Final Year Project Report yq009086_Nathan_Brown_Puzzle-based game using an open source game engine

by Nathan Brown

Submission date: 01-May-2017 02:32PM (UTC+0100)
Submission ID: 71532742
File name: CS3IP16-yq009086-Final_Project_Report.docx (24.97M)
Word count: 29380
Character count: 166202

CS3IP16

School of Mathematical, Physical and Computational Sciences
Individual Project – CS3IP16

Puzzle-Based Game Using an Open-Source Game Engine

Student Name: Nathan Brown

Student Number: 23009086

Supervisor: Hong Wei

Date Submitted: 01/05/2017

ABSTRACT:

Commercially and individually produced video games have become a large part of the entertainment industry over recent years, with games being assigned budgets on par with, if not exceeding that of some films. Because of this, many individuals wish to start producing original games using their own ideas for all areas of the game, from storyline, through art, to logic and gameplay. It has been shown that games can both be used as a useful tool in terms of enjoyment, as well as in an educational, or even rehabilitative way. Whatever reason for producing new game content, some tools exist for upcoming developers, of both technical and none technical backgrounds, to explore ways to allow their ideas to be tried and tested. The design of these tools needs to be carefully considered in order for both industry developers to produce professional content, as well as new designers to be able to put across their concepts. Broad concepts of open-source game engines will be investigated, followed by an in-depth analysis of one specific tool. The ease at which the package can be picked up will also allow for evaluation of the tool's effectiveness for programming a puzzle based game. To do this, a product of this nature will be implemented to determine the quality and speed at which a concept for a video game can be created.

Table of Contents

INTRODUCTION:

In this report, the effectiveness of the game engine in being able to produce a prototype for a game, in a quick and reliable way, will be investigated. This will be a challenging task assessing programming ability and usage of a previously unknown tool that is used frequently in industry. Following from research, an appropriate game will be developed to both generate an understanding of game creation concepts in relation to both programming and level design. The created game will involve puzzles, various areas and applications of logic and deductive reasoning to demonstrate the engine's capabilities. Following each stage of the software design process, an evaluation will take place with regard to its ease of use relating back to the new game designer. An overall use of the chosen engine will be reviewed along with a look into how technical scripting elements were coded in order to produce the final products features.

PROBLEM ARTICULATION:

As technology advances, game engines and 'modding' (modifications) have given players the opportunity to become involved to a level where they can implement their own creative ideas, either through modification of a game or producing one of their own. The principle issue being addressed is how the construction of a product such as a puzzle game is programmed and implemented. One problem to date has been the deterrent of complex game design environments, requiring in-depth programming expertise, detailed understanding of physics & related mathematical computations (lighting and water mechanics). From this, a secondary objective from the problem is learning how to translate an individual's imaginative gaming ideas into reality, despite these challenging barriers. The latest software tools created to assist upcoming developers need to be assessed to determine whether they are suitable for the creation of complex content and ,in creating this puzzle game, experience will be gained on how industry used environments can be used to develop features through detailed programming.

Problem solution benefits:

As this project progresses to completion, certain individuals will be involved in both development process (monitoring progress) and being end users. The project supervisor is tasked with overseeing project progress and providing useful feedback on advancements made, and also is there to aid with any potential issues in the project. The supervisor may benefit from this by seeing the project progressing, with the developer being benefitted with informative feedback. End users, both experienced, and not experienced, in puzzle games benefit from the problem solution. Those new to puzzle games get a challenge to solve a problem, or may be interested in the graphical style or accompanying storyline of the product. The experienced puzzle game players may be interested to see how puzzles compare with other games in the same genre, or look for interesting concepts or ways to solve difficult puzzles that they have not previously faced.

Both aspiring and technical savvy developers could benefit from this solution also. From the report and review of game engines, an appropriate tool could be revealed to them based on their technical level of expertise. If designing a similar product, they could gain ideas from features included in game being produced and also gain an overview for the features of the game engine (for example, the extent of debug tools, realistic physics calculations). For the individual undertaking the project, numerous benefits can be seen, from enhancing programming knowledge to developing skills with an industry piece of software, subsequently gaining employability skills.

Upon reviewing the timescales for this project, a number of assumptions and restrictions on the level of detail of the project have been made. The level of detail in certain aspects of the game such as art and storyline will not be able to be developed in entirety within this projects timescale. Although care will be taken to minimise expensive computational functions, the device running the game will restrict whether the game can be played. Both the length of the levels or game world developed and the complexity of other game elements like AI interaction and mechanic complexity will also be limited by time available. In order to determine the success of this project, the developer will be able to gain knowledge on how an industry piece of software is used to create a complex puzzle game. The most appropriate environment to use for this creation should also be evaluated. In terms of user criteria, the user should be able to navigate through all produced levels without any significant faults or errors occurring. Furthermore, the focus of both level and feature design should be on user engagement and enjoyment.

The final product will demonstrate the use of an industry standard game engine, alongside how that engine can be used to produce an entertaining and challenging result. The level of complexity to both an experienced and a new user of the software will be discussed at each stage in development, for example in design.

LITERATURE REVIEW/INITIAL RESEARCH:

For this product to be created, one needs to understand how three aspects of game design (puzzle game theory, leading puzzle games in the genre and game engine design) have progressed to their current state. Within this review, the puzzle gaming genre; related puzzle gaming theory and the progression and utility of Game Engines will be investigated to provide a comprehension of the reasons behind this project. Any areas of relevance to the current project will be highlighted, alongside the impact that they may have on the final product.

Progression of Puzzle game theory:

From the outset, it is clear that deciding on how to build a puzzle game is a momentous task. Many interpretations of what discerns a good puzzle can be argued, as what is challenging for some may be trivial for others. It can also be argued that any existing concept of a game (either a board game or computer game) can be altered, to be made into a puzzle, by manipulating some of its existing mechanics or rules to increase the level of thinking or difficulty required. For example, a deck of cards can be used to produce games that require intense thinking or astute observation to comprehend the rules, or how to solve the puzzle. The card game Mao requires players who do not know the rules (upon starting the game) to learn them through a set of penalties given by player/players that know the rules [1]. This sort of puzzle involves the user understanding where they went wrong, adapting their ongoing strategy accordingly.

When considering how to apply good puzzle gaming theory to computer games, it is worth consulting literature based on knowledge from existing and successful puzzle game developers. An article from Rock, Paper, Shotgun discusses developers views on how to make a good puzzle game [2]. This shows that recurring topics come up in terms of how a puzzle game should operate. It should be based on the user understanding the puzzle before solving it. It should also be minimalist in terms of providing the player with tools to solve problems, so they learn to apply the tools to a puzzle. This also allows them to learn something new about the different applications of the mechanics, rather than to just solve each puzzle differently.

An article by Dev.mag that also interviewed many puzzle game designers summarises and categorizes the key concepts on how puzzle games seem to be designed [3]. Again this article shows puzzles must have a clear objective in mind and use simple, but polished mechanics. It comments on the designing in such a way that it is gameplay focussed and goal-oriented, designing puzzles on paper beforehand to understand the logic and produce a solid final riddle for the user to solve. For user interaction, teaching the player something new should be done in isolation, meaning the only thing that can be done in any given tutorial is the mechanic being taught. Puzzles should be based on flexible mechanics and have staggered difficulty, from simple to complex. This article also discusses the importance of early and frequent testing, observing the player and giving no hints, asking for as much feedback as possible. This article appears to highlight the important areas to consider when developing a product of this nature

To be able to understand a user's focus when interacting with the game world, a paper was reviewed that looked at the psychological classifications of needs of human beings in reference to their fulfilment through playing games [4]. Although based on the role-playing genre, the paper remarks that these findings could be readily applied to many types of game. The paper mentions that the primary goal for game designers is to ensure player enjoyment. Citing another paper, Bostan 2009, the paper further lists user needs under the following categories: Materialistic; Power; Affiliation; Achievement; Information and Sensual. When applied to computer games, the satisfaction of these needs is limited by games content and rules of play, with information being the most prominent aspect in puzzle/adventure style games to satisfy the need for achievement. Finally, the paper mentions themes emerging based around retention, acquisition, dominance and awareness, with storytelling playing an important role.
This provides good insight into what features should be present in a puzzle game, especially when compared with the game under review in this paper, Fallout 3, which uses a set of skills that can be used for different purposes.

Progression of puzzle game genre:

It is clear that the puzzle game genre has an eclectic range of possibilities, as puzzle elements can be applied to many other genres of gaming. Puzzle gaming also has its intrinsic types, such as hidden object finding, logic puzzles, physics puzzles, real time strategy and even digital representations of existing physical games such as Sodoku or Monopoly. Therefore it is worth exploring the existing products that have previously been and are currently still being used, in the video game industry.

Initially, puzzle games consisted of one base mechanic that the user was required to utilise in a variety of different scenarios. Games such as Tetris, worked on the concept of rotating various 2-dimensional shapes so that they formed full lines, which were then removed from the gameboard[5]. From this, puzzle games adapted into multiple different permutations, with newer concepts being developed on how to solve a particular problem. As computing power advanced, the ability to introduce more complicated mechanics became possible. This meant that more detailed puzzles could be made revolving around using movement, around an environment, or even more advanced 2-dimensional puzzle games based on more detailed logic. More modern offering similar styles of play to 'Tetris' include the 'CandyCrush' series of games, which revolve around matching tiles of successive colours in order to remove them from the game, where new ones take their place [6], with the aim to be to achieve a particular score in a time limit whilst completing other side objectives in more advanced levels.

Platformer games such as 'Donkey Kong' involved jumping onto different sections of a fixed screen in order to collect items or advance to another stage. Evolving from this were scrolling platformers that allowed for the perception of larger worlds by scrolling the world as the character moved. This proved to be an iconic step in gaming in general, with the appearance of games such as 'Jump Bug' and the video game 'Super Mario' [7], but also provided grounding for another way for developers to incorporate puzzles into their games. Puzzle platformers became popular with many different games being produced with extensive variety in both the puzzles being solved and the way they were completed, a relevant example being that of the title 'Super Meat Boy'. This game had the player navigate a 2D environment, utilising momentum and the ability to stick to, and slide down, walls to avoid traps set up in each level [8]. The way in which this game handles respawning also made it even more popular as the respawn time is instant, not focusing on lives, with the setback of dying to the player being minimal. This had the benefit of allowing the player to re-enter the game and retry what they had attempted, forcing them to reconsider their strategy without the frustration of preserving lives, making intensely difficult portions of the game feel more balanced [9] as a large setback was not present upon death .

More recently, games have even further adapted the 2-dimensional mechanics to introduce innovative ways of solving problems. This has been done through adding features such as simulated Gravity and environment interaction, as can be seen in 'Limbo' [10]. Use of perspective to allow ways of traversing a 2-dimensional environment in an almost 3-dimensional way as demonstrated in 'Fez' [11], and interesting ways of combining physics and resource management in games such as 'Unravel' [12], introduces novel new approaches to puzzle game design. All of these popular puzzle games approach their puzzles in completely different ways, although core themes such as collecting items or navigating to new locations remains the same, the solutions to these problems are handled through the games core mechanics. This is an important concept to note for the product that is going to be created.

As advances in technology allowed for the creation of 3 dimensional games, the notion of 3D platformers was also introduced. 'Alpha Waves' allowed the player to navigate spring like platforms in a 3D environment, creating a set of jumping puzzles involving timing and placement of the players shadow to ensure the connection with the platform [13]. Further advancements to this 3 dimensional platforming genre were games such as 'Super Mario 64', which involved an eclectic mix of puzzles requiring use of jumping, timing, camera movement and enemy avoidance to get to the desired objective (collecting 7 stars per course, with additional secret and extra stars)[14]. Again, more puzzle-centric 3 dimensional games were released, one of which was considered an extreme success due to its never before used mechanic of instantaneous teleportation of a player. Valve's 'Portal' (and additionally its successor 'Portal 2') allowed the user to connect between two points in the game world by firing entry and exit portals at different surfaces of the levels. This mechanic, along with other mechanics such as glass walls that gave the player an overview of a puzzle to help them to solve it [15], allowed for a wide range of puzzles to be developed using different aspects of physics and momentum. These puzzles still revolved around a simple goal (placing a box on a pad to open a door) but introduced complications such as a box being encased in a glass box that the player has to determine how to get to.

Currently, the puzzle game genre appears to be constantly redefined by new and intelligently designed puzzle mechanics. The recent publication of Snake pass [16], where the user has a limited grip (resource) and uses it to climb the environment, shows a further novel adaptation of a similar core mechanic (resource management). From all of the past and current versions of puzzle games, it appears that the main concept, that is used, is to keep the primary goal simple and, using simple tools or facts about the environment, design intelligent solutions to reach your goal. The book, 'The Art of Computer Game Design' highlights the importance of limiting the information given to the player as it forces them to use guesswork and facts about the environment to influence their next move [17]. All of the discussed existing products and literature above utilise this premise, showing that the product going to be created requires the same approach.

An article by the Guardian presents the trends that will impact gaming in the future, such as gaming integrated with social media; live streaming game content; and the idea of 'gamers as creators' through mediums like game modifications (mods) [18]. Current, ever evolving games such as Minecraft have allowed users to generate code that only affects small parts of the game in the form of mods. Not only this, but the freedom over the content that can be added allows users to improve their programming abilities, alongside creating game-altering mods [19]. Because of freedoms such as mods, as well as the release of open-source game engine software, it became more feasible for the average user to be able to produce a professional style product in small teams or by themselves, with the need for only a willingness to learn the specific engine and develop an understanding of programming languages and logic.

The lecture 'What is a Puzzle?' by Scott Kim defines games as a rule-based system in which the goal is for the player to win, which is not dissimilar to a physical puzzle, meaning that puzzles can easily be turned into games and vice versa [20]. This, alongside simplistic implementation (of controls and gameplay) [21], shows that the player can focus on solving the problem presented to them and is also reflected by all of the reviewed products of this genre. The theme of the discussed literature on the puzzle game genre also reinforces the message conveyed in the extract 'Making Sense Of Software: Computer Games and Interactive Textuality' by Ted Friedman, where a computer game is defined as a process of demystification whereby a user adapts their strategy by means of trial and error [22].

Progression of game engines:
Creating a game from scratch used to mean, for the average person, that all types of programming needed to be taken into account, from more basic concepts such as gameplay decisions or how the level would be generated, to more complicated issues regarding programming the entirety of the physics behaviour for the game. Depending on the complexity of the game this person wanted to implement, this would require a deep understanding of physics and the way in which objects interact and behave.

As technology has developed, not only has more documentation for programming in a variety of different languages become available to the average person, the appearance of specific programs to deal with graphic related content have also appeared [23]. This has meant that users can more easily create content they wish to convert into a game. Furthermore, companies have started to release the software utilised to create their games, which removes most of the load on the programmer with regards to aspects that would usually take a long time to study and implement such as physics calculations [24]. This, alongside other features such as drag and drop objects which can be rescaled to more easily develop levels and lighting (another complex programming feature), have also been simplified and converted so that users can easily tweak lighting levels and the way light behaves.

The purpose of a game engine is to manage the game at all phases. A lecture series by Doron NussBaum details the three stages of game control (initialisation, game loop and termination) and how the collections of libraries (engines) handle them [25]. In this lecture, DirectX is also looked at with regards to how to set up Input; Graphics and display. From this it is clear that programs such as DirectX, although useful to the developer to some extent, would still put off the average aspiring developer from trying to create a product as the language used is still to a major extent unknown and complex in appearance.

Many produced game engines aim to further abstract the notion of low level programming related to complex calculations and efficiency concerns, even to the extent of building additional layers atop game engines to 'soften the learning curve' to programmers, especially related to training simulations, as shown in the publication detailing the Delta3D product [26] . This shows that game engines are not only useful for aspiring developers, but the knowledge of how to utilise one is also good for visualising any sort of simulation. This is

reinforced by the utilisation of a game engine for landscape visualisation [27] and to aid in search and rescue simulation [28].

Game Engines can be useful for rapid development of games through use of various editors. A paper covering real positioning in virtual environments found this, alongside detailing the use of the Unreal Engine, mentioning useful tools, programmability and licencing of the product [29]. Without game engines, developers would need to consider many things in relation to environment generation, graphics rendering and modelling objects. The benefits of such engines is further reinforced by a paper on Console Architectures and Game Engines, mentioning both an extensive representation of the components of a typical game engine, as well as discussing the difficulties and strategies for rendering large 3D environments [30].

Due to both the benefits and complexities introduced by the above literature, it is clear that aspiring developers may wish to take advantage of these tools, especially if they have little experience with programming. Although the complexities discussed are worth noting as it may mean that certain projects (for example large scale worlds or city visualisations) are not suitable and may require a more tailored approach or engine to carry out the task necessary.

The outputs that an individual wishes to gain from a game engine can differ depending on the application being developed, with considerations such as the scope of the project, whether it is single player or multiplayer, and how much detail of the world needs to be included. A paper entitled '3D Games As a New Reality', details the applications, realism, structure of the engine architecture, from sound to rendering, as well as relevant features and limitations of such a tool [31]. This exhaustive study found that game engines have evolved to deliver recognisable, but not entirely convincing, visual effects, providing the developer with a large array of features such as fully textured environments, shadows, reflections, physics and sound. It also found that a game engine is not without its drawbacks, such as insufficient computing power to implement fully realistic physics; inability to handle large scale multiplayer connections spanning over 128 players, and the precision and detail level of the environment. Despite these drawbacks, the tools such as the Unreal Engine are still prevalent in industry, speeding up development and can be used for a variety of applications.

From the relevant literature on game engines, a wide range of products exist, both free and paid for, that can allow someone who wishes to produce content for a game to do so rapidly, with little experience of how the underlying mechanisms work. Industry standard game engines have now evolved, with many successful games being produced using them. Due to this, it could be difficult to discern which engine is most applicable for any given user's needs. Analysis has been performed on the most useful game engines used currently based on their individual merits and features [32]. During research, the Unity, Unreal and CryEngine have been mentioned multiple times, suggesting that those particular engines could be preferred by established developers. Comparative articles exist evaluating which to choose of these select few [33] , weighing up pros and cons such as 2D and 3D capabilities, learning curve, and costs relating to the success of the finished product. Due to the diverse range of applications that these products could be needed for, it could be argued that, for an aspiring developer, the choice is down to personal preference based on experience. The extent to which these engines simplify the game design process will be further explored as the project progresses.

Conclusion:

From the above initial research and literature review, some key concepts have been realised about the product that will be developed. From analysis of puzzle game theory, the key finding was to keep the concepts, user interface, and goals, simple. The puzzles should not being frustrating to the user, but provide an appropriate level of intrigue and challenge, with a focus on player enjoyment. A limited number of core mechanics should be implemented, which the player can be quick to pick up and apply to more complicated puzzles at later stages in the game. The above articles failed to mention the level of knowledge necessary for an individual to implement a game however.

From reviewing of the previous and existing titles in the puzzle game and platform genre, it is clear that most successful products follow the concepts as discovered in the literature on puzzle game theory. 'Tetris' started with an initial core mechanic, with more recent titles still basing puzzles around only a select number of mechanics. These modern puzzle games provide the user with new, innovative, puzzles with more ingenious ways in which to utilise their mechanics.
An industry standard game engine should be used for creation of the final product, as lots of documentation exists on them so it would be more easily available for novice users to experiment with the environment and

understand how it operates, as well as experienced developers being able to utilise their many features. Games can be used to allow players to create things from within the game itself, as well as providing opportunities to learn skills like programming through mods. However, game engines can be used to allow players to create content from scratch. Abstract and new concepts can pay off if created correctly (such as 'Portal' and 'Snake Pass') but the more abstract the puzzle concept, the more care needs to be taken when designing it.

Therefore it will be important to investigate how effective and easily a game engine such as those described can be picked up and used to create a product. This project will take an open source engine and use it to produce a puzzle game in the form of a desktop application. Not only will this determine how skilled an individual needs to be for creating a product such as a game using this software, but will also test all aspects of project management, testing and development and how straightforward these stages are. Furthermore, creating projects such as this has further benefits in developing skills in logical reasoning, technical knowledge through research, and experience relevant for any potential careers within a technical field.

The current existing state of the literature reflects that improvements can be made only in areas of enhanced realism, more detailed environments and increased support for multiplayer for game engines. With regards to the puzzle game genre, improvements can only be made through more interactive artificial intelligence characters and new ways of designing mechanics.

THE SOLUTION APPROACH:

From the above literature, the principles of puzzle games have been evaluated to determine what works and what should be left out. This has been reinforced by looking at existing titles that present puzzles as their main concepts and the relationship between physical puzzles. Another element to consider for the design phase would be to take a cursory look at the way the levels in each game are presented to the player, as the way these are structured will be important in the later phases of design. From this, a set of technical objectives and final product requirements can start to be developed.

Pre-existing games and their features

Having researched both leading puzzle game titles and more independent titles, along with relevant academic material, it is clear that the game to be developed needs a specific type of mechanic or mechanics in order to make it unique and interesting, such a mechanic needs to be investigated in order to determine which is most appropriate for the product being developed. Successful games in this genre seem to involve some sort of abstract manipulation of a real world concept such as Time or Gravity. The title 'Braid' introduces some interesting mechanics on the reversing of time [34], with the aforementioned title of Limbo utilising gravity. Due to this, it appears that appropriate mechanics to implement would involve Gravity and Time.

As the project initiation document mentions, the aim is for the solution to include some sort of artificially controlled entity within the world. Therefore, concepts were reviewed as to what the most common behaviours for artificial game entities are. The title 'Hitman' was reviewed in terms of enemy intelligence as it incorporates stealth based mechanics such as hiding and avoidance. In terms of reviewing friendly intelligence, companion based behaviours like following or aiding in tasks also seemed most appropriate.

Comparison of game engines

Within the literature review, many game engines were mentioned, with particular attention drawn to the Unreal Engine, CryEngine and Unity engine. The article based on their comparison [35] was used to weigh up their relative merits and drawbacks, a table summarising the information is detailed below:

Table 1: Pros and Cons of Game Engines

Game Engine	Pros	Cons
Unity	Many features Simple Interface Cross Platform Support Large Asset Library Free for personal edition	Limited editing capabilities No modelling/building features
Unreal Engine 4	Advanced Graphic Capabilities New particle system Now uses C++ Free to initially use	Steep learning curve No backward compatibility for new generation consoles Separate license required for each console
CryEngine	Superior graphics to Unity Intuitive and powerful level design No Royalty fee	More difficult to grasp Monthly subscription

Game engine summary evaluation

From the above evaluation, the chosen engine has been Unity due to its feature rich, but simple layout, with large asset library for different needs of the developer. From the Unity website, it is clear that there is a range of import formats for models, which should mean that other external programs can be used to build models should the need arise. It also supports C# and JavaScript which are strongly typed languages so they may be more intuitive to the new programmer.

Research into chosen engine

In order to begin working on the project, the way in which the Unity editor worked needed to be explored. This was achieved by first installing Unity, followed by visiting the Unity forums and determining how detailed the documentation on how to use the editor was. Unity appears to have extensive documentation on how to use all of the parts of the editor and tutorials are present on how to implement a basic form of game in the form of videos [36]. Although a basic knowledge of programming appears to be assumed from these videos, they are easy to follow and explain how to use the editor and the programming environment sufficiently. Further research into the writings on how to use Unity revealed a useful book entitled 'Unity 5 From Zero to Proficiency (Foundations)', which further extended the basic knowledge attained from following the first few tutorials on the Unity website[37].

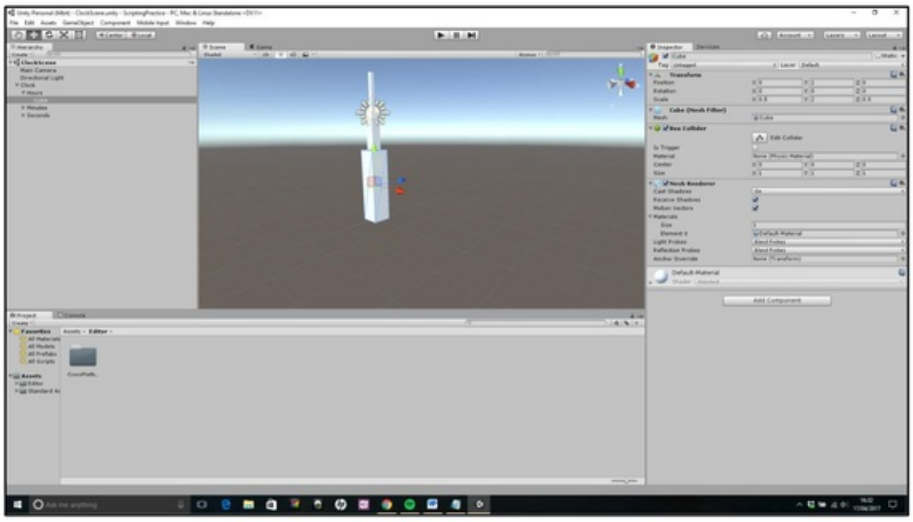


Fig.1 Overview of the Unity interface.

Just from these two sources, Unity was quick to pick up and most aspects of the editor interface were well explained and intuitive to use. This reflects that this particular tool may be very well suited to the average developer who wishes to create a game.

Chosen Solution

The final chosen solution is to use the Unity game engine to produce a Puzzle Platformer game that takes the shape of the 2.5D perspective. As the range of puzzle games spans across a range of types, from first person to third person, involving many different concepts, a certain degree of freedom exists as to what format the game should take. Therefore the style of the game to be developed has been influenced from personal preference. The flexibility of the Unity engine allows for any of the above discussed perspectives or mechanics to be introduced if so wished by the developer, making this particular engine a perfect candidate to complete this project.

The game itself will revolve around 3 core mechanics that the user will gain in order to solve puzzles within the world. The user interface should be minimal, with the controls being intuitively introduced to the player. In general, the use of a game engine removes the need for the developer to consider any external issues such as optimising for a given device, as this is all taken care of. As this project also considers how a novice in programming would be able to pick up and use this software, this comes as a welcome benefit.

As developers also need to be able to be flexible in how they design new levels and parts of the game, the design of each component should be such that time spent on adding new features once the base game is finished is minimised. By focusing on base mechanics and allowing the complexity of the puzzles being solved to get increasingly difficult, this will allow both new and experienced users of puzzle games to gain enjoyment out of the final product.

Other outlined features of the final product will be as follows:

- An entity that interacts with the user in some way
- Checkpoint saving system
- Various levels of puzzle complexity
- Easily reworked levels

DESIGN:

User Interface Design:

As the interface to the game as well as the aesthetical design needs to be minimal and not clutter the screen at any point, the amount of conveyed information should either be temporary on screen or should exist at the edges in order to allow the user to properly interact with the product. The two main conceptualised user interfaces would be that of the title screen (presenting levels and exit game functionalities) and the in-game interface itself.

The title screen needs no other information aside from the levels that can be played and a way for the player to exit the game. As more levels are added to the game, a paging system would be useful as to not clutter up the level select screen with too many buttons; the figure below shows an initial design concept for the main menu:

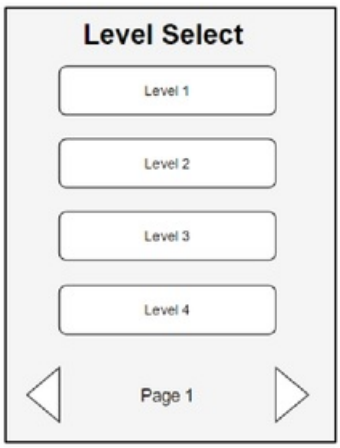


Fig.2 Conceptual level design main menu

In terms of minimising the interface presented to the user during gameplay, rather than constant reminders on how to perform certain actions, the player could be reminded of how to use such an action where it is required through pop-up dialogs. These could be triggered when a player walks through a specific area of the environment, which could show the player how to use certain controls or notify them of something important, such as whether they have reached a checkpoint or have visited a certain location in the game.

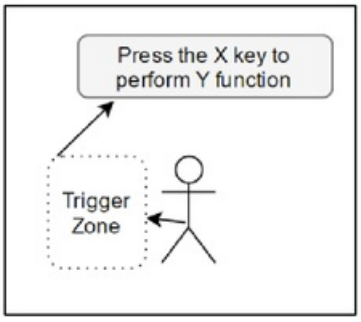


Fig.3 Player interaction with a trigger that displays a message to the user

As one required feature of the end product is to have interactions with an artificial entity, how the player interacts with the entity needs to be considered in terms of user interface. For example, if the user is to be interacting with a non-hostile entity of the world, then interface interactions could be solely in the form of pop-up dialogs, perhaps with player options that affect the resultant behaviour that is exhibited by the entity. Conversely, if the player is dealing with an enemy, how the player and enemy interact in terms of damage or health also needs to be considered in terms of interface. From the initial research and literature review, the title

'Super Meat Boy' was discussed. This particular game deals with the player's death by instantaneously respawning them at the start of the level.

As the levels being developed have a puzzle-centric approach in mind, it would be sensible to adopt a similar approach to handling death between the enemy and the player. This is due to the fact that the focus should be on allowing the user to reattempt the puzzle quickly, with the artificial object acting as a learning experience that the player should avoid this particular enemy. This would then allow for no health bar or damage calculation interface to be incorporated, further simplifying the user's experience.

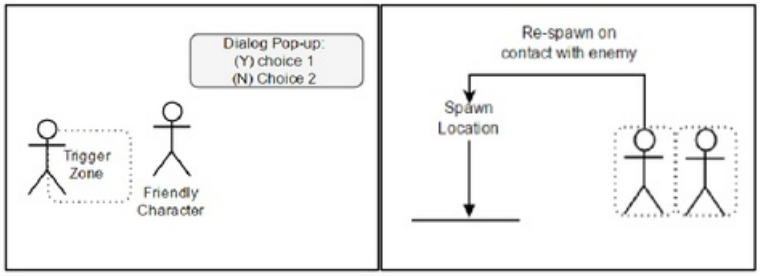


Fig.4 Friendly/Hostile interactions to minimise interface.

Level Design:

As the mechanics that were to be included in the final product were already drafted, level designs could be conceptualised on paper. For each level, the player needs a goal that is simplistic (for example, navigate to this point), but the transition to the goal may be complex and require use of the player's knowledge of the mechanics to solve. Additionally, in some cases, the player may be able to reach the goal in multiple ways, by utilising different mechanics. This will enhance the replayability and will allow more complicated levels to be designed, however it is important that the goal can definitely be reached by at least one method.

Due to this, from the defined mechanics, a large set of potential puzzles could be produced. Potential categories of puzzle and solution could be as follows:

1.) **Solution by direct observation**
The player could solve the problem directly by observing something simple about their environment, such as to jump over a gap to reach the other side or to turn on a switch to enable progression to the next stage of the level. This is the simplest type of puzzle, and may not be as enjoyable for the player to solve as it requires no lateral thinking.

2.) **Solution by needed item**
The player could require a necessary item to help solve the problem that is hinted by the environment, for example a key to a locked door. This notifies the player on what to do and what is needed, which will require the player to locate the key based on previous or existing knowledge, allowing more creative puzzle solving.

3.) **Entity Interaction/ Mechanic Puzzles**
Utilising the mechanics that are demonstrated by the game, such as time manipulation, in order to overcome problems perhaps offers the most flexible way to approach designing levels. As the player can be taught these concepts early on, they can learn that all puzzles can be solved using one of, or a combination of, them. This strongly suggests that the program being developed should include a tutorial section in order to introduce the player to these basic ways of solving problems.

From these options, it appears that the best approach is to use them in combination, with the core mechanics as a base to solve all problems. Then, with the addition of less complex solutions such as direct observation, will strike a good balance of difficulty amongst the created puzzles.

Perspective/View Design:

As the decided theme is 2.5D, the angle of perspective needs to highlight the 3 dimensional world that is constrained to 2 dimensional movement. It also needs to be considered whether the area of focus in a given level should be on the player (following them as they move) or whether an overview of the level should be presented within which the player moves around. As the Unity engine will provide the main environment for adding objects to a scene, and can handle both 2D and 3D games [38], the 2.5D effect can be achieved by creating a 3D project in Unity, angling Unity's camera object to an aesthetically pleasing angle, and restricting a 3D character to only move in 2 dimensions. The camera can then be used to follow the player as they move, meaning that levels larger than the camera size can be explored.

Initial Level Designs:

Paper designed tutorial levels were drafted along with a conceptual finished level to be explored when the product would be completed. For demonstration purposes, three tutorial levels would be produced as well as a fragment of a potential level that would show the combination of the mechanics, simulating more advanced puzzles that the user would have to solve at a later stage in the game. Once the project has concluded, fully complete levels can be produced, incorporating more puzzles in different formats. The following figures show the initial design concepts for levels:

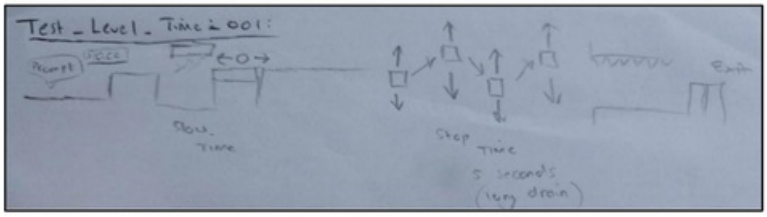


Fig.5 Initial concept tutorial 1

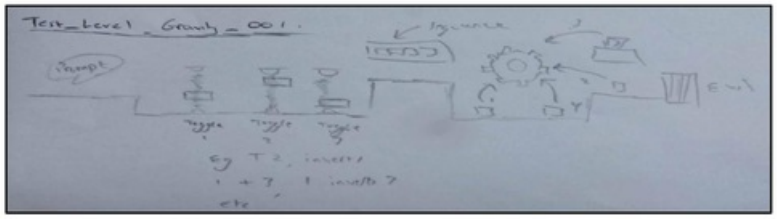


Fig.6 Initial concept tutorial 2

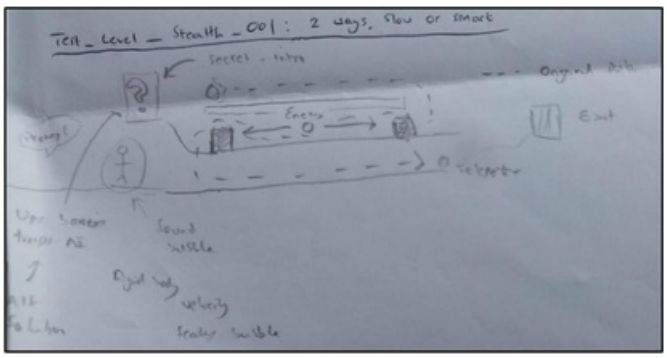


Fig.7 Initial concept tutorial 3

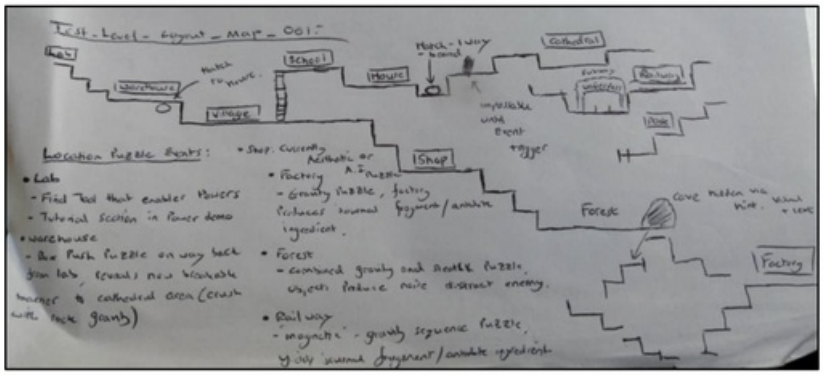


Fig.8 Initial concept of entire level

An updated concept for the level fragment that consisted of multiple level mechanics was also drafted, based upon content from the previous tutorial levels:

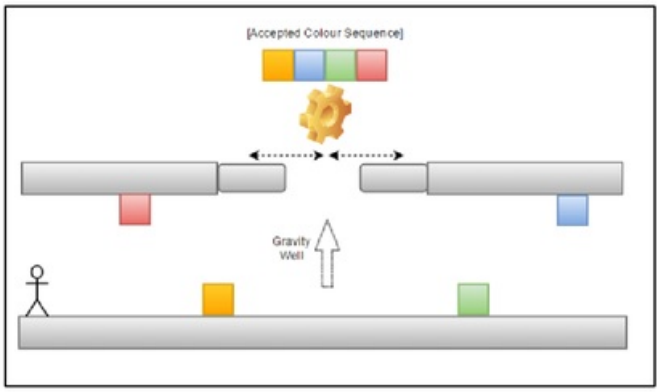


Fig.9 Multi-mechanic level concept

The above concept revolves around both gravity and time manipulation; the player must slow down the moving platforms in order to gain access to the object accepting the sequence. Following this, they must alter gravity in order to walk on the ceiling, which will enable them to push the relevant blocks into the gravity well, which will carry them up to be checked against the correct sequence.

The behaviours that need to be exhibited in these levels were then translated into concepts to form a basis for the programming that needs to be done in order to implement them.

Aesthetic and Story Designs:

As the primary nature of the project is to design and implement the behaviours of the product, the way the game looks, alongside inclusion of a storyline, will not be considered in depth because of time constraints. For a game of this genre however, simple and clean textures could be applied to enhance the visual appeal to the user. If this game was to progress further in areas such as storyline, this could have an impact on considered ethical issues within the project. For example, if the story content or graphics reflected violent or other adult themes (such as drug use), then consideration about its game rating would have to be made prior to its release [39]

Mechanics/Logic Design:

Stealth:

Initially, stealth was conceptualised as a value that related to the sound that the player made. The more sound generated, the higher the value of the variable. This would then effect whether or not you could be seen by other sensing entities in the world. This idea then progressed to a potentially more realistic representation of sounds that involved the sound bouncing off surfaces, and if a sound collided with a sensing entity, the location of the player would become known to it. Implementing radial sound in this way however could be expensive computationally as calculations would have to be made for all surfaces the sound bounced off. Because of this, a more simplified model needed to be determined.
As sound is created in all directions from the player when they move, sound could be simplified to that of a sphere, with its radius representing the area of sound that is being produced. If possible within Unity, advantage could be taken of the known mathematical shape, potentially reducing computational complexity through already known equations, with only the properties of the sphere needing to be changed as more or less noise is produced by the player.

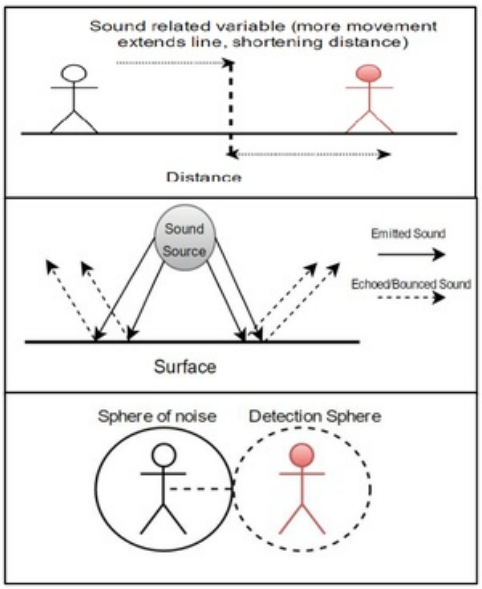


Fig.10 Sound mechanic progression

Time Manipulation:

For time manipulation, there is a variety of different possibilities of altering time and speeds that could then be applied to puzzles. Time could be slowed in a specific area of the level, with puzzles revolving around slowing parts of the world down in order to traverse the environment or to allow for specific timings to be involved in the puzzle. Another approach would be to slow all objects but the player down, to achieve a similar effect.

Gravity Manipulation:

Again, as the nature of gravity simulation can present a number of different variations of puzzles to be solved, a few possibilities will be considered here. From the level design concepts, the idea of moving platforms with gravity in order to organise ways to get across gaps or other obstacles could create some interesting logical puzzles.

Gravity could also be incorporated into supplying a specific object with other objects in the world, using centred gravity. This could allow for the player to push objects that are affected by the central gravity to solve a sequencing puzzle (as shown in the level designs). Finally, the most flexible gravity-related mechanic appears to be the allowance for the player to walk on different surfaces, such as ceilings and walls, by inverting gravity.

Applying this practically, the player's controls and variables that effect their gravity and movement would have to be inverted when the world was altered.

Artificial Intelligence:

Initial Ideas:

For the interactive portion of the game relating to artificial intelligence, the complexity of interaction with the player will heavily dictate this entities complexity and behaviours. At its most basic level, the non-player character (NPC) would have to determine where the player is in the world. This could be implemented in the form of a following behaviour where the NPC simply scans the world to identify the player's position and navigates towards that location.

This could then be further advanced to include other behaviours, for example avoiding obstacles in-between the target and the NPC's position. This could involve both horizontal and vertical movement so would require calculations of, not only where obstacles exist but also, the best path to take to the current target. As the developing genre of game is to consist of platforms separated so that the player can jump between them, the NPC character must also have the ability to do this in some form.

For hostile characters, more complexity could be added by simulating different senses such as hearing or sight. The hearing parameter could give an approximation of the targets location, whereas sight would give an accurate location description. Due to this, the implementation of the NPC could become very complex and difficult to follow; meaning a way to simplify its implementation should be considered.

Finite State Automata:

Within the compilers module studied in previous years, the theory behind finite state Automata was learned. This is how to describe a machine using different states, with certain criteria causing a transition between these states. Much like this, an artificial entity in the world could be described with different states, such as listening, seeing and walking. Hints at how this can be implemented, along with an introduction to finite state automata and its limitations, are discussed in an extract from 'Game Programming Patterns' by Robert Nystrom, with certain programming concepts such as enums and switch statements being suggested as a way to determine the new state of the machine based on specific conditions [40].

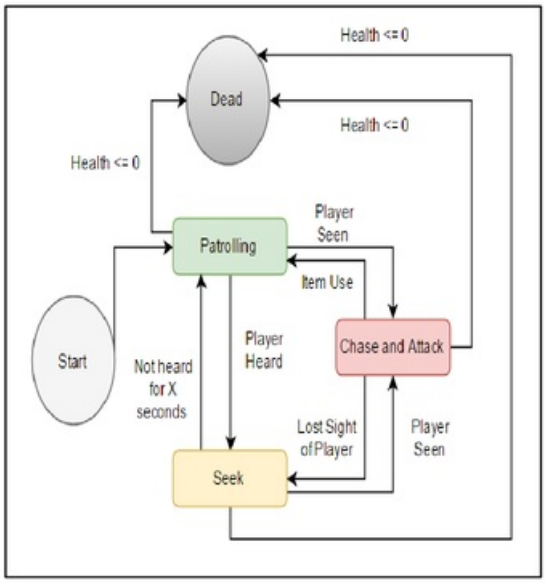


Fig.11 Diagram of states for Finite State Machine (FSM)

Additionally, as the world is segregated into platforms, a way in which the NPC determines where walkable platforms exist, and how to traverse them, needs to be taken into account, suggesting that a pathfinding algorithm needs to be implemented. This could be done by producing a map during load time of all of the walkable and non-walkable spaces in the world that is then used by the character to navigate.

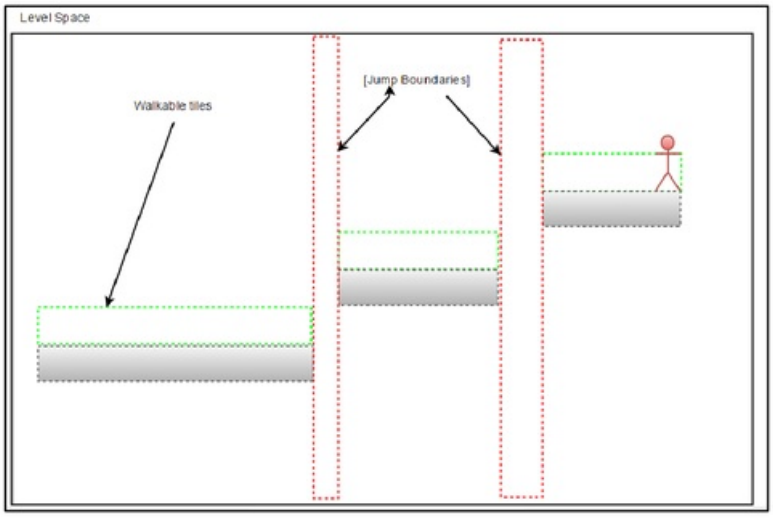


Fig.12 Concept for world mapping for NPC/AI

After evaluating the possible designs necessary, thought can be given to how this will be set up within the game engine environment itself. In most self-developed games, a central loop known as a game loop is required to handle most of the processing done within a game. This could be updating physics, checking for collisions or waiting for events. In general, it acts to advance the state of the game slightly [41].

The Unity engine already implements this game loop for the developer, meaning that the game is advanced frame by frame using one of Unity's many methods. Some of these are applied in the scripts that are being run, and either act as one time run functions to initialise certain parameters(Unity's Awake() and Start() functions, or as continuous updates that are made once per frame or more (Unity's Update(), LateUpdate() and FixedUpdate() functions) [42].

Therefore, every script written in Unity that is attached to an object in the game is updated through these functions. This can be visualised as the game engine acting as a central hub, will all scripts utilising it to execute their logic. At this stage, the potential functions that the game needs to perform can be estimated, resulting in a list of potential scripts. The figure below demonstrates the potential scripts and how they would interface with the game engine.

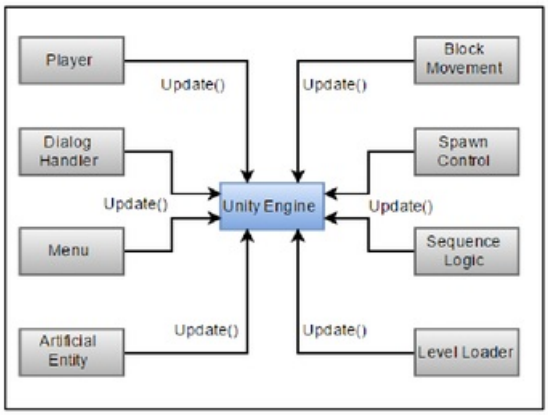


Fig.13 Game Interaction Diagram with Unity acting as a central hub

Due to the nature of the interaction needed between certain parts of the game world, some scripts may need references to other scripts in order to work effectively. For example, the script controlling the NPC behaviour may need a reference to the player object in order to access properties like its location. From the potential scripts outlined in the above figure, potential dependencies between scripts will be identified below.

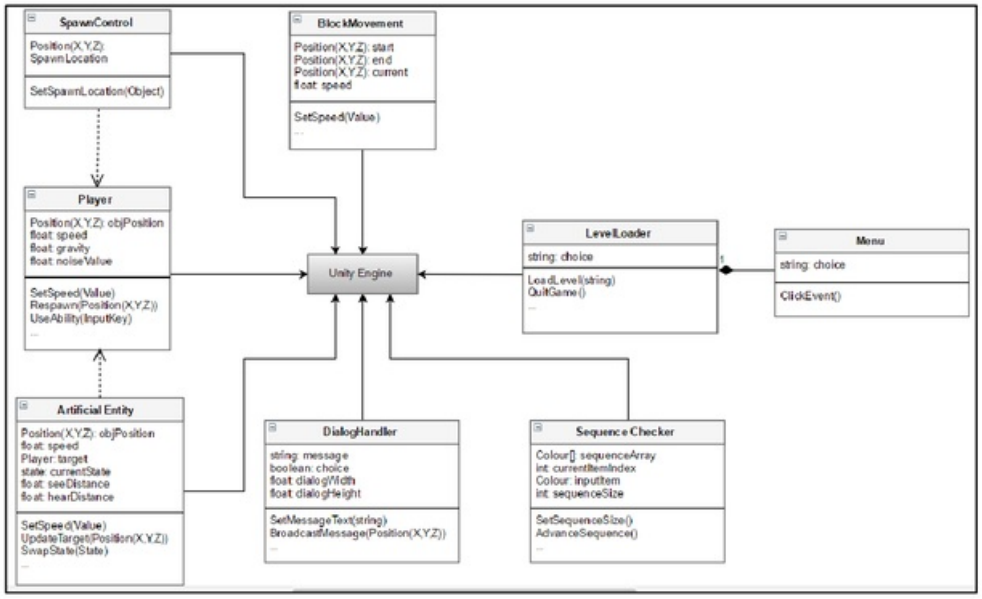


Fig.14 Conceptual Interactions of Scripts

Application of Design to the Unity Editor:

Through experimentation with the Unity editor, as well as inspecting its documentation, potential ways to implement each mechanic have been realised, below is an outline of how designed mechanics could be translated into functions using the C# language and Unity features.

Unity Editor: Level Design

Unity structures its environment based on scenes. Each scene corresponds to either a 3D or 2D environment where the developer can import assets or manipulate pre-existing features to construct a game world. There is a large variety of primitive shapes offered within the editor that can be independently scaled, rotated and translated in the scene. As the nature of the game being created revolves around platforms, the primitive cube shapes pair perfectly with constructing platforms in a very short space of time.

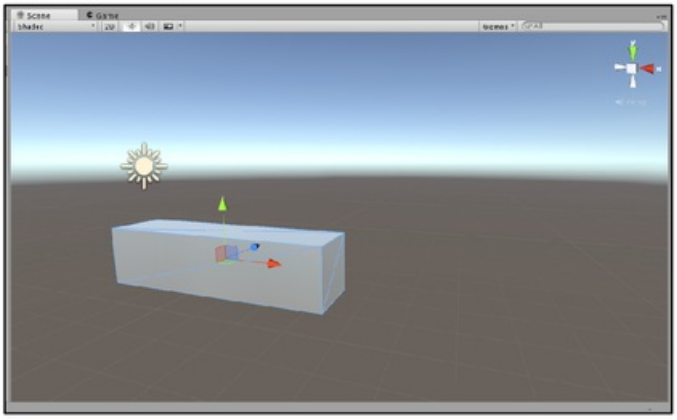


Fig.15 Primitive cube altered to produce a platform

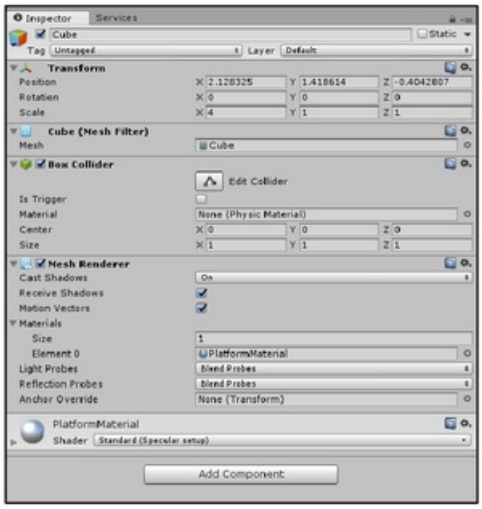


Fig.16 Unity inspector showing cube properties

From this, a quick mock-up of the first level was produced in order to verify that Unity's primitive shapes could be used to construct a platformer like design.

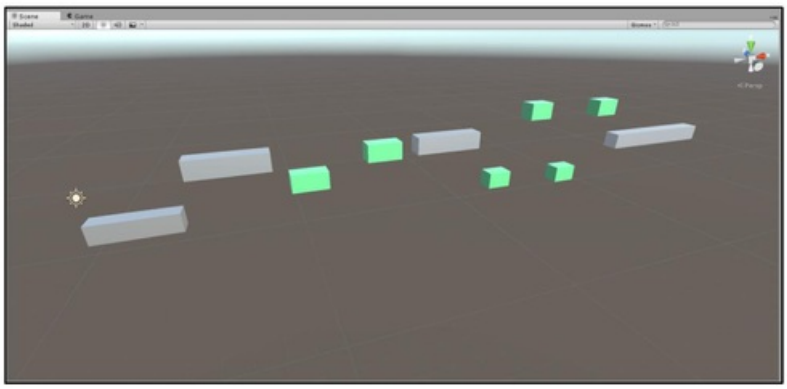


Fig.17 First level concept drafted into Unity

Furthermore, inspecting the properties of the shapes revealed that they have a transform component, storing their position as a 3-dimensional vector. This could be manipulated in scripts to provide motion of platforms, which would enable the controlling of object speeds for the time mechanic.

Unity Editor: User Interface:

Unity also contains a User Interface system, allowing for simple on screen text to be displayed [43], and camera objects that can be positioned and attached to other objects. This allows for easy ways to both follow the player using an in-built camera object and display information to the user without cluttering the interface all of the time. Textures, materials, sounds and 3D models can also be imported with relative ease, simplifying any need to provide visual detail to the world.

Unity Editor: Stealth Mechanic:

Collisions in the game engine are handled by colliders, which are primitive shapes that have wireframe representations and can be set so they can either be ignored by entities and other objects (enabling trigger events to occur), or act as barriers to restrict movement and provide a surface for other objects to rest on/interact with.

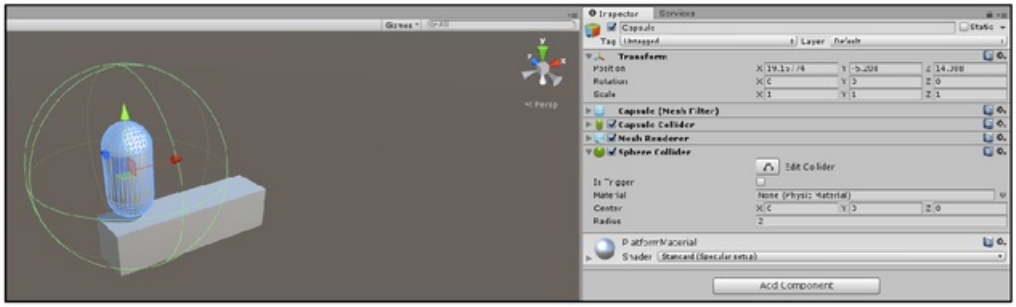


Fig.18 SphereCollider component attached to Unity's capsule primitive

The collider shown in the above figure could be utilised for the stealth mechanic as the SphereCollider could be used to detect when a part of the sphere has passed into an object, meaning sound could be represented this way.

Unity Editor: Time Mechanic
Research into how Unity handles its updates revealed a Time class that includes a property of 'Time.deltatime'. Deltatime handles the time elapsed since the last frame. The associated documentation states that, if adding or subtracting to a value in a frame, it should be multiplied with this value [44]. This suggests that the speed of objects can be controlled with a speed variable multiplied by Time.deltaTime. This demonstrates that an objects speed of translation could be moderated by simply altering a speed variable within a script, meaning that the apparent slowing down of objects with respect to time is possible. If more complicated physics were necessary, Unity's Rigidbody component could be used in a similar way with adding forces such as velocity to an object with respect to Time.deltaTime.

Movement and Gravity Mechanics:
For all required movement by the game, objects transform components can be translated and rotated by using Unity's Vector3 library, as it contains many potentially useful methods of moving between vectors (.MoveTowards and .Lerp), as well as Vector maths[45]. As an object's transform in Unity is stored as the type Vector3, most movement throughout the game world will heavily involve this library. These library functions could also be used to translate an object towards a point acting as a center of gravity, which can be useful in implementing the gravity mechanics.

For the player's movement, Unity has a CharacterController class that allows the developer to easily perform collision constrained movements. This appears to be a more simplified way of implementing player-based behaviour and may save more time in the implementation phase as the script attached to the object will only allow it to move when the Move() function is called, allowing for direction specification and gravity calculation [46] [47].

For the gravity mechanic that involves checking against a sequence of different blocks, simple data structures such as Lists or Arrays could store the expected sequence, which is then compared to the objects entering the sequence checker at runtime.

Artificial Entity Mechanics:

As this script would need to allow player like movement, it may also need to include a CharacterController object, with the difference of predefined movement directions based on a target. If gravity or collisions need not be obeyed, then simple vector movement may suffice for the artificial entity. As found earlier, for a finite state version of an artificial intelligence, programming structures such as enums may be useful. If the AI needs to pathfind around the world based on walkable and non-walkable spaces, a grid could be created based on the 3D environment (although this could be computationally complex for 3D environments with platforms at varying elevations).

To enable the sensing of the AI, intersection tests of noise spheres could simulate the hearing sense. Raycasting is a technique that has been used for such concepts as determining if a player has been hit in a first person shooter game. A ray is drawn from the point of the gun that extends until it intersects with something. Calculations are then completed based on what the ray hits. This has also been used in rendering applications [48]. Unity has a Raycast library that allows the developer to specify a ray origin, and then get information on

where the ray collides [49]. This could be appropriate for the sight sense as information about both the player and platforms could be gained from casting

Design Phase Evaluation: Relative Ease of use to end user

Upon completion of the design phase of this project, the extent to which prior technical knowledge was needed should be evaluated in order to see if an inexperienced user could design both the levels and necessary behaviours. It has been determined that the availability and detail for Unity documentation on all scripting information is detailed and, in most cases, shows an example implementation of the element being demonstrated. This should mean that a user would have relative ease in integrating these scripts with the world they are trying to develop. Designing level layouts with the primitive shapes is very simple, with intuitive controls for scaling, rotating and positioning objects, meaning that a new developer could quickly gain a visual representation for their world. If the new developer wished to add more complicated models, these could be created in other applications and then imported as an asset or downloaded. This is also true of scripts, which could allow an aspiring developer to gain robust, working code that solves more complex tasks than they perhaps can initially create themselves, but also gives them an opportunity to browse that code to further their own understanding.

Conversely however in more complicated theoretical designs like the FSM or how to properly perform object movement, some degree of knowledge of maths is required on concepts like Vectors and more complex programming concepts like structs, enums and switch statements. Additionally, decisions made on the potential computational complexity of any given solution would not be able to be made by someone with little knowledge of how expensive different checking methods are, this therefore may result in an inexperienced user producing an inefficient, or perhaps unusable product, which may put them off? Although concepts like inheritance need not be explored fully, although possible, as unity's scripting system in attaching to objects can eliminate the need for it for games of certain complexities. Similarly, the correct data structures to be used for a specific mechanic (such as arrays for checking a sequence) may not be known to a novice developer, meaning that they may produce inefficient code or form bad coding habits from not using the most appropriate method of implementation.

IMPLEMENTATION:

Using the outcomes of the design sections, scripts to implement their relative behaviours in the world needed to be produced. Firstly, pseudocode representations of the logic behind each feature/class to be implemented were drafted, with this then being translated into code. During implementation, classes that were not foreseen during the initial design process were needed in order to produce a coherent finished product

Player Control and Ability Use

This class needed to handle the basic movements of a user, such as moving and jumping, as well as being able to produce sound upon moving, detect enemy collisions and influence the environment through abilities. The figure below shows a pseudocode implementation of the player mechanics. All pseudocode assumes constant updating utilising Unity's update methods.

```
begin
initialise variables
initialise noiseRadius
initialise currentworldpos

if(user input){

    //For Abilities
    CheckKeyPressEvent();

    move(UserinputDirection);
    UpdateWorldpos();

    //For detection purposes
    UpdateNoiseLevel();

    //For Jumping
    if(input = Jumpkey){
        VerticalMove();
    }
}
//For Collisions
    if(CollideWithEnemy = true){
        RespawnPlayer();
    }
end
```

Fig.19 Initial Pseudocode for player class

From this basis, the program was implemented using Unity in combination with visual studio. The script was attached to a primitive 'Capsule' shape and was given, along with its existing components, a sphere collider to handle noise and a CharacterController to aid with movement.

```
//Check if the controller is on the ground and the camera has not been rotated
if (controller.isGrounded && camMovement.hasRotated == false)
{

    //Get horizontal movement input from the player
    moveDirection = new Vector2(Input.GetAxis("Horizontal"), 0);
    //Relate the move direction to the objects transform (for movement)
    moveDirection = transform.TransformDirection(moveDirection);

    //Multiply the movement transformation by the objects speed
    moveDirection *= speed;

    //Handles vertical movement using a the Jump button (space)
    if (Input.GetButton("Jump"))
    {
        //Affects the y component of the players Vector2 movement
        moveDirection.y = jumpSpeed;
    }

    //If the camera has rotated, that means the player is upside down
}
else if (controller.isGrounded && camMovement.hasRotated == true)
{

    //Get the input from the player and negate it so that the movement when flipped is the same
    moveDirection = new Vector2(Input.GetAxis("Horizontal"), 0);
    moveDirection = transform.TransformDirection(-moveDirection);
    moveDirection *= speed;

    if (Input.GetButton("Jump"))
    {
        moveDirection.y = jumpSpeed;
    }
}

//Check if the player is holding down the move buttons and if they are, update the noiseSphere
updateNoiseRadius();

//Calculate the gravity effect on the controller by subtracting from its y component over time
moveDirection.y -= gravity * Time.deltaTime;

//Move in the direction as supplied by the user over time
controller.Move(moveDirection * Time.deltaTime);
```

Fig.20 Code snippet of character movement and jumping

First a check was made as to whether the controller was on the ground through the inbuilt 'isGrounded' Boolean and what orientation the world was in (inverted or not). This was then used to get horizontal axis information from the user input, supplying the direction of movement (for example, the left arrow key input press corresponded to a certain related 2-dimensional vector of the form <-1,0>). This was then translated into movement through the Move() function for the CharacterController attached to the player object. The speed of the player was controlled through its speed variable and the Jumping mechanic by altering the y-component of the movement vector using the jump speed variable. Gravity was simulated by subtracting a Gravity value from the player's movement vector over time using Time.deltatime. As the CharacterController class works off collisions, the isGrounded function would ensure that the player would not just endlessly fall in the world. The basics for movement were adapted from existing Unity documentation on the CharacterController class's move function.

Noise was updated through a function that altered the radius of the SphereCollider object attached to the player object as shown in the figure below:

```
private void updateNoiseRadius()
{
    //If the player is moving, track how long they are holding the button down
    if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.Space))
    {

        //If the moveTime is greater than the maxNoise stop incrementing
        if (moveTime >= maxNoise)
        {
            moveTime = maxNoise;
        }
        else
        {
            //Gradually increase the time moving by the acceleration of the player upto a maximum
            moveTime += acceleration;
        }

    }
    else
    {
        //If the player is not moving, reduce the players noiseSphere down to the minNoise value
        if (moveTime <= minNoise)
        {
            moveTime = minNoise;
        }
        else
        {
            moveTime -= acceleration;
        }
    }
    //Update the radius of the players spherecollider to represent the noise
    noiseCollider.radius = moveTime;


}
```

Fig.21 Noise updating code as the player

The CharacterController class does not deal with Physics directly, therefore in order for the player to be able to interact with other physical objects, they must have RigidBody components attached and the developer must utilise Unity's inbuilt function 'OnControllerColliderHit()', which in its documentation demonstrates how to allow a CharacterController to interact with Rigidbody objects [50]. This was utilised to allow the player to push blocks that would become useful during the gravity and sequence based puzzles implementation.

```
//Handles the collisions between the object and any other object it connects with that has a Rigidbody
void OnControllerColliderHit(ControllerColliderHit hit)
{
    //If the body that is collided with has no rididbody or isKinematic (no interaction with physics)
    Rigidbody body = hit.collider.attachedRigidbody;
    if (body == null || body.isKinematic)
    {
        return;
    }


    if (hit.moveDirection.y < -0.3F)
    {
        return;
    }

    //Define a new vector that the hit object is to travel along
    Vector3 pushDir = new Vector3(hit.moveDirection.x, 0, hit.moveDirection.z);

    //Apply velocity to the collided objects rigidbody to enable a pushing behaviour based on pushpower
    body.velocity = pushDir * pushPower;
}
```

Fig.22 Implementation of player push interaction with blocks

The Input class used within the movement portion of player implementation can be used to determine any key pressed by the user using methods such as (GetKeyDown(),GetKeyUp() and GetKey()). These were used to handle the player's different abilities to change the world in various ways.

```
//When X is pressed
if (Input.GetKeyDown(KeyCode.X))
{
    //Toggle whether the world is flipped or not
    if (isFlipped == true)
    {
        isFlipped = false;
    }
    else
    {
        isFlipped = true;
    }

}

if (isFlipped)
{

    //If the camera on the player has not rotated (inverted), call rotate
    if (camMovement.hasRotated == false)
    {
        camMovement.rotateCamera();
    }


}
if (!isFlipped)
{

    //If the camera on the player is currently inverted
    if (camMovement.hasRotated == true)
    {
        //Rotate back to normal
        camMovement.rotateCamera();
    }

}
```

```
    //If the user presses the H key
    if (Input.GetKeyDown(KeyCode.H))
    {
        //Find all instances in the world where an object is tagged with 'isSlowable'
        GameObject[] myArray = GameObject.FindGameObjectsWithTag("isSlowable");

        foreach (GameObject g in myArray)
        {
            //Get the objects movement component to access its speed
            platformMove myobject = g.GetComponent<platformMove>();

            //Get the speed before you modify and store in a var
            originObjSpeed = myobject.speed;
            myobject.setSpeed(slowAmount);
        }
    }
    else if (Input.GetKeyDown(KeyCode.J))
    {
        //When J is pressed, set the object speed back to its original value
        GameObject[] myArray = GameObject.FindGameObjectsWithTag("isSlowable");
        foreach (GameObject g in myArray)
        {
            platformMove myobject = g.GetComponent<platformMove>();
            myobject.setSpeed(originObjSpeed);
        }
    }
}
```

Fig.23 Code snippet showing time manipulation (right) and gravity inversion calls(left)

The X key is designated for altering the gravity of the world and relies on another Class called cameraFollowScript. This allows the Camera focussing on the player to rotate, adding a visual effect to the game as well as handling the gravity mechanics which will be discussed below. The H and J keys were used to implement time slowing effects in the world. The Unity environment has a way to assign objects to certain groups using reference words known as 'Tags' [51]. These were used to identify all of the objects present in the scene that could be slowed down by the player.

All objects in the current scene with the tag 'isSlowable' are added to an array. This array is then iterated through; getting the attached movement script to that object (meaning that its speed can be modified) and using the setSpeed method of the platformMove class to increase or decrease object movement speeds. This successfully implements the apparent slowing down and speeding up of certain parts of the gameworld.

Related Class: cameraFollowScript

As described in the design section, the player also requires the ability to alter gravity in order to walk on ceilings of levels. This class was implemented to complement the Player class by manipulating an attached Camera object to the player. To provide a visual effect of the world turning, the transform of the Camera was rotated slowly through the self-defined .rotateCamera() function. Once the camera rotation reached a specified value of 180 degrees, the world would then be rotated. Initially, the CharacterController attached to the player was going to be rotated, with calculations performed to alter what parts of the CharacterController were grounded. However in practice this proved difficult to implement as Unity has global gravity acting on CharacterControllers that acts downwards. Due to this, and to reduce potential calculations involved, a parent object was created with all objects needing to be rotated set as its child in Unity's hierarchy view, with the transform of this parent object being altered so that the world itself rotated 180 degrees, achieving the same effect.

```
public void rotateCamera()
{
    //Get the current value of rotation about the z axis of the object
    currentcamRotation = transform.rotation.z;

    //Define a target rotation of 180 degrees
    Vector3 targetRotation = new Vector3(0, 0, 180);

    //If the world has not been rotated
    if (hasRotated == false)
    {
        //If the current rotation of the camera is equal to 180 degrees
        if (transform.eulerAngles == targetRotation)
        {
            //Set hasRotated to true
            hasRotated = true;
            //Find the parent gravity object and set its transform(rotating the world)
            transform.parent.gameObject.transform.Rotate(new Vector3(0, 0, 180));

            //Move the camera slightly so that it still centers the player
            transform.position = followTarget.transform.position + camDelay;
            camDelay = transform.position - followTarget.transform.position;

        }
        else
        {
            //Rotate the camera smoothly clockwise on each call
            transform.eulerAngles = Vector3.MoveTowards(transform.rotation.eulerAngles, new Vector3(0, 0, 180), 100 * Time.deltaTime);

        }

    }
    else if (hasRotated == true)
    {
        if (transform.eulerAngles == targetRotation)
        {
            //Set it to 0 degrees and set hasRotated to false
            hasRotated = false;
            //Find the parent gravity object and set its transform
            transform.parent.gameObject.transform.Rotate(new Vector3(0, 0, 180));

        }
        else
        {
            //Rotate the camera smoothly anti-clockwise on each call
            transform.eulerAngles = Vector3.MoveTowards(transform.rotation.eulerAngles, new Vector3(0, 0, 180), 100 * Time.deltaTime);
        }
    }
}
```

Fig.24 Smooth camera rotation and world inversion cone snippet

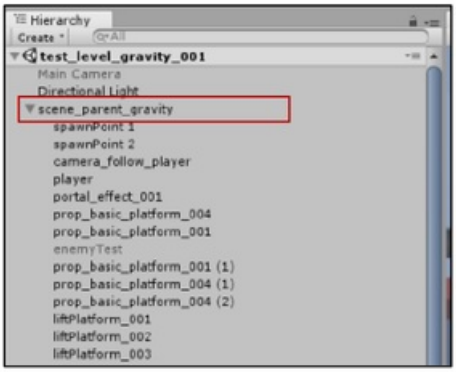


Fig.25 Hierarchy view of scene with gravity parent object

Finally, basic functions for the player were implemented in order to handle respawning after death and interactions with enemy based objects:

```
//Create function so if you stray too far from the platform you respawn
//Requires there to always be a spawnPoint with that name and all to be tagged respawn
public void Respawn()
{
    //Get the new location of the latest spawnpoint object and set it to that
    GameObject[] spawns = GameObject.FindGameObjectsWithTag("Respawn");
    if (isFlipped)
    {
        //If the world is flipped and you need to respawn, do so at an inverted spawnpoint
        foreach (GameObject g in spawns)
        {
            if (g.name == "spawnPoint 1")
            {
                transform.position = g.transform.position;
            }
        }
    }
    else
    {
        transform.position = currentSpawnPoint;
    }

}

//Handles when the player comes into contact with the enemy
void OnTriggerStay(Collider other)
{
    //If the collided object has a tag of enemy and you are sufficiently close to that object
    if (other.tag == "enemy" && Vector3.Distance(transform.position, other.gameObject.transform.position) < 1.2)
    {
        //Handles the case of a specific enemy type, requiring change of that particular enemy's state
        if (other.name == "testEnemy")
        {
            //Reset the enemy's position
            other.GetComponent<testu>().transform.position = other.GetComponent<testu>().initialSpawn;
            other.GetComponent<testu>().swapState(testu.state.PATROLLING);
        }
        //Then respawn
        Respawn();
    }
}
```

Fig.26 Respawn and enemy interaction behaviours

If the player collides with one type of enemy, the script would access that enemy's AI script and alter its state back to patrolling and return the associated AI body to where it initially spawned in the world. In all other cases, it will cause the player to respawn on contact with the enemy. Respawning is handled either by the currentSpawnPoint variable (set in another script handling spawn points) or at 'spawnPoint 1' if the world is inverted at the time of the players death. Below is a final figure showing the player movement along a platform with their movement affecting a noise sphere.

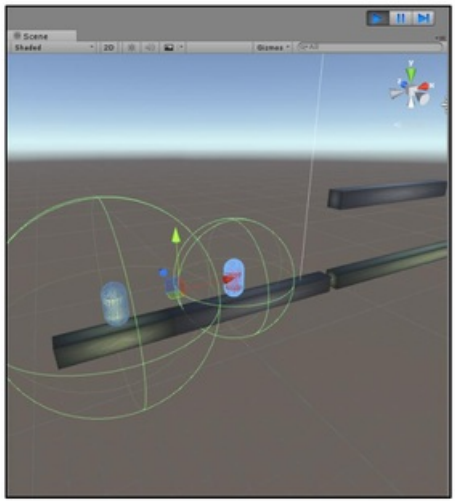


Fig.27 Demonstration of noise sphere mechanic

Artificial Entity (EnemyFollow)

This class needed to be able to move like a player, utilising noise and visual information on the player to identify their location and track them. Additionally, the layout of the environment needed to be known so that it could be properly traversed. Again, an initial pseudocode representation of the class to be implemented was constructed.

```
Begin
initialise variables
map environment to set up pathfinding
initialise state
initialise spawn point
pick intial target

switch(current state){

    case state:Dead
    //Remove from game
    Destroy associated gameobject;
    break;

    case state:Patrol

    if(target = reached){
        //Pick a new target
        currentTarget = pathfinding[randomVal];
    }else{
        for(all pathfinding map points){
        //Pick a random position to walk to
        currentTarget = pathfinding[randomVal];
    }

    Move towards currentTarget;
    }
    //Check if you can see or hear the player
    if(CanSeePlayer = true){
        Transition to Chase state;
        break;
    }
    if(CanHearPlayer = true){
        Transition to Seek state;
        break;
    }
    break;

    case state:Seek
    //Move towards where player was heard
    currentTarget = pathfinding[heardPosition];
    Move towards currentTarget;
    break;

    case state:Chase
    //Move towards where player was seen
    currentTarget = pathfinding[seenPosition];
    Move towards currentTarget;
    break;
}
end
```

Fig.28 Initial pseudocode for NPC/AI implementation

When implementing into Unity, many more complexities arose as to how to deal with features of the environment like elevation and enemy positioning, as well as calculating realistic movement between the platforms of the world based on distance. Challenges were also faced when designing the mapping function to inform the artificial entity about the environment it was traversing.

Firstly, it was required for the AI to move around and successfully arrive at a given point in space. This was done quickly with the Vector3.MoveTowards() function that specifies a start and end vector for the enemy to move towards. Although this method worked, it did not take into account the environment and therefore would ignore any collisions. Due to this, it was realised that, in order for the enemy to navigate the environment properly, it needed to behave like the player. Following this, Vector3 movement was exchanged for CharacterController movement through the move function. The final object was created in the Unity editor with an attached capsule collider, sphere collider for sensing radius and a Rigidbody for collisions.

As the game was to implement platforms, the first part of the class was used to setup all possible walkable points on the map. This was done by creating a Waypoint class that stored both the position in 3D space that the walkable point existed, but also a Boolean value to determine if the enemy was currently on the edge of that platform, the figure below shows the code implementation.

```
//Used to store walkable points in the map
public class Waypoint
{

    //This class will store 2 variables, one for if the object is an edge, and the Vector3 position of that block

    public bool onEdge;
    public Vector3 walkablePoint;


    //Constructor for the creation of a waypoint

    public Waypoint(float xComp, float yComp, float zComp, bool edge)
    {
        this.onEdge = edge;
        this.walkablePoint = new Vector3(xComp, yComp, zComp);
    }


}
```


Fig.29 Waypoint class implementation

World Mapping:
From this basis, platforms were placed in the Unity world using primitive cube shapes and altering their scales, which were then tagged with the identifier 'Platform' to alert the script that it should consider that platform as walkable in the world. Utilising this information, a function was created that mapped the initial positions of all objects marked Platform, and calculated the number of walkable tiles on top of each based on a 1x1x1 Cube primitive.

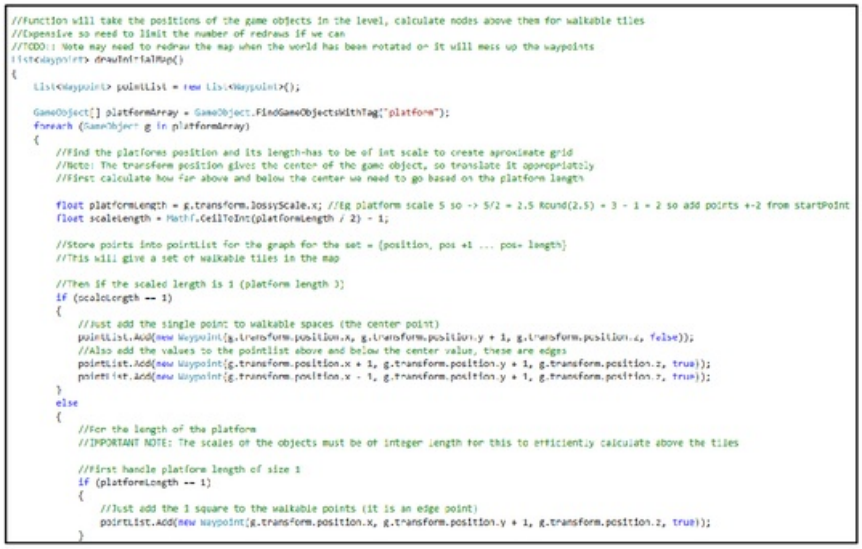


Fig.30 Mapping of world space into walkable and edge spaces of platforms

This function created a list of Waypoints and, after finding all objects with the tag 'platform', calculated points above each platform that approximated walkable spaces in the world. This was achieved by getting the scale of the platform (the extent to which a 1x1x1 cube had been altered in the X direction to represent a platform), and calculating how many 1x1x1 cubes would fit into a platform of that length.

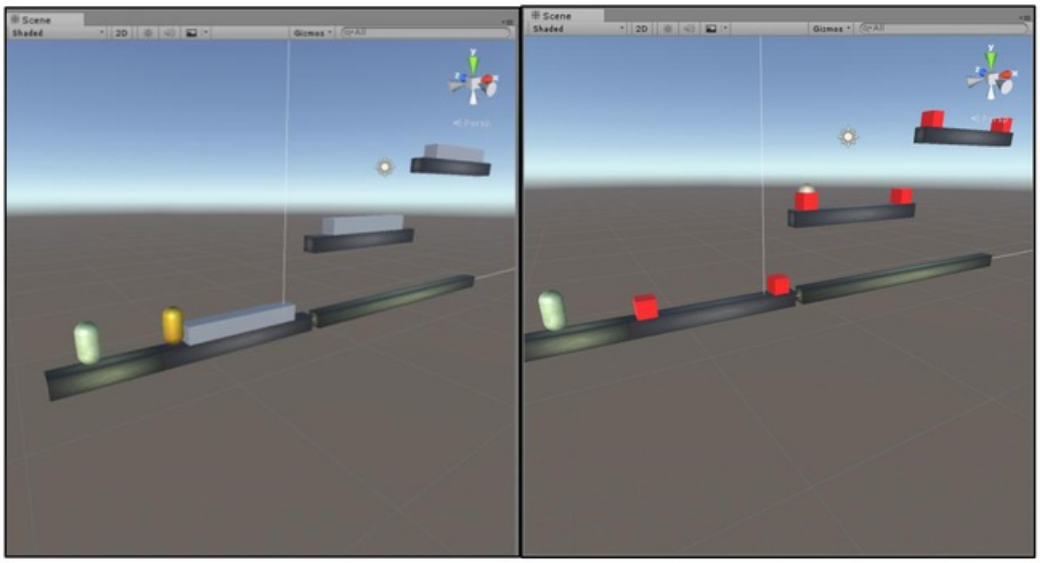


Fig.31 Showing debug functions detailing the edges and walkable spaces as seen by the FSM AI

Once that was done a simple state system was set up using an enum for the FSM, which handled the transitions between states as in the pseudocode. In order to allow the FSM to use its senses in all states, checks also needed to be made to see if the enemy could see and hear the player in each state. For example, if in the seek state, once the entity has navigated to the last heard location, it still needs to continually check if it can still hear the player, or whether the player has been seen in-between movements. Additionally, a counter needed to be included to transition back to less aggressive states if the player had not been seen or heard after a period of time.

Functionality then needed to be created to allow the FSM to look in a specified direction (left or right) to try to locate the player. This was done through Raycasting in Unity. This function returns true if the player is seen to the left and false if seen to the right. For sound, a Bounds intersection test between the enemy and the player's SphereCollider components was performed, giving an approximation of the player's location at the contact point of the two spheres.

```
    //Can you still hear the player?
    if (myCollider.bounds.Intersects(sensingPerson.noiseCollider.bounds))
    {
        //Reupdate the last heard location of the player by using bounds
        lastHeardLocation = sensingPerson.transform.position;
    }
    else
    {
        //Increment wait counter
        waitCounter++;

    }
    //The final break will act as the refresh for still being in seek

    break;
}
```

```
//Function that casts a ray allowing the AI to 'look' in different directions
public bool Look(Vector3 lookDirection)
{
    //Cast a ray in the specified direction and see if you can see the player
    RaycastHit connect;
    Ray sightRay = new Ray(transform.position, lookDirection);

    if (Physics.Raycast(sightRay, out connect, sightLength))
    {
        //Player has been seen
        if (connect.collider.tag == "Player")
        {
            //You can see the player, return true
            return true;
        }
    }
    else
    {
        //You cannot see the enemy in this direction
        return false;
    }

    return false;

}
```

Fig.32 Hear function (top) and look function (bottom) code snippets

To help with true navigation of the world space, it needed to be ensured that the enemy could determine whether its current target was to the left or the right of it before attempting to move and that the position it was walking to was one of the list of walkable map points, to ensure that it did not run off the edge of platforms when it reached them. From this the LeftOrRight and FindClosestNavpoint function were created.

```
//Function to determine whether the object is to the left or right of its target
public bool LeftOrRight(Waypoint target)
{

    if (transform.position.x > target.walkablePoint.x)
    {
        return true;
    }
    else if (transform.position.x < target.walkablePoint.x)
    {
        return false;
    }
    else
    {
        return false;
    }

}
```

Fig.33 Comparing x components of each vector to determine if the player was to the left or right

```
//Finds the closest point in the walkable list to the player and returns it
private Waypoint findClosestNavPoint(Vector3 targetPoint)
{
    float minDistanceToTarget = Mathf.Infinity;
    float distance = Mathf.Infinity;
    int index = -1;

    //Search for the closest point in your list to the point supplied by the function parameter
    for (int i = 0; i < pathFindArray.Count; i++)
    {
        distance = Vector3.Distance(pathFindArray[i].walkablePoint, targetPoint);

        if (distance < minDistanceToTarget)
        {
            minDistanceToTarget = distance;
            index = i;
        }
    }
    Debug.Log("Navigating to point " + pathFindArray[index].walkablePoint);
    return pathFindArray[index];
}
```

Fig.34 Implementation of approximating the closest pathfinding point to a target

The move functionality of the AI also turned out to be non-trivial, as the AI needed to move vertically as well as horizontally. Therefore a function needed to be created that would allow the AI to not only determine whether the player was to the left or right of them, but also above or below. This calculation only needed to be performed on an edge as that is where vertical movement would occur.

Once on the edge however, the AI needed to know the closest platform to move to so that it did not simply move to the edge closest to the player. Therefore a jump function was coded to calculate, out of all possibilities, the closest platform to the enemy, but that would also get them closer to their target (a waypoint or the player position approximated by a waypoint). This was done by performing a D&C technique where the relative elevations of the player and enemy were calculated to determine if the player was above or below the enemy. All edge points that were above/below the player were considered, calculating the distances between the enemy and each edge. The shortest one was picked and this could be repeated until the enemy landed on the same platform as the target. At each stage after a jump, it should be recalculated whether the target is to the left or right from an update in the target detection part of the script.

Issues arose, however, as to when the enemy navigated to another edge, as they had no way of determining which direction any given platform faced, which meant that they could choose to walk off the edge of the platform as they had no idea as to where platforms were as they were not numbered/indexed. To remedy this, after completing a jump, a cooldown was introduced. Before jumping again, the script would diagonally cast 2 rays to act as intersection tests (to the south east and south west of the player). If only 1 ray would hit an object, then the direction that the enemy should move would be in the X component direction of that ray (east or west).If both rays hit a platform, distance calculations would need to be made, with the shortest distance ray being the platform that is currently being stood on, as all other platforms would be at differing elevations. Following this calculation, the enemy was moved slightly into the centre of the platform so that they could re-evaluate the target position without being on an edge to prevent the constant execution of the jump behaviour.

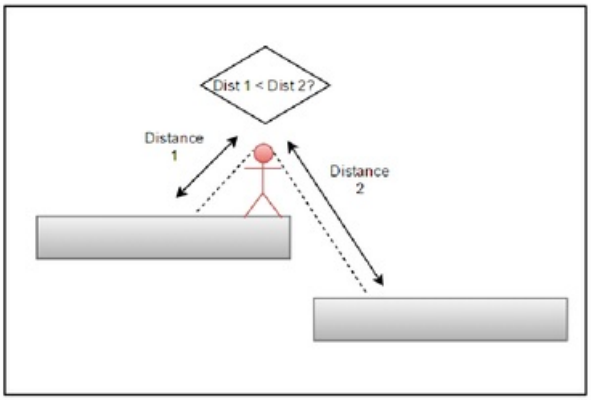


Fig.35 Visual demonstration of platform identification mechanic

```
//Check whether you're above or below the target
if (isAbove == true)
{
    //The target is above you
    //Calculate distance between you and all edge nodes in PathfindArray that have a higher y component than yours, and pick the closest one
    //Variable to store closestEdge
    float lowestDist = Mathf.Infinity;
    int totalCount = 0;
    int neededIndex = 0;
    foreach (Waypoint way in pathFindArray)
    {
        float tmpDist;
        //If the y component is bigger and its an edge tile,consider it
        if (way.walkablePoint.y > currentPosition.walkablePoint.y && way.onEdge == true)
        {
            //Calculate the distance between them
            tmpDist = Vector3.Magnitude(way.walkablePoint) - Vector3.Magnitude(currentPosition.walkablePoint);
            //If the new distance is smaller than the current lowest one
            if (Mathf.Abs(tmpDist) < lowestDist)
            {
                //Set lowest to the current distance
                lowestDist = tmpDist;
                //Store the current index of the smallest one
                neededIndex = totalCount;
            }
        }
        totalCount++;
    }
    //After this you have the closest edge to your player, so teleport to that edge
    transform.position = pathFindArray[neededIndex].walkablePoint;
    //Set hasJumped to true
    hasJumped = true;

    //You've just jumped so check which way you need to move based on the platform youre on
    PlatformMove();

}
```

Fig.36 Code snippet of elevation test from NPC/AI (D&C- points only above/below player considered- not both)

```
//Function to determine which way the AI should move on a platform following a jump
public void PlatformMove()
{
    //Cast 2 rays from your position diagonally left and right
    RaycastHit swHit, seHit;
    Ray swRay, seRay;
    swRay = new Ray(new Vector3(transform.position.x, transform.position.y + 1, transform.position.z), new Vector3(-1, -1, 0));
    seRay = new Ray(new Vector3(transform.position.x, transform.position.y + 1, transform.position.z), new Vector3(1, -1, 0));

    //Check if both of them collide with something
    if (Physics.Raycast(swRay, out swHit) && Physics.Raycast(seRay, out seHit))
    {
        //Get both the rays distances before the object they hit, the closest intersection of the ray is the direction to move
        if (swHit.distance < seHit.distance)
        {

            //The platform is on the left of you, so move that way off the edge and update your position
            transform.position = new Vector3(transform.position.x - 2, transform.position.y, transform.position.z);
            //Update your position
            currentPosition = findClosestNavPoint(transform.position);

        }
        else if (swHit.distance > seHit.distance)
        {

            //The platform is on the right of you, so move that way off the edge and update your position
            transform.position = new Vector3(transform.position.x + 2, transform.position.y, transform.position.z);
            //Update your position
            currentPosition = findClosestNavPoint(transform.position);
        }
    }
    else if (Physics.Raycast(swRay, out swHit))
    {

        //The south west ray cast hit something
        //Move to the west to get off the edge
        transform.position = new Vector3(transform.position.x   2, transform.position.y, transform.position.z);
        //Update your position
        currentPosition = findClosestNavPoint(transform.position);
    }
```

Fig.37 Code snippet of ray casting process as described visually above

All of the behaviour regarding movement, after the initial state transitions, is handled by the Move() function, which bases the type of movement to perform on the state its in and its current position in the world (on an edge or not).

From the basis of the class when only simple movement existed, a similar class was constructed with the same behaviours but operating using the Vector3.MoveTowards() movements. This allowed similar behaviour to be exhibited without concern for the environment being traversed. This paired perfectly with a ghost-like enemy that could hear and see the player, but with the added challenge to the player of ignoring objects when it traversed the environment.

Block/ Platform Movement:

In order to implement the parts of the environment that moved back and forth between two points, a script was needed to determine how fast and far any platform would move in the world. This would then allow flexible future designs for levels as the script could be attached to other objects to exhibit similar behaviour (for example slowing moving objects down or to act as open and closing doors). The pseudocode was defined as follows:

```
Begin
initialise startPosition
initialise currentPosition
initialise target

if(currentPosition = target){
    targetReached = true;
}
if(targetReached = true){
    target = startPosition;
    targetReached = false;
}
else{
    //Move towards your target
    Move(target);
    //Update position
    currentPosition = position;
}

End
```

Fig.38 Initial pseudocode for platform movement

Upon implementation, this class was simple to construct, as Unity has a 3-Dimensional Vector sorting the position of the object, meaning that its position and its target position could be easily navigated between.

```
public class platformMove : MonoBehaviour
{

    //Stores where the platform should bounce between
    public Vector3 startPos, endPos, currentPos;
    //Stores the rate at which the platform should move relative to time
    public float speed;
    //False is whether or not a target has been reached
    private bool isTargetReached = false;

    //Initialise the start position for the object
    void Start()
    {
        currentPos = startPos;
    }

    //Updates after each physics calculation (not each frame like update)
    void FixedUpdate()
    {

        //first check to see if we got to the target place (currentpos = endpos)
        if (currentPos == endPos)
        {
            //We have reached the target
            isTargetReached = true;

        }
        else if (currentPos == startPos)
        {
            //We need to go back to the end position
            isTargetReached = false;
        }

        //If we have reached the endPos Vector
        if (isTargetReached)
        {
            //Move towards the startPos Vector
            transform.position = Vector3.MoveTowards(currentPos, startPos, speed * Time.unscaledDeltaTime);
        }
        else
        {
            //Move towards the the end again
            transform.position = Vector3.MoveTowards(currentPos, endPos, speed * Time.unscaledDeltaTime);

        }

        //Update current position
        currentPos = transform.position;
    }
```

Fig.39 Implementation of platform move in C#

The class simply checks if the objects position is at its end position and sets the reached Boolean to true. If the target has been reached, the MoveTowards function is called to translate the platform to the location that it first started at. This repeats endlessly, allowing for moving platforms to be placed in the world that the player has to navigate. A separate public function was also introduced to set the speed of the given platform. This means that external classes like the player can access and modify the speed properties of the platforms at runtime.

Two differences were created in this class compared with others constructed. Unity's default Update() function was replaced by FixedUpdate() and any reference to the object's movement was multiplied by UnscaledDeltaTime as opposed to DeltaTime. This is due to the fact that framerate would dictate how fast an object moves and therefore may be different for different users. By using the above substitutes, any object with a Rigidbody component that this script is attached to will calculate its movement at fixed intervals independent of framerate. This allows for no slowdown in the physics calculations which could cause unexpected behaviour of the objects in the scene at run time if operating on a less capable machine. This also means that, if necessary, an adaptation to this script could be made such that an object could obey physics-based properties and still move (for example a box moves along and can fall down holes as it is effected by gravity.

Sequence Checking:

This class would be used in any part of the game where checking against a sequence would be necessary. In this particular case, the sequence checking logic was implemented alongside code that attracted specific items (cubes) towards the attached object's centre. Once the cube had been pulled into the object far enough, it was consumed and the colour of the incident cube checked against a sequence. Initially, the pseudocode implementation just contained the sequence checking logic as shown in the figure below:

```
Begin

Object currentItem;
Colour neededItemArray[];
int index;

currentItem = incidentObject;

if(currentItem.getColour() = neededItemArray[index]){
    //Increment index- correct combination
    index++;
    Remove currentItem from world
}else{
    //Incorrect sequence combination -reset it
    index = 0;
    Remove currentItem from world
}
End
```

Fig.40 Initial pseudocode of sequence checking script

When programming this into Unity, certain in-built features such as tags were used in order to determine what items were passing through the objects field of effect. Additionally, a force on any cube in a radial area that brought the object closer to the centre of the sequence checker was created. Unity's OnTriggerStay() function [52] was used for the collision detection of the object to be sequenced and the area of effect of the sequence checker.

```
public class gearPuzzleLogic : MonoBehaviour
{

    //Used to pull all objects towards the gear and check against a sequence
    //Stores the sequence of colours that will be checked and colours to parse
    private List<Color> sequenceList = new List<Color>();
    public List<string> colourConverts = new List<string>();
    public SphereCollider pullSphere;
    public float pullRadius;
    //Public variable to control how quickly the gear pulls objects
    public float pullForce = 5;

    //An optional gameobject event that could occur from the correct sequence
    public GameObject optionalEndTrigger;

    //Stores the current point in the sequence
    private int currentSeqIndex = 0;

    void Start()
    {
        //For each string in the list
        for (int i = 0; i < colourConverts.Count; i++)
        {
            //Try and parse to a colour type
            Color temp = Color.clear;
            ColorUtility.TryParseHtmlString(colourConverts[i], out temp);

            if (temp == Color.clear)
            {
                Debug.LogError("Colour not parsed correctly, revisit the string value of colour");
            }
            else
            {
                //Set the value in the sequence list to that colour
                sequenceList.Add(temp);
            }

        }

        //Set the pull area to the user defined value
        pullSphere.radius = pullRadius;
    }
```

Fig.41 Initial setup of colour parsing

Firstly, a list of colours were supplied that could be edited in the Unity editor to easily customise the length of the sequence to be checked and the colours in a sequence. Upon the object with this script being created in the scene, a list of strings is passed to the sequence checker which then uses the 'ColorUtility' class's 'TryParseHtmlString' function to convert the literal text into a colour [53]. If it fails, an error is thrown, if not the successfully converted colour is added to the sequence list to be checked. The extent to which the object attached to the script attracts cubes to be checked against the sequence is also initialised here and is set to a value that can be altered in Unity's inspector for the developers ease.

Following this, logic was implemented to allow the attached object to attract cubes of a specific tag, at a certain speed.

```
void Update()
{
    //If all colours have been successfully matched in sequence
    if (currentSeqIndex == sequenceList.Count)
    {
        //Set the optional event to active
        if (optionalEndTrigger != null)
        {
            optionalEndTrigger.SetActive(true);
        }

        //Reset the sequence index count
        currentSeqIndex = 0;
    }
}
void OnTriggerStay(Collider sequenceCube)
{
    //For all objects inside the spherecollider with the tag 'seqCube'
    if (sequenceCube.tag == "seqCube")
    {

        //Pull the cube towards your transform
        Vector3 directionToMove = transform.position - sequenceCube.transform.position;
        sequenceCube.transform.Translate(directionToMove * pullForce * Time.deltaTime);

        //Check the colour of the cube coming in against the sequeunce list, if its correct, accept and advance the sequence index
        //But if it violates the sequence, reset it
        if (sequenceList[currentSeqIndex] == sequenceCube.gameObject.GetComponent<Renderer>().material.color &&
            gameObject.GetComponent<BoxCollider>().bounds.Intersects(sequenceCube.GetComponent<Collider>().bounds))
        {
            //This is the correct colour in the sequence, so destroy the given cube and advance the index
            GameObject.Destroy(sequenceCube.gameObject);
            currentSeqIndex++;

        }
        else if (sequenceList[currentSeqIndex] != sequenceCube.gameObject.GetComponent<Renderer>().material.color &&
            gameObject.GetComponent<BoxCollider>().bounds.Intersects(sequenceCube.GetComponent<Collider>().bounds))
        {
            //Incorrect sequence has been entered, so reset the sequence back to the start
            GameObject.Destroy(sequenceCube.gameObject);
            currentSeqIndex = 0;
        }

    }

}
```

Fig.42 Implementation of gravitational attraction towards the centre of the sequencer

At each frame update, the current index was checked to see if it was equal to the number of elements in the sequence list. If this was the case, action could be taken in the form of setting another object (defined in Unity's inspector) to active. This was used to enable the end portal to a level in the final implementation. The OnTriggerStay function executes its code as long as another object is in the current objects Collider space. If the current cube inside the pull radius of the SphereCollider attached to the object had a tag of 'seqCube' then the cube was moved towards the centre of the object using Vector3 movement. A direction vector was created from the current position of the sequencer to the cube to be pulled, subsequently attracting the cube towards the sequencer center using the .Translate method.

Once the cube had entered the sequencers internal collider, which is in the same shape of a cube, the cube's colour was tested against the current colour being looked for. If the colour was correct, the currentseqIndex was updated, so the next colour could be looked for, and if the colour was incorrect, the sequence would reset. In both cases, the game object associated with the cube is destroyed to make way for new entries to the sequence. A setter function was produced so that the developer can change both the force applied to the object when it enters the sphere of attraction, and also the radius of the sphere.

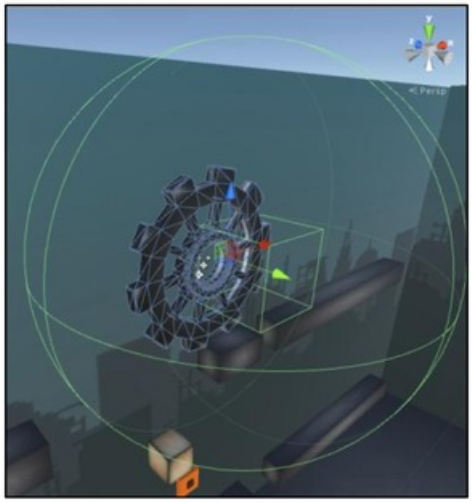


Fig.43 Display of both attraction sphere and destruction collider of the sequencer

Related Class: Cube/Primitive Spawning

In order for the above sequencer to have an object to check against a sequence, another class was produced that created primitive cube shapes at a given location of a specific colour. This was then used in conjunction with the above class to allow for a specific event to occur only if the correct sequence was entered.

```
// Update is called once per frame
void Update () {
    //If the cube has strayed too far from its start position, destory it
    if(sequenceCubePrefab != null)
    {

        Vector3 dist = transform.position - sequenceCubePrefab.transform.position;
        Debug.Log(dist);
        //Find the distance between the objects transform and its maxDistance apart
        if (Mathf.Abs(dist.x) > maxDistApart || Mathf.Abs(dist.y) > maxDistApart || Mathf.Abs(dist.z) > maxDistApart)
        {
            Destroy(sequenceCubePrefab.gameObject);
            isSpawned = false;
        }
    }
}

void OnTriggerStay(Collider other)
{
    if(Input.GetKeyDown(KeyCode.Z) && other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
    {
        if(isSpawned == false)
        {
            //Spawn an instance of your own cube colour
            sequenceCubePrefab = GameObject.CreatePrimitive(PrimitiveType.Cube);
            sequenceCubePrefab.transform.position = spawnPos;
            sequenceCubePrefab.transform.localScale = new Vector3(1, 1, 1);
            sequenceCubePrefab.transform.name = currentName;
            sequenceCubePrefab.tag = "seqCube";
            sequenceCubePrefab.AddComponent<Rigidbody>().useGravity = false;


            //Try to set the colour via string with the user specified one
            Color temp = Color.clear;
            ColorUtility.TryParseHtmlString(setColour, out temp);
            //Set the cube to that colour
            if(temp == Color.clear)
            {
                Debug.LogError("Could not parse colour correctly");
            }else
            {
                sequenceCubePrefab.GetComponent<Renderer>().material.color = temp;
            }

        }

    }
}
```

Fig.44 Implementation of primitive cube spawning

When the player is inside the collision zone that this script is attached to, if they press the z key, the script will attempt to spawn a prefab of a 1x1x1 cube. Also in the update method is a control on whether or not a cube should exist after a certain distance away from its spawn location. If the distance in any direction exceeds a certain value the gamer object is destroyed. Unity's GameObject.CreatePrimitive is used to create the 1x1x1 cube, and then its properties such as colour and world interaction are changed as it spawns into the world.

Dialog and UI display:

From the initial design, there needed to be a way to display information to the user without interfering with the world on the screen for too long. Therefore a script needed to be implemented to produce this feature. Additionally, how the main screen appears where the user selects their levels needs to be considered:

```
Begin

string displayText;
int dialogWidth;
bool choices,displayed;

//Calculate Chars Per Line
int dialogLength = NumberOfLines();

//Subdivide displayText into X lines
DivideText();
//Display the text to the screen
DisplayText();

if(choices){
    if(textDisplayed){
        Input = userInput;
        if(Input is in choiceList){
            //Execute Choice Code
            ChoiceExec();
        }
    }
}

if(DialogNotNeeded){
    Remove from screen;
}

End
```

Fig.45 Initial pseudocode for dialog display

For the pop-up dialogs to be used in game, the following figure shows the implementation. Due to time constraints, the choice mechanic was not introduced but could be possible for future adaptations if features such as questlines were added to the game.

```
//On starting, work out the appropriate length for the text in the text field and set the width and height of the popup accordingly
void Start()
{
    if (manualReSize == false)
    {
        //Attempt to auto resize the box to the text
        //Assume average line length of 27 characters
        //So 150/27 = characters per unit rect width approx 5.55
        //15 is the min size for one line
        popupWidth = 150;
        //Work out the number of lines needed
        int numLines = Mathf.CeilToInt(popupText.Length / 27);
        if (numLines > 1)
        {
            popupHeight = 2f * (numLines * 16);
        }
        else
        {
            popupHeight = 20 * 2f;
        }

    }
}
void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        togglePopup = true;
    }
}


void OnTriggerExit(Collider other)
{
    if (other.tag == "Player")
    {
        togglePopup = false;
    }

}
```

Fig.46 Approximation of dialog box width and height according to letter size

If the player entered the trigger zone of the object (defined by its collider), the space that characters took up on one line was calculated, and then the number of lines for the text entered was approximated. The length of the created box was then influenced by the popUpHeight variable which multiplied the space one line took up by the number of lines needed to show the information. In order to ensure that the information was positioned correctly in the scene for the player, the OnGUI() method was used to create a GUIBox in the environment at a suitable position related to the script's attached object position [54].

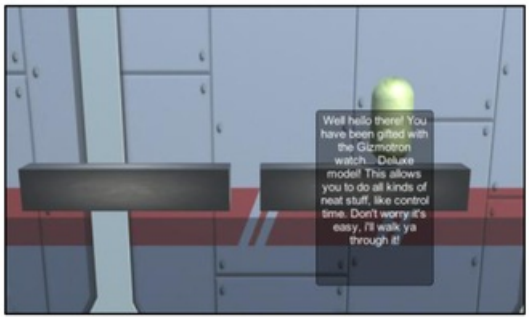


Fig.47 Display of dialog upon trigger zone entry

Displaying Level Select:

When implementing the level select screen, Unity's in-built UI system simplified the process significantly. A camera was simply placed in the desired position in the scene and the Unity UI buttons were added to the scene onto a canvas set to the Camera Screen space. This meant that a background could be included. Additionally, an AudioListener component was added to the canvas [55] so that music could also be included in a looping fashion for effect.

```
public class LoadLevel : MonoBehaviour
{

    //Used for returning to the start menu
    public string levelSkip;

    //Handles the level selection on the main menu, loading the level chosen by the user
    public void OnClick(string levelToLoad)
    {
        SceneManager.LoadScene(levelToLoad);
    }

    //If button clicked with no arguments, quit the application
    public void OnClick()
    {
        Application.Quit();
    }

    //Function to allow scene transition in game, when a level is finished
    void OnTriggerEnter(Collider other)
    {

        //If the user wishes to attach a level skip to an object set to not null
        if (levelSkip != null || levelSkip == "")
        {
            if (other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
            {
                //Load the scene relevant to the skip
                SceneManager.LoadScene(levelSkip);
            }

        }
    }
}
```

Fig.48 Level select code and level transition in game (at the end of a level)

Implementation of the level loading script was again very simple using Unity's in built package. Once the buttons for the UI were set up correctly, their associated OnClick() function needed to be mapped to the level to be loaded, which was done via a string parameter. The Scene Manger was then used to determine what scene was to be loaded [56].

Fig.49 Final level select screen

Related Class: Place Name Display:

Similar logic was used to display and fade place names to the user (for example if they enter a city). For this, a canvas was created with controls grouped on it using a CanvasGroup. Unity's UI buttons were used to align the name to the screen using a Screen Space – Overlay on the canvas so it appeard on top of everything. Due to time constraints, the button acting as a place name proved an effective way to create a desired fade effect without the need for creating animations for objects. In order to fade the button, the alpha component of the CanvasGroup was accessed and, using Unity's Coroutines [57], the alpha was gradually faded in when a trigger was entered and out when the trigger was left [58]:

```
void OnTriggerExit(Collider other)
{
    if(other.tag == "Player" && other.GetComponent<CapsuleCollider>().GetType() == typeof(CapsuleCollider))
    {
        toggleGUI = false;
        fadeGUI();
    }

}

void fadeGUI()
{
    //If the toggleGUI is true, fade the ui into the screen
    if (toggleGUI)
    {
        StartCoroutine(BeginFade());
    }else
    {
        //Fade the UI out, we no longer need it
        StartCoroutine(EndFade());
    }

}


//Slowly changes the alpha of the canvasGroup component from 1 to 0
IEnumerator EndFade()
{

    while(canvas.alpha > 0)
    {
        canvas.alpha -= Time.deltaTime;
        yield return null;
    }
}

//Slowly changes the alpha of the canvasGroup component from 0 to 1
IEnumerator BeginFade()
{
    while(canvas.alpha < 1)
    {
        canvas.alpha += Time.deltaTime * 2;
        yield return null;
    }
}

void SetPlaceName()
{
    placeNameText.text = nameText;
}
}
```

Fig.50 Place name fade text code snippet

Helper Classes: Spawning (player and objects), Box translation and teleportation:
Using similar logic to the above classes, triggers and translations were used to implement various features such as player spawn points (the trigger zone sets the current spawn of the Player by using a reference to the player control script); teleporting the player back to a location; activating event specific objects at the right time and moving boxes in a specified order to produce a puzzle that inverted specific platforms depending on a button press.

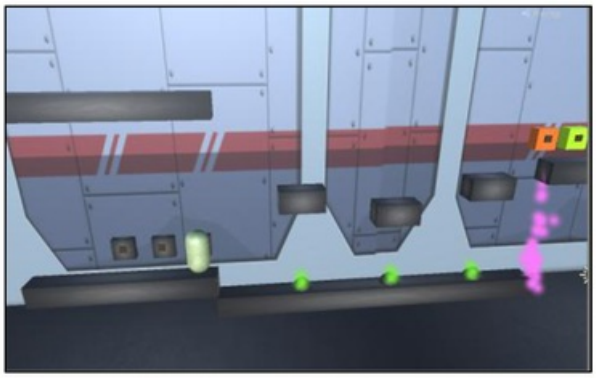


Fig.51 display of platform inversion puzzle to reach an elevated part of the level

Optimisation:

At times during the implementation and alpha testing, latency issues were experienced whilst playing which meant that changes needed to be made in order to more efficiently create the final product. In the Unity editor, some latency issues were fixed by disabling shadows cast by objects that were unhelpful or unnecessary, which helped the performance of the running game. During the first implementation of the AI script, when approaching an edge, the script would calculate the distances between its current object location and all other walkable points on the map. This caused the movement to lag the game significantly. This was remedied by reducing the amount of calculations needed for a jump by only checking edges and only those edges in the direction of travel being considered.

As mentioned prior, the primitive cubes spawned by a button press could cause the game some latency issues if too many of them are added to the scene. The destruction of these cubes if they stray too far from their point of origin prevents this from having too much of an impact on the gameplay.

Ease of implementation to the average user:

After the implementation phase of the project, a lot more information has been gathered on how easy Unity was to use when creating the desired features for the product. In terms of the level design implementation, the 'drag and drop' nature of the primitive shapes allowed for some rapid level development, as well as visual clarity on how the objects interacted and were positioned. It was easy to set up any desired lighting for the scene, as well as attach any imported asset such as art, to any object in the scene. Importing of models is also easy and quick, meaning the user can have more detailed objects, potentially even with animations, in the scene. Unity's terrain feature was also experimented with, along with environment prefabs, showing that a detailed terrain including mountains, water and trees for a 3D environment is very quick and simple to create.

Unity also allows intuitive linking of objects through its inspector. If a property is declared as public, the user can drag and drop another object in the current scene to it with the same property, making the implementation of dependencies on other scripts more straight forward. Values in this inspector can also be easily changed by the developer and can also be changed in runtime without having to recompile. This allows the developer to see changes quickly which allows for both reduced time to find errors (by testing many scenarios in one run) or to observe the effect a change of a variable has on the game environment.

In terms of scripting, Unity also provides some prefab scripts and controllers such as the Ethan model 3rd person controller script that provides a script with a fully modelled character that has a camera can move, jump and crouch. Despite this not being used for this particular project, aspiring developers are able to import it into the scene and observe how it operates, and also are able to view its code, meaning they could also learn how the complex behaviour was implemented. The Unity from zero to proficiency book (as used in the research phase) also detailed both car and plane prefabs, meaning that a user could easily create a variety of different games. Unity also has a lot of in-built classes that help simplify the coding task such as OnControllerColliderHit, CharacterController and ColourUtility. This could allow the user an easy way of performing tasks such as physics-based collisions without needing to implement or have an in depth understanding of properties like velocity and laws of motion describing an objects movement.

Applying the scripting complexity to this project however revealed that, although some scripts were trivial to implement and could have been produced by someone with little experience in programming, it was not the case with others. The more complicated scripts such as the AI controller required knowledge of other areas of programming/computer science knowledge such as finite state machines. In general, the more complicated the feature implemented, the more programming knowledge was needed to be drawn upon in terms of both computational efficiency and class/programming structure, therefore it seems as though a developer should have at least a basis in programming, computer science, or logic for Unity to be viable for large-scale, complex projects. In general, Unity appears effective for producing simple worlds for the basic user, but, skills such as programming and complexity of decisions are vital to producing any product that has a great deal of depth. Without this knowledge, utilising Unity could become frustrating for the novice developer.

TESTING:

Testing was performed throughout the project with basic tests being able to be formed as soon as the design phase was completed. As the program developed had no input parameters to test aside from in game keyboard input. The main focus for the testing was to assess whether the behaviour of the game conformed to expected behaviours and whether or not the product as a whole satisfied the original objectives of this project. These were assessed in a black box manner, assessing the behaviour of the mechanic under test, in terms of visual and behavioural output.

Development/Alpha testing:

Whilst programming the solution, multiple techniques were used to test components as they were written to ensure they were correct. Where the result of the test did not meet the expected outcome, further debugging tools were used to identify the issue. Both Unity and Visual Studio's debugging features came in useful.

Visual Studio debugging:

As the environment used for scripting in C# was Visual Studio, features such as breakpoints were taken advantage of for halting the code at specific points to assess values of variables and to be able to step through the program to determine where a potential fault may arise. This was especially helpful in ensuring correct behaviour in the AI script as a lot of state transitions needed to be considered depending on a variety of factors.

Unity Editor Debugging:

Many features were used in Unity in order to assess whether the logic behind a written piece of code was correct as well as its relevant performance. Unity's Debug class allows for two functions which were particularly useful when testing during development. Debug.Log and its variants allowed for messages to be output into Unity's console window whilst the game was running in the environment[59]. This was useful for inspecting suspect areas of code where errors were occurring and logging values of particular variables at runtime. During the development of the artificial entity script and the player script, this feature was used to determine values of camera rotation when the player rotated the world. Also the artificial entity's current target and where directionally it was in relation to the entity's current position was logged in the console.

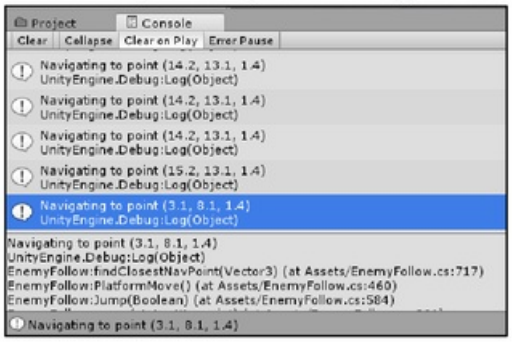
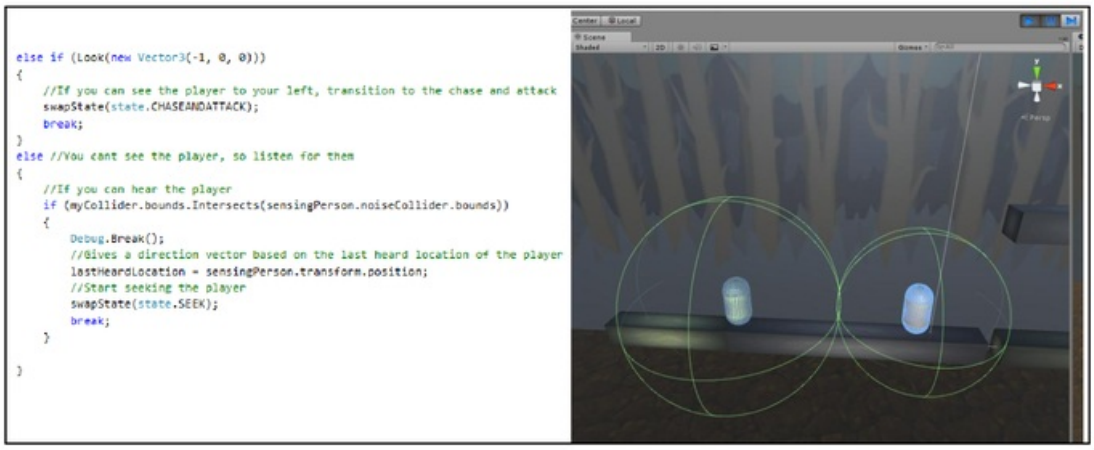


Fig.52 Output of Debug.Log to Unity's console

Another useful feature was Unity's Debug.Break function which pauses the simulation at the point in the script that the break function was called. This allowed for identification of exactly which point in execution a piece of the script was entered, which was useful in determining when events happened in the world. In particular, this feature was used to alter the behaviour of the noise spheres as the enemy was not correctly tracking the player when the bounds of their corresponding spheres overlapped. The debug.break feature was implemented at the conditional statement where the intersection was tested. The game was never paused, showing that the portion code was not accessed, this was then amended and the test performed again to pause at the correct point of intersection [60].



```
else if (Look(new Vector3(-1, 0, 0)))
{
    //If you can see the player to your left, transition to the chase and attack
    swapState(state.CHASEANDATTACK);
    break;
}
else //You cant see the player, so listen for them
{
    //If you can hear the player
    if (myCollider.bounds.Intersects(sensingPerson.noiseCollider.bounds))
    {
        Debug.Break();
        //Gives a direction vector based on the last heard location of the player
        lastHeardLocation = sensingPerson.transform.position;
        //Start seeking the player
        swapState(state.SEEK);
        break;
    }

}
```

Fig.53 Demonstrating Unity being paused at the exact moment of intersection of spheres

The use of the Unity inspector was also very useful during testing as another way to keep track of a particular objects property from position to developer-defined variables. This was used in order to verify that the place name text when entering a trigger zone did alter the alpha property of the associated canvas group component over time:

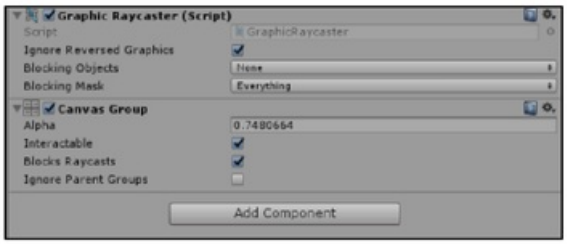


Fig.54 Alpha value changing periodically in Unity's inspector

Finally, in order to ensure that the code being implemented or the types of materials and animations in each scene would not cause too much latency, Unity's in-built profiler tool was used at points where latency may occur (sections with expensive computations) to ensure there was no unnecessary spikes in the proportion of the processing time spent of scripts or physics calculations:

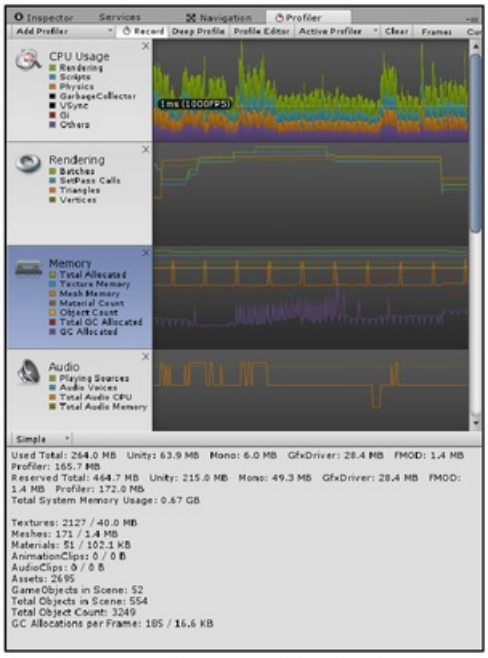


Fig.55 Output of Unity's profiler after level was run

Functionality Testing:

In order to effectively test the behaviours of the game, a table was constructed for relevant tests and their associated scene. Some tests are identical throughout scenes, so have no level association (such as the player).

Table 2: Test Case List For Functionality/ Behaviour Testing

Test ID	Description	Expected Outcome	Actual Outcome	Action Taken
TC1::LvlSelect	ButtonClick event	Take user to appropiate level	Level loaded correctly	N/A
TC2::LvlSelect	Quit Click Event	Stop Editor/Exit Game	No action from button in editor	The Application.Quit method does not work in editor, only after building does it function as expected, no action needed
TC3::LvlSelect	Music Play On Awake	Looping background track plays when level loads	Music plays and loops correctly	N/A
TC4::LvlSelect	Background Image Test	Image loads correctly and no other portion of the world space is visable. Buttons correctly appear and are of reasonable size	Image encompasses entire sceen, button placement sufficient	N/A
TC5::Player	Input: AD and Arrow Keys for Direction	A and leftArrow map to left character movement, D and right arrow to right movement	Movement observed correct, test success	N/A
TC6::Player	Input: Spacebar	Player executes a jump	Player jumps corectly	N/A
TC7::Player	Gravity Application	Player descends after jump due to gravity effect over time	Gravity correctly applied until surface contact, test success	N/A

TC8::Player	Ability Use Test Input: H,J,X keys	H slows slowable objects, J speeds object back up, X rotates world	Abilities behave as expected across scenes, test successful	N/A
TC9::Time	Platform Move	Platforms move endlessly back and forth upon starting scene	Platforms at correct locations and move back and forth correctly	N/A
TC10::Time	Platform Slow Down	On Input key H, movement speed of all slowable objects reduced	All moving platforms slow down	Speed changed so objects are slowed to a more significant rate to create an easy tutorial of the ability.
TC11::Time	Platform Speed Up	On Input key J, movement speeds of all slowable objects return to original speed	All moving platforms speed up effectively	N/A
TC12::Time	Portal/Exit Test	Upon completing the jump puzzle, the portal is activated and displayed to the user	Trigger zone works as expected, portal opened correctly	N/A
TC13::Time	Level Transition Test	When player enters portal, level select screen loaded	Level select screen loaded properly	N/A
TC14::Time	Level Respawn Test	If player falls out of map, reset players position	After a short duration, player is reset	N/A
TC15::Time	Textures Load Correctly	Upon loading of the scene, no objects have blank textures	All objects have textures loaded	Resolution/Tiling of some textures altered for better appearance
TC16::Time	Popup text correctly displayed	On entering trigger zone, popup displays to user (all text visable)	Popup successfully displayed	N/A
TC17::Gravity	Pop up text correctly displayed	All trigger zones display correct popup text	All pop up text displayed correctly	N/A
TC18::Gravity	Input: Z key platform move	Pressing Z key in trigger zone of button causes 2/3 platforms to change elevation	Platforms move as expected	N/A
TC19::Gravity	Input: Z key multiple clicks	Pressing the Z key repeatedly should just reverse the change by the first click of Z	Platforms correctly revert to their initial position if clicked more than once	N/A
TC20::Gravity	Reset Portal Test	If player gets stuck at first puzzle, teleporter should place them at the buttons	Player placed at start of level	Changed teleport target to be in front of the buttons
TC21::Gravity	Cube Primitive Spawn test	When Z pressed in front of coloured button, cube of that colour spawned correctly	Correct colour cube spawned on keyboard input	N/A
TC22::Gravity	Cube Destroy Test	If cube is too far from spawn location, remove it from world	Cube correctly destroyed	N/A

TC23::Gravity	Cube Interaction	Player pushes cube to move in the direction of movement	Cube moves in the way the player pushes it	N/A
TC24::Gravity	Sequencer Absorb Test	When cube pushed into area of pull effect, it is pulled into the center of the sequencer and destroyed	Cube correctly goes towards center of object and gets destroyed on contact(Multiple cubes can distrupt this process)	Disruption process encourages the player to not spawn too many cubes so no action necessary.
TC25::Gravity	Increment Sequencer Test(observed in editor)	Value of sequence counter increases when correct item entered	Value correctly increments up to a maximum value	N/A
TC26::Gravity	Correct Sequence Test	Portal displays after correct combination entered	Portal displayed after correct sequence entry	N/A
TC27::Gravity	Incorrect Sequence Test	No behaviour change	As expected	N/A
TC28::Gravity	Player fall test: Upright	Player resets at spawn	Player respawns correctly	N/A
TC29::Gravity	Player fall test: Inverted	Player resets at inverted spawn location	Player respawns on ceiling tile after falling off inverted world	N/A
TC30::Gravity	Correct Colour Parse(observed in editor)	String values effectively map to Colours	Inputted string values correspond to the correct colours	
TC31::Gravity	Input X: World rotation test	World is rotated 180 degrees when X pressed with smooth camera rotation	World rotates, but not all objects	Last added objects to the scene needed to be included in the parent gravity objects
TC32::Gravity	Mutiple world rotation tests	No errors arise in multiple inversions of gravity	Slight change in camera perspective after each inversion	ReStructure the camera delay on the rotation script for the camera to account for this effect
TC33::Gravity	End portal activate test	Particle effects shown when portal active	Effects display Correctly	N/A
TC34::Gravity	Portal Level Screen Test	When colliding with the portal, the player is returned to level select	Level Select Menu Loaded	N/A
TC35::Gravity	Particle effect test	All particle effects display when intended	Button related particle effects display when button pressed, all others function as expected	N/A
TC36::Gravity	Correctly loaded textures	No object in the scene is a primitive colour	No object untextured	N/A
TC37::Gravity	User control on Gravity inversion	Left and right buttons still move player left and right (right does not become left and vice versa) upon inversion	Controls remain the same when inverted and on normal ground	N/A

TC38::Stealth	Graphic Load Test	All graphics and textures load correctly	All objects have textures on level load	N/A
TC39::Stealth	Player Fall Test: Upright	Respawn player at latest spawnpoint	If player dies at the start, respawned at start. If checkpoint is reached, player respawns at checkpoint	N/A
TC40::Stealth	AI move test	AI correctly moves between waypoints, changing direction after time	AI correctly navigates to its waypoint and updates them over time	N/A
TC41::Stealth	AI hear test	When player makes enough noise, AI navigates to point of sphere intersection	AI successfully navigates towards point of intersection	N/A
TC42::Stealth	AI state reduction	Upon not hearing or seeing the player for a period of time, the alert level of the AI will decrease	AI transitions from Chase->Seek->patrol states over time if player not seen	N/A
TC43::Stealth	AI see test	Raycast effectively allows AI to see and chase player	AI can detect player both to the left and the right of it.	N/A
TC44::Stealth	AI elevation test	AI correctly moves up or down depending on target position	Correct elevation calculations made at each jump point	N/A
TC45::Stealth	AI speed increase test	When AI sees player, the speed of movement increases	Speed increases	Factor of speed increase altered to make AI move faster than before
TC46::Stealth	AI contact player test	Player is respawned immediately on contact with enemy	Player respawns correctly at their current respawn point	N/A
TC47::Stealth	Non-Walkable Area test(observed in editor)	If player is on non walkable ground to AI, closest possible space is found	AI effectively gets to a position as close as possible to the player	N/A
TC48::Stealth	End portal link test	Portal correctly links player back to level select screen	Level select screen loaded correctly	N/A
TC49::Stealth	Player sound detectable test	When player moves, the AI can pick up on location	The more noise the player makes, the earlier the AI begins to track them	N/A
TC50::Stealth	Popup test	Popup displayed clearly and in the correct position	Popup display as expected	N/A

TC51::Stealth	AI correct path choice test	If target is below AI, they will pick the closest edge to jump to	Closest edge picked accept at extreme cases	No action taken due to time constraints, but can be solved with a conditional to treat when reaching an edge on the far side of a platform
TC52::Stealth	CheckPoint test	Respawn position of player changes once the checkpoint is passed	Checkpoint effectively alters respawn location	N/A
TC53::FinalLvl	Place Name Fade test	Place name fades in on trigger enter and fades out on trigger exit	Alpha of button effectively faded in and out.	N/A
TC54::FinalLvl	Graphic Load test	All models and textures load appropriately	Enemy in final level missing texture	Applied enemy texture to the Ghost AI entity.
TC55::FinalLvl	Moving platform test	Platforms move back and forth indefinitely from the start	Platforms exhibit correct movement	N/A
TC56::FinalLvl	Cube spawn test	Cube of relevant colour spawned when Z pressed	Correct cube spawned	N/A
TC57::FinalLvl	Cube destroy test-distance	If the cube is too far from spawn position, remove from level	Singular spawned cubes correctly removed if too far from spawn point	N/A
TC58::FinalLvl	Cube destroy test-platform contact	If cube contacts the moving platforms it is destroyed after a short while	Cube destroyed if in contact with the platform for long enough	N/A
TC59::FinalLvl	Gravity Inversion Test	Player can rotate the world by pressing X	World correctly rotates	N/A
TC60::FinalLvl	Inverted Button Press test	Button on the ceiling can be pressed properly, causing a particle effect in the level	Particle effetc demonstrated	N/A
TC61::FinalLvl	Player fall test: Upright	Respawn player at checkpoint	Player respawns on ground at current checkpoint	N/A
TC62::FinalLvl	Player fall test: Inverted	Respawn player at inverted checkpoint	Player respawns at the designated inverted spawn point	N/A
TC63::FinalLvl	Ghost AI: See,Hear tests	Floating AI can exhibit all functions of the stealth AI in the air	All functions of Ghost AI work as expected	N/A
TC64::FinalLvl	Ghost AI: Contact player test	Player respawned on immediate contact with ghost AI	Player instantly respawned at current respawn point	N/A
TC65::FinalLvl	Ghost AI: state reset test	Once the player dies, the AI returns to its original position and sets state to patrolling	AI correctly resets its state to patrolling but does not return to first spawn point	Code changed to include an initial spawn point which is returned to after each player death

TC66::FinalLvl	Respawn Player Test	Player respawns at correct position after death or jumping off the level	As expected	N/A
TC67::FinalLvl	Pull radius change test	Once the button on ceiling pressed, the radius of pull expands allowing cubes into sequencer	Pull radius increases, allowing the cubes to float up into the sequencer	N/A
TC68::FinalLvl	Player Noise: AI Ghost interaction	When the player produces a lot of movement, the AI effectively hears and tracks player	Noise mechanic works effectively	N/A
TC69::FinalLvl	Correct sequence test/portal activation	Correct entered sequence spawns portal on the ceiling	Once correct sequence entered, portal effects are observed	N/A
TC70::FinalLvl	Incorrect Sequence test	No change in behaviour of level	As expected	N/A
TC71::FinalLvl	Portal level link test	Walking through the portal returns player to main menu	Level select menu displayed as expected	N/A

Beta Testing:

Once the product had been finished, as discussed in project meetings, users were invited to play the game to determine if more errors could be uncovered. This would allow for ways of testing not previously conceived, as an intended user would be trialling the product. This stage of testing took the form of a Silent Observation. Each testing candidate was introduced to the game and shown how to select a level from the level select screen and then asked to play the game. The candidate was then observed to see what areas of the game they were struggling with. Following their completion of the game, they were asked for feedback on what they had found challenging, as well as their thoughts on factors such as playability, usability and interest.

In order to effectively test this, candidates were selected that had a varying amount of experience in playing games of this type before, to allow for the most extensive range of possibilities to be tested. Below is a table summarising the information gained from this testing phase.

Table 3: Beta Testing Results

Candidate Name	Experience Level	Silent Observations	Candidate Feedback	Action Taken
K.Taylor	Beginner	Found it difficult to control the characters momentum; unsure how to approach the sequencer puzzle. Struggled with movement and jumping	*"I didn't know how to move the character or jump"* *"Really enjoyable, but the later level is very tricky"*	Pop up included in the first level explaining momentum, jumping and directional movement for the player
C.Smith	Intermediate	Completed the puzzles in a logical way as expected. Struggled at the beginning of the gravity level; Shadowing by a game object caused the celing button in the final level to be difficult to see	*"It needs to be explained how to reset your position, in some areas I did not know that in order to reset the puzzle I needed to jump off the map"*	Disabled shadow casting on the problematic object to allow the button to be seen. Included a popup that detailed to the player how to reset from a puzzle if stuck

A.Becker	Intermediate	Completed the puzzles with relative ease	*"It doesn't tell you how to jump but other than that a really fun game"*	Addressed above
S.Manley	Experienced	Seemed to grasp quickly how the mechanics worked and how to use them to solve problems	*"a really interesting game, I can see how this could be used to make some interesting puzzles"* *"The enemy on the last level seems a bit difficult though, it got in the way a lot when I was trying to solve the problem"* *"I would defintely play this if more levels were created"*	Changed enemy in last level so that movement was slower and the change in path direction also happened less frequently.
V.Rao	Beginner	Struggled with jumping and did not notice some portals were available when the level was completed. Completed some puzzles in interesting ways	*"The game doesn't tell you how to move or jump"* *"The buttons on the gravity level were hard to see and I wasn't sure what I was meant to do with them"*	First issue was addressed in another test case; made particle effects on portals more easily visable and highlighted buttons through pop up dialog

From this stage of testing, a lot more feedback was received as to how the game operates and some key changes were made in order for users to have a more enjoyable experience. After all stages of testing, the project initiation document can now be consulted to determine if the original objectives of this project have been met. This, alongside user feedback on enjoyment, forms the basis of acceptance testing for this product.

Consulting the PID in the appendix, all criteria have been met, producing a sufficiently challenging game with various mechanics implemented. Complex behaviours have been implemented in the form of the AI element and use of save points and progress tracking through level completion gives the end user some sense of achievement. From feedback from players, the game is challenging and enjoyable, reflecting its suitability for the target audience. The secondary goal of evaluating Unity in terms of ease of use has also been achieved.

Testing Limitations:
Some testing limitations exist in working on a project such as this. For example, time constraints on the project meant that only a finite amount of time can be dedicated to testing, therefore some issues could still be unexplored. Test suites could go a long way to help automate this process and save the developer time. Exhaustive testing is also unrealistic, although a product such as a game would benefit greatly from either an increased number of Beta testers or a scheme where the game is rolled out as unfinished versions which are then explored by players who report any found bugs. This way, users become integrated into the development process which could result in a more successful product upon final release.

DISCUSSION:

Upon concluding the testing section, how the project has progressed can be reviewed. Comparing the results to the Project Initiation Document (PID), the extent to which the initial objectives, outputs and features were achieved can be seen.

Objectives and Outputs Success:

One objective of the project was to determine the optimal approach for designing a game and ensure that it's output was best suited for the target audience. This objective was satisfied through the research stage in deciding the game engine and language to use, as well as using an industry used tool. From the testing, the appropriateness of the product for the target audience has been realised, as the more experienced players from the Beta testing enjoyed the game more, with the lesser experienced participants willing to learn the new skills. Because of this, the first objective has been successfully fulfilled.

Research into how algorithms were used by industry to achieve their game's resulting mechanics was also another key objective of the project, alongside utilising that knowledge to produce complex behaviours that enhanced the users enjoyment and encouraged them to play multiple times. This proved more difficult to achieve as some algorithms that were of public knowledge (such as the A* pathfinding algorithm) needed to be heavily adapted to conform to the platform like nature of the product under development. Additionally, it was found that how current developers devise their mechanics in a technical sense is not always known, meaning that most of the mechanics needed to be designed using speculation on how existing products may produce the results they do. Providing a sufficient level of enjoyment and challenge to the player has also been successfully achieved as the test results show participants enjoyed completing the puzzles. Some Beta test results suggested that the difficulty spike was too high in some places, which provided useful feedback on how to alter the gameplay experiences. Through the use of Unity, any change to the difficulty were very straight forward to alter to better fit the user's sense of fairness about any given level.

Another output of the project was to produce interesting level designs. The inclusion of the 3 mechanics aided in creating a vast range of possible level layouts, increasing the playability and level of interest of the game to the player. Deciding on a level based system instead of a persistent world was a good design choice as it allowed for a linear increase in level difficulty. This was reflected in the testing with a particular participant solving a problem utilising a different strategy than was envisioned for a particular level.

Consulting the project specification in the PID, it can be seen whether or not the initial features for the game have been produced. Interactive AI behaviour has been implemented in the form of an enemy with senses of sight and hearing, acting as a deterrent to the player. An effective tutorial section was successfully implemented after the testing stage provided useful amendments to the implemented tutorial levels. When playing the tutorial levels, most participants of the Beta tests did not require reminding of the abilities at their disposal to solve any given problem. Adapting difficulty can be reflected in the increase in difficulty of each level, with multiple mechanics being combined to present the largest challenge to a player. This was augmented with the ability for difficulty to be appropriately scaled based on feedback from the Beta testing phase.

Both the Alpha and Beta testing phases identified the fulfilment of the Saving, Progress tracking, and hints system, highlighting the importance of a thorough testing phase. Saving was achieved through checkpoint systems, progress tracking through the level based approach that allowed the user satisfaction of solving a particularly tricky puzzle. The pop-up dialog system provided the grounding for a good hint based system, with feedback from testers further extending its usefulness by pointing out areas that may need more guidance for the player.

Reasons for used test process:
The importance of user interaction and enjoyability in a product of this nature is paramount. Because of this, it was decided that, as well as development testing that ensured each feature worked as expected, Beta testing should also be included as users are at the center of focus for any software solution, especially one linked so closely as a source of entertainment.

This user centric approach spurred the implementation of certain features such as the AI with different senses, as each design and implementation decision made was to maximise user experience. In a further version of the product, additional content can be added to extend user enjoyability further, including a more consistent art style and fleshed out storyline with engaging dialog, providing the player a reason to navigate through the levels.

Ease of use for a new developer:
From design, implementation and testing, it has given an appreciation of how challenging it could be for an aspiring developer with little knowledge of programming to undertake a task such as this. Unity provides a range of good features for optimisation, both manual and automated by Unity, as well as some easy ways to develop levels. Through the use of prefabs, already robust coding solutions for features , such as 3rd person character movement, can be used to aid the developer in gaining fast results with little background knowledge. However, as the ideas wanting to be implemented become more complex, the reliance on pre made assets and scripts could become unrealistic, meaning that an understanding on how to program to some degree, or a willingness to learn is necessary. Learning through something such as the Unity editor however would greatly soften the learning curve needed to learn programming from the initial concepts.

SOCIAL, LEGAL, HEALTH, ETHICAL ISSUES:

The nature of this project as it stands at completion had no issue with Social, Legal or Ethical issues. The only potential legal concern is one due to copyright with regards to the assets used for materials and models for objects, however these were all sourced under Creative Commons License (CC) and therefore permission had been granted through the varying CC levels. If more time had been available, and a further in depth storyline had been added to the game, this could raise potential ethical issues depending on its content. If the game addressed adult themes in the game, this could have an impact on the player's behaviour which highlights the importance of having an age rating on a game. It could also be argued that health issues could arise from over playing of a game, such as inactivity and extended periods viewing monitors. This could be reduced by adding disclaimers and warnings to take regular breaks from using the product. Conversely however, it could be argued that a game such as this improves an individual's lateral thinking and problem-solving skills.

CONCLUSION/FUTURE IMPROVEMENTS:

The principle aim for this project was to assess how a puzzle game was produced using a game engine used in industry, assessing ease of use from learning it from first principles. The way in which games such as puzzle games are built was also part of the core aim. The produced product was a platform based puzzle game, incorporating the use of object slowing, world inversion and stealth to present problems to the user which they had to solve. These mechanics were applied in a variety of different scenarios (levels) where the orchestration of the level also played a part in the difficulty of the puzzle being solved.

A secondary aim to the project was to assess the relative ease of use of Unity to a new developer with little programming knowledge. Following the project's completion, an appreciation has been gained for the progressive difficulty in implementing features that require complex logic, which may be out of scope for newly beginning programmers. Unity uses a modular approach that is very simple to get on with and tutorials allow for a user to learn things very easily, especially if some programming is already known. Basic level designs can be produced in a short space of time, allowing for an easy way to visualise concepts.

Different design techniques were explored in terms of how developers structure their levels, which were then applied to the Unity program. Through the use of the game engine, alongside the visual studio scripting environment, a great deal has been learned in how best to approach a programming solution in terms of a game, and how to apply those approaches to the industry software. Being previously versed in C#, the coding environment was immediately familiar, allowing for a head start on understanding the scripting element of Unity and how game developers may go about producing some of the features that leading games possess today.

Reading multiple sources for the research stage and Unity documentation on a previously unknown platform improved both academic information retrieval and development environment tools.

The overall success of the project was largely determined by the testing phase, which revealed that required features worked as intended and appropriately reflected a good user experience. Within the project, designing the worlds and how the levels would work with the chosen mechanics was an enjoyable experience, allowing for a creative outlet in a technical project. The implementation of the mechanics was also enjoyable in terms of overcoming an immense technical challenge, however it was difficult to gauge how long a given feature would take to implement, meaning that some areas of the project such as producing the AI, overran significantly.

Given the time constraints of the project, time has been managed effectively and the project was finished on schedule. Structuring the work in terms of milestones helped with timely completion of deliverables. Where parts of the project overran, the introduction of weekly or fortnightly mini deadlines ensured that the overrunning was compensated for. Limitation of the projects scope was vital in its success as an almost endless set of features could have been included, however, focusing on 3 main concepts allowed for the project to stay on track. It was found that, during the initial project phases, the PID needed to be altered in order to align with the research that was carried out on what makes a good puzzle game.

There are a number of future improvements that this project could benefit from in terms of Art and Design, Storyline and technical content. A stylised approach to the game could be produced, allowing for a unique art style that would help players identify a game of this nature from other similar products. A compelling back story will also attract a wider audience and to put across an image of the world you as a developer have created. Features such as cut scenes and background information scattered around the environment can also be used to put across the narrative for the game that involves the player piecing it together. In terms of technical content, more levels can be introduced to increase the variety of puzzles to be solved across a range of different environments. Scoring could also be updated into a more literal sense, with either the time taken to complete a level acting as a metric or how many attempts were required. This promotes replayability and would be simple to implement moving forward.

Interaction could further be developed by the introduction of a second player, either locally or connected via the internet. This would further increase the range of puzzles as they could be tackled together meaning more cooperative solutions being needed for any given problem. Providing the player with more flexible mechanics could also be an improvement, for example, the time or area time could be slowed down in, 90 degree intervals of gravity rotation enabling walking on walls, and introduction of an inventory system to give items that aid with stealth and distraction. More mechanics could also be implemented, such as different AI behaviour for day and night, which would also affect the stealth capabilities of the player. All of the above adaptations could lead to more detailed, engaging and enjoyable puzzles in the long run.

REFERENCES:

[1] https://www.pagat.com/eights/mao.html [Accessed 30/04/17]
[2] https://www.rockpapershotgun.com/2015/01/22/how-to-make-a-puzzle-game/ [Accessed 30/04/17]
[3] http://devmag.org.za/2011/06/04/how-are-puzzle-games-designed-conclusion/ [Accessed 30/04/17]
[4] B.Bostan, U.Kaplancali *'Explorations In Player Motivations: Game Mechanics'* [Available from: http://silentblade.com/presentations/Bostan_Kaplancali_2009.pdf] [Accessed 30/04/17]
[5] http://tetris.com/about-tetris/ [Accessed 30/04/17]
[6] http://candycrushsaga.com/how-to-play [Accessed 30/04/17]
[7] http://uk.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 [Accessed 30/04/17]
[8] http://www.xblafans.com/super-meat-boy-bandage-guide-world-one-3183.html [Accessed 30/04/17]
[9] http://www.gameskinny.com/16wli/5-games-that-use-death-in-unique-and-inventive-ways [Accessed 30/04/17]
[10] http://playdead.com/games/limbo/ [Accessed 30/04/17]
[11] http://www.fezgame.com/ [Accessed 30/04/17]
[12] http://www.unravelgame.com/ [Accessed 30/04/17]
[13] http://uk.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 [Accessed 30/04/17]
[14] http://uk.ign.com/articles/1996/09/26/super-mario-64 [Accessed 30/04/17]
[15] https://developer.valvesoftware.com/wiki/Game_Mechanics_(Portal_2) [Accessed 30/04/17]
[16] http://www.snake-pass.com/ [Accessed 30/04/17]
[17] C.Crawford '*The art of computer game design'* pp. 37-39 [Available from: https://pdfs.semanticscholar.org/f3ed/05a823c7079ec46eba0fed5e942cb8829f73.pdf] [Accessed 30/04/17]
[18] https://www.theguardian.com/technology/2015/jul/23/16-trends-that-will-change-the-games-industry [Accessed 30/04/17]
[19] http://mod-minecraft.net/nature-overhaul-mod/ [Accessed 30/04/17]
[20] S.Kim *'What is a puzzle'* [Available from: http://cs.wellesley.edu/~cs215/Lectures/L17-IntroGamesJigsawPuzzle/ScottKim-What_is_a_Puzzle.pdf] [Accessed 30/04/17]
[21] http://www.gamasutra.com/view/feature/130268/a_detailed_crossexamination_of_.php?page=29 [Accessed 30/04/17]
[22] T.Friedman *'Making Sense Of Software: Computer Games and Interactive Textuality'* [Available From: http://web.stanford.edu/class/history34q/readings/Cyberspace/Friedman_Making%20Sense.html] [Accessed 30/04/17]
[23] http://www.directxtutorial.com/Lesson.aspx?lessonid=11-4-2 [Accessed 30/04/17]
[24] https://www.unrealengine.com/ [Accessed 30/04/17]
[25] D.Nussbaum *'Game Rendering'* [Available From: http://people.scs.carleton.ca/~nussbaum/courses/COMP3501/notes/Game_rendering_directx_review_part_1_sept_2012_%5bv1%5d.pdf] [Accessed 30/04/17]
[26] http://ieeexplore.ieee.org/abstract/document/1438251/?reload=true [Accessed 30/04/17]
[27] A.Herwig, P.Paar *'Game Engines: Tools For Visualization and Planning?'* [Available From: https://www.researchgate.net/profile/Philip_Paar/publication/268212905_Game_Engines_Tools_for_Landscape_Visualization_and_Planning/links/5464ab2b0cf2cb7e9dab8bc5.pdf][Accessed 30/04/17]
[28] J.Craighead, J.Burke, R.Murphy *'Using the Unity Game Engine to Develop SARGE: A Case Study'* [Available From: https://www.researchgate.net/profile/Jeffrey_Craighead/publication/265284198_Using_the_Unity_Game_Engine_to_Develop_SARGE_A_Case_Study/links/55d33fdf08aec1b0429f32f4.pdf] [Accessed 30/04/17]
[29] R.De Chiara, V.Di Santo, U.Erra, V.Scarano *'Real Positioning in Virtual Environments Using Game Engines'* [Available From: http://www.isislab.it/papers/egita07.pdf][Accessed 30/04/17]
[30] E.Prakash, J.Wood *'Games Technology: Console Architectures, Game Engines and Invisible Interaction'* [Avilable From: https://dspace.lboro.ac.uk/dspace-jspui/bitstream/2134/20295/4/cgat-keynote.pdf] [Accessed 30/04/17]
[31] D.Johnston *'3D Game Engines as a New Reality'* [Available From: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.6924&rep=rep1&type=pdf] [Accessed 30/04/17]
[32] http://www.develop-online.net/news/the-top-14-game-engines-the-list-in-full/0114330 [Accessed 30/04/17]
[33] https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose [Accessed 30/04/17]
[34] http://www.gamevortex.com/gamevortex/soft_rev.php/4177/braid-xbox-360.html [Accessed 30/04/17]
[35] https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose [Accessed 30/04/17]
[36] https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial [Accessed 30/04/17]

[37]P.Felicia *'Unity 5: From Zero To Proficiency (Foundations)'2015*
[38] https://docs.unity3d.com/Manual/2Dor3D.html [Accessed 30/04/17]
[39] https://www.esrb.org/ratings/ratings_guide.aspx [Accessed 30/04/17]
[40] http://gameprogrammingpatterns.com/state.html [Accessed 30/04/17]
[41] http://gameprogrammingpatterns.com/game-loop.html [Accessed 30/04/17]
[42] http://wiki.unity3d.com/index.php?title=Event_Execution_Order [Accessed 30/04/17]
[43] https://unity3d.com/learn/tutorials/topics/user-interface-ui [Accessed 30/04/17]
[44] https://docs.unity3d.com/ScriptReference/Time-deltaTime.html [Accessed 30/04/17]
[45] https://docs.unity3d.com/ScriptReference/Vector3.html [Accessed 30/04/17]
[46] https://docs.unity3d.com/ScriptReference/CharacterController.html [Accessed 30/04/17]
[47] https://docs.unity3d.com/ScriptReference/CharacterController.Move.html [Accessed 30/0417]
[48] http://lodev.org/cgtutor/raycasting.html [Accessed 30/04/17]
[49] https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Accessed 30/04/17]
[50] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnControllerColliderHit.html [Accessed 30/04/17]
[51] https://docs.unity3d.com/Manual/Tags.html [Accessed 30/04/17]
[52] https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html [Accessed 30/04/17]
[53] https://docs.unity3d.com/ScriptReference/ColorUtility.TryParseHtmlString.html [Accessed 30/04/17]
[54] http://answers.unity3d.com/questions/26676/can-you-make-a-guibox-follow-a-players-positionsol.html [Accessed 30/04/17]
[55] https://docs.unity3d.com/Manual/class-AudioListener.html [Accessed 30/04/17]
[56] https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html [Accessed 30/04/17]
[57] https://docs.unity3d.com/Manual/Coroutines.html [Accessed 30/04/17]
[58] http://stackoverflow.com/questions/39336423/fadein-fadeout-ui-button-in-unity-5 [Accessed 30/04/17]
[59] https://docs.unity3d.com/ScriptReference/Debug.Log.html [Accessed 30/04/17]
[60] https://docs.unity3d.com/ScriptReference/Debug.Break.html [Accessed 30/04/17]

Game Asset Sources Used In Unity Editor for Final Product:

1.) All Credit to Artist: http://www.newgrounds.com/art/view/bizmasterstudios/forest-background-for-parallax-scrolling
Used for level select background

2.) By Attribution: " "Night of Chaos" Kevin MacLeod (incompetech.com)
Licensed under Creative Commons: By Attribution 3.0
License: http://creativecommons.org/licenses/by/3.0/
Source: http://incompetech.com/music/royalty-free/music.html
Usage: Level Select Music

3.) No Credit Needed but referenced
Source: http://tf3dm.com/download-page.php?url=arv-craft-48189
Usage: Aesthetic for gravity manipulation devices

4.) Non Commercial Use, Free to Use
Source: http://tf3dm.com/3d-model/gear-20205.html
Usage: Used in game as gravitational object

5.) No credit needed
Source: https://opengameart.org/content/ruined-city-background
Usage: Backdrop for final level

6.) Under Creative Commons - CC-3.0
Source: https://opengameart.org/content/grungy-title-screen-background
Usage: Floor for Final Level

7.) No credit needed
Source: https://opengameart.org/content/background-for-games-2
Usage: Backdrop for final level

8.) Under Creative Commons - CC-3.0
Source: https://opengameart.org/content/night-forest-background
Usage: Backdrop for stealth level

9.) Under Creative Commons - CC-3.0
Source: https://opengameart.org/content/tileable-200x200-dirt-texture
Usage: Floor for stealth level
10.) Under Creative Commons CC 0
Source: https://opengameart.org/content/sci-fi-background
Usage: Backdrop for gravity and time levels
11.) Under Creative Commons CC-3.0
Source: https://opengameart.org/content/dirty-tiles
Usage: Player and enemy Textures

Creative Commons Licencing:

CC-3.0: http://creativecommons.org/licenses/by/3.0/

CC 0: https://creativecommons.org/share-your-work/public-domain/cc0/

Drawing Tool Utilised for Design:

https://www.draw.io/

BIBLIOGRAPHY:

Historical Analysis of platformer games:
http://www.gamasutra.com/view/feature/130268/a_detailed_crossexamination_of_.php?page=1

General Overview of how games have evolved:
http://booksite.elsevier.com/samplechapters/9780123749536/01~Front_Matter.pdf

Overview List/ Literature on Usability in games:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.8294&rep=rep1&type=pdf

Extra Reading on Game Engines:
http://ahnet2-dev.cch.kcl.ac.uk/files/gameenginedevelopments-1.pdf

Recommended Game Engines:
http://www.worldofleveldesign.com/categories/level_design_tutorials/recommended-game-engines.php

APPENDICES:

Appendix 1: Project Initiation Document:

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**23009086**
Student Name	**Nathan Brown**
Email	**n.j.brown@student.reading.ac.uk** **njbrown207@gmail.com**
Degree programme (BSc CS/BSc IT)	**BSc CS (Computer Science)**
Supervisor Name	**Hong Wei**
Supervisor Signature	
Date	

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	Own Project
1.2	**Project Title**
	Puzzle based game using an open source game engine
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	To design a game containing puzzles using an industry standard development environment to explore how developers go about creating new games.

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Nathan Brown, BSc CS, n.j.brown@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
	Hong Wei, h.wei@reading.ac.uk
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
1.8	**Company Address**
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**

SECTION 2 – Project Description

2.1	**Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**

The background research for the project before beginning work on the implementation will start with research into what makes a good puzzle game and a comparison into how each of the game engines will benefit or hinder a prospective game developer (pros and cons). From this, one game engine will be selected and then the use of online documentation and any other sources possible will be used to increase proficiency with the tool. For example, tutorials based on video sharing sites such as YouTube or from the tool's website (question and answer forums and in-built tutorials).

When deciding on what sort of mechanics to implement in the game, research will be done into what is already out on the market, including successful puzzle ideas and other such successful games. This will be done through the use of forums where discussion can be had for example Reddit; video reviews from YouTube and written reviews from websites such as IGN. Researching various puzzle games will give ideas on how to bring in a positive experience element to the game being created for the users. Throughout the project, continual research will need to be done if problems arise that cannot be solved with individual knowledge alone, this could be through contact with a supervisor or finding internet or book resources on the subject.

Research will also be done into the best way to test the project; this will inevitably involve end user testing and feedback when the project is nearer completion. However, during the development time (alpha testing phase) each segment of the game will be tested for potential bugs in the gameplay itself, as well as any errors in the scripting portion of the project. This feedback from users and troubleshooting the code should allow for a more robust end product. More market research will also be completed as to opinions on the aesthetic feel to the game i.e preferences of textures, models, dialogue options and potentially music. The feedback from this research will enable a more immersive final product to be produced.

Where possible, the best solution to any problem being faced will be selected in order to improve the efficiency of the final product. This could be, for example, if the game engine does not have a built in feature to support certain player or enemy movement. Research will be completed into how best to implement the movement through scripting so that the feature runs as smoothly as possible when it has been created.

References:
Research into how to make a puzzle game:
https://www.rockpapershotgun.com/2015/01/22/how-to-make-a-puzzle-game/

Help sections for choices of game engine:
http://docs.unrealengine.com/latest/INT/Videos/
https://www.cryengine.com/tutorials
https://unity3d.com/learn/tutorials

Sites used to research existing puzzle games:
https://www.youtube.com/user/IGNentertainment

2.2	**Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.

	Objectives: 1. Perform research into different game engines, programming languages and previous PC or Console games to decide the optimal choice for the product being developed. 2. Research into the logic behind some of the mechanics of the products researched to help design relevant algorithms and mathematical processes for the product. 3. Create a sufficiently challenging game that provides the target audience with an interesting product with various mechanics governing the puzzles. 4. Develop the puzzle game using the most appropriate game engine and language. This will include designs and implementation of the processing for the game, alongside any interactions with external entities such as a database to store data. The design for the game GUI will also be developed and implemented. 5. Carry out alpha testing as the game is being developed, and Beta testing with end users as the product nears completion. Overall carry out relevant functional and non-functional requirement testing as well as white and black box testing for functions the game is to perform. 6. Continue to keep a project log as the project develops to note ideas and concepts that were thought of, alongside keeping track of all decisions made. 7. Write a project report detailing the research, design, implementation, and testing process of the product and its fitness for purpose. 8. Reflect on good and bad points of the project for future undertakings. Outputs: 1. Produce the best suited product for the target audience, and the best way for the product to perform 2. Produce a product with complex behaviours and scripts that enhance the playability of the game as well as the users' enjoyment. 3. Produce interesting overall level designs to create many ways to challenge the user. 4. Produce a report detailing the comparisons between the different possible choices of game engine and give reasons for why the one being used in the project has been chosen. As well as this, detail any external or internal storage mechanisms being used. 5. Produce a detailed testing plan and report detailing the type of testing performed on the product and what aspects of the product were tested. 6. Produce a written copy of all important decisions made within the project with time and dates and citations of relevant sources. 7. Combine all aspects of the software development lifecycle into one coherent document (analysis, design, implementation, testing). 8. Produce a reflective piece detailing all opinions on the project after it has been carried out for potential improvements to the end product.
2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.

	Initial project features that would need to be included are as follows: 1. Artificial Intelligence (AI) based players that will populate the levels and have interactive behaviour with the user. 2. A tutorial based section of the game or manual will instruct the player on how to interact with characters and how to play the game. This means that the player needs little to no prior knowledge in order to utilise the game. 3. A range of puzzles involving various applications of logic to complete. Games of a similar type will be researched to gain inspiration on how these puzzles will be structured. An example of which could be portals puzzle logic with weighted platforms. 4. A minimum of 3 different mechanics being used to complete puzzles 5. Adapting difficulty levels of puzzles. 6. Allow the user some control over how to solve a puzzle in some parts of the game 7. Save points through autosave or some other means. 8. Progress tracking system. 9. Hints or skips for harder problems.
2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval?**
	Ethical issues with a game of this type would be related to the story based element. For example, whether there were any adult themes or ethical issues within the dialog or text of the final product, such as violence, use of drugs or sexual themes. Another social aspect concerned could be how real world skills would be gained from such an entertainment product, for instance, increased dexterity or cognitive reasoning/problem solving. A potential perception of a social impact from playing the game could be whether the game was sufficiently intellectually engaging. Strategies to prevent any negative or social impact would need to be taken into account when designing the mechanics and the written content. Also, in order to determine if there was any potentially upsetting content, the game will be subject to testing from different groups of individuals and their feedback given.
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	As the aim for this project is to determine to what extend a game can be produced from open source software (much like an individual would do if they were starting to create games for the first time), there should be no costs involved and no purchases necessary for the project. The only software required will be the open-source game engine used and any assets that need to be created or purchased to implement into the game with regards to character models or scene backdrops.
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**

As all of the content needed can be accessed online, there is no foreseeable specific resources that will be needed for this project.

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1	**Background Research**	2	
1.1	Research into pre-existing games		Report showing findings
1.2	Research into what makes a good puzzle game		Report showing findings
1.3	Comparison of potential game engines		Comparison Report
1.4	Pros and cons list of each engine		Pros and cons list in report
1.5	Familiarisation of how to use chosen engine		Findings and sources used to learn the tool noted
2	**Analysis and design**	5	
2.1	Gather requirements for project		Requirements Specification
2.2	Design connections between level, player and ai		UML diagrams
2.3	Level/ World Design concepts		Description/diagrams of levels/world
2.4	Entity interaction design (with player and world/level)		Skeletal pseudo code models of how some features will work.
2.5	Source assets to use in the game		Document detailing where and how each asset was created/collected (e.g blender model)
3	**Develop prototype**	5	
3.1	Initial setup tests/ mechanics	…	Outputted report of how mechanics work
3.2	Level/ world navigation	…	Output of any coding needed to aid the traversal of the world
3.3	Ai interaction testing		Code snippets explaining the interaction of the ai with the player etc.
4	**Testing, evaluation/validation**	3	
4.1	Development testing/Unit Testing		Tests for each unit of the game when it has been created
4.2	Alpha Testing		Testing of the game to assess general features by the developer
4.3	White and Black Box Testing		Testing cases for varying input data both looking at the internal code and just looking at inputs and outputs
4.4	Requirements Testing		Analysis of whether the created product reflects the initial requirements for the

			system.
4.5	Beta Testing		Test results from the beta test users and their comments on the game.
4.6	Acceptance Testing		Analysis of whether or not the product created reflected how users wanted it to work (this would be gained from feedback at earlier stages in analysis)
5	**Assessments**	2	
5.1	write-up project report	1	Project Report
5.2	produce poster	0.5	Poster
5.3	Prepare for project demonstration	0.5	Project Presentation
TOTAL	**Sum of total effort in weeks**	**17**	

SECTION 4 - Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

	START DATE: 23/09/16												
	Project Weeks												
Project stage	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
2 Analysis/Design													
3 Develop prototype.													
NB: Development may continue due to changes recommended in Testing													
4 Testing, evaluation/validation					Logbook								
NB: Testing will continue throughout development.													
5 Assessments						Demo	Poster	Report					

AREA HEALTH AND SAFETY RISK ASSESSMENT FORM (RA1)

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)			

SECTION 1 : Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels	X	**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	
2.	Fall of objects		**7.**	Heating & ventilation	X	**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping		**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration	X	**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	
5.	Display screen equipment	X	**10.**	Electrical Equipment	X	**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	X

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2. For more complex activities or projects you are advised to use Form RA2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	Risk Level (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
5	Excessive time spent looking at the computer screen	Regular breaks, making sure to work in well-lit conditions. Change colour settings on laptop		X		
6	Low light may pose certain hazards	Making sure there is adequate lighting in place.			X	
7.	Temperature and air flow may have an impact on work/ ability to concentrate	Ensure a comfortable and healthy working environment. Open windows			X	
Name of Assessor(s)			**SIGNED**			
Review date						

Health and Safety Risk Assessments – continuation sheet

Assessment Reference No	
Continuation sheet number:	

SECTION 2 continued: Risk Controls

Hazard No.	Hazard Description	Existing controls to reduce risk	Risk Level (tick one)			Further action needed to reduce risks
			High	Med	Low	*(provide timescales and initials of person responsible for action)*
10	Use of computer systems and charging cables with water or power surges	Use of surge protection equipment to prevent loss. Take care to ensure no water products are used in proximity to the electrical equipment		X		
14	Noise and vibration of external sources may effect work	Work in quiet spaces such as library or lab away from external noise sources.			X	

	Failure to back up work may lead to loss of work.	Regularly back up work both to a hard drive and the cloud and keep all versions updated.			X	Depending on how long ago a backup was made delays can be varied.
Name of Assessor(s)			**SIGNED**			
Review date						

Appendix 2: Code

1.) boxMove Class

```
using UnityEngine;
using System.Collections;

public class boxMove : MonoBehaviour {

    //Stores the transforms for objects that will be effected by this scripts behaviour
    public Transform object1;
    public Transform object2;
    public Transform object3;

    //Stores the shifts for the transform of the objects
    public float shiftobj1, shiftobj2, shiftobj3;

    //Booleans to prevent user from multiple button presses in a short space of time
    private bool isPressed = false;
    private bool hasMoved = false;

    void OnTriggerStay(Collider other)
    {
```

```
        if(isPressed == false)
        {
            //If the player enters the button and presses Z
            if (other.tag == "Player" && Input.GetKeyDown(KeyCode.Z) && other.GetType() == typeof(CapsuleCollider))
            {
                isPressed = true;

                //If hasMoved is false, then they are moving from origin to position
                if (hasMoved == false)
                {
                    object1.position = Vector3.Lerp(object1.position, new Vector3(object1.position.x, object1.position.y + shiftobj1,
object1.position.z), 1);
                    object2.position = Vector3.Lerp(object2.position, new Vector3(object2.position.x, object2.position.y + shiftobj2,
object2.position.z), 1);
                    object3.position = Vector3.Lerp(object3.position, new Vector3(object3.position.x, object3.position.y + shiftobj3,
object3.position.z), 1);
                    hasMoved = true;
                }else if(hasMoved == true)
                {
                    //Then we should move them from position back to the origin
                    object1.position = Vector3.Lerp(object1.position, new Vector3(object1.position.x, object1.position.y - shiftobj1,
object1.position.z), 1);
                    object2.position = Vector3.Lerp(object2.position, new Vector3(object2.position.x, object2.position.y - shiftobj2,
object2.position.z), 1);
                    object3.position = Vector3.Lerp(object3.position, new Vector3(object3.position.x, object3.position.y - shiftobj3,
object3.position.z), 1);
                    hasMoved = false;
                }

            }
        }else
        {
            //isPressed is true, so set it back to false
            isPressed = false;
        }

    }

}
```

2.) Camera Follow Script Class

```
using UnityEngine;
using System.Collections;

public class cameraFollowScript : MonoBehaviour
{

    //Attached game object to apply camera transform on
    public GameObject followTarget;
    public bool hasRotated = false;
    public float currentcamRotation = 0;

    //Store the offset to keep the camera from the target
    Vector3 camDelay;

    // Use this for initialization
    void Start()
    {
        camDelay = transform.position - followTarget.transform.position;
    }


    void LateUpdate()
    {
        if (hasRotated == false)
        {
            //Stores the position of the target and updates the cam position to camDelay offset value behind player
            Vector3 actualPosition = followTarget.transform.position + camDelay;
            transform.position = actualPosition;

        }
        if (hasRotated == true)
        {

            //Stores the position of the target and updates the cam position to camDelay offset value behind player
            Vector3 actualPosition = followTarget.transform.position + camDelay;
            transform.position = actualPosition;
```

```
        }


    }

    public void rotateCamera()
    {
        //Get the current value of rotation about the z axis of the object
        currentcamRotation = transform.rotation.z;

        //Define a target rotation of 180 degrees
        Vector3 targetRotation = new Vector3(0, 0, 180);

        //If the world has not been rotated
        if (hasRotated == false)
        {
            //If the current rotation of the camera is equal to 180 degrees
            if (transform.eulerAngles == targetRotation)
            {
                //Set hasRotated to true
                hasRotated = true;
                //Find the parent gravity object and set its transform(rotating the world)
                transform.parent.gameObject.transform.Rotate(new Vector3(0, 0, 180));

                //Move the camera slightly so that it still centers the player
                transform.position = followTarget.transform.position + camDelay;
                camDelay = transform.position - followTarget.transform.position;

            }
            else
            {
                //Rotate the camera smoothly clockwise on each call
                transform.eulerAngles = Vector3.MoveTowards(transform.rotation.eulerAngles, new Vector3(0, 0, 180), 100 *
Time.deltaTime);

            }

        }
        else if (hasRotated == true)
        {
```

```
            if (transform.eulerAngles == targetRotation)
            {
                //Set it to 0 degrees and set hasRotated to false
                hasRotated = false;
                //Find the parent gravity object and set its transform
                transform.parent.gameObject.transform.Rotate(new Vector3(0, 0, 180));

            }
            else
            {
                //Rotate the camera smoothly anti-clockwise on each call
                transform.eulerAngles = Vector3.MoveTowards(transform.rotation.eulerAngles, new Vector3(0, 0, 180), 100 *
Time.deltaTime);
            }
        }
    }


}
```

3.) Destroy Cube Class

```
using UnityEngine;
using System.Collections;

public class DestroyCube : MonoBehaviour {

    private float counter;

       void OnTriggerStay(Collider other)
    {
        counter++;
        //Destory all sequence cubes you come into contact with
        if (other.tag == "seqCube" && counter > 50)
        {
                GameObject.Destroy(other.gameObject);
```

```
            counter = 0;
        }
    }
}
```

4.) Enemy Follow Script

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


//Used to store walkable points in the map
public class Waypoint
{

    //This class will store 2 variables, one for if the object is an edge, and the Vector3 position of that block

    public bool onEdge;
    public Vector3 walkablePoint;


    //Constructor for the creation of a waypoint

    public Waypoint(float xComp, float yComp, float zComp, bool edge)
    {
        this.onEdge = edge;
        this.walkablePoint = new Vector3(xComp, yComp, zComp);
    }



}

public class EnemyFollow : MonoBehaviour
{

    //Use an Enum to store the possible states for our AI
```

```
    private enum state { START, PATROLLING, SEEK, CHASEANDATTACK, DEAD }

    //Variables used so enemy can detect the player through noise
    public SphereCollider myCollider;
    public MovingPlayer sensingPerson;

    //Various variables for the AI
    public float health = 0;
    public float speed = 0.5f;
    public bool isDead = false, isHeard = false, isSeen = false;
    public float moveTime = 0;
    public float timeBetweenMoves = 4f;
    //Controls how long the enemy waits after not seeing/hearing player
    public float waitCounter = 0;
    //Controls the max amount of time the enemy will wait
    public float maxWaitTime = 3f;
    //Controls how far the enemy can see
    public float sightLength = 3f;
    public bool hasJumped = false;
    public int timeBetweenJumps = 100;
    public float jumpSpeed = 5.0f;
    public float gravity = 1.0f;
    private float acceleration = 0.02f;
    private bool isFlipped = false;
    private Vector2 moveDirection = Vector2.zero;
    private int timeOnEdge = 0;



    //Stores where the player was last seen or heard
    public Vector3 lastHeardLocation;
    public Vector3 lastSeenLocation;


    //Stores the enemy's current state, defaults to start
    private state currentState = state.START;

    //Waypoints of walkable map points, current target and current position
    private List<Waypoint> pathFindArray = new List<Waypoint>();
    private Waypoint currentWaypoint;
```

```
    private Waypoint currentPosition;

    //Get the enemys character controller component
    private CharacterController controller;

    public GameObject prefab;
  void Start()
    {
        //This array stores walkable points for the AI
        pathFindArray = drawInitialMap();

        //Stores the initial tracking waypoint for the AI
        currentWaypoint = pathFindArray[Random.Range(0, pathFindArray.Count)];
        //Stores the ai's initial position in the array of walkable points and sets its position to that
        currentPosition = pathFindArray[Random.Range(0, pathFindArray.Count)];
        transform.position = currentPosition.walkablePoint;
        controller = GetComponent<CharacterController>();

        //for (int i = 0; i < pathFindArray.Count; i++)
        //{
        //    Debug.Log("Walkable point " + i + " is " + pathFindArray[i].walkablePoint);
        //    Debug.Log("Is the object an Edge? " + pathFindArray[i].onEdge);
        //    //If the platform is an edge, turn the prefab red
        //    //if (pathFindArray[i].onEdge == true)
        //    //{
        //    //    Debug.Log("Setting colour to red");
        //    //    prefab.GetComponent<Renderer>().material.color = Color.red;
        //    //    Instantiate(prefab, pathFindArray[i].walkablePoint, Quaternion.identity);
        //    //}
        //    if (pathFindArray[i].onEdge == false)
        //    {
        //        Debug.Log("Setting colour to grey");
        //        prefab.GetComponent<Renderer>().material.color = Color.grey;
        //        Instantiate(prefab, pathFindArray[i].walkablePoint, Quaternion.identity);
        //    }
        //}

    }

    void FixedUpdate()
```

```
    {

        switch (currentState)
        {

            //Initialises AI with health
            case state.START:
                {
                    health = 10;
                    //Transition state to patrolling
                    swapState(state.PATROLLING);
                    break;
                }
            case state.PATROLLING:
                {
                    gameObject.GetComponent<Renderer>().material.color = Color.white;
                    //First check if the enemy is dead, if so set dead state and set isDead to true
                    if (health <= 0)
                    {
                        //Used to determine to the player if the enemy is dead or not
                        isDead = true;
                        swapState(state.DEAD);
                        break;
                    }


                    //If you look right and see the player, switch to Chase and Attack
                    if (Look(new Vector3(1, 0, 0)))
                    {
                        swapState(state.CHASEANDATTACK);
                        break;
                    }
                    else if (Look(new Vector3(-1, 0, 0)))
                    {
                        //If you can see the player to your left, transition to the chase and attack
                        swapState(state.CHASEANDATTACK);
                        break;
                    }
                    else //You cant see the player, so listen for them
                    {
```

```
                        //If you can hear the player
                        if (myCollider.bounds.Intersects(sensingPerson.noiseCollider.bounds))
                        {
                            //Gives a direction vector based on the last heard location of the player
                            lastHeardLocation = sensingPerson.transform.position;
                            //Start seeking the player
                            swapState(state.SEEK);
                            break;
                        }

                    }
    //If the moving in the random direction for a certain duration
                    if (moveTime >= timeBetweenMoves)
                    {
                        //Pick a new direction
                        currentWaypoint = pathFindArray[Random.Range(0, pathFindArray.Count)];

                        //Reset the move time for this movement
                        moveTime = 0;

                        setSpeed(1);
                        //Move towards the new target
                        move(state.PATROLLING, currentWaypoint);

                    }
                    else
                    {
                        setSpeed(1);
                        //Move in a the same direction
                        move(state.PATROLLING, currentWaypoint);
                        //Increment moveTime
                        moveTime++;
                    }

                    //If none of the above conditions occur, skip to the next iteration of update (next frame)
                    break;
                }

            case state.SEEK:
                {
```

```
                    gameObject.GetComponent<Renderer>().material.color = Color.yellow;
                    //First check if the enemy is dead, as it could occur in any state
                    if (health <= 0)
                    {
                        //Used to determine to the player if the enemy is dead or not
                        isDead = true;
                        swapState(state.DEAD);
                        break;
                    }
                    //The player can now be heard
                    isHeard = true;

                    //If the enemy has not heard the enemy for maxWaitTime seconds
                    if (waitCounter >= maxWaitTime)
                    {
                        //The player cannot be heard
                        isHeard = false;
                        //Reset waitcounter
                        waitCounter = 0;
                        //Transition to patrolling
                        swapState(state.PATROLLING);
                        break;
                    }


                    //Set your target to the closest point in your list to the heard location
                    currentWaypoint = findClosestNavPoint(lastHeardLocation);

                    //Set to normal movement speed and move in that direction
                    setSpeed(1);
                    move(state.SEEK, currentWaypoint);

//Look left and right for the player, if you can see them, transfer state
                    if (Look(new Vector3(1, 0, 0)))
                    {
                        lastSeenLocation = sensingPerson.transform.position;
                        swapState(state.CHASEANDATTACK);
                        break;
                    }
                    else if (Look(new Vector3(-1, 0, 0)))
```

```
                {
                    lastSeenLocation = sensingPerson.transform.position;
                    swapState(state.CHASEANDATTACK);
                    break;
                }


                //Can you still hear the player?
                if (myCollider.bounds.Intersects(sensingPerson.noiseCollider.bounds))
                {
                    //Reupdate the last heard location of the player by using bounds
                    lastHeardLocation = sensingPerson.transform.position;
                }
                else
                {
                    //increment wait counter
                    waitCounter++;

                }
                //The final break will act as the refresh for still being in seek

                break;
            }

        case state.CHASEANDATTACK:
            {
                gameObject.GetComponent<Renderer>().material.color = Color.red;
                //First check if the enemy is dead, as it could occur in any state
                if (health <= 0)
                {
                    //Used to determine to the player if the enemy is dead or not
                    isDead = true;
                    swapState(state.DEAD);
                    break;
                }

                //The player can now be seen!
                isSeen = true;

                //If the enemy has not seen the enemy for maxWaitTime seconds
```

```
                    if (waitCounter >= maxWaitTime)
                    {
                        //The player cannot be seen
                        isSeen = false;
                        //Reset waitcounter
                        waitCounter = 0;
                        //Now try to hear the player
                        swapState(state.SEEK);
                        break;
                    }

//Set the seen and heard locations to the same vector in case of sight loss
                    lastHeardLocation = lastSeenLocation;

                    //Set the current target to the closest waypoint in the AI's list to where player was seen
                    currentWaypoint = findClosestNavPoint(lastSeenLocation);

                    //Chase player with increased speed
                    setSpeed(2);
                    move(state.CHASEANDATTACK, currentWaypoint);


                    //Make a check if you can still see the player
                    //If you can, reupdate last seen and heard positions
                    //Look left and right for the player
                    if (Look(new Vector3(1, 0, 0)))
                    {
                        lastSeenLocation = sensingPerson.transform.position;
                        break;
                    }
                    else if (Look(new Vector3(-1, 0, 0)))
                    {
                        lastSeenLocation = sensingPerson.transform.position;
                        break;
                    }
                    else
                    {
                        //else you have tried everything and cannot see the player
                        //Increment the wait counter for the next pass
                        waitCounter++;
```

```
                    }

                    //Break will allow for the AI to continue to move in that direction on the next pass
                    break;
                }

            case state.DEAD:
                {
                    //Destroy the gameObject, we no longer need it
                    Destroy(this.gameObject, 2f);
                    break;
                }

            default:
                {
                    //If all else fails, just patrol
                    currentState = state.PATROLLING;
                    break;
                }
        }


    }

//Swaps the AI from one state to a specified other state
    void swapState(state switchStateVal)
    {
        currentState = switchStateVal;
    }

    //Stores the current state and the node in world space to move to
    void move(state myState, Waypoint target)
    {
        //If patrolling, target will be exact, if Seeking or Chase and Attack, target will be the result of ClosestnavPoint function
        //If current poistion is waypoint position you have reached your target and need an update
        if (currentPosition.walkablePoint == target.walkablePoint)
        {
            switch (myState)
            {
```

```
                case state.PATROLLING:
                    //Pick a new random position
                    currentWaypoint = pathFindArray[Random.Range(0, pathFindArray.Count)];
                    break;
                case state.SEEK:
                    //You have reached the players last heard location, player death handled in player script so just return to
FixedUpdate()
                    return;

                case state.CHASEANDATTACK:
                    //You have reached the players last seen location, player death handled in player script so just return to
FixedUpdate()
                    return;

                default:
                    return;


            };


        }
        else
        {
            //First and foremost, check if you are already on an edge
            if (currentPosition.onEdge == true)
            {
                //Is the target above or below you
                //To do this, get the current position transform point and compare the Y components
                float myElevation = currentPosition.walkablePoint.y;
                float targetElevation = target.walkablePoint.y;

                if (myElevation < targetElevation)
                {
                    //Jump to the target above you
                    Jump(true);
                }
                else
                {
                    //The target is below you so jump to below
```

```
                    Jump(false);
                }
            }
            else
            {
                //If the target is to the left of you (func returns true)
                if (LeftOrRight(target))
                {
                    //If they are on the left of you, move 1 unit to your left and update your current position
                    controller.Move(Vector3.left.normalized * speed * Time.deltaTime);
                    //Update your current position (Approximated through the closestNavPoint Function)
                    currentPosition = findClosestNavPoint(transform.position);
                }
                else
                {
                    //If they are on the left of you, move 1 unit to your left and update your current position
                    controller.Move(Vector3.right.normalized * speed * Time.deltaTime);
                    //Update your current position (Approximated through the closestNavPoint Function)
                    currentPosition = findClosestNavPoint(transform.position);

                }

            }

        }



    }
//Function to determine whether the object is to the left or right of its target
    public bool LeftOrRight(Waypoint target)
    {

        if (transform.position.x > target.walkablePoint.x)
        {
            return true;
        }
        else if (transform.position.x < target.walkablePoint.x)
        {
            return false;
```

```
        }
        else
        {
            return false;
        }

    }

    //Function to determine which way the AI should move on a platform following a jump
    public void PlatformMove()
    {
        //Cast 2 rays from your position diagonally left and right
        RaycastHit swHit, seHit;
        Ray swRay, seRay;
        swRay = new Ray(new Vector3(transform.position.x, transform.position.y + 1, transform.position.z), new Vector3(-1, -1, 0));
        seRay = new Ray(new Vector3(transform.position.x, transform.position.y + 1, transform.position.z), new Vector3(1, -1, 0));

        //Check if both of them collide with something
        if (Physics.Raycast(swRay, out swHit) && Physics.Raycast(seRay, out seHit))
        {
            //Get both the rays distances before the object they hit, the closest intersection of the ray is the direction to move
            if (swHit.distance < seHit.distance)
            {

                //The platform is on the left of you, so move that way off the edge and update your position
                transform.position = new Vector3(transform.position.x - 2, transform.position.y, transform.position.z);
                //Update your position
                currentPosition = findClosestNavPoint(transform.position);

            }
            else if (swHit.distance > seHit.distance)
            {

                //The platform is on the right of you, so move that way off the edge and update your position
                transform.position = new Vector3(transform.position.x + 2, transform.position.y, transform.position.z);
                //Update your position
                currentPosition = findClosestNavPoint(transform.position);
            }
        }
        else if (Physics.Raycast(swRay, out swHit))
```

```
        {

            //The south west ray cast hit something
            //Move to the west to get off the edge
            transform.position = new Vector3(transform.position.x - 2, transform.position.y, transform.position.z);
            //Update your position
            currentPosition = findClosestNavPoint(transform.position);
        }
        else
        {
            //The south east ray cast hit something or neither did, meaning weve failed
            //Move to the east to get off the edge
            transform.position = new Vector3(transform.position.x + 2, transform.position.y, transform.position.z);
            //Update your position
            currentPosition = findClosestNavPoint(transform.position);

        }

}

    //Jumps to a newly calculated platform if able
    public void Jump(bool isAbove)
    {
        if (hasJumped == true)
        {
            //You cannot jump just yet, so don't
            //Subtract from the jumpCounter
            timeBetweenJumps--;
            //Check if the jump count is 0
            if (timeBetweenJumps <= 0)
            {
                //Time between jumps has hit zero again
                timeOnEdge++;
                //You can jump again
                hasJumped = false;
                //Reset the timebetween jumps variable
                timeBetweenJumps = 200;
            }
        }
        else
```

```
        {
            //Check whether you're above or below the target
            if (isAbove == true)
            {
                //The target is above you
                //Calculate distance between you and all edge nodes in PathfindArray that have a higher y component than yours, and
pick the closest one
                //Variable to store closestEdge
                float lowestDist = Mathf.Infinity;
                int totalCount = 0;
                int neededIndex = 0;
                foreach (Waypoint way in pathFindArray)
                {
                    float tmpDist;
                    //If the y component is bigger and its an edge tile,consider it
                    if (way.walkablePoint.y > currentPosition.walkablePoint.y && way.onEdge == true)
                    {
                        //Calculate the distance between them
                        tmpDist = Vector3.Magnitude(way.walkablePoint) - Vector3.Magnitude(currentPosition.walkablePoint);
                        //If the new distance is smaller than the current lowest one
                        if (Mathf.Abs(tmpDist) < lowestDist)
                        {
                            //Set lowest to the current distance
                            lowestDist = tmpDist;
                            //Store the current index of the smallest one
                            neededIndex = totalCount;
                        }
                    }
                    totalCount++;
                }
                //After this you have the closest edge to your player, so teleport to that edge
                transform.position = pathFindArray[neededIndex].walkablePoint;
                //Set hasJumped to true
                hasJumped = true;

                //You've just jumped so check which way you need to move based on the platform youre on
                PlatformMove();

            }
            else
```

```
            {
                //Target is below you
                //Calculate distance between you and all edge nodes in PathfindArray that have a higher y component than yours, and
pick the closest one
                //Variable to store closestEdge
                float lowestDist = Mathf.Infinity;
                int totalCount = 0;
                int neededIndex = 0;
                foreach (Waypoint way in pathFindArray)
                {
                    float tmpDist;
                    //If the y component is smaller and its an edge tile,consider it
                    if (way.walkablePoint.y < currentPosition.walkablePoint.y && way.onEdge == true)
                    {
                        //Calculate the distance between them
                        tmpDist = Vector3.Magnitude(way.walkablePoint) - Vector3.Magnitude(currentPosition.walkablePoint);
                        //If the new distance is smaller than the current lowest one
                        if (Mathf.Abs(tmpDist) < lowestDist)
                        {
                            //Set lowest to the current distance
                            lowestDist = tmpDist;
                            //Store the current index of the smallest one
                            neededIndex = totalCount;
                        }
                    }
                    totalCount++;
                }

                //After this you have the closest edge to your player, so telelport to that edge
                transform.position = pathFindArray[neededIndex].walkablePoint;
                hasJumped = true;

                //You've just jumped so check which way you need to move based on the platform youre on
                PlatformMove();

            }
        }
    }

//Function that casts a ray allowing the AI to 'look' in different directions
```

```
public bool Look(Vector3 lookDirection)
{
    //Cast a ray in the specified direction and see if you can see the player
    RaycastHit connect;
    Ray sightRay = new Ray(transform.position, lookDirection);

    if (Physics.Raycast(sightRay, out connect, sightLength))
    {
        //Player has been seen
        if (connect.collider.tag == "Player")
        {
            //You can see the player, return true
            return true;
        }
    }
    else
    {
        //You cannot see the enemy in this direction
        return false;
    }

    return false;

}

//Set the movement speed of the AI
public void setSpeed(float movespeed)
{
    speed = movespeed;
}

//Function will take the positions of the game objects in the level, calculate nodes above them for walkable tiles
//Expensive so need to limit the number of redraws if we can
//TODO:: Note may need to redraw the map when the world has been rotated or it will mess up the waypoints
List<Waypoint> drawInitialMap()
{
    List<Waypoint> pointList = new List<Waypoint>();

    GameObject[] platformArray = GameObject.FindGameObjectsWithTag("platform");
    foreach (GameObject g in platformArray)
```

```
        {
            //Find the platforms position and its length-has to be of int scale to create aproximate grid
            //Note: The transform position gives the center of the game object, so translate it appropriately
            //First calculate how far above and below the center we need to go based on the platform length

            float platformLength = g.transform.lossyScale.x; //Eg platform scale 5 so -> 5/2 = 2.5 Round(2.5) = 3 - 1 = 2 so add
points +-2 from startPoint
            float scaleLength = Mathf.CeilToInt(platformLength / 2) - 1;

            //Store points into pointList for the graph for the set = {position, pos +1 ... pos+ length}
            //This will give a set of walkable tiles in the map

            //Then if the scaled length is 1 (platform length 3)
            if (scaleLength == 1)
            {
                //Just add the single point to walkable spaces (the center point)
                pointList.Add(new Waypoint(g.transform.position.x, g.transform.position.y + 1, g.transform.position.z, false));
                //Also add the values to the pointlist above and below the center value, these are edges
                pointList.Add(new Waypoint(g.transform.position.x + 1, g.transform.position.y + 1, g.transform.position.z, true));
                pointList.Add(new Waypoint(g.transform.position.x - 1, g.transform.position.y + 1, g.transform.position.z, true));
            }
            else
            {
                //For the length of the platform
                //IMPORTANT NOTE: The scales of the objects must be of integer length for this to efficiently calculate above the
tiles

                //First handle platform length of size 1
                if (platformLength == 1)
                {
                    //Just add the 1 square to the walkable points (it is an edge point)
                    pointList.Add(new Waypoint(g.transform.position.x, g.transform.position.y + 1, g.transform.position.z, true));
                }
                else
                {
                    //Add the center point
                    pointList.Add(new Waypoint(g.transform.position.x, g.transform.position.y + 1, g.transform.position.z, false));

                    //For the scaled length, place walkable navpoints either side of the center until the edge of the platform
                    for (int i = 1; i <= scaleLength; i++)
```

```
                    {
                        //If the i counter equals the scale length, we have an edge
                        if (i == scaleLength)
                        {
                            //Add the points in the normal way, but tag them as edges
                            //Add the point in center (i==0) and the points +- platformLength
                            pointList.Add(new Waypoint(g.transform.position.x + i, g.transform.position.y + 1,
g.transform.position.z, true));
                            pointList.Add(new Waypoint(g.transform.position.x - i, g.transform.position.y + 1,
g.transform.position.z, true));
                        }
                        else
                        {
                            //Otherwise, you are not on an edge, so add them to the list normally
                            pointList.Add(new Waypoint(g.transform.position.x + i, g.transform.position.y + 1,
g.transform.position.z, false));
                            pointList.Add(new Waypoint(g.transform.position.x - i, g.transform.position.y + 1,
g.transform.position.z, false));

                        }

                    }
                }



            }

        }
        return pointList;

    }


    //Finds the closest point in the walkable list to the player and returns it
    private Waypoint findClosestNavPoint(Vector3 targetPoint)
    {
        float minDistanceToTarget = Mathf.Infinity;
        float distance = Mathf.Infinity;
        int index = -1;
```

```
        //Search for the closest point in your list to the point supplied by the function parameter
        for (int i = 0; i < pathFindArray.Count; i++)
        {
            distance = Vector3.Distance(pathFindArray[i].walkablePoint, targetPoint);

            if (distance < minDistanceToTarget)
            {
                minDistanceToTarget = distance;
                index = i;
            }
        }
        Debug.Log("Navigating to point " + pathFindArray[index].walkablePoint);
        return pathFindArray[index];
    }



}
```

5.) Fade Place Name Class

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class fadePlaceName : MonoBehaviour {

    public bool toggleGUI;
    public CanvasGroup canvas;
    public Text placeNameText;
    public string nameText;

    void Start()
    {
        toggleGUI = false;
    }


    void OnTriggerEnter(Collider other)
    {
```

```
        if(other.tag == "Player" && other.GetComponent<CapsuleCollider>().GetType() == typeof(CapsuleCollider))
        {
            toggleGUI = true;
            SetPlaceName();
            fadeGUI();
        }
    }


    void OnTriggerExit(Collider other)
    {
        if(other.tag == "Player" && other.GetComponent<CapsuleCollider>().GetType() == typeof(CapsuleCollider))
        {
            toggleGUI = false;
            fadeGUI();
        }

    }

    void fadeGUI()
    {
        //If the toggleGUI is true, fade the ui into the screen
        if (toggleGUI)
        {
            StartCoroutine(BeginFade());
        }else
        {
            //Fade the UI out, we no longer need it
            StartCoroutine(EndFade());
        }

    }


    //Slowly changes the alpha of the canvasGroup component from 1 to 0
    IEnumerator EndFade()
    {

        while(canvas.alpha > 0)
        {
```

```
            canvas.alpha -= Time.deltaTime;
            yield return null;
        }
    }

    //Slowly changes the alpha of the canvasGroup component from 0 to 1
    IEnumerator BeginFade()
    {
        while(canvas.alpha < 1)
        {
            canvas.alpha += Time.deltaTime * 2;
            yield return null;
        }
    }

    void SetPlaceName()
    {
        placeNameText.text = nameText;
    }
}
```

6.) Gear Puzzle Logic Class (Sequencer)

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

public class gearPuzzleLogic : MonoBehaviour
{

    //Used to pull all objects towards the gear and check against a sequence
    //Stores the sequence of colours that will be checked and colours to parse
    private List<Color> sequenceList = new List<Color>();
    public List<string> colourConverts = new List<string>();
    public SphereCollider pullSphere;
    public float pullRadius;
    //Public variable to control how quickly the gear pulls objects
    public float pullForce = 5;

    //An optional gameobject event that could occur from the correct sequence
    public GameObject optionalEndTrigger;
```

```
    //Stores the current point in the sequence
    private int currentSeqIndex = 0;

    void Start()
    {
        //For each string in the list
        for (int i = 0; i < colourConverts.Count; i++)
        {
            //Try and parse to a colour type
            Color temp = Color.clear;
            ColorUtility.TryParseHtmlString(colourConverts[i], out temp);

            if (temp == Color.clear)
            {
                Debug.LogError("Colour not parsed correctly, revisit the string value of colour");
            }
            else
            {
                //Set the value in the sequence list to that colour
                sequenceList.Add(temp);
            }

        }

        //Set the pull area to the user defined value
        pullSphere.radius = pullRadius;
    }
    void Update()
    {
        //If all colours have been successfully matched in sequence
        if (currentSeqIndex == sequenceList.Count)
        {
            //Set the optional event to active
            if (optionalEndTrigger != null)
            {
                optionalEndTrigger.SetActive(true);
            }

            //Reset the sequence index count
```

```
            currentSeqIndex = 0;
        }
    }
    void OnTriggerStay(Collider sequenceCube)
    {
        //For all objects inside the spherecollider with the tag 'seqCube'
        if (sequenceCube.tag == "seqCube")
        {

            //Pull the cube towards your transform
            Vector3 directionToMove = transform.position - sequenceCube.transform.position;
            sequenceCube.transform.Translate(directionToMove * pullForce * Time.deltaTime);

            //Check the colour of the cube coming in against the sequeunce list, if its correct, accept and advance the sequence
index
            //But if it violates the sequence, reset it
            if (sequenceList[currentSeqIndex] == sequenceCube.gameObject.GetComponent<Renderer>().material.color &&
                gameObject.GetComponent<BoxCollider>().bounds.Intersects(sequenceCube.GetComponent<Collider>().bounds))
            {
                //This is the correct colour in the sequence, so destroy the given cube and advance the index
                GameObject.Destroy(sequenceCube.gameObject);
                currentSeqIndex++;

            }
            else if (sequenceList[currentSeqIndex] != sequenceCube.gameObject.GetComponent<Renderer>().material.color &&
                gameObject.GetComponent<BoxCollider>().bounds.Intersects(sequenceCube.GetComponent<Collider>().bounds))
            {
                //Incorrect sequence has been entered, so reset the sequence back to the start
                GameObject.Destroy(sequenceCube.gameObject);
                currentSeqIndex = 0;
            }

        }

    }

    public void setPullRadius(float value)
    {
        //Set a runtime value for the radius that the pull takes
        pullSphere.radius = value;
```

```
    }
}
```

7.) Load Level Class

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class LoadLevel : MonoBehaviour
{

    //Used for returning to the start menu
    public string levelSkip;

    //Handles the level selection on the main menu, loading the level chosen by the user
    public void OnClick(string levelToLoad)
    {
        SceneManager.LoadScene(levelToLoad);
    }

    //If button clicked with no arguments, quit the application
    public void OnClick()
    {
        Application.Quit();
    }

    //Function to allow scene transition in game, when a level is finished
    void OnTriggerEnter(Collider other)
    {

        //If the user wishes to attach a level skip to an object set to not null
        if (levelSkip != null || levelSkip == "")
        {
            if (other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
            {
                //Load the scene relevant to the skip
                SceneManager.LoadScene(levelSkip);
            }
```

```
        }
    }
}
```

8.) Moving Player Class

```
using UnityEngine;
using System.Collections;

public class MovingPlayer : MonoBehaviour
{
    //Variables for the player
    public float speed = 5.0f;
    public float jumpSpeed = 5.0f;
    public float gravity = 1.0f;
    public float minY = 0, maxY = 0;
    public float slowAmount = 1;
    public float pushPower = 2.0f;
    private Vector2 moveDirection = Vector2.zero;


    //Variables for the players noise sphere
    private float originObjSpeed = 1;
    private float moveTime = 1.0f;
    private float maxNoise = 5.0f;
    private float minNoise = 1.0f;
    private float acceleration = 0.02f;
    private bool isFlipped = false;
    public SphereCollider noiseCollider;

    //Current position in world space the player can respawn at
    private Vector3 currentSpawnPoint;
    //Debug for detailing where other platform is
    public Transform startPlatform;

    //Set ip the initial noise radius of the player and set the initial spawn point
    void Start()
    {
        //Initialise Variables
```

```
        noiseCollider.radius = 1.0f;

        //Set initial spawn
        currentSpawnPoint = transform.position;
    }


    void Update()
    {
        //At any point, if the player is outside the map bounds as defined by max and min y, then respawn the player
        if (transform.position.y < minY || transform.position.y > maxY)
        {
            //Respawns at the latest checkpoint triggered by the player
            Respawn();
        }


        //Get the main camera's rotation variable so we can see what current position the camera is at
        GameObject playerCam = GameObject.FindGameObjectWithTag("MainCamera");
        cameraFollowScript camMovement = playerCam.GetComponent<cameraFollowScript>();
        CharacterController controller = GetComponent<CharacterController>();

        //Check if the controller is on the ground and the camera has not been rotated
        if (controller.isGrounded && camMovement.hasRotated == false)
        {

            //Get horizontal movement input from the player
            moveDirection = new Vector2(Input.GetAxis("Horizontal"), 0);
            //Relate the move direction to the objects transform (for movement)
            moveDirection = transform.TransformDirection(moveDirection);

            //Multiply the movement transformation by the objects speed
            moveDirection *= speed;

            //Handles vertical movement using a the Jump button (space)
            if (Input.GetButton("Jump"))
            {
                //Affects the y component of the players Vector2 movment
                moveDirection.y = jumpSpeed;
            }
```

```
        //If the camera has rotated, that means the player is upside down
    }
    else if (controller.isGrounded && camMovement.hasRotated == true)
    {

        //Get the input from the player and negate it so that the movement when flipped is the same
        moveDirection = new Vector2(Input.GetAxis("Horizontal"), 0);
        moveDirection = transform.TransformDirection(-moveDirection);
        moveDirection *= speed;

        if (Input.GetButton("Jump"))
        {
            moveDirection.y = jumpSpeed;
        }
    }

    //Check if the player is holding down the move buttons and if they are, update the noiseSphere
    updateNoiseRadius();

    //Calculate the gravity effect on the controller by subtracting from its y component over time
    moveDirection.y -= gravity * Time.deltaTime;

    //Move in the direction as supplied by the user over time
    controller.Move(moveDirection * Time.deltaTime);



    //When X is pressed
    if (Input.GetKeyDown(KeyCode.X))
    {
        //Toggle whether the world is flipped or not
        if (isFlipped == true)
        {
            isFlipped = false;
        }
        else
        {
            isFlipped = true;
        }
```

```
        }

        if (isFlipped)
        {

            //If the camera on the player has not rotated (inverted), call rotate
            if (camMovement.hasRotated == false)
            {
                camMovement.rotateCamera();
            }


        }
        if (!isFlipped)
        {

            //If the camera on the player is currently inverted
            if (camMovement.hasRotated == true)
            {
                //Rotate back to normal
                camMovement.rotateCamera();
            }

        }


        //If the user presses the H key
        if (Input.GetKeyDown(KeyCode.H))
        {
            //Find all instances in the world where an object is tagged with 'isSlowable'
            GameObject[] myArray = GameObject.FindGameObjectsWithTag("isSlowable");

            foreach (GameObject g in myArray)
            {
                //Get the objects movement component to access its speed
                platformMove myobject = g.GetComponent<platformMove>();

                //Get the speed before you modify and store in a var
                originObjSpeed = myobject.speed;
```

```
                myobject.setSpeed(slowAmount);
            }
        }
        else if (Input.GetKeyDown(KeyCode.J))
        {
            //When J is pressed, set the object speed back to its original value
            GameObject[] myArray = GameObject.FindGameObjectsWithTag("isSlowable");
            foreach (GameObject g in myArray)
            {
                platformMove myobject = g.GetComponent<platformMove>();
                myobject.setSpeed(originObjSpeed);
            }
        }
    }


    private void updateNoiseRadius()
    {
        //If the player is moving, track how long they are holding the button down
        if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D)
|| Input.GetKey(KeyCode.Space))
        {

            //If the moveTime is greater than the maxNoise stop incrementing
            if (moveTime >= maxNoise)
            {
                moveTime = maxNoise;
            }
            else
            {
                //Gradually increase the time moving by the acceleration of the player upto a maximum
                moveTime += acceleration;
            }

        }
        else
        {
            //If the player is not moving, reduce the players noiseSphere down to the minNoise value
            if (moveTime <= minNoise)
            {
```

```
                moveTime = minNoise;
            }
            else
            {
                moveTime -= acceleration;
            }
        }
        //Update the radius of the players spherecollider to represent the noise
        noiseCollider.radius = moveTime;


    }

    //Handles the collisions between the object and any other object it connects with that has a Rigidbody
    void OnControllerColliderHit(ControllerColliderHit hit)
    {
        //If the body that is collided with has no rididbody or isKinematic (no interaction with physics)
        Rigidbody body = hit.collider.attachedRigidbody;
        if (body == null || body.isKinematic)
        {
            return;
        }


        if (hit.moveDirection.y < -0.3F)
        {
            return;
        }

        //Define a new vector that the hit object is to travel along
        Vector3 pushDir = new Vector3(hit.moveDirection.x, 0, hit.moveDirection.z);

        //Apply velocity to the collided objects rigidbody to enable a pushing behaviour based on pushpower
        body.velocity = pushDir * pushPower;
    }

    public void SetRespawn(Vector3 newSpawn)
    {
        //Set this targets transform to the newspawn value
        currentSpawnPoint = newSpawn;
```

```
    }

    //Create function so if you stray too far from the platform you respawn
    //Requires there to always be a spawnPoint with that name and all to be tagged respawn
    public void Respawn()
    {
        //Get the new location of the latest spawnpoint object and set it to that
        GameObject[] spawns = GameObject.FindGameObjectsWithTag("Respawn");
        if (isFlipped)
        {
            //If the world is flipped and you need to respawn, do so at an inverted spawnpoint
            foreach (GameObject g in spawns)
            {
                if (g.name == "spawnPoint 1")
                {
                    transform.position = g.transform.position;
                }
            }
        }
        else
        {
            transform.position = currentSpawnPoint;
        }

    }

    //Handles when the player comes into contact with the enemy
    void OnTriggerStay(Collider other)
    {
        //If the collided object has a tag of enemy and you are sufficiently close to that object
        if (other.tag == "enemy" && Vector3.Distance(transform.position, other.gameObject.transform.position) < 1.2)
        {
            //Handles the case of a specific enemy type, requiring change of that particular enemy's state
            if (other.name == "testEnemy")
            {
                //Reset the enemy's position
                other.GetComponent<testu>().transform.position = other.GetComponent<testu>().initialSpawn;
                other.GetComponent<testu>().swapState(testu.state.PATROLLING);
            }
            //Then respawn
```

```
            Respawn();
        }
    }


}
```

9.) Platform Move Class

```
using UnityEngine;
using System.Collections;

public class platformMove : MonoBehaviour
{

    //Stores where the platform should bounce between
    public Vector3 startPos, endPos, currentPos;
    //Stores the rate at which the platform should move relative to time
    public float speed;
    //False is whether or not a target has been reached
    private bool isTargetReached = false;

    //Initialise the start position for the object
    void Start()
    {
        currentPos = startPos;
    }

    //Updates after each physics calculation (not each frame like update)
    void FixedUpdate()
    {

        //First check to see if we got to the target place (currentpos = endpos)
        if (currentPos == endPos)
        {
            //We have reached the target
            isTargetReached = true;
```

```
        }
        else if (currentPos == startPos)
        {
            //We need to go back to the end position
            isTargetReached = false;
        }

        //If we have reached the endPos Vector
        if (isTargetReached)
        {
            //Move towards the startPos Vector
            transform.position = Vector3.MoveTowards(currentPos, startPos, speed * Time.unscaledDeltaTime);
        }
        else
        {
            //Move towards the the end again
            transform.position = Vector3.MoveTowards(currentPos, endPos, speed * Time.unscaledDeltaTime);

        }

        //Update current position
        currentPos = transform.position;
    }

    //Controls the speed at which the platforms move
    public void setSpeed(float input)
    {
        speed = input;
    }


}
```

10.) Popup dialog Script

```
using UnityEngine;
using System.Collections;

public class PopupDialog : MonoBehaviour
{

    //public boolean for developer if they wish to add their own size for a popup window
    public bool manualReSize = false;

    //Used for storing the width and length of the popup window
    public float popupWidth;
    public float popupHeight;


    public string popupText;
    private bool togglePopup = false;

    //On starting, work out the appropriate length for the text in the text field and set the width and height of the popup
accordingly
    void Start()
    {
        if (manualReSize == false)
        {
            //Attempt to auto resize the box to the text
            //Assume average line length of 27 characters
            //So 150/27 = characters per unit rect width approx 5.55
            //15 is the min size for one line
            popupWidth = 150;
            //Work out the number of lines needed
            int numLines = Mathf.CeilToInt(popupText.Length / 27);
            if (numLines > 1)
            {
                popupHeight = 2f * (numLines * 16);
            }
            else
            {
                popupHeight = 20 * 2f;
            }

        }
```

```
    }
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            togglePopup = true;
        }
    }


    void OnTriggerExit(Collider other)
    {
        if (other.tag == "Player")
        {
            togglePopup = false;
        }

    }

    void OnGUI()
    {
        if (togglePopup)
        {


            Vector3 position = Camera.main.WorldToScreenPoint(transform.position);
            position.y = Screen.height - position.y;

            //Create a GUIBox and display it above where the attached gameobject is
            GUIStyle myBoxStyle = new GUIStyle(GUI.skin.box);
            myBoxStyle.normal.textColor = Color.white;
            myBoxStyle.wordWrap = true;
            myBoxStyle.fontSize = 15;

            GUI.Box(new Rect(position.x - 10, position.y - 40, popupWidth, popupHeight), popupText, myBoxStyle);
        }
    }


}
```

11.) Portal Generator Script

```
using UnityEngine;
using System.Collections;

public class PortalGenrator : MonoBehaviour {

    //This class is used to spawn a portal after a certain task has been achieved trigger entered
    //Right now the script is only trigger based, but this could be adapted for when ceratin quest objectives have been completed

    //Gets the gameobject associated with the end portal and activates/ deactivates it
    public GameObject portalObject;


    void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
        {
            //Player has got to the needed part of the level to unlock the next objective/exit/logicpath
            if(portalObject.activeSelf == false)
            {
                //Activate the portal game object
                portalObject.SetActive(true);

            }

        }
    }
}
```

12.) Spawn Cube Class

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

```
public class SpawnCube : MonoBehaviour {

    //Stores the list of currently existing objects that cannot exist together
    public List<string> checkList = new List<string>();
    //Stores the name of the to be created object
    public string currentName;
    //Stores the gameobject to be spawned
    private GameObject sequenceCubePrefab = null;
    //Used to determine what sort of box to spawn
    public string setColour;
    //Used to determine the location the box will spawn
    public Vector3 spawnPos;
    //Used to determine if an object has been spawned already
    private bool isSpawned = false;

    public float maxDistApart;
    void Start () {
       //If no colour has been selected, set the default to white
    if(setColour == null)
        {
            setColour = "white";
        }
       }

       // Update is called once per frame
       void Update () {
        //If the cube has strayed too far from its start position, destory it
        if(sequenceCubePrefab != null)
        {

            Vector3 dist = transform.position - sequenceCubePrefab.transform.position;
            Debug.Log(dist);
            //Find the distance between the objects transform and its maxDistance apart
            if (Mathf.Abs(dist.x) > maxDistApart || Mathf.Abs(dist.y) > maxDistApart || Mathf.Abs(dist.z) > maxDistApart)
            {
                Destroy(sequenceCubePrefab.gameObject);
                isSpawned = false;
            }
        }
```

```
    }

    void OnTriggerStay(Collider other)
    {
        if(Input.GetKeyDown(KeyCode.Z) && other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
        {
            if(isSpawned == false)
            {
                //Spawn an instance of your own cube colour
                sequenceCubePrefab = GameObject.CreatePrimitive(PrimitiveType.Cube);
                sequenceCubePrefab.transform.position = spawnPos;
                sequenceCubePrefab.transform.localScale = new Vector3(1, 1, 1);
                sequenceCubePrefab.transform.name = currentName;
                sequenceCubePrefab.tag = "seqCube";
                sequenceCubePrefab.AddComponent<Rigidbody>().useGravity = false;


                //Try to set the colour via string with the user specified one
                Color temp = Color.clear;
                ColorUtility.TryParseHtmlString(setColour, out temp);
                //Set the cube to that colour
                if(temp == Color.clear)
                {
                    Debug.LogError("Could not parse colour correctly");
                }else
                {
                    sequenceCubePrefab.GetComponent<Renderer>().material.color = temp;
                }

            }

        }
    }
}
```

13.) Teleport Player Class

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

```
public class teleportPlayer : MonoBehaviour {

    public Vector3 teleportTarget;

       void OnTriggerEnter(Collider other)
    {

        if(other.tag == "Player" && other.GetType() == typeof(CapsuleCollider))
        {
            //Teleport player back to where you want the target to be
            other.transform.localPosition = teleportTarget;
        }

    }
}
```

14.) Toggle Singularity Class

```
using UnityEngine;
using System.Collections;

public class ToggleSingularity : MonoBehaviour {

    //Stores the reference for the sequencer object and the value that is to be used to effect said object
    public gearPuzzleLogic sequencer;
    public float singularityVal = 0;
    public ParticleSystem singularityFX;
    //Ensures the singularity is only activated once and cannot be toggled - could change this to allow toggling with more complex
puzzles
    private bool isActivated = false;

    void OnTriggerStay(Collider other)
    {
        //If the player enters the trigger zone and presses Z
        if(other.tag == "Player" && other.GetType() == typeof(CapsuleCollider) && Input.GetKeyDown(KeyCode.Z))
```

```
        {
            //Set activated to false and alter the pull radius of the singularity
            if (isActivated == false)
            {
                //Enable the singularity effect
                singularityFX.gameObject.SetActive(true);
                sequencer.setPullRadius(singularityVal);
                isActivated = true;
            }
        }
    }
}
```

15.) Trigger Spawn Point Class

```
using UnityEngine;
using System.Collections;

public class TriggerSpawnPoint : MonoBehaviour {

       void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player")
        {
            //Set the players new spawn point to this one
            other.GetComponent<MovingPlayer>().SetRespawn(transform.position);
        }
    }
}
```

Final Year Project Report yq009086_Nathan_Brown_Puzzle-based game using an open source game engine

GRADEMARK REPORT

FINAL GRADE

/100

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64

PAGE 65

PAGE 66

PAGE 67

PAGE 68

PAGE 69

PAGE 70

PAGE 71

PAGE 72

PAGE 73

PAGE 74

PAGE 75

PAGE 76

PAGE 77

PAGE 78

PAGE 79

PAGE 80

PAGE 81

PAGE 82

PAGE 83

PAGE 84

PAGE 85

PAGE 86

PAGE 87

PAGE 88

PAGE 89

PAGE 90

PAGE 91

PAGE 92

PAGE 93

PAGE 94

PAGE 95

PAGE 96

PAGE 97

PAGE 98

PAGE 99

PAGE 100

PAGE 101

PAGE 102

PAGE 103

PAGE 104

PAGE 105

PAGE 106

PAGE 107

PAGE 108

PAGE 109

PAGE 110

PAGE 111