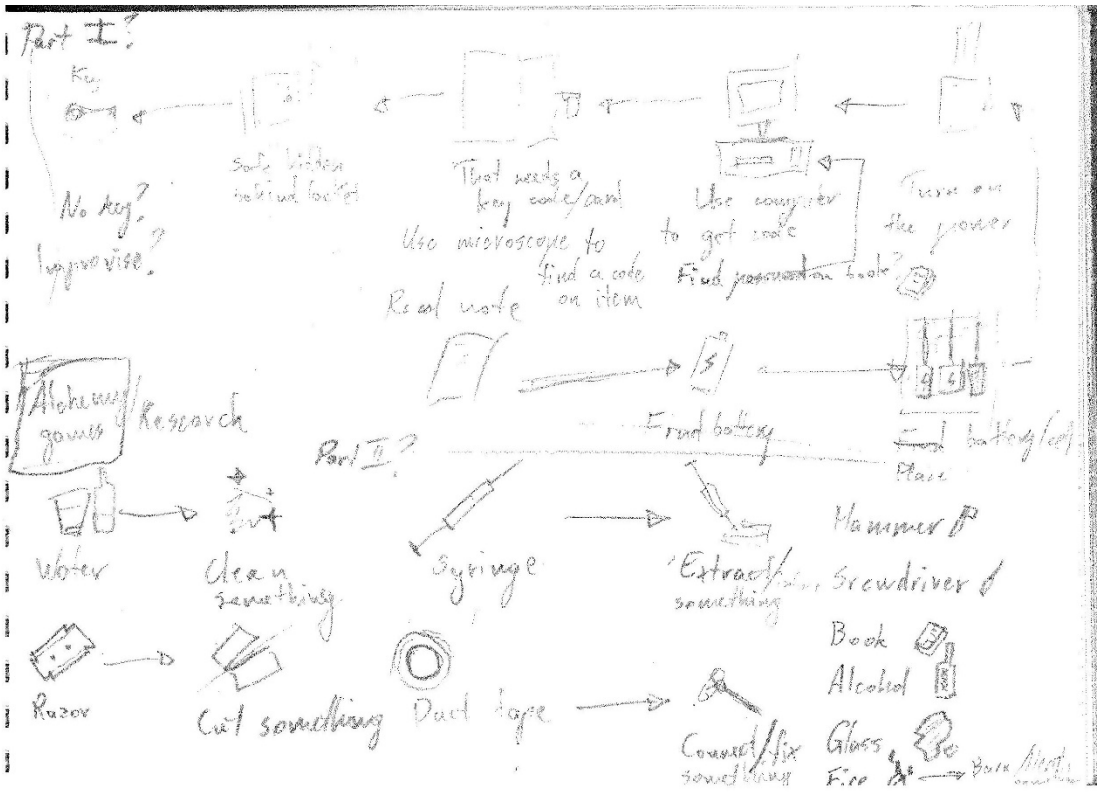


Fig. 3: This was part of the thinking and brainstorming process during the development of clues and puzzles before the final version of the game was decided. Many of these ideas did not reach the final stage as they were found to be too tedious, too difficult or too time-consuming to be considered for development. Concepts explored included step-by-step game progression, items and props, their interactions, uses and purposes.

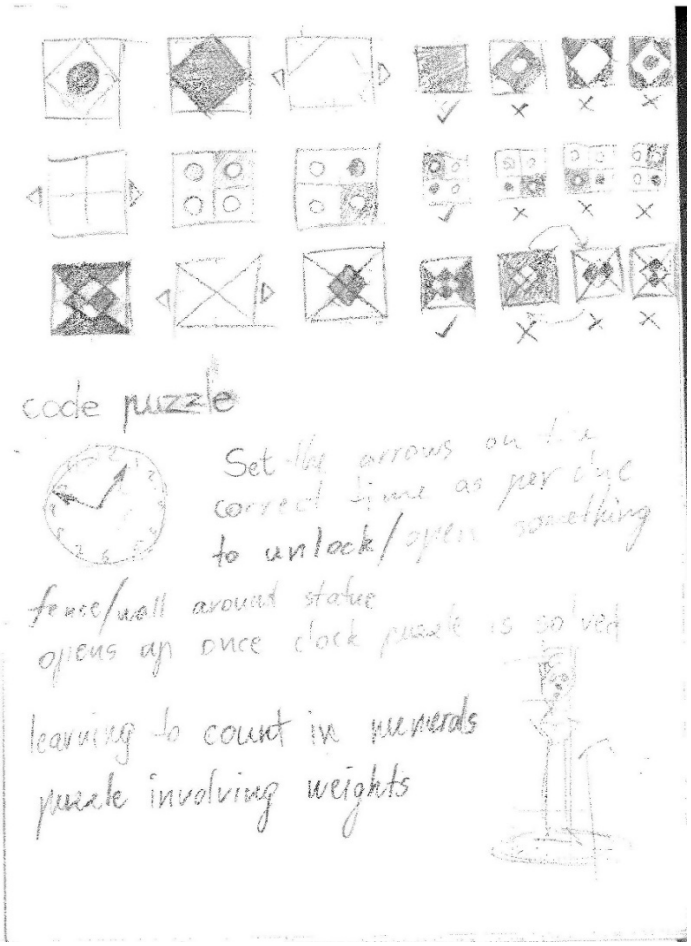


Fig. 4: One of the puzzles that did make it to the final game was the so-called Image Puzzle. Above is a drawing of its design, including all the different images, only one of which is correct in each row. The puzzle was designed in a way that the player would be able to recognize a pattern in each row, and the correct image would follow the logic of the other two images. Below the puzzle are a few ideas for a few more puzzles that did not reach the final game. One included a statue that would be closed off and would have to be accessed by turning they arrows on a clock to a particular time. After that it would have to be moved to a particular location in the level to perhaps unlock a new puzzle or even as part of the ending. The main problem was that it lacked any purpose, and having it in the game seemed pointless unless a good explanation was found, i.e. a story. It made matters more complicated and difficult to handle, so at the end the idea was abandoned. A similar problem was found with the other puzzles noted here.

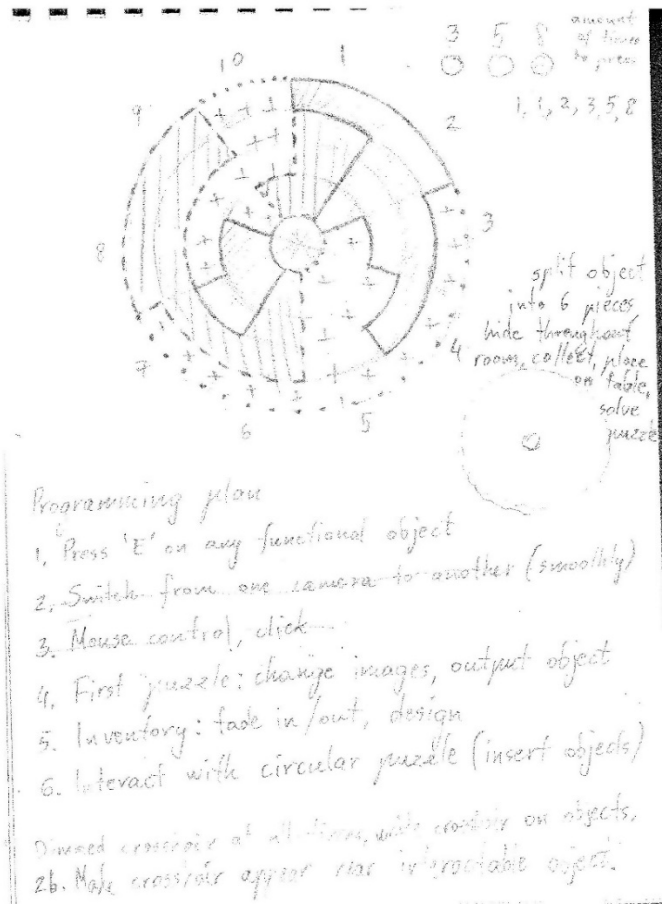


Fig. 5: The final puzzle, or the so-called Platform Puzzle was going to be more complex when first created. It was supposed to be a circular shape made up of several different platforms that would have to be enabled by placing pieces of a disc on a table nearby. The disc was supposed to represent the same pieces on the circular shape. Once all the pieces were collected, the platforms would rise from the ground, but at different heights. The final task the player would have to do is to level out the height in order to press the button hidden in the centre that would unlock the door. This puzzle was problematic due to its complexity, and implementing it became too time-consuming. The decision was made to leave whatever had been completed and simplify the solution. Instead, three spheres would have to be brought to the table and would have to be touched a certain number of times in order to solve the puzzle. Below the puzzle is a 'programming plan' that was only a small part of the coding process. Making to-do lists proved to be extremely helpful during the entire development process.

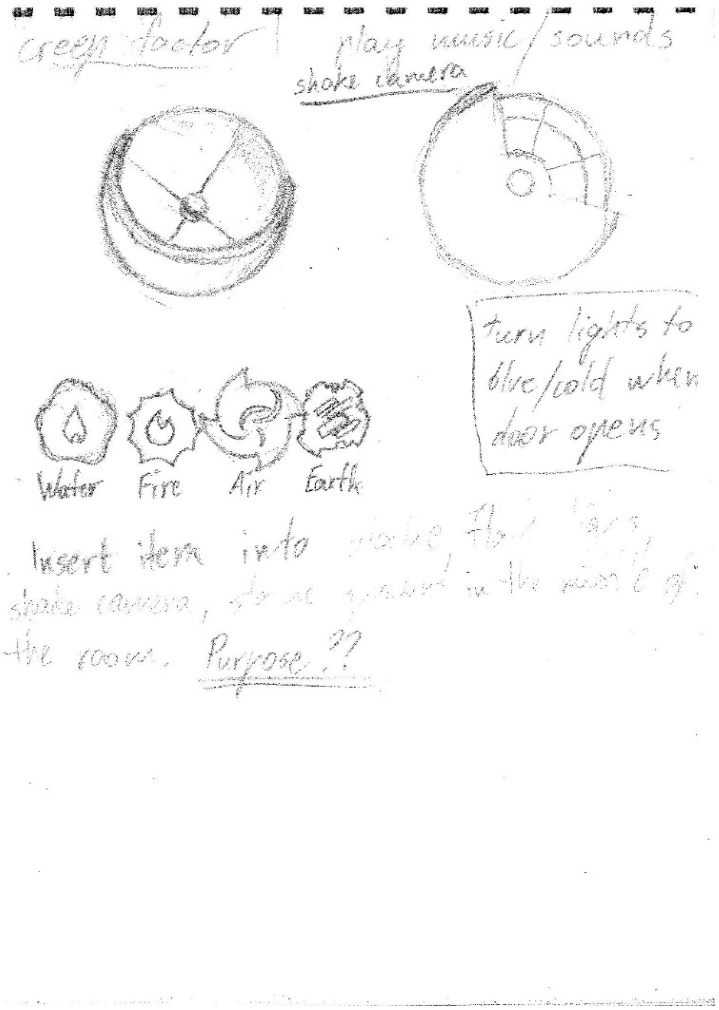


Fig. 6: Some items that were considered as pieces of any puzzles. Concepts were explored such as the ability to examine the items themselves, hiding puzzles in them and providing more meaning to them. A few more elements such as camera shaking and horror elements such as flashing lights and music were considered. Some of these ideas did make it to the final game, while others, such as camera shaking, were discarded due to their complexity.

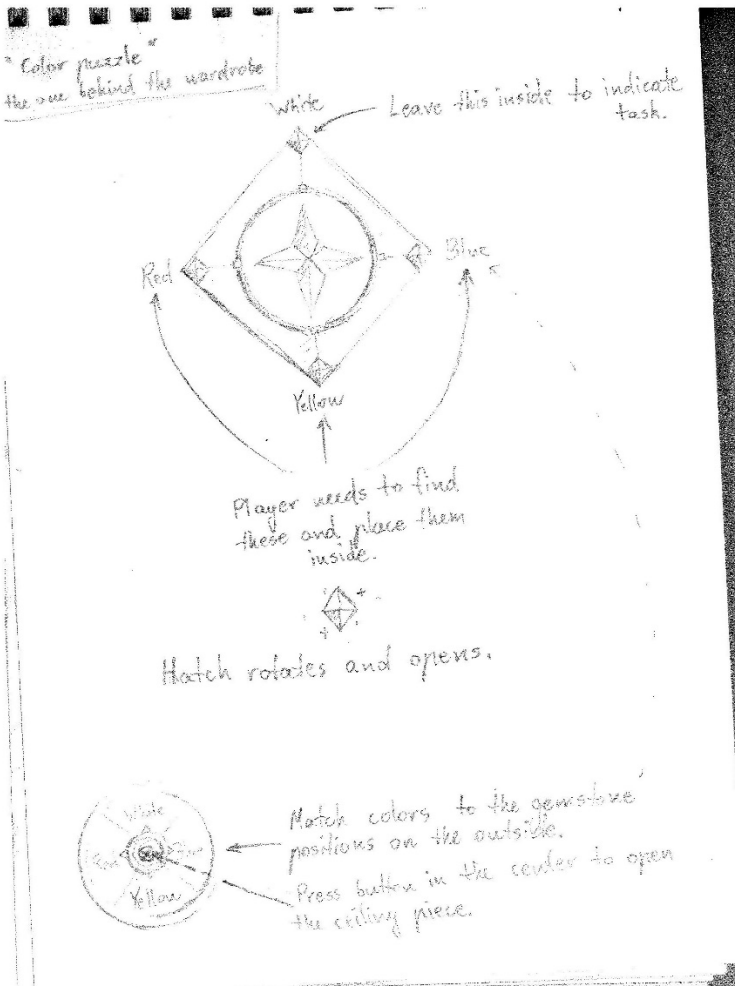


Fig. 7: The Colour Puzzle was the last puzzle to be designed and was surprisingly relatively straightforward. It consisted of multiple layers: one was the opening of the door, then the colour matching, and, finally, pressing the button in the centre to lower the platform in the ceiling. The code was fairly similar to the one in the Image puzzle as the mechanics were mostly the same – press a button to change between colours and the hatch unlocks when they all match the positions of the gemstones on the outside. The challenge was getting the items from the inventory to the puzzle itself as they player had to use the collected gemstones on the outside of the puzzle to open the door.

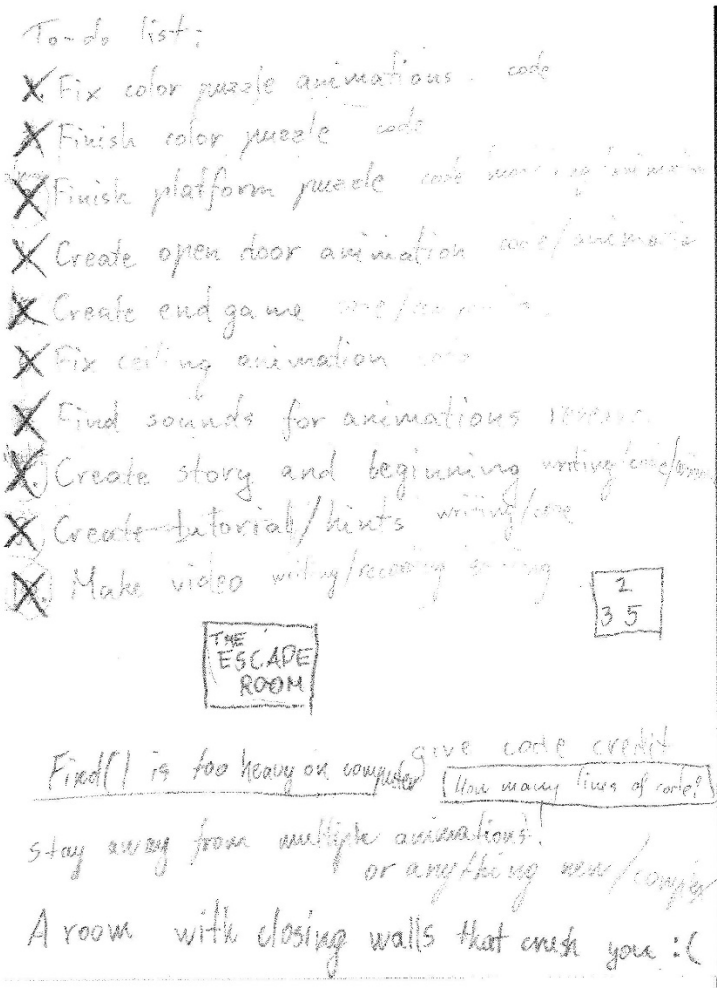


Fig. 8: Another to-do list showing some of the completed tasks during development and the final stages of the project. Some scribbles of the logo are visible, and a new idea for the ending is was noted down at the bottom. A few warnings have been noted down, including avoiding complicated animations, and any new features, as the main objective at the time was to finish all the elements that had made their way to the final game.

Appendix 3: Third-party assets

Below is a list of custom assets that were used in the project.

Materials obtained from:

Textures: https://www.textures.com/search?q=wood

FreePBR: https://freepbr.com/c/floors/page/2/

cc0textures: https://cc0textures.com/list.php?cat=Wood

cgBookcase: https://www.cgbookcase.com/

3D models:

Nefertiti bust by Gargore: https://assetstore.unity.com/packages/3d/nefertiti-56818

Small plants by Keilbaum: https://assetstore.unity.com/packages/3d/vegetation/plants/small-plants-6930

Yughues Free Decorative Plants by Nobiax / Yughues: https://assetstore.unity.com/packages/3d/props/interior/yughues-free-decorative-plants-13283

Essentials:

Post Processing Stack by Unity Technologies: https://assetstore.unity.com/packages/essentials/post-processing-stack-83912

Standard Assets by Unity Technologies: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351

Scripts:

Item and Inventory scripts by Brackeys: https://github.com/Brackeys/RPG-Tutorial/tree/master/Finished%20Project/Assets/Scripts

Appendix 4: Technical Documentation

Below is a list of the most important classes in the game that were written and documented. Please note that some scripts were written by other developers and are not present in this section of the appendix. They can be found in the third-party assets section.

```
using UnityEngine;
using UnityEngine.UI;

public class PlayerController : MonoBehaviour {


    /* Controls the interaction between the player and the objects by using a
     * ray cast and retrieving information from any objects that can be used */


    public new Camera camera; // Camera object that is attached to the player
    private readonly float distance = 10; // Measures the distance between the player
camera and the object
    public Image cursor; // Image that stores the cursor icon
    public static bool mouseClicked = false; // Boolean value that becomes true when
mouse is clicked

    public static GameObject interactable; // Any object that the player is looking at
and has a tag of "Useable" or "Object" will be stored in this game object

    Color cursorColor; // Colour of the cursor icon

    // Update is called once per frame
    void Update()
    {
        cursorColor = cursor.color; // Assign a colour to cursorColor

        cursorColor.a = 0.1f; // Set its alpha to 0.1

        Ray ray = camera.ScreenPointToRay(Input.mousePosition); // Create a ray cast that
goes from the camera to where the mouse is pointing, i.e. middle of the screen
        RaycastHit hit; // Get information from ray cast

        // If ray hits an object with the tag "Useable" or "Object", set cursor colour
to white, assign the game object to 'interactable' and set mouseClicked to true
        if (Physics.Raycast(ray, out hit, distance) && hit.transform.gameObject.tag ==
"Useable" ||
            Physics.Raycast(ray, out hit, distance) && hit.transform.gameObject.tag ==
"Object")
        {
            cursor.color = Color.white;
            interactable = hit.transform.gameObject;

            if (Input.GetMouseButtonDown(0) && !mouseClicked)
            {
                mouseClicked = true;
            }

        }
        // Otherwise set cursor's alpha back to 0.1
        else
        {
            cursor.color = cursorColor;
        }
    }
}
```

```
using UnityEngine;

public class ImagePuzzle : MonoBehaviour
{


    /* Controls the Image Puzzle, the buttons and rendering of materials,
     * i.e., the images. Designed to be used for all three image frames */


    public Material[] materials; // An array that stores all the available image
materials in each row
    public MeshRenderer meshRenderer; // Mesh renderer of the image frame
    private int n = 0; // Integer that will be used to count between images

    public GameObject leftButton;
    public GameObject rightButton;

    public GameObject topImage;
    public GameObject middleImage;
    public GameObject bottomImage;

    // Used to keep track of puzzle progression
    public static bool part1_solved = false;
    public static bool part2_solved = false;

       // Use this for initialization
       void Start () {

        meshRenderer = GetComponent<MeshRenderer>();

    }

       // Update is called once per frame
       void Update () {

        // If mouse is clicked
        if (PlayerController.mouseClicked == true)
        {
            // If 'interactable' is not empty and its name is "Puzzle painting"
            if (PlayerController.interactable != null &&
PlayerController.interactable.name == "Puzzle_painting")
            {
                part1_solved = true; // Painting is removed
            }

            // If player is looking at an object named "Right Button", button is pressed
and image is changed to the next one
            if (PlayerController.interactable.transform.name ==
rightButton.transform.name)
            {
                ButtonPress.button_pressed = true;

                if (n >= 3)
                {
                    n = 0;
                }
                else
```

```
{
                    n++;
                }

                meshRenderer.material = materials[n];
                PlayerController.mouseClicked = false;

            }
            // If player is looking at an object named "Left Button", button is pressed
and image is changed to the previous one
            if (PlayerController.interactable.transform.name ==
leftButton.transform.name)
            {
                ButtonPress.button_pressed = true;

                if (n <= 0)
                {
                    n = 3;
                }
                else
                {
                    n--;
                }

                meshRenderer.material = materials[n];
                PlayerController.mouseClicked = false;

            }

        }

        // If all three images have been correctly picked, the puzzle is solved
        if (topImage.GetComponent<MeshRenderer>().material.name == "img-3-1 (Instance)"
&&
            middleImage.GetComponent<MeshRenderer>().material.name == "img-4-1
(Instance)" &&
            bottomImage.GetComponent<MeshRenderer>().material.name == "img-8-1
(Instance)")
        {
            part2_solved = true;
        }

    }
}
```

```
using UnityEngine;

public class ColorPuzzle : MonoBehaviour
{

    /* Contains the majority of the game objects and booleans
     * required for this puzzle. Tracks the completion. */

    // Booleans that become true once all gems are placed on puzzle
    public static bool redSlot = false;
    public static bool blueSlot = false;
    public static bool yellowSlot = false;

    // Game objects for coloured panels inside puzzle
    public GameObject topPanel;
    public GameObject bottomPanel;
    public GameObject leftPanel;
    public GameObject rightPanel;

    public GameObject centerButton;

    public static bool part1_solved = false;
    public static bool part2_solved = false;

    public static bool center_button = false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // If all three gemstones are placed on the puzzle, part 1 is solved
        if (redSlot == true && blueSlot == true && yellowSlot == true)
        {
            part1_solved = true;
        }

        // If all colours on the panels are correct, part 2 is solved
        if (ChangeColor.top_solved == true && ChangeColor.bottom_solved == true &&
            ChangeColor.left_solved == true && ChangeColor.right_solved == true)
        {
            part2_solved = true;
        }

        // For the center button
        if (PlayerController.mouseClicked == true)
        {
            if (PlayerController.interactable.transform.name ==
centerButton.transform.name)
            {
                center_button = true;
                ButtonPress.button_pressed = true;
                PlayerController.mouseClicked = false;
            }

        }
    }

}
```

```
using UnityEngine;

public class ChangeColor : MonoBehaviour
{

    /*  Changes material on panels inside Colour Puzzle  */

    public Material[] materials; // Array for storing materials
    public MeshRenderer meshRenderer;

    private int n = 0; // Used for counting

    public AudioSource audioSource;

    public GameObject button;

    public GameObject section; // Panel section

    public static bool top_solved = false;
    public static bool bottom_solved = false;
    public static bool left_solved = false;
    public static bool right_solved = false;

    // Start is called before the first frame update
    void Start()
    {

        meshRenderer = GetComponent<MeshRenderer>();
        audioSource = GetComponent<AudioSource>();

    }

    // Update is called once per frame
    void Update()
    {
        // For switching between colours, same as in Image Puzzle
        if (PlayerController.mouseClicked == true)
        {
            if (PlayerController.interactable.transform.name == button.transform.name)
            {
                audioSource.Play();

                if (n >= 3)
                {
                    n = 0;
                }
                else
                {
                    n++;
                }

                meshRenderer.material = materials[n];
                PlayerController.mouseClicked = false;

            }

        }
```

```
// If sections have correct colours, puzzle is solved
        if (section.name == "Top Panel" && meshRenderer.material.name == "White glass
(Instance)")
        {
            top_solved = true;
        }

        if (section.gameObject.name == "Left Panel" && meshRenderer.material.name == "Red
glass (Instance)")
        {
            left_solved = true;
        }

        if (section.gameObject.name == "Bottom Panel" && meshRenderer.material.name ==
"Yellow glass (Instance)")
        {
            bottom_solved = true;
        }

        if (section.gameObject.name == "Right Panel" && meshRenderer.material.name ==
"Blue glass (Instance)")
        {
            right_solved = true;
        }

    }
}
```

```
using System.Collections;
using UnityEngine;

public class PlatformPuzzle : MonoBehaviour {


    /* Controls the Platform Puzzle */

    // For tracking the sate of the puzzle
    public static bool part1_solved = false;
    public static bool part2_solved = false;
    public static bool door_button_pressed = false;

    public bool found = false; // Becomes true when all spheres are placed on table

    public Material glowMaterial, glassMaterial, redMaterial; // Materials for spheres

    public GameObject tableSlot_1, tableSlot_2, tableSlot_3; // Slots for spheres to go
in
    public GameObject doorButton; // Button found in the center
    public GameObject light_1, light_2, light_3, light_4; // Lights used for flickering

    GameObject sphere_1, sphere_2, sphere_3;

    int count_1, count_2, count_3; // Used for counting shpere taps

    bool sphere_1_solved, sphere_2_solved, sphere_3_solved; // Become true if spheres are
tapped enough times
    bool sphere_1_stop, sphere_2_stop, sphere_3_stop; // Used to stop the spheres from
blinking

    bool waitActive, clickActive, flickerActive; // Used as triggers for Enumerators

    AudioSource audioSource;

    // make spheres flash red for 2 seconds
    IEnumerator RedFlash()
    {
        waitActive = true;

        sphere_1 = GameObject.Find("Sphere 1(Clone)");
        sphere_2 = GameObject.Find("Sphere 2(Clone)");
        sphere_3 = GameObject.Find("Sphere 3(Clone)");

        sphere_1.GetComponent<MeshRenderer>().material = redMaterial;
        sphere_2.GetComponent<MeshRenderer>().material = redMaterial;
        sphere_3.GetComponent<MeshRenderer>().material = redMaterial;

        yield return new WaitForSeconds(2.0f);

        sphere_1.GetComponent<MeshRenderer>().material = glassMaterial;
        sphere_2.GetComponent<MeshRenderer>().material = glassMaterial;
        sphere_3.GetComponent<MeshRenderer>().material = glassMaterial;

        waitActive = false;

    }
```

```
// make spheres flash white for half a second
    IEnumerator WhiteFlash()
    {
        clickActive = true;

        if (!waitActive)
        {
            PlayerController.interactable.GetComponent<MeshRenderer>().material =
glowMaterial;
        }

        yield return new WaitForSeconds(0.5f);

        if (!waitActive)
        {
            PlayerController.interactable.GetComponent<MeshRenderer>().material =
glassMaterial;
        }

        clickActive = false;
    }

    // make lights flicker
    IEnumerator Flicker()
    {
        flickerActive = true;

        light_1.GetComponent<Light>().enabled = false;
        light_2.GetComponent<Light>().enabled = false;
        light_3.GetComponent<Light>().enabled = false;
        light_4.GetComponent<Light>().enabled = false;

        yield return new WaitForSeconds(0.2f);

        light_1.GetComponent<Light>().enabled = true;
        light_2.GetComponent<Light>().enabled = true;
        light_3.GetComponent<Light>().enabled = true;
        light_4.GetComponent<Light>().enabled = true;

        yield return new WaitForSeconds(0.2f);

        light_1.GetComponent<Light>().enabled = false;
        light_2.GetComponent<Light>().enabled = false;
        light_3.GetComponent<Light>().enabled = false;
        light_4.GetComponent<Light>().enabled = false;

    }

    // Use this for initialization
    void Start () {

        audioSource = GetComponent<AudioSource>();

       }

    // Update is called once per frame
    void Update () {
```

```
// check if table contains all spheres
        if (!found)
        {
            if (GameObject.Find("Sphere 1(Clone)") != null && GameObject.Find("Sphere
2(Clone)") != null && GameObject.Find("Sphere 3(Clone)") != null)
            {
                part1_solved = true;
                Destroy(tableSlot_1);
                Destroy(tableSlot_2);
                Destroy(tableSlot_3);
                found = true;
                return;
            }
        }

        // if all spheres are placed on table
        if (PlayerController.interactable != null && !part2_solved && found)
        {
            if (PlayerController.interactable.name == "Sphere 1(Clone)" ||
            PlayerController.interactable.name == "Sphere 2(Clone)" ||
            PlayerController.interactable.name == "Sphere 3(Clone)")

            // flash sphere if mouse is clicked
            {
                if (!waitActive)
                {
                    if (Input.GetMouseButtonDown(0))
                    {
                        if (!clickActive)
                        {
                            StartCoroutine(WhiteFlash());
                        }
                    }
                }

                // if player taps on Sphere 1, increase count; if count is 2, mark as
solved; if count is more than 2, flash red, reset count on all spheres
                if (PlayerController.interactable.name == "Sphere 1(Clone)" &&
PlayerController.mouseClicked && !sphere_1_stop)
                {
                    count_1++;

                    if (count_1 == 2)
                    {
                        sphere_1_solved = true;
                    }
                    else if (count_1 > 2)
                    {
                        if (!waitActive)
                        {
                            StartCoroutine(RedFlash());
                        }

                        count_1 = 0;
                        count_2 = 0;
                        count_3 = 0;

                        sphere_1_solved = false;
```

```
                    sphere_2_solved = false;
                    sphere_3_solved = false;
                }

                PlayerController.mouseClicked = false;

            }

            // if player taps on Sphere 2, increase count; if count is 3, mark as
solved; if count is more than 3, flash red, reset count on all spheres
            if (PlayerController.interactable.name == "Sphere 2(Clone)" &&
PlayerController.mouseClicked && !sphere_2_stop)
            {
                count_2++;

                if (count_2 == 3)
                {
                    sphere_2_solved = true;
                }
                else if (count_2 > 3)
                {
                    if (!waitActive)
                    {
                        StartCoroutine(RedFlash());
                    }

                    count_1 = 0;
                    count_2 = 0;
                    count_3 = 0;

                    sphere_1_solved = false;
                    sphere_2_solved = false;
                    sphere_3_solved = false;
                }

                PlayerController.mouseClicked = false;

            }

            // if player taps on Sphere 3, increase count; if count is 5, mark as
solved; if count is more then 5, flash red, reset count on all spheres
            if (PlayerController.interactable.name == "Sphere 3(Clone)" &&
PlayerController.mouseClicked && !sphere_3_stop)
            {
                count_3++;

                if (count_3 == 5)
                {
                    sphere_3_solved = true;
                }
                else if (count_3 > 5)
                {
                    if (!waitActive)
                    {
                        StartCoroutine(RedFlash());
                    }

                    count_1 = 0;
```

```
count_2 = 0;
                        count_3 = 0;

                        sphere_1_solved = false;
                        sphere_2_solved = false;
                        sphere_3_solved = false;
                    }

                    PlayerController.mouseClicked = false;

                }

                // if all spheres are solved, the puzzle becomes solved
                if (sphere_1_solved && sphere_2_solved && sphere_3_solved && part1_solved
&& !part2_solved)
                {
                    sphere_1_stop = true;
                    sphere_2_stop = true;
                    sphere_3_stop = true;

                    sphere_1 = GameObject.Find("Sphere 1(Clone)");
                    sphere_2 = GameObject.Find("Sphere 2(Clone)");
                    sphere_3 = GameObject.Find("Sphere 3(Clone)");

                    PlayerController.mouseClicked = false;

                    sphere_1.GetComponent<MeshRenderer>().material = glowMaterial;
                    sphere_2.GetComponent<MeshRenderer>().material = glowMaterial;
                    sphere_3.GetComponent<MeshRenderer>().material = glowMaterial;

                    part2_solved = true;

                }
            }

        }

        // if puzzle is solved and door button is pressed, flash lights and open door
        if (part2_solved)
        {
            if (PlayerController.interactable.name == "Door Button" &&
PlayerController.mouseClicked)
            {
                door_button_pressed = true;
                ButtonPress.button_pressed = true;

                if (!flickerActive)
                {
                    StartCoroutine(Flicker());
                }

                PlayerController.mouseClicked = false;
            }
        }

    }
}
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UseItem : MonoBehaviour
{

    /*  Controls use of items  */

    Inventory inventory;

    IDictionary<int, string> dict = new Dictionary<int, string>() {

        // keys are item ID's, values are object names that the items need to be used
with
        // Example: 1 - Red Gemstone ID, "Red Slot" - where the gemstone needs to be
placed
        // This technique allows you to add as many items as you wish

        {1, "Red Slot"},
        {2, "Blue Slot"},
        {3, "Yellow Slot"},
        {4, "Table Slot"},
        {5, "Table Slot"},
        {6, "Table Slot"},

    };

    public Image slot1, slot2, slot3, slot4, slot5;

    bool slot1_active, slot2_active, slot3_active, slot4_active, slot5_active;

    bool alpha1_pressed, alpha2_pressed, alpha3_pressed, alpha4_pressed, alpha5_pressed;

    Item item;

    // Start is called before the first frame update
    void Start()
    {
        inventory = Inventory.instance;

        slot1_active = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (PlayerController.interactable != null) // if player is looking at an
interactable object
        {
            if (Input.GetKeyDown(KeyCode.Alpha1)) // if button 1 is pressed
            {
                if (inventory.items[0] != null) // check if first inventory slot is not
empty
                {
                    if (dict.ContainsKey(inventory.items[0].key)) // check if item key
matches with the key in dictionary
                    {
```

```
                        if (dict[inventory.items[0].key] ==
PlayerController.interactable.name) // check if value name is the same as the object that
the player is looking at
                        {
                            item = inventory.items[0]; // assign first item in list to
'item'
                            item.Use();                // use item
                            inventory.Remove(item);    // remove item from inventory
                            alpha1_pressed = false;
                            return;
                        }
                    }
                }
            }

            if (Input.GetKeyDown(KeyCode.Alpha2))
            {
                if (inventory.items[1] != null)
                {
                    if (dict.ContainsKey(inventory.items[1].key))
                    {
                        if (dict[inventory.items[1].key] ==
PlayerController.interactable.name)
                        {
                            item = inventory.items[1];
                            item.Use();
                            inventory.Remove(item);
                            return;
                        }
                    }
                }
            }

            if (Input.GetKeyDown(KeyCode.Alpha3))
            {
                if (inventory.items[2] != null)
                {
                    if (dict.ContainsKey(inventory.items[2].key))
                    {
                        if (dict[inventory.items[2].key] ==
PlayerController.interactable.name)
                        {
                            item = inventory.items[2];
                            item.Use();
                            inventory.Remove(item);
                            return;
                        }
                    }
                }
            }

            if (Input.GetKeyDown(KeyCode.Alpha4))
            {
                if (inventory.items[3] != null)
                {
                    if (dict.ContainsKey(inventory.items[3].key))
                    {
```

```
                        if (dict[inventory.items[3].key] ==
PlayerController.interactable.name)
                        {
                            item = inventory.items[3];
                            item.Use();
                            inventory.Remove(item);
                            return;
                        }
                    }
                }
            }

            if (Input.GetKeyDown(KeyCode.Alpha5))
            {
                if (inventory.items[4] != null)
                {
                    if (dict.ContainsKey(inventory.items[4].key))
                    {
                        if (dict[inventory.items[4].key] ==
PlayerController.interactable.name)
                        {
                            item = inventory.items[4];
                            item.Use();
                            inventory.Remove(item);
                            return;
                        }
                    }
                }
            }

        }

    }
}
```

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class EndRoom : MonoBehaviour
{

    /*  Controls the events in final room  */

    public GameObject wall, player, playerController, blackCanvas, credits, sound, crush;

    float distance; // Tracks distance between player and wall with sign

    bool crushActive;

    public static bool player_inside, start;

    public Animator wall_1, wall_2, wall_3, door_1, door_2;

    public AudioSource audioSource, audioSource_wall, audioSource_door,
audioSource_crush;
    public AudioClip wall_clip, door_clip, crush_clip;

    bool canvasOn;

    bool trigger1, trigger2, trigger3;

    bool play;

    // Walls move in between pauses, when the last wall crushes player, cut to black,
display credits, go back to menu
    IEnumerator Crush()
    {
        crushActive = true;

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_1, "Wall 1 Move 1", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_2, "Wall 2 Move 1", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_3, "Wall 3 Move 1", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_1, "Wall 1 Move 2", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_2, "Wall 2 Move 2", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_3, "Wall 3 Move 2", wall_clip);
```

```
        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_1, "Wall 1 Move 3", wall_clip);

        yield return new WaitForSeconds(3.0f);

        playAnimation(wall_2, "Wall 2 Move 3", wall_clip);

        yield return new WaitForSeconds(5.0f);

        playAnimation(wall_3, "Wall 3 Move 3", wall_clip);

        yield return new WaitForSeconds(3.0f);

        blackCanvas.SetActive(true);

        canvasOn = true;

        yield return new WaitForSeconds(3.0f);

        credits.SetActive(true);

        yield return new WaitForSeconds(3.0f);

        credits.SetActive(false);

        SceneManager.LoadScene("Menu");

    }

    // Start is called before the first frame update
    void Start()
    {
        blackCanvas.SetActive(false);
        credits.SetActive(false);
    }

    void playAnimation(Animator animator, string trigger, AudioClip clip)
    {
        Debug.Log("Playing");
        play = true;

        animator.SetTrigger(trigger);

        if (audioSource != null && play)
        {
            audioSource_wall.PlayOneShot(clip);
            play = false;
        }

    }

    // Update is called once per frame
    void Update()
    {
        if (canvasOn)
        {
            if (!trigger3)
```

```
{
                audioSource_crush.PlayOneShot(crush_clip);
                trigger3 = true;
            }
        }

        distance = Vector3.Distance(wall.transform.position, player.transform.position);

        // if distance between player and wall is 25, play music
        if (distance <= 15)
        {
            start = true;
        }
        // if distance is less than 45, it means player is inside room
        else if (distance <= 45)
        {
            player_inside = true;
        }
        // if sound is playing, begin wall animaions
        if (start && !crushActive)
        {
            StartCoroutine(Crush());
        }
        // if player is getting further from the wall, or closer to the doors, shut doors
        if (start && distance >= 35)
        {
            if (!trigger2)
            {
                playAnimation(door_1, "Close Door", door_clip);
                playAnimation(door_2, "Close Door", null);
                trigger2 = true;
            }
        }
        // play scary music
        if (start)
        {
            if (audioSource.clip.name == "scaryscape" && !trigger1)
            {
                audioSource.Play();

                trigger1 = true;
            }

        }

    }
}
```