2017
2018

Programming Project

DOCUMENTATION

AMARKUMAR PATEL
CANDIDATE NUMBER: 5132

OCR A LEVEL COMPUTER SCIENCE

Contents

Bibliography

Fixing test reference 1.3.5	https://forum.unity.com/threads/why-does-my-character-not-stop-moving-immediately-after-i-release-the-movement-buttons.250021/
Fixing test reference 2.2.2	https://forum.unity.com/threads/how-to-make-rigid-body-stop-moving-after-collision.88066/
Unity scripting reference	https://docs.unity3d.com/ScriptReference/
User Input	https://docs.unity3d.com/ScriptReference/Input.html
Understanding nested if statements	https://www.tutorialspoint.com/csharp/nested_if_statements_in_csharp.htm
Understanding arrays and iterating through	https://www.dotnetperls.com/loop-string-array
Set game object to active when condition is met	https://unity3d.com/learn/tutorials/topics/scripting/activating-gameobjects
Considering a solution to high score issue	https://support.unity3d.com/hc/en-us/articles/115000341143-How-do-I-read-and-write-data-from-a-text-file-

1.0 Analysis

1.1 Problem Identification

Today, more and more of the younger generation are being exposed to the technological advances, which are causing changes in lifestyle. This rapid acceleration in technology today has led to a change in the way the younger generation learn. As a result, recently there has been an increase in the number of educational games, however there are not many new generation games with simple controls that focus on developing mental (brain) skills such as concentration, visual perception and reaction time. The development of these skills is important to the younger generation because they are key mental skills that will be used throughout life. The skills are highly used when for example playing sports or in general during physical activities as they progress through life.

Consequently, there is a need in the market to create games with simple controls that focus on developing skills (as mentioned above) in an entertaining way. The development of a game would be the best solution because the younger generation need to be encouraged to develop their mental skills from an early stage in their lifestyle. The use of an entertaining game would mean that my stakeholders would want to keep playing the game meaning that that they would constantly be training their brain skills. Temple Run, Subway Surfers and Sonic Dash are a new genre of platform games called endless runners which do develop skills, however these are example of games that have a very strong emphasis on an entertaining factor rather than a skill developing (learning) factor, therefore players' minds are being distracted from development of the important skills. My game that I wish to create will be entertaining however it will not distract players from developing the skills that it will focus on developing.

1.2 Use of computational Methods

Computational methods allow the problem to be solved as other methods would not be as efficient. Additionally, other methods of development may not result in the same overall solution being developed. My solution will involve regularly generated object; therefore, the use of computational methods will allow this to be achieved without complication by an iterative process. Since the players view of the screen will need to be constantly updated computational methods makes this possible as these methods are very good at responding to user inputs and working through a set of instructions to decide on a certain output.

Not only are computational methods suitable for it sequential nature, they are also suitable due to their speed. As my solution relies on generating graphics throughout the game, computational methods have fast processing and calculation speeds. Fast processing speeds will improve the overall user experience as players will see smooth positioning of spawned obstacles in a virtual game environment. Also, fast processing speeds would also mean the player does not experience any delay between for example, the user inputs and the corresponding output or the generation of graphical objects. Computational methods are able to process and calculate in real time making my solution feasible by computational methods as calculation speeds, faster than humans are capable of, are required.

The use of a non-computational method would result in a different solution being developed which may not be successful and will not correspond to the solution I wish to develop for the stakeholders (6-16 year olds). Also, I believe that if a non-computational method was to be used the solution will most certainly be weak in areas such as iteration and response to user inputs. Existing solutions to the problem that I wish to develop a solution to, exist today and do not always make use of computational methods. For example, classic maze puzzles which make use of pen and paper to train mental skills are a big success. They are a success because every puzzle has a different solution and there are an infinitive different types of maze puzzles all which have their own twist.

Despite the success, of existing solutions that make use of non-computational methods a computational method to develop my solution outweighs non-computational methods because features such as changes in game playing speed, generation of graphical objects in a virtual game environment, response to user input are example of features of my solution which are impossible to achieve without the use of computational methods. Furthermore, the use of computational methods would allow a solution which appeals to my stakeholders to be developed.

1.3 Stakeholders

The stakeholders for the project will be 6 to 16 year olds. This age range will the stakeholders because the solution aims to develop mental skills such as concentration, visual perception and reaction time so the earlier they develop these skills the more beneficial it will be for their lifestyle. Children and teenagers (6 to 16 year olds) require a quick gaming experience due to their busy lifestyle therefore, my solution will meet their requirements because the objective of the game is relatively simple (navigate through a large number of cubes without colliding with any of them) therefore players will not have to read long instructions to play the game. Additionally, the levels will not be requiring too much time to get through. The controls (mainly just be the left and right arrow keys) make the solution appropriate, for my stakeholders due to how simple it will be to play the game and the controls also reinforce the ideas of the developing mental skills because players will be focused and concentrating on playing the games rather than mastering complex controls.

Another requirement of my stakeholders is that they require an entertaining game otherwise the game will not be replayed by my stakeholders. My solution will meet this requirement as the design of the game and other aspects such as visual effects will make the game entertaining. Furthermore, the objective of the game would reinforce the entertaining aspect due to how addictive it will be as the objective is very similar to popular games such as ‘Cubefield’ and ‘CrossyRoad’.

Furthermore, the stakeholders within this age range include different players such as competitive and casual players. Competitive players requirements will be meet as my solution will involve different levels that would increase in difficulty as they progress through them which adds to the competitive nature to the game. Causal players will just simply be looking to progress through levels. Casual player’s requirements will also be met as the games objective is simple to understand and the game itself has simple controls therefore there will be reduced demands on the time and skill required to learn how to play the game. I will be conducting further research to ensure that my solution appeals to my stakeholders that I have aimed my game at.

The game that I will develop will not just be suitable for 6 to 16 year olds, however a wider audience can play it, but 6 to 16 year olds are the main stakeholders that I will target for my project. I will conduct a survey to ensure that my game appeals to my stakeholders that I have targeted my game.

Overall, the stakeholders for my project will be 6 to 16 year olds and my game will need to meet their requirements. My solution will be appropriate due to the various reason mentioned in the previous paragraphs. The stakeholders will use the proposed solution to help them to learning and develop important mental skills that they will use outside of a game environment.

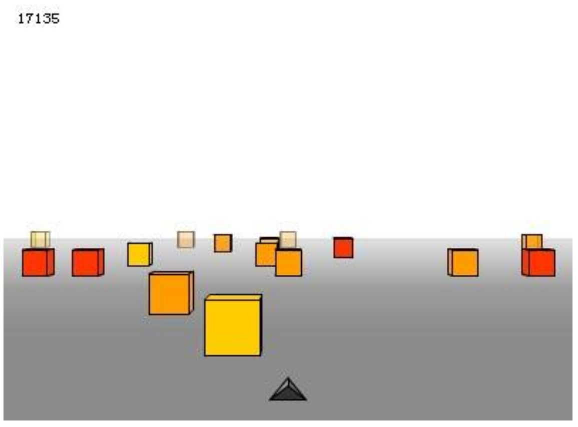
1.4 Research

1.4.1 Existing solutions (Games)

Cubefield

Based on the problem I have discovered and the gap in the market that I have found, I conducted research to find existing solutions. For my research I decided to look at different types of games that are played by, not only my stakeholders but also a wider audience. I also considered looking at games that are played across different devices because they include different features that PC and Online games do not always include. I have included a list of features from the existing game that interest me.

The first game I decided to research is Cubefield, which is an online 3D game developed by Yoacrde.



This existing solution involves the player, represented by a small grey arrow to navigate through the cube filed without stopping. The aim of this game is, to last as long as you can without colliding with any of the cubes. The player must go through a certain distance before progressing to the next stage (without stopping) where upon the speed increases. As soon as the player collides with a cube the game is over, and the player can either stop playing or restart from the beginning. Cubefield is a simple game to play and the only controls that are involved are the left and right arrow keys. They player can also move to the left and right as much as they want without any restriction. This game has no way of saving scores. It only saves the top score that players obtain. Despite the success of this game there are no other versions of the game. Overall, the aim of the game and it simplicity is what has inspired my solution.

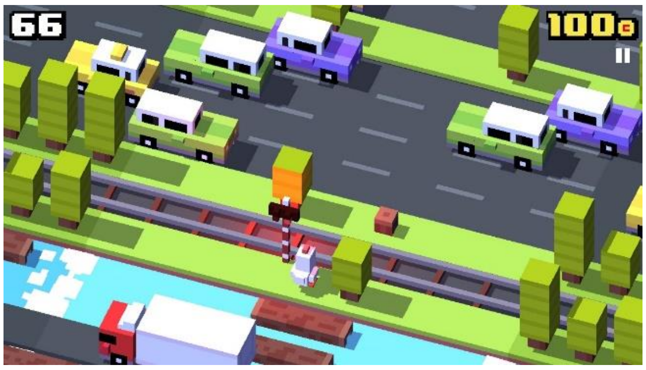
Feature of Cubefield:

- Player progression through the levels of the game without stopping
- Change in difficulty as the player progresses through the game
- Score depending on distance
- Saves only the players top score which is then seen on the start menu of the game
- Menu to start the game including the instructions and controls in the same section

Crossy Road

Crossy Road is a more modern existing solution to the type of game I wish to develop. Crossy Road is a 3D mobile game developed by Hipster Whale.

This existing solution involves the player, playing as a virtual character to make their way across busy roads, train tracks, streams and grass by moving in 4 possible directions which are forward, backward, left and right. As the players progresses across the obstacles and their score increases the speed of the game increases. Players have to constantly try to



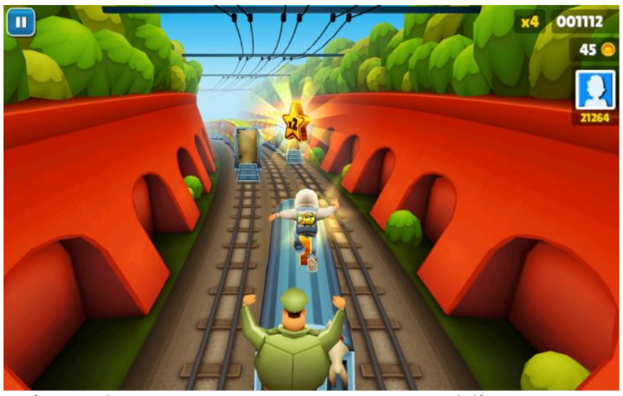
move forward, trying to avoid obstacles that could run them over. Also, players must avoid waiting to cross for a long period of time or move too fact back, otherwise the virtual character (the player) is taken by a giant bird of prey where upon the game ends. The aim of this game is to cross as many roads as possible without being hit by any cars or trains, drowning or being caught by the prey. The game also includes coins which the player can collect whilst playing the game. These coins can then be used to unlock new virtual characters (mascot) which players can play as. Players can also, get free gifts which general includes a coin prize.

Despite how entreating this game is, there is a lot to learn in this game due to the number of additional features and settings that players can change to create their perfect game experience. This game has a tutorial when the players first opens the game however, this tutorial cannot be accessed again and there is not instructions page provided.

Features

- High scores leader board
- Simple controls, however the game is played on a touchscreen
- Change in difficulty as the player progresses through the game – increase in speed
- Random procedurally generated object which are of different types
- Coin collection whist playing the game – player has to move over the coin to collect it
- Instruction tutorial to show how to play when it is payers first time playing the game

<u>Endless Runner Game</u>



Subway surfers is a 3D platform, mobile game developed by Kiloo and SYBO Games. Temple Run, is also another 3D platform, mobile game game developed Imangi Studios. Both of these games are a new genre of platform games called “endless runners” and are both very similar in their aim and game features.

Subway surfers involves the player trying to escape from the inspector and his dog by avoiding collisions with trains and other obstacles. Temple run, is very similar however the player has to escape a monster whilst avoiding tree roots and turning in junctions. In both games, the player is always running forwards with a chance to collect additional items for more points and power ups. In the picture above the “x2” power up can be seen, this mean that the players points are doubled if they collect it. As the player’s progresses through the game and their score increases, the speed of the game increases. Subway Surfers and Temple run both have touch screen controls however, they use 4 way motion (left, right, up and down) but the controls cause the virtual character to move in different ways. For example, if the player swipes up the character jumps, if the players swipes down they slide. The game also includes coins which the player can collect whilst avoiding other obstacles during their endless run. These coins can then be used to upgrade the power-ups and can be used to unlock new virtual characters which players can use to play the game.

This is an example of an existing solution with features I would not like to include in my game. The number of ways of controlling the character would not be used in my game as the original simplicity feeling would be lost. Also, the use of power-ups, the various graphical effects and that changing character convey a very strong emphasis on an entertaining factor rather than a skills developing factor, therefore players minds are being distracted from development of the important skills that my solution will aim to develop.

Features:

- Increase in speed of the game as the player moves a greater distance
- Random procedurally generated object which are of different types
- Coin collection whist playing the game – player has to move over the coin to collect it
- Score depends on distance
- Instruction tutorial to show how to play when it is payers first time playing the game
- Menu with simple user interface

1.4.2 Primary Research (Survey)

In order to understand my stakeholders and find out what they want I will be carrying out some primary data collection. The method of data collection that I will be using will be a survey. The survey will be conducted with mixed number of younger children from primary schools and teenagers from a secondary school. I will conduct my survey on 10 young children and 10 teenagers and will involve both genders. This will allow me to get wider opinion as my game does not target any certain gender. My survey would include questions that will allow me to identify my stakeholders gaming habits such as how often they play games and their preferred methods of playing games. It will also include questions that will influence my decision to add certain features to my overall game solution.

From my primary data collection, I predict to find that my stakeholders prefer method of playing games is PC because my stakeholders many not own their own personal handheld device. I also predict to find that players would want to keep a track of their high scores because this would be a factor that would influence their decisions to keep wanting to play the game, as they would want to try beating their previous score. Furthermore, I also predict to find that my players are looking for a game that involves individual levels, rather than an “endless runner” sort of game due to lifestyles that my stakeholders live as they will be looking for a quick gaming experience.

1.4.3 Survey Results

These are the results that I have processed into a tally having conducted my primary data collection:

For what duration do you play games in a week?

0	1 to 5 hours	6 to 10 hours	11 to 15 hours	15 + hours
0	5	11	2	2

What is you frequently used method of playing games?

PC	Games console	Mobile Phone/Tablet	Other
12	4	4	0

Would you prefer an “endless runner” game or a number of individual levels?

Endless runner	Individual levels
8	12

For a game with individual levels how many levels would you play?

1 to 5	6 to 10	11 to 15	15 +
2	8	9	1

For a game with individual levels, would you like the levels to be the same duration?

Yes	No
4	16

Would you prefer obstacles in games to be per-placed or randomly generated?

Pre-placed	Randomly Generated
9	11

As you progress through the game what changes would you like to see?

Change in background	Obstacles changed to different colours	Changes in game speed	No change to be made
3	3	14	0

What types of effect would you like to be seen in the game?

Destruction i.e screen shake	Interfere with the screen i.e obstacles that interfere view of screen view	The object being moved to disappear for a short period of time	No effect to be seen
10	10	0	0

Would you like to use the arrow keys or customise the keys used to play the game?

Use arrow keys	Customise keys
17	3

Would you like to see your high score displayed to you?

Yes	Don't mind
15	5

What is your preferred method of learning how to play a new game be?

Video Tutorial	Interactive tutorial	Reading instructions	Learning Yourself
10	7	0	3

1.4.4 Research Conclusion

Having researched existing solutions to my proposed problem and conducting a survey on a group of my stakeholder I have considered some potential features for my proposed computational solution. Firstly, from my research I have found that the most frequently used method of playing games was by using PC's. The game that I wish to develop will be a PC based game because my stakeholders will not only just have access to a PC to play the game, but the keyboard and the mouse will also be vital. The keyboard will be used to control the player controlled entity and the mouse will be used to control the menu.

From my research of existing solutions, I found that most of the games did have a menu which allow the player to interact with in order to start the game, access some sort of in-game shop, to view high scores and change other setting. The use of a menu would be important to allow players to interact with other elements rather than just the game.

Another game features that I found across most of the existing solutions was a change in game speed as they player progress through the game. This would be an important potential feature for my solution because not only does my survey show that my stakeholders want to see changes in speed, but it will further reinforce the aim of my solution. As my solution aims to develop certain brain skills, reaction times of players would truly be tested if there is change in game speed because players will need to adapt themselves to have faster a reaction time if they want to obtain a high score.

From my survey, I found that players would be looking for effects such as screen shakes (destruction) and obstacles that interfere the view of the players screen. These are features that are included in existing console games such as Mario Kart. Screen shakes (destruction) and obstacles that interfere the view of the players screen features are features that I would include in my game because not only do they appeal to my stakeholders, but they will also test their concentration. If they do not provide their full attention to the game and they come to experience one of these effects, then they could possibly collide with and obstacle where upon the game will end.

From the existing solution “Crossy Road” I found that the obstacles that players had to avoid were randomly placed each time the game was played. I have considered the feature of randomly placed obstacles as a part of my game because it means that players can play the game more than once as each time the player will not know what to expect and it add variation to the game. However, from my survey having obstacles pre-placed was also a popular feature, therefore I will consider both options. Some part of the game the players will experience pre-placed objects and for the rest the obstacles will be spawned randomly.

1.5 My solution

Overall, the game that I have decided to develop is a 3D PC game with a simple aim which is to move a player controlled entity in two directions and avoid a number of obstacles progressing through as many levels as possible. My solution will develop certain mental skills. Concentration and reaction time, skills will be developed by my game because player will have to move a player controlled entity through a number of randomly spawned cubes without colliding with any of them. Furthermore, my game will develop visual perception skills as players have to analyse what they see in advance in order to plan their next move.

The game will have levels which the player will be able to progress through. I have decided to use levels because when I conducted my survey the majority of my stakeholders preferred individual levels rather than an endless runner game. Also, the use of levels suits different types of player within the age range I wish to aim my game at. For casual players, game with levels are sometimes the best option because they will not spend much time playing due to other commitments that they will have, however they would prefer to play a few levels of the game for entertainment rather than play a long game that does not finish. Also, for competitive players levels are also appropriate because the different levels would increase in difficulty by increasing the speed as they progress through them which adds to the competitive nature to the game

Randomly placed obstacles will be included within my game because it creates variation as it means that the levels can be played over and over again as players will not know what to expect. Some obstacles will be pre-placed because at the beginning of the game pre-placed obstacles will allow players to prepare their mind set for playing the game. Some part of the game the players will experience pre-placed objects and for the rest the obstacles will be randomly spawned.

The scoring system that I will use for my game would be based on the distance that the player reaches across the different levels. As soon as the player collides with an obstacle such as a cube the game will end where upon the score will be displayed to the player. From my survey most of my stakeholders wanted their high score to be displayed to them therefore, a scoreboard will be a feature of my game that will keep a record of the top five scores obtained by the player. The scoreboard will be used to allow the players to track their progress through the game.

Simple controls will be used in my game. The left and right arrow keys on the keyboard will be used by players to move the entity. From my survey, I found that a majority of my stakeholders would prefer to use the arrow keys therefore there will not be any need to create an option to allow players to customise the controls they use to play the game, like other game do today.

Effects such as screen shakes (destruction) and obstacles that interfere the view of the players screen will be include in my game. These effects will be displayed/experienced by the player if they move into a certain object in the path. The object that will cause these effects will be decided upon the design stage.

A menu screen will be the first screen that players will see before they start to play the game. From the menu screen players will be able to access other elements rather than just going into playing the game. They will be able to access a tutorial on how to play the game, view the scoreboard and be able to start to play the game from the menu. The menu screen will need to be designed with a graphical interface not only just to make sure that it easy to use but to make the game engaging.

From the menu of my game I will include a “How to play” option to help players understand how to play the game. From my survey that I conducted, the instructions on how to play the game will be deliver to players in a video tutorial. Therefore, I will need to embed a video tutorial into my game which is accessible from the menu. I also personally think that a video tutorial is the best way to play the game because if the player ever does forget how to play the game they can always go back and watch the tutorial again.

Summary of solution;

- Levels will be included to allow the player to progress through, however the changes in speed will be made each level
- Obstacles will be both pre-placed and randomly placed
- Scoring system that saves the top five score will be included
- Simple controls to control the entity will be used
- Effects such as screen shakes (destruction) and obstacles that interfere the view of the players screen will be included
- Menu screen with a graphical use interface will be included
- “How to play” video tutorial will be included and will be accessible from the menu

1.6 Risks

1.6.1 Limitations

Firstly, a major limitation will be time management. Due to the number of features, that my solution includes I will need to make sure that I use the time that I have appropriately. The game, I wish to develop can include a number of features that would enhance the game. However, due to time constrains and the fact that I will be developing the solution myself I will use the results that I gather from my survey to consider features that need to be implemented into the game first (essential using a priority list to organise my features). Once high priority features have been included into my game, I would then consider the addition of other features and elements. Furthermore, I will need to consider time for not just development, but I will also have to allocate time to produce the documentation.

Another limitation of the proposed solution is the number of levels that I will be able to include in my game. As mentioned above, I will be developing the solution myself therefore, I cannot allocate task such as creation of more levels to other people. Due to this reason, I will not be creating any more than 6 to 10 levels. My survey that I conducted show that 8 people out of 20 would play 6 to 10 levels however, 9 out of 20 would play 11 to 15 levels. Even though more of my stakeholders would prefer 11 to 15 levels I will only be creating only 6 to 10 levels.

Development of features such as the process of generating the randomly placed obstacles that player have to avoid will be another limitation. The development of code for this part of the project will be challenging and could become complex. This part will be challenge due to how experienced I am with C#. I have written code in C# before however, I have not used all elements of the language therefore I will need to research some elements that could lead to an efficient code being written for the game. In addition, to help with development of features I will make sure that I am clear with what I am trying to achieve, and I will research how similar features have been developed and coded in other games. The support of examples and my understanding will majorly help with the development and coding the solution.

Furthermore, another limitation of my solution is the score saving system that I have decide to include. My scoring system will save the top five score that a single player has obtained. However, I am unable to create a network scoring system that allows other players to view each other scores on different PCs. This is a feature that available on many existing solutions such as Crossy Raod, Temple run, Subway Surfers. Due to how inexperienced I am with network gaming I will not be able include this feature as a part of my solution.

1.7 Requirements

1.7.1 User Hardware and Software

Hardware

Keyboard, mouse and Monitor

These basic peripherals which are required by all PC will allow player to interact with the game. The keyboard controls (mainly the left and right arrow keys) will be used by the player the move the player controlled entity in the game. The mouse will not necessarily be required when playing the game however, players will need to use to allow interaction with the menu when the game starts. The monitor will display the game to the player.

CPU - 1Ghz speed

The game will not require a very fast CPU to run due to the size of the game. The game will be of a small size and therefore, a CPU with a speed of 1Ghz will be suitable. My game might not involve the use of the complete speed of the CPU however, for hardware requirements it is important that I specify this speed because it is likely that the CPU will be carry out other operation on the PC. A speed of 1Ghz will be suitable as there will be no limitations for example, experiencing a delay between the user inputs and the corresponding output or the generation of graphical objects.

RAM - 1 GB RAM

No much RAM will be required in particular to run the game, however having more RAM than required is important because the computer that the game will be played on will have for example be running an operating system and other software, that will need to be kept up to date whilst the game is running. More than 2GB of RAM is not a requirement however more RAM would mean that players have a better gaming experience as the PC will not be slow.

Hard Disk – an additional 120 MB

The file size of my game once I have finished developing it will not particularly be very large therefore 120 MB of additional storage will be enough storage to allow the game to run. At the end of development, I will export my game as an executable file. An executables file will be the most suitable method for exporting and distributing my game because players will not need any special software to play the game. However, this would mean that players will require a windows PC to play the game.

GPU - Integrated with the CPU

Due to the simple nature of the game, and the lower graphical intensity within the game, a separate GPU will not be a requirement, however the CPU will need to have the ability to process graphics in real time.

Software

Operating system - Windows 2007

The PC that players will be using to play the will require on operating system which is windows 2007 or above. It is recommended that windows 2007 or above is used to avoid any errors when running the game. Furthermore, a windows 2007 operating system or above will be required as my game will be exported and distributed as an executable file. A windows operating system will be the most suitable to open the file.

1.7.3 Developer Hardware

Hardware

Keyboard, mouse and monitor

The peripherals will be needed to allow me as the developer to interact with the development software that I will be using which is unity and Visual studio. These peripherals will also be used by the players, therefore as I will be testing the game it is important that the peripherals that I use during testing have a similar specification to those that my players might use. For example, I will need to make sure that I test the game on a monitor that has resolution of 1920x1080 and possibly others such as 1366x768 and 1024x768 to make sure that the game functions as expected. Although not directly related to the game but another peripheral that I will use is a portable storage medium to store the files that I will create and work on during development.

CPU – 1.8Ghz speed

It is important that I use a CPU with sufficient speed that will allow me to run Visual Studio and Unity both together during development. Development of the game may not involve use of the complete speed of the CPU however, it is important I use a CPU with this speed or higher because the CPU is likely to be carrying out other operations in the background on the PC as well. A CPU of speed 1.8Ghz or higher is suitable as there will be no limitations such as a delay between other operations taking place or programs crashing resulting in losing work if it is not saved on a regular basis.

RAM – 1GB RAM

A minimum of 1GB of RAM will be need since RAM stores the programs, applications and data that are in use by the PC. This much RAM will be enough to allow Visual Studio along with Unity to run at the same time as well as running the operating system and other applications.

Hard Disk – 5GB or more

The system that I will be using to develop the game will need to have enough hard disk space so that I can store the software applications Unity and Visual Studio.

1.7.4 Developer software

The language that I will be using to develop my solution will be C#. I will be using C# not only due the experience I have with using the language, but it will allow features of the game to be implemented with ease. Also, another reason for using C# is because it is supported by the Unity game engine, which I will also be using.

The game engine that will use to develop my solution will be Unity. I have decided to use Unity because of how simple it is to use the Unity editor. This will be important because it will make development of the game simpler as the built in visual editor, will allow the project to be built and modified with ease and it will not require much learning.

Furthermore, another reason for using the Unity game editor is due to the ability to run the development of the game in the game window (game view). This will allow me as the developer, to test the game within Unity and preview how the game will work on target devices – those that will be used by my stakeholders to play the game. In addition, Unity allows the game to be run simultaneously while changing game properties (e.g. the speed that game object moves). This will be particularly useful when testing usability features of the game as I will be experiment how changes that are made have an impact on the game.

Microsoft Visual Studio will be the integrated development environment (IDE) that I will be using to develop my code. I have chosen this IDE rather than any other IDE because of how familiar I am with using it. Furthermore, it has a number of feature that will allow me to code the solution to the problem comfortably. The fact that visual studio supports programming in any language that the programmer selects is another reason the influenced why I will use Microsoft Visual Studio. I will be programming in C# as mentioned above. The IDE is free, however there is a paid version of the IDE which include furthermore features. For my project the free, Visual Studio Community version will be suitable due the relatively small scale of the project.

The features of the IDE and how I will use them, have been summarised in the table below.

Development of code	Programmers can write code efficiently with losing accuracy (i.e. identities variables that do not exist). As elements of the language are being used in the code, the IDE shows an explanation of what the elements will do. This will help understand elements of the language that I have not used before. The automatic indentation makes it is easy to

	see what lines of code belong to what functions and procedures.
Debug Tools	As the code is typed by the programmer, the IDE check for errors therefore, minimising the number of errors such as syntax errors to be fixed when the code is run for the first time. The speed that Visual Studio finds and suggests fixes to bugs will be another advantage.
Testing Tools	Visual studio built in testing tools will be used to execute testing. Visual studio has the ability to carry unit testing. This will be useful as the code can be broken down in testable parts as I develop. Also, the Visual studio has the ability to test each line of the code. This might be useful for my project.
Integration with Unity	As I also be using Unity to develop my game, Visual studio has the ability to develop with Unity. Visual studio will allow the scripts to be created and edited automatically because when you double click on a script Visual Studio will open.

1.8 Success Criteria

Reference No	Criteria	Reason	How criteria will be achieved
1.1	Game environment	Lay foundations of the project which will allow other features to be added	Platform/ground plane will be added to the game so the object that the player will be represented by can be added
1.2	Movement	Will allow the user to move the player controlled game object on a platform	Player is able to use the left and right arrow keys to move the entity on the platform
1.3	Player game view	Players should have a view of the game that will allow them to play it and see obstacles that they need to avoid	Camera in the game will be set to a suitable position to allow the game to be played by
2.1	Pre-placed obstacles	Pre-placed obstacles at the beginning of each level will allow the user to adjust to the level before experiencing randomly positioned obstacles	Will be placed on the platform when design the level, and same effect should be seen as 2.2 when players collides into these obstacles
2.2	Destructive Collisions	Allows the players to collide with object which then stops the player from moving and gives rise to an effect	Player controlled entity will be moved across to the destructive obstacle. Colliding into these types of obstacles will no longer allow player to move and

			should give rise to certain effect
2.3	End of game	Players must progress through the game without colliding with any of the obstacles where upon the game ends.	Player entity will be moved to avoid all obstacle and the game should end.
2.4	Randomly positioned obstacles	Randomly positioned obstacles will make the game more entertaining and will mean the user can play the game more than once as the path the players follow will be different each time	Will be randomly generated on the platform whilst players is playing the game. The same effect should be seen as 2.2 when players collides into these obstacles
2.5	Restricting player movement	Player must not be able tom move of the ground plane	Move player game object to the edge of the ground plane
3.1	Scoring system	A single score will be obtained by players, which will depend upon the distance travelled across the levels without colliding with the obstacles.	Play multiple levels of the game to ensure that the score from each level is added to get a final score.
3.2	Score update as player progress	The score will need to be displayed live to the user as they are playing the game	Play the game to make sure that the score for that particular level increases the further progression made. Check format and display
4.1	Levels	The player will progress through levels in the game.	Complete the first level of the game and then press the button to play the next level
4.2	Changes in speed as more levels complete	Gives variety to the game play and adds an entertaining element to the game.	Get to Level 5 and a change in game playing speed should be apparent
5.1	Main menu	Easy for player to navigate with other elements rather than just the game itself	Click the button on many different screens (i.e. when playing, looking at high scores) to return to the main menu
5.2	Navigation through menu (controls)	Graphical user interface to allow the user to play the game or see their top five high-scores	Move the mouse cursor to click on the option to allow interaction will the main menu
6.1	High-scores screen	Show players the scores, to beat which adds to the competitive nature of the game	When the high-score screen is accessed, score should appear in order of highest to lowest and possibly show what level the player reached.

6.2	Updates in high-score screen	High-score screen update to show five of the most recent top scores achieved.	Obtain a high-score that is not on the high-scores screen and then ensure it appears there in the correct order
7	Visual effects: screen shake	Screen shakes will disturb the player and will really test the player concentration.	Player controlled entity will be moved across to the destructive obstacle. Colliding into these types of obstacles will give rise to a certain effect.
8.1	Game notification when game ends	Once the players have collided with an obstacle the game will end showing a message to the player of the play can restart the game.	Collide with an obstacle where upon the game will end and display a message to the player. Press the restart button to play again
8.2	Game notification when level completed	Once the players have completed a level the game will end showing a message to the player and the next level will start	Complete a level. A message should show that the level has been completed. The next level should load.
9.1	Video tutorial	Tutorial will be created by myself which players can watch in order to learn how to play the game	A short quick tutorial showing how to play the game
9.2	Embedding video tutorial	Video will be embedded so player will not have to be redirected to an external link	Press the “how to play” button on the main menu. Action should results in a video file starting to play
10	Export game to and executable file	Executable file will be used as players will not need to install any special software to play the game	Run the executable file on windows 2007 and above. Checking the all aspects of the game successfully work
11	Testing the end solution	Test plans will be produced and used when testing aspects of the solution during development and when completely test the final solution.	Thorough testing will be carried out on my solution to ensure it contains no bugs and that my solution full work.

2.0 1st Sprint

2.1 Design

The first part of the solution that I will be designing will involve the game itself. My game will have around six or more levels, however my first few sprints will primarily focus on level one. This will allow me to focus on the main concepts of the game. Once I have developed the first level and have carried out testing, I will be able to able to adapt the first level to create further levels. The aim of the first sprint is to establish basic game features. To do this, I have broken down the sprint into individual subtasks:

1. Setup game environment
2. Create the player control entity (player controlled game object)
3. Add control to the player game object
4. In scene camera set-up

I have decided to break down the sprint into the following subtasks and will carry out these tasks in the corresponding order because the outcome of one task is dependent upon the previous task. For example, the player controlled game object cannot be added without the setup of the game environment. Furthermore, the break down will enable me to focus on the player game object because the speed, the directions that it moves and the way the object moves are all important aspects which will determine the development of other features such as how the camera in the game scene follows the game object.

Before I start to program any of my solution, I will need to plan algorithms as they will provide step by step instructions on how-to solve the problem. The use of algorithms will make sure that I am not just solving the problem, but it will allow me to plan an efficient solution to the problem. I will use algorithms to plan any code that I will develop within each sprint. The algorithms will solve individual problems and then once coded they will be implemented into the final solution.

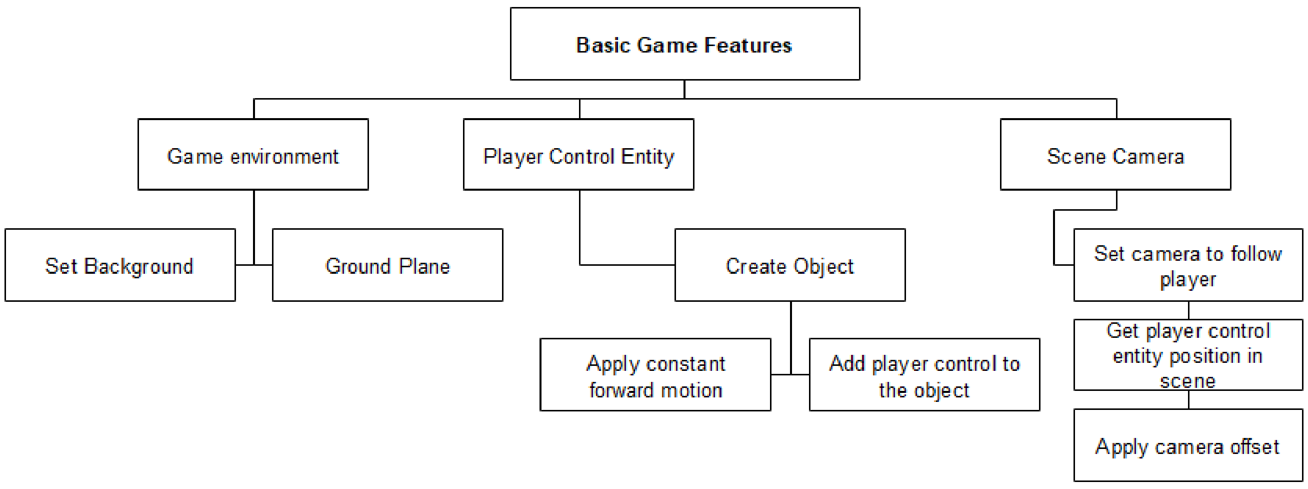

Basic Game Features
Game environment
Player Control Entity
Scene Camera
Set Background
Ground Plane
Create Object
Set camera to follow player
Get player control entity position in scene
Apply camera offset
Apply constant forward motion
Add player control to the object

Figure 1: Top-down design for Sprint 1

2.1.1 Setup game environment

To develop the game, I will be using unity. The setup of the game environment will involve creating a ground plane which the player controlled game object will be based upon. The player of the game will control the object across the ground plane. I will also add a solid colour background to the scene.

To create the ground plane, I will use a cube game object. The dimensions that I will use for the ground plane will be (15, 1, 100) on the (x, y, z) axis respectively for now (these dimensions will change in later sprint). These specific dimensions will be used because the x axis determines the width of the ground plane. My game will restrict how much players can move in the x direction, therefore by having the ground plane to a specific width would enforce this rule. The dimensions in the z axis determine the length of the ground plane. The length of the ground plane will indicate how long it will take players to complete the level.

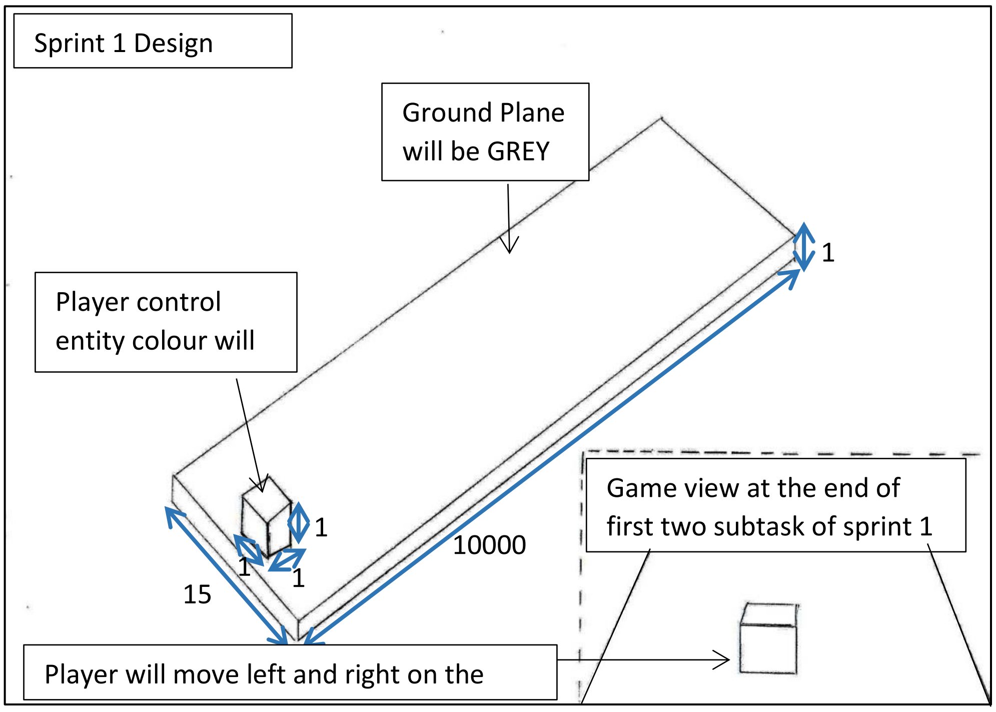
The colour of the ground plane will be grey. An alternative option would be to use an image that will wrap around the ground plane however this will distract the players attention and will not look as appealing as using a solid colour. The grey colour will be applied to the ground plane because players will view ground plane throughout, the game. Due to it being a solid colour, it will not cause any visual problems for the players and they will be able to focus on the aim of the game.

2.1.2 Creating the Player controlled game object

The player control entity is the object that players will use to navigate through the obstacles to play the game. In my game the player object will be a cube – the player will be represented by a cube. The player object will move across the ground plane therefore it will need to be positioned in the game scene to allow it to do so.

The cube dimensions will be (1, 1, 1) on the (x, y, z) axis respectively. These dimensions will be applied because the cube will not only need to be big enough for the player to see however, it will need to of a reasonable size. This is because players will be moving the cube to avoid obstacles therefore the cube will need to be able to move through the gaps of the obstacles.

The colour of the player object will be blue. I have chosen this colour for the player object as players will be concentrating on this object to move it, avoiding obstacles. This colour will stand out from the obstacles that players must avoid therefore making it visual clear to players their position in the game.



Sprint 1 Design
Ground Plane will be GREY
Player control entity colour will
1
1
1
1
15
10000
Game view at the end of first two subtask of sprint 1
Player will move left and right on the

Figure 2: This diagram shows what my game will look like once the first two subtask of this sprint have been completed. The game view shows what I expect the players to see.

2.1.3 Add control to the player controlled game object

In my game, the player controlled game object will move across the ground plane. The object will be controlled by the player using the left or right arrow keys only. This would mean that I will need to only need to consider movement of the player game object to move along the x-axis.

To add control to the player game object I will need to use physics within unity. I will need to use a Rigidbody component as this will allow objects to have physics properties such as gravity, drag, velocity, etc. Furthermore, the player controlled game object must have a collider attached to it because it will allow interaction with other physics objects in the game (such as obstacles).

To move the object in the scene, the player input will be taken from the keyboard and the input will be applied the to the player game object as a force to the Rigidbody. The algorithm below shows how I will add control to the player control object:

```
private rigidbody rb
public float speed
public float forceForward = 2000
function start ()
    rb = RETURN reference to rigidbody
end function
function fixed update ()
    APPLY forceForward TO rb
    moveAlongX == INPUT direction of movement
    vector 3 moveObject = NEW vector 3 (x, y, z)
    if moveAlongX == right_key THEN
        AddForce to rb to move right
    else if moveAlongX == left_key THEN
        AddForce to rb to move left
    endif
end function
```

In the algorithm private rigidbody, rb is the variable that holds the reference to the Rigidbody. The reference is set in the start function meaning that when the code in the start function is executed it will return a reference to the attached Rigidbody. This is done in the start function because code in this function is called on the first frame the script is enabled and is only called once.

The fixed update function is used to deal with the control of the player game object. Since the player game object will have a Rigidbody attached to it, when applying a force to, it will need to apply the force every fixed frame. In the fixed update function, a forward force is being applied to Rigidbody attached to the player game object to keep it moving on the z axis. The code is in this function because the forward force will be applied throughout the game and not just when the player first starts the game.

Users will be able to move the player game object by either pressing the left or right arrow keys on the keyboard to move it on the x axis. The above algorithm shows that if either key is pressed, then the input is applied as a force to the Rigidbody in the horizontal direction. Even if I am only considering movement in one direction, the user input is being converted into a vector3 variable to allow the force to be applied to the Rigidbody. This is because if in the future the game wants to be development further to allow movement in the other directions the code can be edited with ease to allow it to either move on the y axis (this will allow the player game object to jump) or move in the z axis under the player controls.

To allow the player game object to move faster a variable called speed will be used. This variable will be declared as a public float, as this will allow it to be set accurately in the inspector within unity. I currently do not know specifically what speed will be applied to the object, so the ability to edit it from the inspector will save me from having to open the script each time to change the speed.

Variable Name	Type	Explanation
rb	Object - Rigidbody	Variable will store the reference to the Rigidbody. The name rb is used for this variable as it is just short for Rigidbody. Variable will allow the physics properties of the object to be changed.
speed	float	Speed value which the player game object will move. Declared as a float because the speed will need to be set to an accurate speed that players are able to handle.
forceForward	float	Variable will store the magnitude of the force that will be constantly applied to the Rigidbody of the player game object to move it in along the z-axis. Variable will be public to allow it to be edited in the inspector.
moveAlongX	float	Stores the keyboard input which will be used to add forces. The variable will store float values within the range of -1 and 1.
moveObject	Vector 3	Converts the values of the moveAlongX into a vector 3 value. It will store values that will determine the direction of the force that will be applied to the player game object.

2.1.4 Setting camera to follow player controlled game object in scene

Currently, in the scene I only have one camera. I will only be using one camera, and I will need to set this camera to follow the player controlled game object. The player controlled game object can move across the ground plane however, since the game view is rendered from the cameras in the scene, if the camera stays in a fixed position the players will not be able to see where they are moving.

Due to the nature of the game I am designing, it is suitable for the camera to be set to follow the player controlled game object since this will be what the player will be controlling. This will be a suitable solution to the problem because players will not be concerned with previous events in the game. However, when setting the camera to follow the player controlled game object it is important that it is positioned correctly so that players can see for example, the object they are controlling and other obstacles within the game.

There are two possible ways in which the camera in the scene can be set to follow the player controlled game object. One way would be to parent the camera to the player controlled game object meaning that whenever the player controlled game object moves the camera will move with it. Another way, would be to use the player controlled game objects position and set the camera follow it. The first solution is not every appropriate because if the player controlled game object was to collide with obstacles and cause rotation then the camera will also rotate. This will look confusing to the players. The camera will be setup so that it only follows the position of the player game object. This will be achieved by adding a C# script to the camera.

This is the following algorithm that will enable me to solve this problem:

```
public transform playerTransform
public vector 3 offset
function update ()
        position of camera = position of player object + offset
end function
```

In the algorithm, the playerTransform variable will be created to store a reference to the player controlled game objects position in the scene. The variable has been declared as transform as it allows information such as the position of the player contolled game object to be stored. The variable offset is being used to set how far the camera will be away from player game object. It has been declared as a vector3, as it will allow the position of the camera in the x, y and z directions to be set in the unity inspector.

The update function will be used to allow the camera to be set in the correct position every frame. This algorithm solves the problem of setting the camera to follow the player game object in scene. This is because it will refer to the position of the camera in the scene and it will be set to the position of the player game object with the offset being added to it. The offset will be added to it, so the camera follows the centre of the player controlled game object but also enables the player to have a clear view of the player controlled game object in the scene.

Variable Name	Type	Explanation
playerTransform	Object	Will store reference to the player controlled game objects position in the scene. Transform is a reference data type variable. It is being used because it allows the position (in the x, y and z direction) of the player game object in the scene to be stored.
offset	Vector3	Stores how far the camera will be away from player game object in the x, y and z directions. Will be declared as public to allow the values to be set in the unity inspector as I am unsure on what specific position the camera should be placed in the scene.

2.1.5 Testing

Testing for each sprint will be planned before I start any development. To test my game, I will carry out different types of testing which include:

- Black box testing – involves taking a set of inputs which include valid, invalid and extreme input date entering them into the system and the actual results is compared with the expected results
- White box testing – involves testing all possible paths through the code. Each path must be tested to be sure that the code runs as expected.

The table below describes and explains the different types of testing that will be carried out. Each type of testing will be carried out either using either the Black Box testing or white box testing method as described above.

Type of testing		Description
Unit testing	Performed using the White Box testing method	Unit testing will be carried out before any other form of testing is carried out. Unit testing will ensure that each component or individual units are tested to make sure that they perform as expected. Unit testing is generally the first type of software testing that is carried out therefore it will be carried once an individual unit, program, feature of the game or function is coded.
Integration testing	Performed either using Black Box or White Box testing	Integration testing will be carried out to ensure that individual units and added features are tested when they are combined to be a part of the final program. I will carry out integration testing to uncover errors and unexpected functionality between integrated units. Integration testing will be carried out once individual units have been developed and then combined.
Regression testing	Performed either using Black Box or White Box testing	Regression testing is carried out once a modification has been. Regression testing will be carried out to ensure that the modification made have no impact on other modules of code or it does not have an overall impact on the game functionality. Regression testing will be carried out once an error has been fixed, to make sure the other features of the game still work.
System testing	Performed using the Black Box testing method	System testing will be carried out to ensure that by running the final executable version of the game in different environments (such as a different operating system or monitor resolutions) the game still works as expected. System testing will be carried out once the game has been completely developed.
Usability testing **Acceptance testing**	Performed using the Black testing method	Usability testing will be carried to ensure that the features of the game that I develop are suitable for my stakeholders. It will enable evaluation of the game since the final users (stakeholders) will be testing the game. Usability testing will be carried out when it will be suitable to cerate a complied version of the game to give to my stakeholders to test.

To test my game, I will list test questions that will need to be tested. Specifically, the test questions will be identified as either:

- Verification test questions – these make sure that the features fulfil their intended purpose (these will ensure functionality test is being carried out)

- Experimental test questions – these try the features of the game to make sure that they work with specific cases

For the features of the game that I am testing, I will record the following: feature undergoing testing, type of testing, test question, test question type, cases to test, expected output, actual output and any bugs. For ease of maintenance and presentation, I will record test data in a spreadsheet. Each test will be assigned a test reference number which will be used when referring to it in this documentation. All test will be carried out as they appear in the table, however any experiment tests will be carried out at the end of the sprint.

When testing any form of user input, cases to test will involve valid, invalid and extreme input data because this would make sure that the game does not crash or not function as expected due to an incorrect user input. Each test case will have an expected output, the result of the input being tested will be the actual output. If the expected and actual output are not the same then this will, highlight an error which I will need to fix before moving onto the next subtask/sprint.

As I am using Unity to develop my game, I will test my game using the game view. Using the game view to test the game rather than exporting the game each sprint is suitable. This is because the game view is rendered from the cameras in the game and it represents the final published game at each stage.

At the end of each sprint, I will create a complied version of the game at the point that is in development. These complied versions will be given to my stakeholders for them to review the game and provide me with feedback. Along with the complied versions of the game, I will give my stakeholders a text document which will explain the porotype and will include specific questions about features of the game that have been implemented.

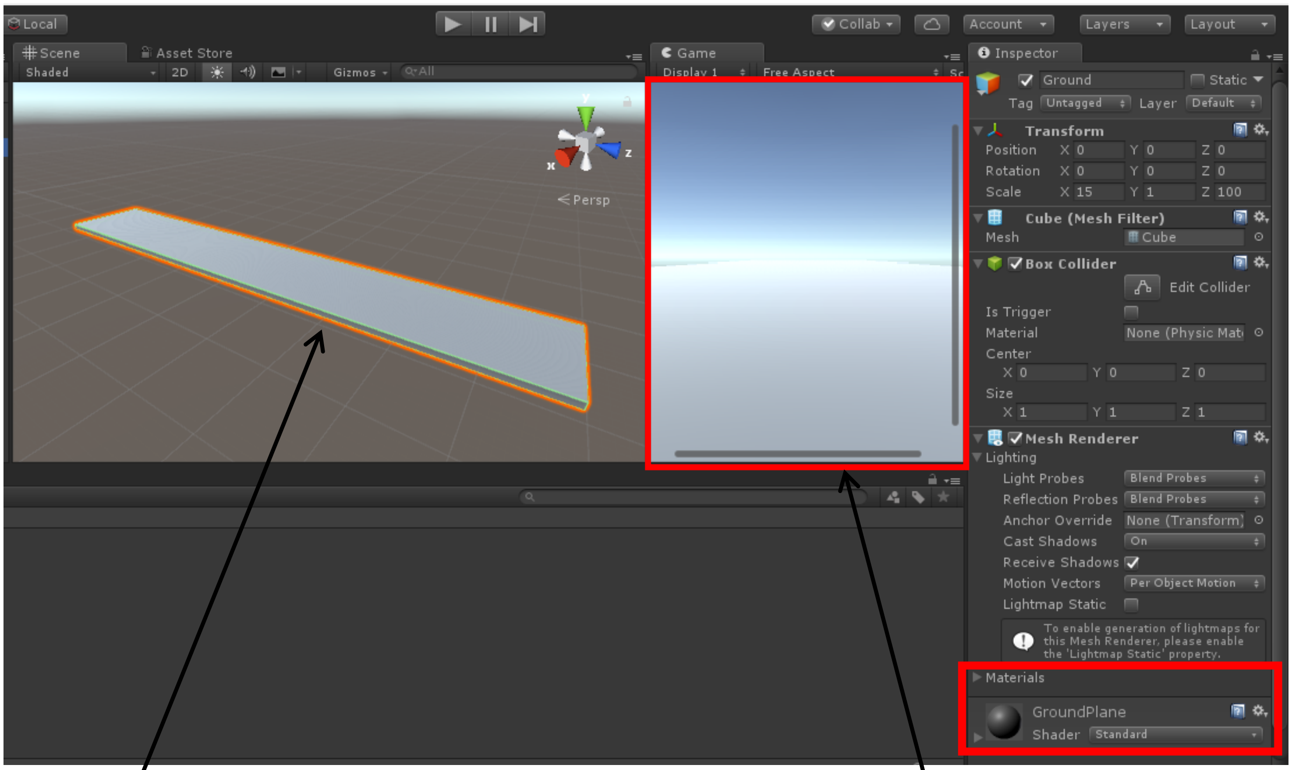
A small group (essentially a focus group) of around 5 people will be select to carry out usability testing and to test the complied version of the game throughout development. These five people will represent the stakeholders for the project.

(In this documentation where a test table has been mentioned refer to appendix A)

2.2 Development

2.1.1 Setting up Game Environment

I will start by creating the ground plane in my game scene. I will create a cube and scale it to size that I have specified in the design section. I will then also apply a grey colour to this ground plane.



Ground — Static
Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	15	1	100

Cube (Mesh Filter)
Mesh: Cube

Box Collider
Edit Collider
Is Trigger: ☐
Material: None (Physic Mat
Center: X 0, Y 0, Z 0
Size: X 1, Y 1, Z 1

Mesh Renderer
Lighting
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Cast Shadows: On
Receive Shadows: ✓
Motion Vectors: Per Object Motion
Lightmap Static: ☐

To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.

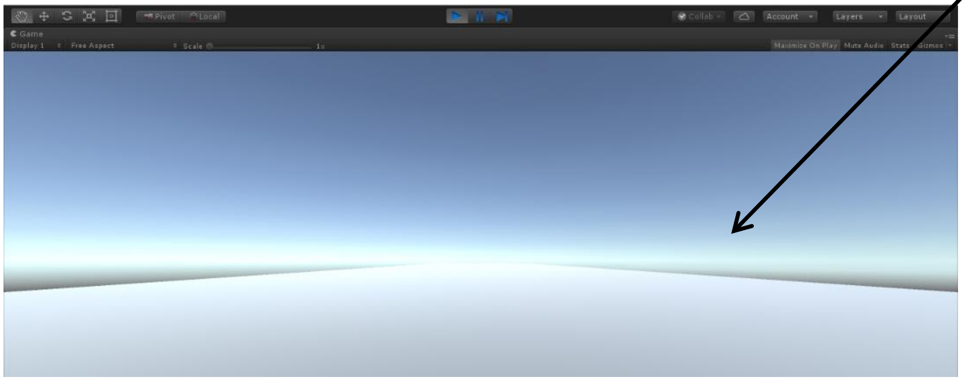
Materials
GroundPlane
Shader: Standard

This is the ground plane that the player object will be placed on. The dimensions of the ground plane have been set to (15, 1, 100) on the (x, y, z) axis respectively. To apply the colour, to the ground plane a material is being used. The only property of the material that has been changed is the colour to grey.

This is the game view. This view is rendered from the cameras that are in the scene. Currently, this is the view that the game player can see. Once I have added the player control entity I will have more of an understanding of where to position the camera.

I have used a grey colour for the ground to plane because players will view the ground plane throughout the game therefore, it will not cause and visual problems.

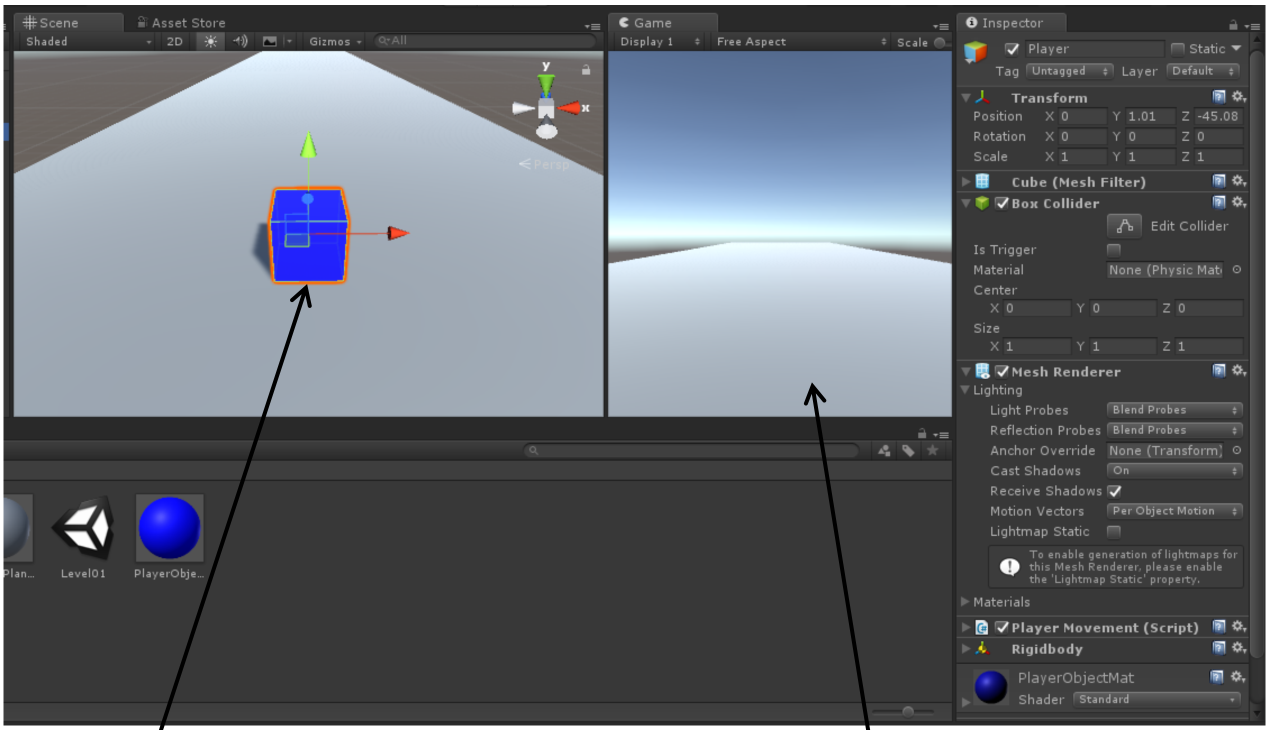
Test reference 1.1



In game view, this is the view that players will see. I think that is enough evidence to show that test succeed since players can see the ground plane which is rendered from the camera that is in the scene.

2.1.2 Adding the player controlled game object

The player control entity will be a cube meaning that the player will be represented by a cube in the game. The player control entity will be positioned on the ground plane so that players can move it to the left or right.

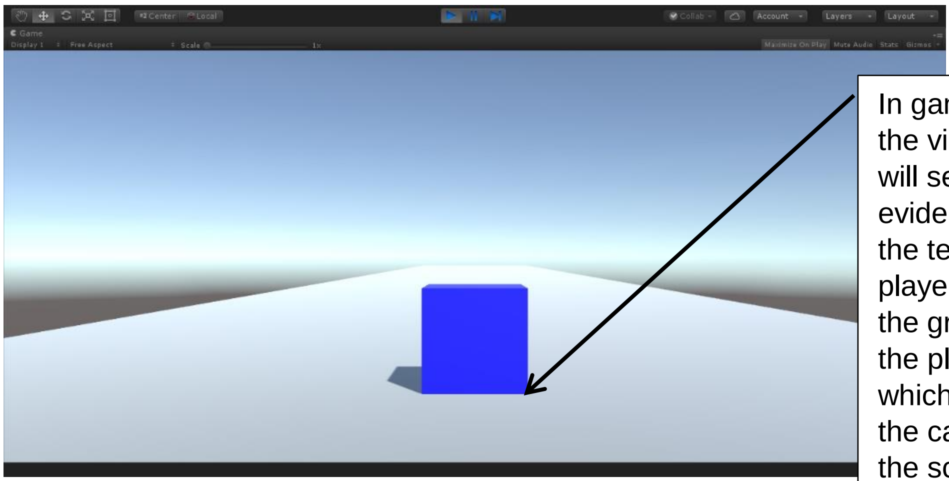


```
Inspector
Player                          Static
Tag  Untagged      Layer  Default
Transform
Position   X 0      Y 1.01    Z -45.08
Rotation   X 0      Y 0       Z 0
Scale      X 1      Y 1       Z 1
Cube (Mesh Filter)
Box Collider
                    Edit Collider
Is Trigger
Material            None (Physic Mat)
Center
  X 0          Y 0          Z 0
Size
  X 1          Y 1          Z 1
Mesh Renderer
Lighting
  Light Probes        Blend Probes
  Reflection Probes   Blend Probes
  Anchor Override     None (Transform)
  Cast Shadows        On
  Receive Shadows     ✓
  Motion Vectors      Per Object Motion
  Lightmap Static
  To enable generation of lightmaps for
  this Mesh Renderer, please enable
  the 'Lightmap Static' property.
Materials
Player Movement (Script)
Rigidbody
PlayerObjectMat
Shader  Standard
```

This cube is the player control entity – it represents the player. The dimensions of the cube are (1,1,1) on the (x, y, z) axis respectively. To apply the colour, to the cube I have used a material. The only property of the material that I have changed is the colour to blue.

Currently, the player control entity cannot be seen in the game view. This is because I need to re-position the camera in the scene to allow the cube to be seen. Before moving onto the next sub-task of this sprint, I will temporarily position the camera, so I can see the object in the game view. However, later the position of the camera in the scene will need to be focused

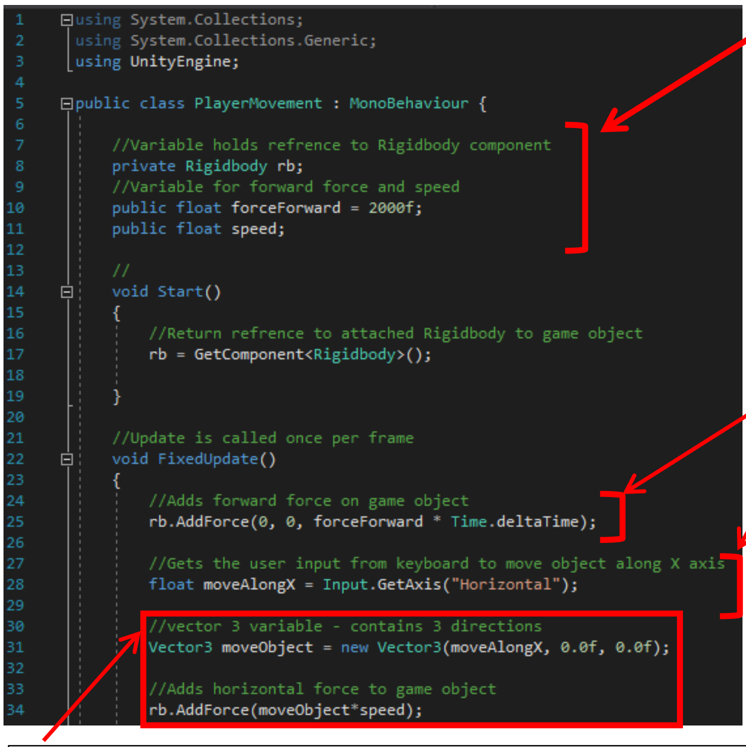
Test reference 1.2



In game view, this is the view that players will see. This is enough evidence to show that the test succeed since players can see both the ground plane and the player control entity which is rendered from the camera that is in the scene.

2.1.3 Adding control to the player controlled game object

To allow my player object to interact with the Unity's physics engine I will need to add a Rigidbody component to game object (the cube which players will control). Then to add control to the player control entity I will need to create a C# script and attach this to the game object.



```csharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerMovement : MonoBehaviour {
6
7      //Variable holds refrence to Rigidbody component
8      private Rigidbody rb;
9      //Variable for forward force and speed
10     public float forceForward = 2000f;
11     public float speed;
12
13     //
14     void Start()
15     {
16         //Return refrence to attached Rigidbody to game object
17         rb = GetComponent<Rigidbody>();
18
19     }
20
21     //Update is called once per frame
22     void FixedUpdate()
23     {
24         //Adds forward force on game object
25         rb.AddForce(0, 0, forceForward * Time.deltaTime);
26
27         //Gets the user input from keyboard to move object along X axis
28         float moveAlongX = Input.GetAxis("Horizontal");
29
30         //vector 3 variable - contains 3 directions
31         Vector3 moveObject = new Vector3(moveAlongX, 0.0f, 0.0f);
32
33         //Adds horizontal force to game object
34         rb.AddForce(moveObject*speed);
```

Following the algorithm for adding control to the player controlled game object, I started by initialising a variable for the constant forwards force that will be applied to the cube. Variables for the speed the cube will move and reference to the Rigidbody were declared.

A constant forward force is being applied to the Rigidbody of the player game object to move it on the z axis

Rather than using two variables to hold which key the players has pressed to move I have used input.GetAxis as it will return a float value in the range between -1 and 1. This is more efficient as players will only be controlling movement on a single axis.

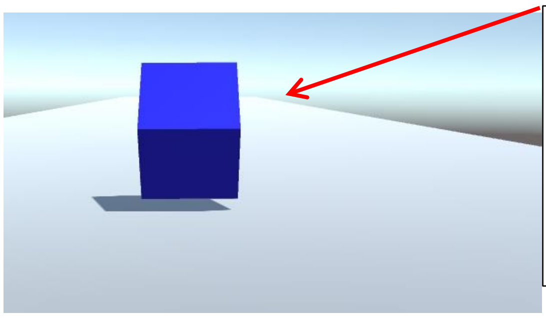
The variable moveObject is being used to get the float value moveAlongX into a vector 3 variable which will allow the force in the correct direction to be added to the Rigidbody. This will allow the player controlled game object to move on the x axis.

Having followed the algorithm that was designed in the design section 2.1.3, and the screenshot above shows the code that has been developed to add control the player control game object.

Line 28 shows the code that is used to get the user input. The method GetAxis returns a float value between -1 and 1 hence why the user input is being stored as a float value. It is important that the value returned by the method is stored with precision because the value will represent the game objects position on the horizontal axis. The string horizontal is being passed to the method because the program only needs to know how if the user wants to move left or right along the horizontal axis. I have decided to use the GetAxis method to get user input because users can either use the left and right or the a and d keys combination to move. For the game object to move along the axis, the user input is being applied as a force in the horizontal direction.

An alternative option would have been to use input.GetKey, which would have returned true if the left key or the right key was pressed. However I have used input.GetAxis because when moving in the left direction it returns a negative float value. This means that I do not need to reverse the direction that the force is being applied on the x axis as the negative float value applies this.

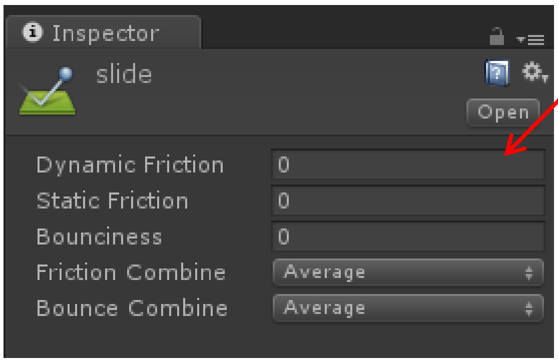
Test reference 1.3.1



When the left arrow key is pressed the player game object does move to the left, however it does not stay on the ground plane when it moves. I suspect that this is due the physics therefore I will fix this before carrying out any further test.

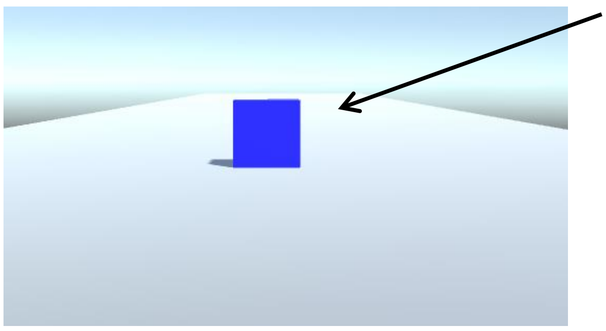
The above bugs that I have found is that the player game object does not stay on the ground plane when it is moved using the left arrow or right arrow keys. I suspect that this is due to physics. Between the game object and ground plane there is friction. To fix this bug I could add some constraints which would freeze rotation on the x axis however, this would cause no visual effect when the player game object collides with obstacles that will be added in a later sprint. An alternative and better fix would be to create a physics material and change the friction properties to zero. The physics material will be applied to the ground.

Bug fix – test reference 1.3.1 – 1.3.2.1



Dynamic Friction	0
Static Friction	0
Bounciness	0
Friction Combine	Average
Bounce Combine	Average

The values for both dynamic and static friction were set by default to 0.6. By setting dynamic friction to zero, the player game object should move smoothly across the ground plane. Setting static friction to 0 will mean that the player game object will move smoothly across the ground plane when the object is still (i.e. when the player first start the game or when they players decides they don’t want to move in the x direction).



When I press the left arrow key the player game object does move to the left smoothly, as there now is no friction between the ground plane and the player game object. Now when the left arrow key or the “A” key is pressed the player game object does move to the left. Also, when the right arrow key or “D” key is pressed the object moves to the right

Test reference 1.3.3 to 1.3.5

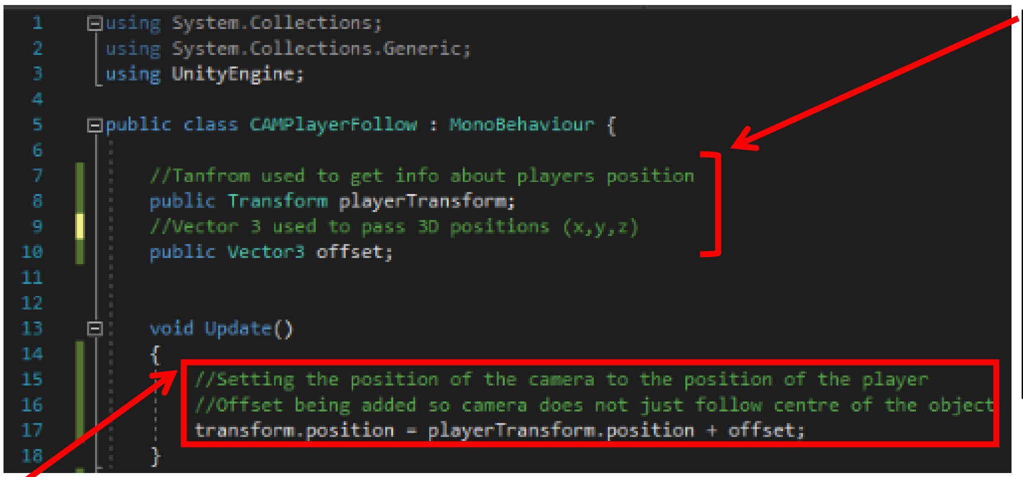
Having performed both test cases specified, they succeeded. The player game object only moves forward if either the left/right arrow keys are pressed, or the A/D keys are pressed. It is important that this test is carried out because if a user does press the wrong key to control the object then the game will not crash. Test reference 1.3.4 tests if players use a combination of using the left/right keys and the A/D letters. For example, if the player attempts to press the D key and the left arrow key, then the player game object continues this move forward as expected.

Test reference 1.3.5 has not succeed since the player game object continues to slide in the direction the players decide to move in. I will fix the bug in the once I have setup the camera in the scene to follow the player game object. I could fix this by changing the amount of forwards force applied, and slowing the speed down however, currently I do not know if these factors are causing the bug. Therefore, setting the camera will allow me to observe the motion of the player game object in game view when players try to control it over a greater period of motion.

2.1.4 Setting camera to follow player controlled game object in scene

Having considered two possible solutions for this problem, I have decided to use the player controlled game objects position and set the camera to follow it. To achieve this, I will need to create a C# script and add this as a component to the camera in the scene.

Having followed the algorithm that was planned in the design of this sprint, the screenshot below shows the code that was developed to set the camera in the scene to follow the player controlled game object.



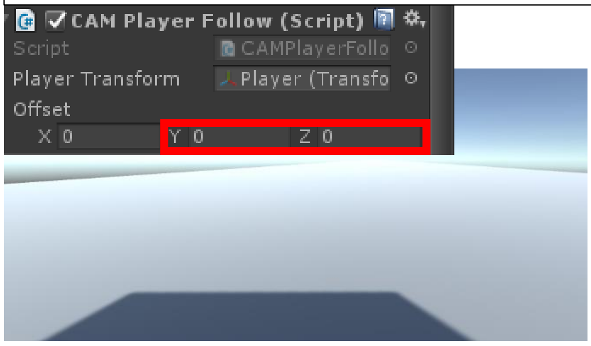
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CAMPlayerFollow : MonoBehaviour {

    //Tanfrom used to get info about players position
    public Transform playerTransform;
    //Vector 3 used to pass 3D positions (x,y,z)
    public Vector3 offset;


    void Update()
    {
        //Setting the position of the camera to the position of the player
        //Offset being added so camera does not just follow centre of the object
        transform.position = playerTransform.position + offset;
    }
```

I started by declaring the variables needed to set the position of the camera: players position (playerTransform) and the position to add to the player controlled game objects position (offset).

The code the sets the position of the camera to the position of the player game controlled object. This means that when the player presses either the left or right key the camera will also move in the same direction (I will make sure of this by carrying out tests). The code also adds the offset which has been set in the Unity inspector.

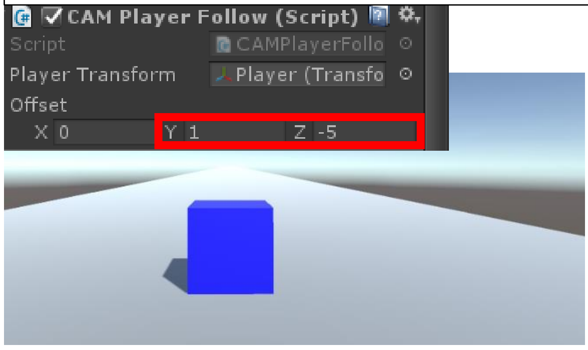
I have used the players game object position in the scene and set that equal to the position of the camera. However, the variable offset if being added so that the camera does not just follow the centre of the player controlled game object. It is important the camera is set in a position that will be suitable for the player to see where they are moving, and when the obstacles are added players should also be able to see them clearly.

Camera position without offset being added



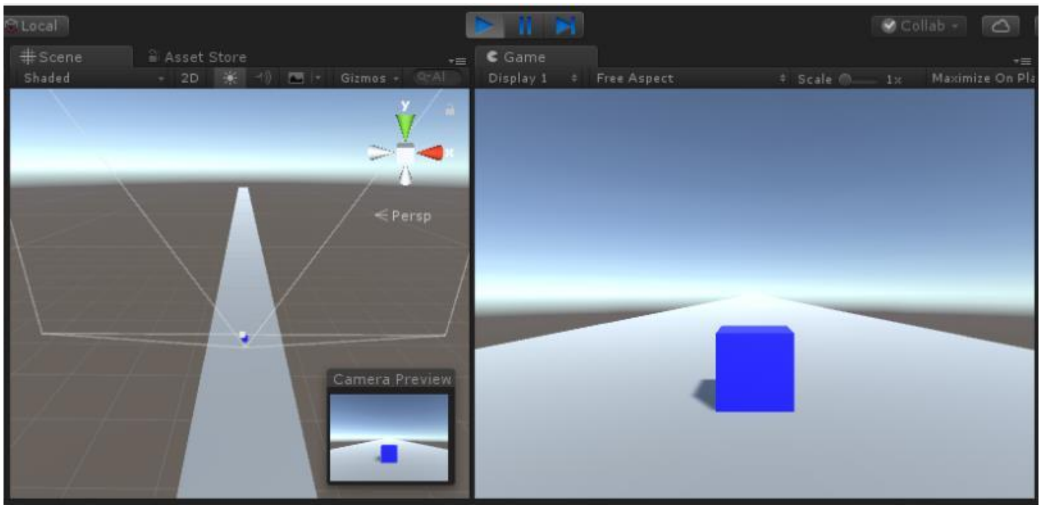
```
CAM Player Follow (Script)
Script               CAMPlayerFollo
Player Transform     Player (Transfo
Offset
  X 0    Y 0    Z 0
```

Camera position with offset being added



```
CAM Player Follow (Script)
Script               CAMPlayerFollo
Player Transform     Player (Transfo
Offset
  X 0    Y 1    Z -5
```

Test reference 1.4 – 1.4.2



Another way of checking if the tests 1.4 – 1.4.2 have succeeded is by selecting the Camera in scene. If the camera frustum (region that can be rendered by a camera) changes as the player game object moves forward, to not show the start of the ground plane, then this means that that the test is passed. The above screenshot provides evidence that the tests has passed as the camera frustum (indicated by the grey lines) changes.

Having performed the tests with each of the specified cases, they all succeed. I will only be having one camera in the scene and this camera has been set to follow the position of the player controlled object. If the player presses either the left or the right key the player controlled object moves and the camera follow its motion. The solution that I have developed to this problem has been successful.

2.3 Testing

Sprint 1 has involved test references 1.1 – 1.4.2. Refer to the testing table for a summary of the tests.

Unit Testing was carried out once the code was developed to add control to the player controlled game object and when setting the camera in the scene. The test table show the test that were carried out once the code for each feature had been developed (refer to tests 1.3.1 – 1.3.5 and 1.4 – 1.4.2). The tests with reference 1.3.3 and 1.3.4 test the robustness of the code developed to add control to the player game object.

Integration testing was carried out once each subtask of the sprint was completed. Integration testing involved carrying out all the test questions that were in the test table from the top. The results of carrying out integration testing can be seen in the integration test table.

Regression testing was carried out when the bug discovered when carrying out tests with reference 1.3.1 - 1.3.2a, was fixed. I carried out test references 1.1 – 1.3.2a again once the bug had been fixed. The results of carrying out these tests again was that there was no impact upon other features of that game that has been implemented in this sprint.

Acceptance testing was not carried out at this point (refer to the review for this sprint for more detail).

2.4 Sprint 1 Review

At the end of the first sprint, I have setup the game environment to includes a ground plane and set the background colour. I have also created the player game object and allowed the user to control the player. Within the scene of the game I have also set the camera to follow the player. This sprint aimed to achieve success criteria 1.1 – 1.3. The end of the first sprint has fulfilled success criteria 1.1 and 1.3.

Success criteria 1.2 has not been fulfilled. Test reference 1.3.5 has not succeed since the player game object continues to slide in the direction the players decide to move in. Having come the end of the sprint, this bug has not been fixed. As mentioned, the bug was supposed to be fixed once I had set the position of the camera however it has not been fixed. This is because I am not sure what is causing this bug. I will start the next sprint by fixing this bug because if it is not fixed the success criteria will not be fulfilled. Apart from the test mentioned, all the other tests have succeeded. This provides further evidence to show that success criteria 1.1 and 1.3 has been achieved.

At the end of this stage, I have created a prototype. I will not be giving this prototype to my stakeholders to gain feedback because this sprint has only focused on basic game features. I could have used the prototype to carry out usability testing, however as there is still a bug with moving the player controlled game object I will carry out any usability testing that I would have carried out in this sprint, later at the end of the next sprint.

This sprint has been successful. At this stage no modifications will need to be made to my solution (refer to the analysis section 1.5 My solution). The aim of this sprint was to establish basic game features. Now that I have done that I can carry on fulfilling further success criteria.

3.0 2nd Sprint

3.1 Design

The previous sprint was successful in establishing basic game features and adding control to the player game object to allow it to move in the horizontal direction. This sprint will further focus on the player controlled game object as I will start this sprint by fixing the bug described by test reference 1.3.5. Furthermore, this sprint will focus on obstacles within the game. This sprint will be broken down into the following individual subtasks:

1. Fix bug described by test reference 1.3.5
2. Create and position obstacles
3. Obstacle and player game object collisions
4. Spawning obstacles

It is important that I start this sprint by fixing the bug described by test reference 1.3.5 because then success criteria with reference 1.2 can be achieved before moving on to achieving any other success criteria. Furthermore, I have decided to order this sprint in order of which tasks will be simple to complete. Creating and positing game collisions objects will be very simple and will not take as much time compared to developing the code for the actions that will take place when the obstacle and the player controlled game object collide.

The reason for breaking down this sprint in the following subtasks above is because it is important that I complete developing features of the game that have an impact on player controlled game object. For example, the collision objects in the game will have an impact on the player controlled game object because if it collides with an obstacle it will result in the game ending. Without the development of the code required between the obstacles and the player controlled game object I cannot go on to look at for example, scoring for the game. Completing my second sprint in this order will allow me to focus on the how the player game object and the obstacles interact with each other and if the correct action is taken when they do.

I will carry on using algorithm to plan any code as it will provide step by step instructions on how to solve the problem.

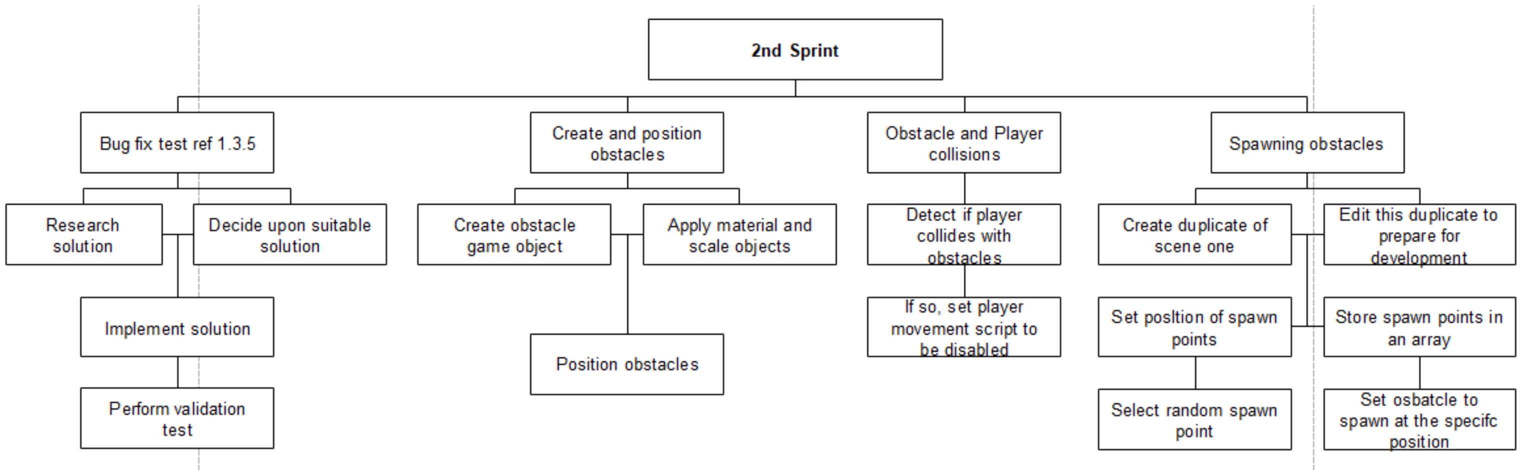

2nd Sprint
Bug fix test ref 1.3.5
Create and position obstacles
Obstacle and Player collisions
Spawning obstacles
Research solution
Decide upon suitable solution
Implement solution
Perform validation test
Create obstacle game object
Apply material and scale objects
Position obstacles
Detect if player collides with obstacles
If so, set player movement script to be disabled
Create duplicate of scene one
Edit this duplicate to prepare for development
Set position of spawn points
Store spawn points in an array
Select random spawn point
Set osbatcle to spawn at the specifc position

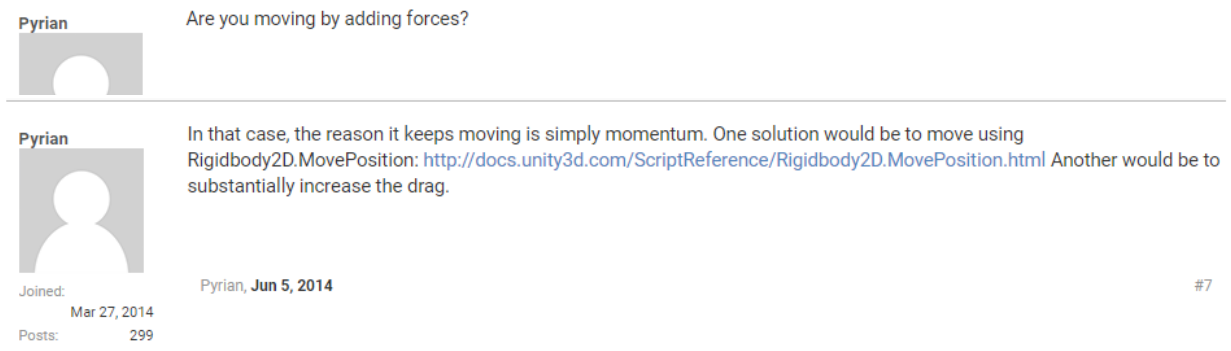
Figure 3: Top-down design for Sprint 2

3.1.1 Fixing bug described by test reference 1.3.5

Having set the camera in the scene to follow the player controlled game object I was able to further observe the motion of the player. The bug that I have found is that when the player taps the left arrow key, the player moves in the left direction however, even if the left key is tapped the player continues to move in the direction that the player wanted to move in. After trying the suggested fixes mentioned (changing the amount of forwards force applied and slowing the speed down) it was evident that the bug was not fixed. I am not very sure to what is causing this bug however, I will be carrying out research to find out how to fix the bug. If it is not fixed it will affect my stakeholders game experience.

As the actual output of this test was that the player moves left and continues to move forward, however also slides in the direction the player wants to move, when carrying out research I knew that in need to find out how to stop a game object (player) sliding along another game object (ground plane).

The following website (site 1.1 – refer to bibliography) describe a similar bug to which I have. It has helped me come up with a solution to fix the bug. The screenshot below shows the part of the forum that helped me to think of a solution to fix the bug.



Pyrian

Are you moving by adding forces?

Pyrian

In that case, the reason it keeps moving is simply momentum. One solution would be to move using Rigidbody2D.MovePosition: http://docs.unity3d.com/ScriptReference/Rigidbody2D.MovePosition.html Another would be to substantially increase the drag.

Pyrian, **Jun 5, 2014** #7

Joined: Mar 27, 2014
Posts: 299

The solution to fix this bug is to change the way in which the player controlled game object moves. Rather than applying a force to the Rigidbody attached to the game object, the solution to the problem will involve moving the position of the game object in the scene. This a suitable solution to the problem because applying a force was resulting in momentum building and therefore causing the player controlled game object to side in the direction. By moving the position of the player controlled game object will mean momentum is not affected.

As mentioned by the forum another solution the problem would be to increase the drag. I have not considered this solution as I am not very experience with the unity physics properties therefore, if I have another problem later in development I will not know if this solution will be causing the problem.

To take into consideration the solutions of this bug I will re-design the algorithm that was devised in section 2.1.3. As the player will only be able to move on the x axis, I will only need to consider allowing the player to move on the x axis in the game. To move the object in the scene, the player input will be taken from the keyboard and the input will be applied to the player game object to move its position in the scene. The algorithm below shows how I will add control to the player control game object:

```
private Rigidbody rb
public float speed
public float forceForward = 2000
function start ()
        rb = RETURN reference to Rigidbody
end function
function fixed update ()
        UserInput ()
        MovePlayer ()
end function
function UserInput ()
        float moveAlongX == INPUT direction of movement
        return moveAlongX
end function
procedure MovePlayer()
        APPLY forceForward TO rb
        if UserInput () > 0 THEN
                move player game object potion to the right
        else if UserInput () < 0 THEN
                move player game object position to the left
end procedure
```

In the algorithm private Rigidbody rb is the variable that holds the reference to the Rigidbody. The reference is then set in the start function meaning that when the code in the start function is executed it will return a reference to the attached Rigidbody. This is done in the start function because code in the start function is called on the first frame the script is enabled and is only called once.

Rather than executing all the code within the fixed update function, I have created another a function called UserInput and a procedure called MovePlayer. I have decided to use these subroutines because it organises the code. If later in development I need to add any code to this script, then any code related to the moving the player controlled game object would be added to the correct method. Furthermore, the use of functions will reduce repeat coding.

The UserInput function will return the value of the variable moveAlongX because this will then be used in the MovePlayer procedure to determine if the position of the player game object should be moved to the left or right from its current position. The UserInput function will be called in the FixedUpdate function because it will need to check the user input every fixed frame. This will allow the player to change the position of the player game object throughout the game.

The MovePlayer procedure will calculate the new position to move the player controlled game object. It will calculate the position to move the Rigidbody attached to the game object. Also, a forward force is being applied to Rigidbody attached to the player game object to keep it moving on the z axis. This function will also be called FixedUpdate function because once it has calculated the new position to move the player it will need to apply the result of the calculation to move the player. This will allow the player to change the position of the game object throughout the game. For parts of this algorithm that have not been explained in this section refer to the design section of sprint 1 (2.1.3 Add Control to the player controlled game object).

Variable Name	Type	Explanation
rb	Object - Rigidbody	Variable will store the reference to the Rigidbody. The name 'rb' is used for this variable as it is just short for Rigidbody. Variable will allow the physics properties of the object to be changed. Will be declared as private.
speed	float	Speed 'value' which the player game object will move. Declared as a float because the speed will need to be set to an accurate speed that players are able to handle.
forceForward	float	Variable will store the magnitude of the force that will be constantly applied to the Rigidbody of the player game object to move it in along the z-axis. Variable will be public to allow it to be edited in the inspector.
moveAlongX	float	Stores the keyboard input which will be used to move the position of the player. The variable will store float values within the range of -1 and 1.

As the above solution should fix the bug, this would mean success criteria with reference 1.2 would be achieved. This means that any development to do with the player controlled game object will be completed. At this stage it is important to carry out user input validation. The table below show what validation I will be carrying out and the planned test data. As the user will mainly be using the keys to control the game object the overall game will not include large amount of input validation.

Input	Validation rule	Explanation of validation	Justification of validation	Test data*	Justification of Test data
Arrows keys	Closed Input	Check to make sure that the player game object moves when keys pressed	Makes sure that user input works		Includes both valid and invalid inputs
WASD keys	Closed Input	Check to make sure that the player game object moves when keys pressed	Makes sure that the user cannot move using any other keys part from this set and the arrow keys		Includes both valid and invalid inputs

* refer to the testing table

3.1.2 Create and position obstacles

The obstacles that I will be adding in this sprint will be the obstacles that players need to avoid. For now, I will look at creating an obstacle, coping it and then positioning them in the scene to create the first level, further on in this sprint I will look at spawning the obstacles for level two of the game.

I will start by creating one obstacle and create a copy of the obstacle. Creating a copy of the obstacle game object is how a prefab will be created. As the same object will be reused in the scene serval times creating a prefab of the object will be useful. This is because if the properties of on one obstacle game object was changed then it will affect all instances produced. If a property of an on obstacle was to change I will want this to be applied to all instance of the obstacle, the use of prefabs will mean that the property will not have to change on all instances individually.

The obstacle dimensions will be (2, 1, 1) on the (x, y, z) axis respectively. These dimensions will be applied because it will differentiate the obstacles from the player game object. By being bigger in size it will be clear to players that these are the objects that they must avoid colliding with. Players will be moving the cube to avoid obstacles therefore the positioning of the obstacles will be important.

The colour of the obstacles will be black. I have chosen this colour for the obstacles as it will stand out from the ground plane and the player controlled object. The colour will also reflect that players must try avoiding the obstacles.

Figure 3 shows a part of the design for the first level. The obstacles will be positioned using the following design where there will be a gap for the player game object to pass the obstacle and advance further into the level. The distance between each set of obstacles will be a fixed amount.

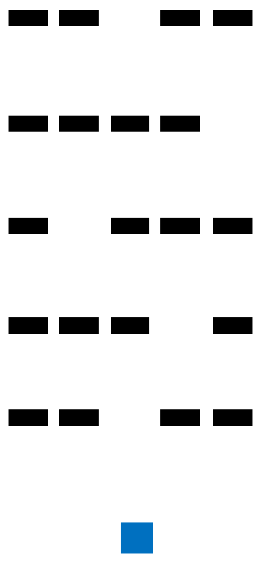


Figure 4: Level 01 Design

3.1.3 Obstacles and player game object collisions

Before I start to design an algorithm to detected if the player collides with an obstacle, it is important to decide on what will happen when the two collide. Two possible events are: a message is displayed to the player to indicate that the game has ended, or the player controlled game object will longer be able to move. Both are possible solutions however, by making sure the player controlled game object is no longer able to move will be the best solution. This is because if the player is no longer able to move then this will be a good indication to the player that the game has ended. Also, displaying a message to the player will be more complicated in terms of the development and this can form a subtask to a later sprint.

To determine if the player controlled game object has collided with an obstacle a script will be attached to the player controlled object. The algorithm bellow shows how a collision will be detected if the player collides with an obstacle.

```
private rigidbody rb
public PlayerMovemnet playerScript
function start ()
    rb = RETURN reference to rigidbody
end function
function OnCollisionEnter (Collision collisionDetails)
    if (collisionDetails == "CollisionObstacle") THEN
        playerScript = Disabled        // Player not able to control object anymore
    end if
end function
```

In the algorithm private rigidbody, rb is the variable that holds the reference to the Rigidbody. The reference is then set in the start function. The variable playerScript will be used to create a reference from the collision script to the PlayerMovemnet script. By creating this reference, it will allow the PlayerMovemnet script to be disabled.

The OnCollisionEnter function is a Unity specific function. I will use this function because when the collider attached to the player controlled game object touches the collider attached to the collision obstacle, the function is called. This means that any code within it will only be executed when the two objects collide. The parameter that is being passed to the function is a class called Collision which contains information such as the object that the player has collided with. A variable called collisionDetails will store the Collision class.

In the OnCollisionEnter function it checks if the player collides with a game object with the name “CollisionObstacle”. If the player does collide with an obstacle, then the PlayerMovemnet script is disabled. This will mean that the player game object will stop moving completely (stop moving in all directions). If the player does not collide with any of the obstacles this script will not be executed.

Variable Name	Type	Explanation
rb	Object - Rigidbody	Variable will store the reference to the Rigidbody. The name ‘rb’ is used for this variable as it is just short for Rigidbody. Variable will allow the physics properties of the object to be changed.
playerScript	Object	Variable will store reference to the PlayerMovemnet script. It will allow the PlayerMovemnet script to be disabled when the player collides with the obstacles. The variable will be declared as public so that the reference can be created in the unity inspector.
collisionDetails	Class	Stores the Collision class. It will allow information such as the object that the player has collided with to be accessed. This information will then be used to determine when the PlayerMovemnet script should be disabled.

3.1.4 Spawning Obstacles

The development of the game so far has focused on level one. Before development of spawning obstacles in the game scene starts, I will create a duplicate of the first scene (level one). This will allow the development of the game so far to be copied and another level (level two) will be created.

The collisions obstacles for level one as mentioned will be copied multiple times and will be positioned in the level one scene. I could do the same for level two however this will mean that the game will be the same all the time and there will be no variation between the levels. Once players have played the game once, the second time they play the game it will no longer be different or challenging. By spawning the obstacles, the computer will position the obstacles and place them in the game scene.

Following the collision obstacle layout for level one (seen in section 3.1.3), the obstacles will need to be spawned is rows. Four obstacles will need to be spawned in a row and at one point no obstacle should be spawned. By having no obstacle being spawned at a certain point, this will indicate to the player that this is the point where they need to control the player game object to pass through the gap.

The obstacles will be spawned across the ground plane. At the start of development five empty game object will be created and their position on the x, y and z axis will be used as the spawn points. The algorithm below shows how the obstacles will be spawned in the game:

```
array pointsToSpawn [ ]
public GameObject obstacle
function Update ( )
        SpawnPoints()
        if (spawn point position on z axis < (length of the ground plane - 100)) THEN
                SpawnObstacles( )
end function
procedure SpawnPoints( )
        vector 3 moveSpawnPoints = NEW vector 3 (x, y, z)
        move position of spawn point on axis by amount moveSpawnPoints
end procedure
procedure SpawnObstcales( )
        int randomPoint = random number between 0 and pointsToSpawn.length
        int i = 0
        for i < pointsToSpawn.length
                if (randomPoint != i) THEN
                        Place obstacle game object at position
                        i = i + 1
end procedure
```

In the algorithm two procedures called SpawnPoints and SapwnObstcales will be used. The SpawnPoints procedure will calculate the position at which the spawn points should be moved once a set of obstacles have been spawned. At the start of the game five spawn points will be created as empty game objects in Unity and their position will be set to 0 on the z axis. The SpawnPoints subroutine will add 40 to the spawn point position on the z axis each time the procedure is called. This subroutine is called in the update function because it will need to calculate the position at which the spawn points should be moved on the z axis each frame to progressively spawn more obstacles.

The SapwnObstcales procedure will create a copy of the obstacles at the specified position in the game. An array will store the position of the five, empty game object which will be used as the spawn points. An index of the elements in the array will be selected at random. This index will then be used to determine at what spawn point position an obstacle should not be spawned. Furthermore, a for loop is being used to iterate through the array and determine if an obstacle should be spawned. The temporary variable ‘i’ is being used to count the iteration of the array. If the value of ‘i’ is less than the length of the array, then this will indicate that obstacles have not been spawned at all the required positions, therefore the iteration of the for loop will continue.

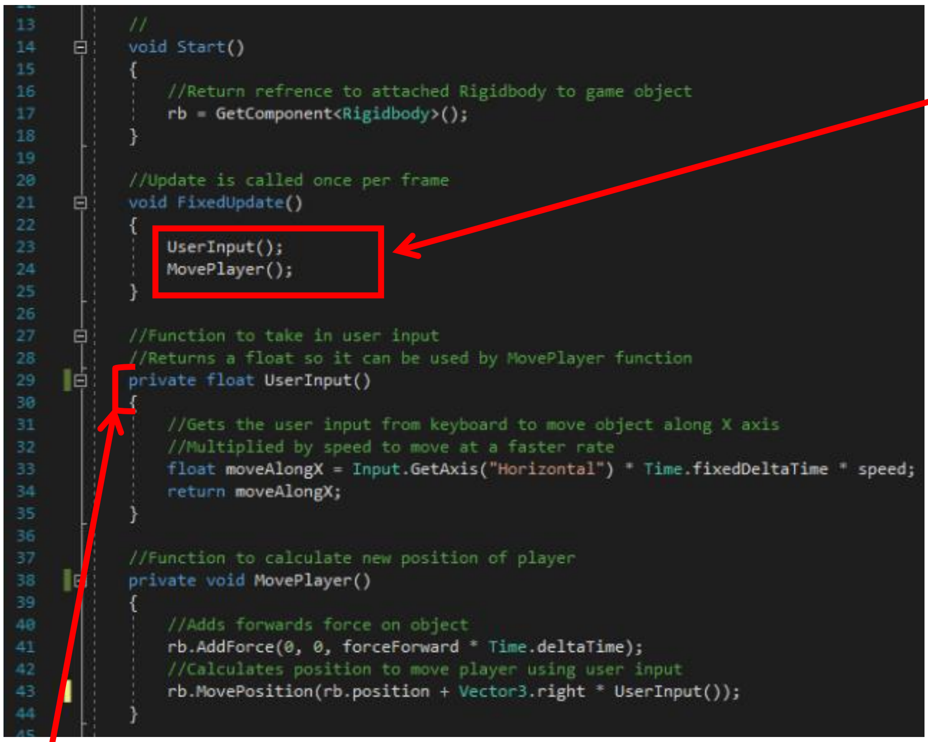
An alternative solution to using an array to store the position of the five empty game objects (which will be used as the spawn points) would be to store each spawn point position using a sperate variable. However, the use of array is being considered because it is easier to handle and if more spawn points want to be used then the size of the array can easily be changed rather than having to create a new variable. Furthermore, a for loop is being used as oppose to a while loop because the loop only needs to be run a set number of times rather than until a condition is no longer true.

Variable Name	Type	Explanation
pointsToSpawn	Object	Array that will store the position of the points where obstacles should be spawned in the scene. Variable is of type Transform because it will store the position which are float value in three dimensions.
obstacle	Object	Will store the game object that should be spawned at the specific spawn points that will be calculated.
moveSpawnPoints	Vector 3	Variable will store the position at which the spawn points should be moved so that the next set of obstacles can be spawned. Variable is of type Vector 3 as it will store values on how much the spawn points should be moved on the x, y and z axis. Will initialise variable within function as the variable will only be used by the function.
randomPoint	Integer	Variable will store an integer value which is the index of an element from the array. In total there will be 5 spawn points so variable will store an integer between 0 and 4. The value of this variable will then be used to determine at which point not to spawn and obstacle. Will initialise variable within function as the variable will only be used by the function.

3.2 Development

3.2.1 Fixing bug described by test reference 1.3.5

To fix this bug describe I will be editing the PlayerMovemnet script that has been attached to the player controlled game object.



```
13      //
14      void Start()
15      {
16          //Return refrence to attached Rigidbody to game object
17          rb = GetComponent<Rigidbody>();
18      }
19
20      //Update is called once per frame
21      void FixedUpdate()
22      {
23          UserInput();
24          MovePlayer();
25      }
26
27      //Function to take in user input
28      //Returns a float so it can be used by MovePlayer function
29      private float UserInput()
30      {
31          //Gets the user input from keyboard to move object along X axis
32          //Multiplied by speed to move at a faster rate
33          float moveAlongX = Input.GetAxis("Horizontal") * Time.fixedDeltaTime * speed;
34          return moveAlongX;
35      }
36
37      //Function to calculate new position of player
38      private void MovePlayer()
39      {
40          //Adds forwards force on object
41          rb.AddForce(0, 0, forceForward * Time.deltaTime);
42          //Calculates position to move player using user input
43          rb.MovePosition(rb.position + Vector3.right * UserInput());
44      }
```

Subroutines created are being called within FixedUpdate because it will need to check the user input and move the Rigidbody attached to the player to a specific position every fixed frame to move the player game object.

The access modifier for both the UserInput and MovePlayer subroutines are both private as they do not need to be accessed by other classes or scripts.

The UserInput function, return type has been set to float because the variable moveAlongX is a float value. This variable is being returned by the function because it will be used when calculating where the player should move in the scene.

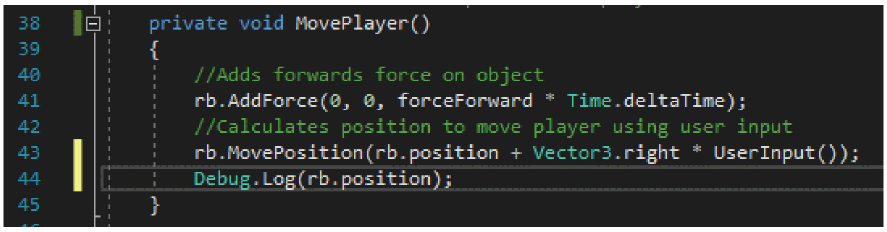
Having followed the algorithm that was designed in the design section 3.1.1, the screenshot above shows the code that has been developed to add control the player controlled game object and to fix the bug described by test reference 1.3.5.

The two line that have changed are line 33 and line 43. Line 33 shows the code that is used to get the user input. The change that has been made is that the user input is being multiplied by Time.fixedDeltaTime and by the speed variable. The user input is being multiplied by the speed variable because this would mean that the player will be able to move at a faster rate which is suitable for the players.

Line 43 shows the code that is used to calculate the position to which the player will be moved. The code rb.MovePosition is used to move the Rigidbody attached to the player game object to a specific position in the scene. The calculation that is being carried out to move the player is the current position (rb.position) is being added to a three dimensions value which is 1,0,0 (Vector3.right). The three dimensions value is being multiped by the return value of the UserInput function because this will allow movement to the left. It will also allow the player game object to not always move by one unit on the x axis.

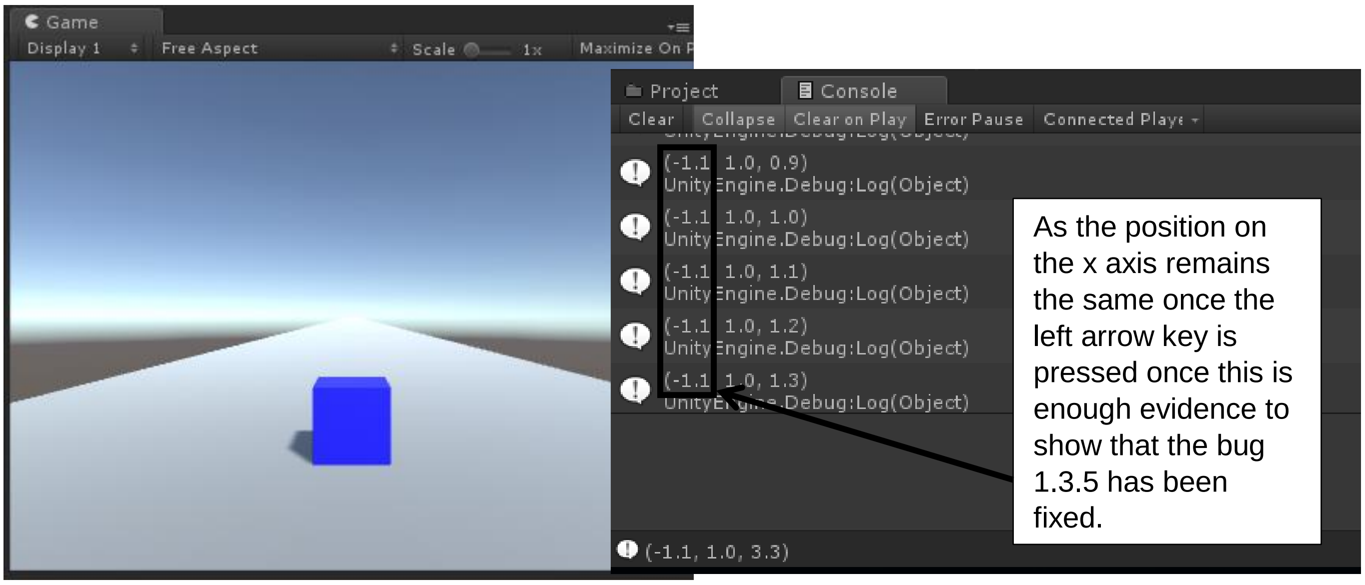
For parts of this code that have not been explained in this section refer to the development section of sprint 1 with heading 2.1.3 Add Control to the player controlled game object.

Test reference 1.3.1a – 1.3.5a



```
38    private void MovePlayer()
39    {
40        //Adds forwards force on object
41        rb.AddForce(0, 0, forceForward * Time.deltaTime);
42        //Calculates position to move player using user input
43        rb.MovePosition(rb.position + Vector3.right * UserInput());
44        Debug.Log(rb.position);
45    }
```

To carry out the following sets of tests I added code (Line 44) to the MovePlayer procedure temporally. This line of code will help carry out the tests with the following test reference 1.3.1a – 1.3.5a because its logs a message to the Unity Console. The message that will be seen in the Unity Console is the parameter that is passed to the Log function – in this case the position of the Rigidbody attached to the player every fixed frame. This will help with testing because I will be able to see how the position changes in the scene as the arrow keys are pressed to input what direction is required to move in.



```
(-1.1  1.0, 0.9)
UnityEngine.Debug:Log(Object)
(-1.1  1.0, 1.0)
UnityEngine.Debug:Log(Object)
(-1.1  1.0, 1.1)
UnityEngine.Debug:Log(Object)
(-1.1  1.0, 1.2)
UnityEngine.Debug:Log(Object)
(-1.1  1.0, 1.3)
UnityEngine.Debug:Log(Object)
(-1.1, 1.0, 3.3)
```

As the position on the x axis remains the same once the left arrow key is pressed once this is enough evidence to show that the bug 1.3.5 has been fixed.

The above screenshots show test reference 1.3.1a. When the left arrow key is pressed once, the player game object does move to the left. Furthermore, it also shows that the bug found when carry out test reference 1.3.5 has been fixed because within the unity console once the left key is tapped it moves to the left and stays in the same position. It does not keep sliding in the left direction across the ground plane. The messages (the player game objects position) shown in the Unity Console provide further evidence to show that the bug found when carry out test reference 1.3.5 has been fixed.

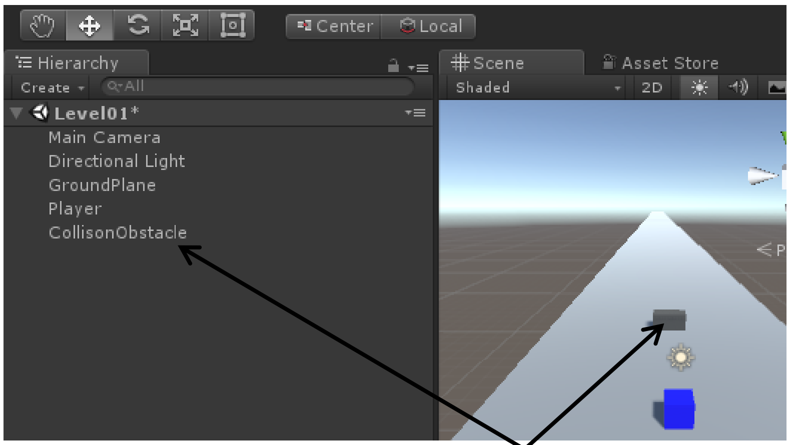
Validation tests

The tests with reference 1.3.1a – 1.3.5a were also used to carry out user input validation. The two validation tests that were carried out were to check that the player game object moves when the arrow or WASD keys were pressed. Test reference 1.3.1a and 1.3.2a show the tests for valid input. Test reference 1.3.3a and 1.3.4a show the tests for invalid input. The user input validation tests described in section design section of this sprint (section 3.1.1) have been successful as only the accepted keys can be used by the player to move the player game object.

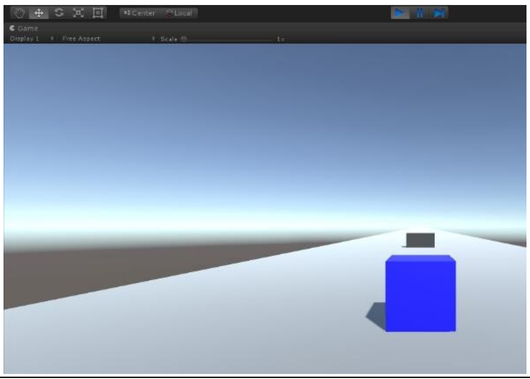
3.2.1 Obstacles and player game object collisions

As explained in the design of this sprint (in section 3.1.2) the obstacles will be reused in the scene many times. To make editing properties of the obstacles easier I will make use of prefabs. I will begin the development of this sprint by creating a single obstacle and then moving on to development of the obstacle and player game object collision code. I will then design the level using the created prefab.

To allow the created obstacles to interact with the Unity physics engine I will need to add a Rigidbody component to game object.



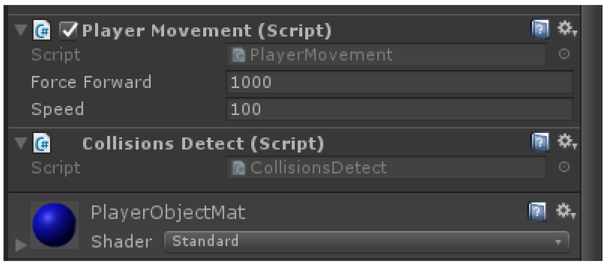
Test reference 2.1



The above shows one obstacle being added in the scene. I have used a cube game object to represent the obstacle in the game. The properties such as the transform and material attached to the game object have been edited so now the game object is ready to be created

This quick test ensures that all currently created game objects appear in game view when in play mode. This is enough evidence to show that the test succeed since players can see exiting and the new game object created.

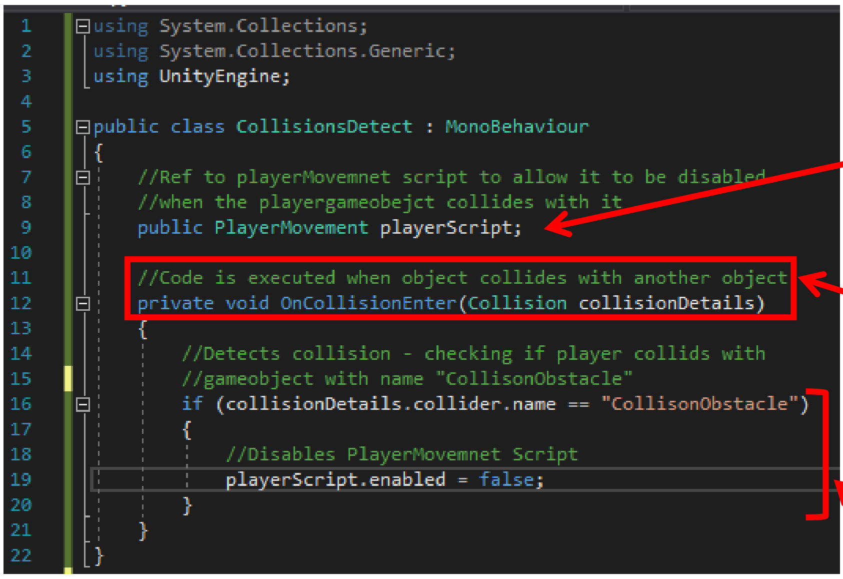
Now that I have created an obstacle I can now develop the code required for the obstacle and player game object collisions. To develop the code required between the obstacles and the player controlled game object I will create another C# script and attach this to the player controlled game object. I could use the script already attached from the previous sprint however, as the code will carry out different functionality, I will use a new script.



Player Movement (Script)
Script PlayerMovement
Force Forward 1000
Speed 100
Collisions Detect (Script)
Script CollisionsDetect
PlayerObjectMat
Shader Standard

A new script called Collisions Detect has been created and added to the played controlled game object. The code in this script will be responsible for detecting collisions between the player controlled game object and the obstacles. Separating it from the player movement script will aid future maintenance as each script carries out different functions.

Having followed the algorithm that was designed in the design section 3.1.3, the screenshot on the next page shows the code that has been developed to detect collisions between the player controlled game object and the obstacles that will be in the scene. Even if the player controlled game object has not been explicitly been referenced in the code but the code still carries out the function as the script is added to the player controlled game object.



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CollisionsDetect : MonoBehaviour
6  {
7      //Ref to playerMovemnet script to allow it to be disabled
8      //when the playergameobejct collides with it
9      public PlayerMovement playerScript;
10
11     //Code is executed when object collides with another object
12     private void OnCollisionEnter(Collision collisionDetails)
13     {
14         //Detects collision - checking if player collids with
15         //gameobject with name "CollisonObstacle"
16         if (collisionDetails.collider.name == "CollisonObstacle")
17         {
18             //Disables PlayerMovemnet Script
19             playerScript.enabled = false;
20         }
21     }
22 }
```

Started by declaring any variables. Reference to the Player Movement script (code annotations used to describe in further detail).

Function “OnCollisonEnter” called when the colliders of the two game objects collide.

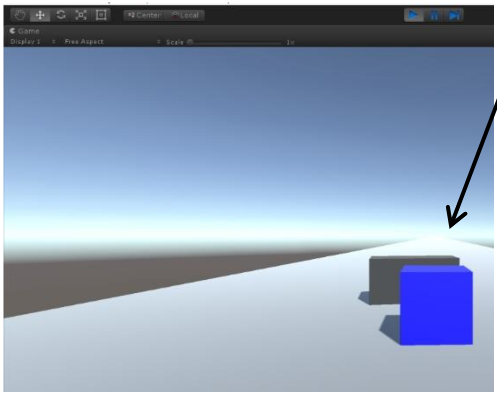
If the player controlled game object hit a collider with the name “CollisionObstacle” then the player movement script attached to the player controlled game object will be set to false, therefore disabling it and not allowing the player to move.

Line 12 of the code use a function called OnCollisionEnter. The function will be called when the collider attached to the player controlled game object touches the collider attached to the collision obstacle. The parameter that is being passed is the Collision class which is given the variable collisionDetails. The Collision class gathers details about the collisions between the two game objects. This is the parameter being passed because information such as the name of the collider that has been hit needs to be checked.

If the player controlled game object collides with the obstacle, then the player movement script will be disabled. Disabling the script will mean the player controlled game object will no longer move and this will indicate to the player that the game has ended.

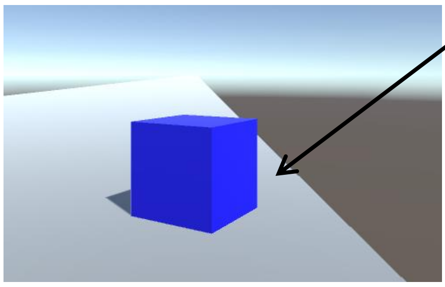
The above code was developed as a solution to the features identified in the success criteria (in the analysis sections) with reference 2.2 and 2.3. Success criteria reference 2.3 has not completely been achieved however I will focus on achieving it completely in a later sprint. When testing I will make sure that I observe the motion between the player controlled game object and the obstacle to achieve 2.1.

Test reference 2.2.1, 2.2.3



When in play mode, if the player controlled object collides with the centre of the obstacle the player stops moving at the point of contact. At this point I also tested for robustness (test reference 2.2.3) by pressing the left and right arrow key to try moving the player. When either key was pressed the player did not move.

Test reference 2.2.2



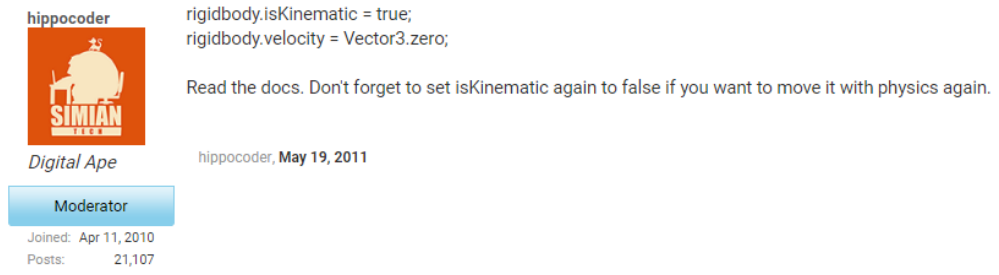
When in play mode, if the player controlled object collides with either the left or right of the obstacle the player does not stop moving at the point of contact (instantly). I am not sure why this is however, it is important that this bug is fixed before I move on because it will affect for example, the players score.

The above bug that I have found is that if the player controlled game object collides with either the left or right of the obstacle the player does not stop instantly (the player stops a few seconds later). I am not very sure to what is causing this bug however, I will be carrying out research to find out how to fix the bug.

Bug fix – Test reference 2.2.2

As the expected output of this test is that the player should stop moving, when carrying out research I knew that I needed to find out how to stop motion after a collision. Both objects (the player and obstacle) have a Rigidbody attached to them. This component allows objects to have physics properties. This lead me to a possible solution which was to disable the velocity somehow when the objects collide.

The following forum https://forum.unity.com/threads/how-to-make-rigid-body-stop-moving-after-collision.88066/ describes a similar bug to which I have come across. It has helped me come up with a solution to fix the bug. The screenshot below shows the part of the forum that helped me.



hippocoder

Digital Ape

Moderator

Joined: Apr 11, 2010

Posts: 21,107

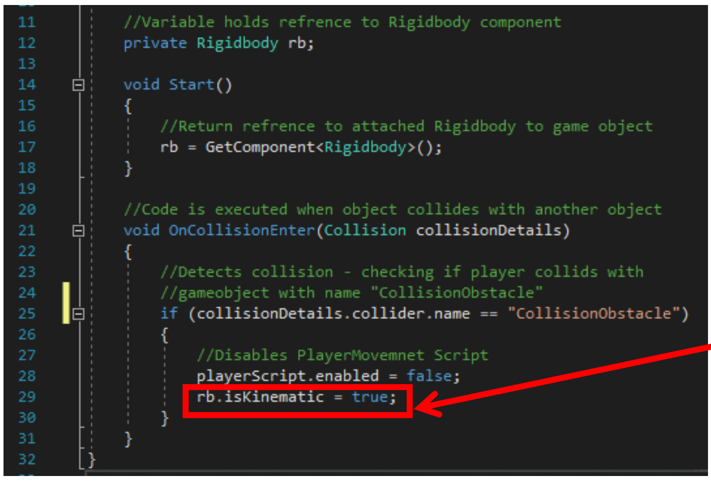
```
rigidbody.isKinematic = true;
rigidbody.velocity = Vector3.zero;
```

Read the docs. Don't forget to set isKinematic again to false if you want to move it with physics again.

hippocoder, May 19, 2011

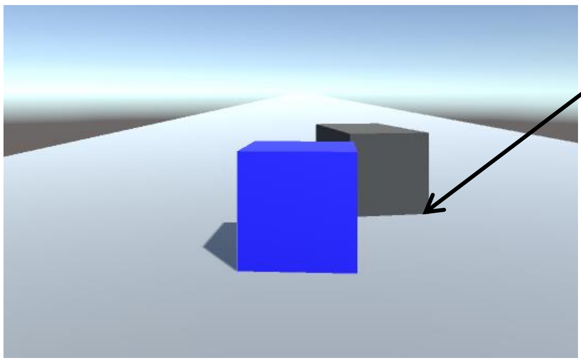
The solution to fix this bug will be to not allow any physics properties of the Rigidbody component to affect the player controlled game object, the instance when the player collides with an obstacle. By not allowing any physics properties to affect the Rigidbody would mean that the velocity (speed) the player is moving will be zero.

An alternative solution to fixing the bug would be to set the velocity of the Rigidbody to zero and set the angular velocity (will stop the player form rotating when it collides) to zero. This will result in the player to stop instantly upon collision. However, the first solution to fix the bug is appropriate because if the player collides with the object the game be set to end. Not allowing any physics properties to affect the Rigidbody attached to the player will be suitable.

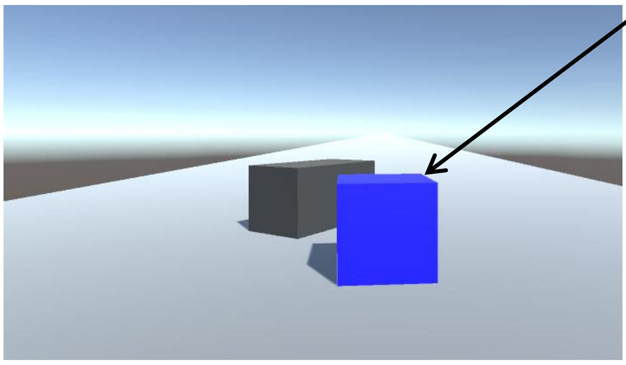


```
    //Variable holds refrence to Rigidbody component
    private Rigidbody rb;

    void Start()
    {
        //Return refrence to attached Rigidbody to game object
        rb = GetComponent<Rigidbody>();
    }

    //Code is executed when object collides with another object
    void OnCollisionEnter(Collision collisionDetails)
    {
        //Detects collision - checking if player collids with
        //gameobject with name "CollisionObstacle"
        if (collisionDetails.collider.name == "CollisionObstacle")
        {
            //Disables PlayerMovemnet Script
            playerScript.enabled = false;
            rb.isKinematic = true;
        }
    }
}
```

To fix the bug, I started by creating a reference to the Rigidbody attached to the player controlled game object. In the function which is called when two game objects collide I added the following line of code. Now, if the player controlled game object hit a collider with the name "CollisionObstacle" then the player movement script attached to the player will be disabled. Also, physics properties such as the velocity will not affect the Rigidbody component.

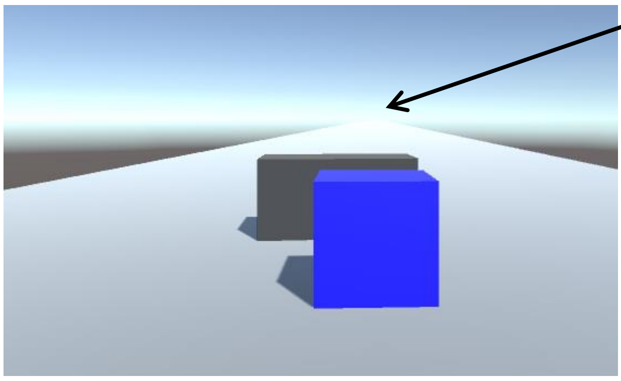


When in play mode, if the player controlled object collides with the left of the obstacle the player now does stop moving at the point of contact (instantly). However, I have now noticed that the obstacle moves more than I expected it to do so.



When in play mode, if the player controlled object collides with the right of the obstacle the player now does stop moving at the point of contact (instantly). However, I have now noticed that the obstacle moves more than I expected it to do so. The next test further focuses on this.

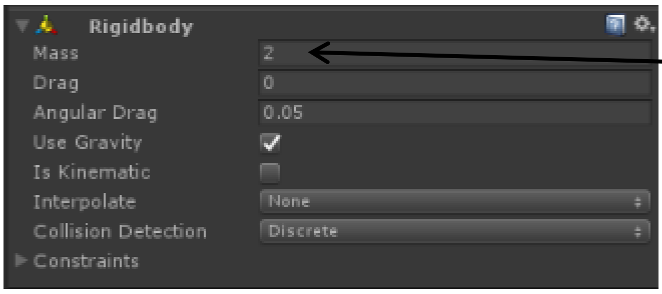
Test reference 2.2.6



When the player collides with the obstacle, the obstacles is pushed forwards and the player controlled game object does stop moving. This is what I want to happen when the two collides however currently the obstacles is being pushed too far.

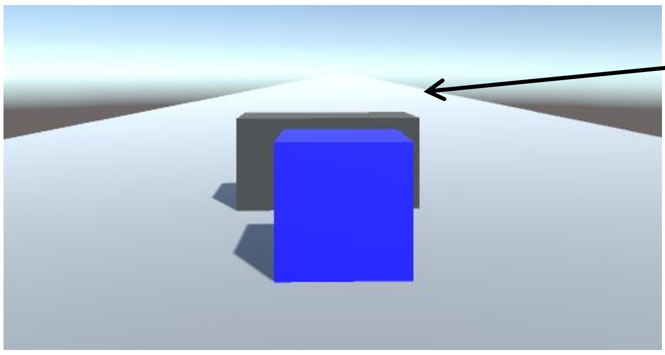
The errors that I have found is that when the player controlled game object collides with the obstacle, the obstacle move far more forward than I expected it to do so. This could cause problems later in development when adding more obstacles. After checking the properties of both game object, the reason for this error, I think it is due to both the game objects (the player and obstacle) having the same mass. To fix this error, I will need to increase the mass of the obstacle. Fixing the error in this way will still cause the obstacle to move forward however not as much as it does currently.

Bug Fix – test reference 2.2.6



```
Rigidbody
Mass                  2
Drag                  0
Angular Drag          0.05
Use Gravity           ✓
Is Kinematic
Interpolate           None
Collision Detection   Discrete
Constraints
```

As there is a Rigidbody component attached to the obstacle I can changes its mass. The default value for the mass was 1. By setting the mass to 2, the obstacle should not forward as much, when the player collides into it.



When the player and the obstacle collide, the player stops moving. The obstacle still slides forward and then stops moving. This solution has not fixed the bug/error. At this moment I will leave the bug unfixed, however if it causes problems later in development I will need to fix it.

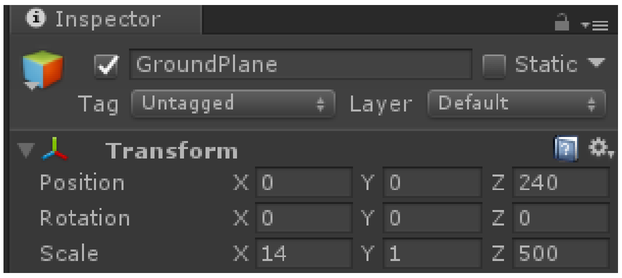
The development of this sprint so far has focused on the solution to features identified in the success criteria (in the analysis sections) with reference 2.2 and 2.3. Success criteria reference 2.2 will be achieved if the player controlled object collides into obstacles and gives rise to a certain effect. Currently the only effect is the obstacles move forwards and the player stops moving. I could however, make the two obstacles bounce of each other when they collide. This will be a question that I will ask at the end of the sprint when carrying out usability feedback.

3.2.2 Create and position collision objects

As mentioned at the start of sprint one, my game will have around six or more levels. Level one of the game will not look at spawning the obstacles. I will look at the development of spawning obstacles for level two of the game in the next subtask of this sprint.

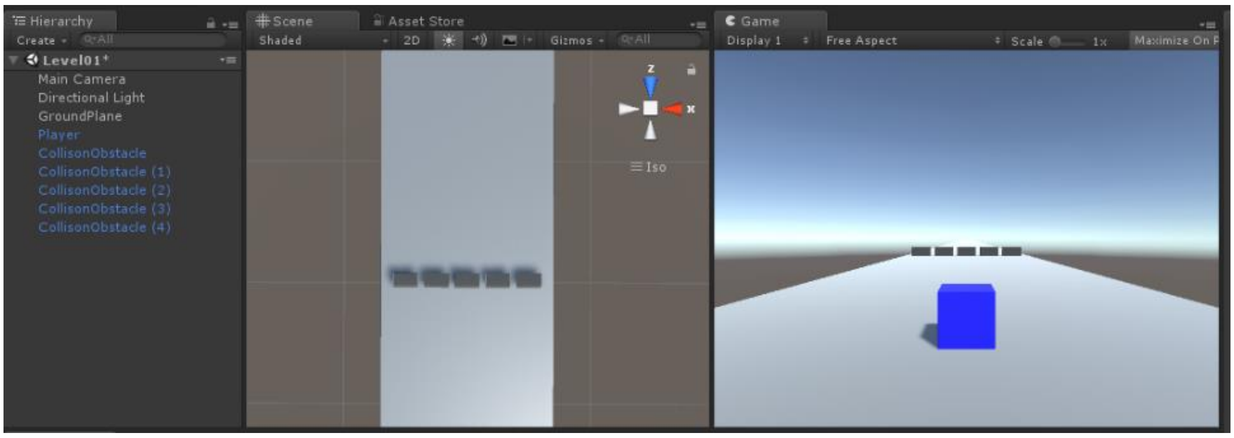
As I have already created an obstacle in the scene and I would like many of these obstacles I will need to create a copy of the game object that I have created. Creating a copy of the game object is how the prefab will be created. Once I have created the prefab, I will place it in the scene a multiple number of times to create an interesting first level that will allow the players to get use to the concepts of the game.

Before I go onto developing Level one of the game using the prefabs, I will change the dimensions of the ground plane. It is important that I decide upon the final dimensions of the ground plane now because this will influence how long player will spend playing level one.



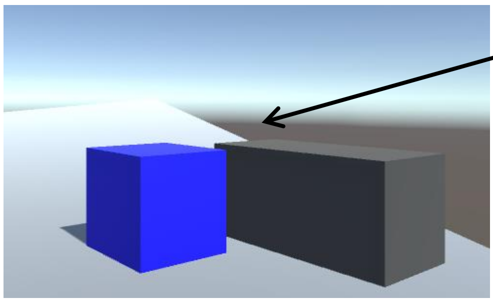
Position	X 0	Y 0	Z 240
Rotation	X 0	Y 0	Z 0
Scale	X 14	Y 1	Z 500

This is the ground plane that the player object will be placed on. The dimensions of the ground plane have been changed to (14, 1, 500) on the (x, y, z) axis respectively. I have also changed the position of the ground plane so that the start of the ground plane is close to where the player controlled game object is.



I have positioned the obstacles in the scene. Before I go any further I will carry out testing to make sure that the same actions take place when the player controlled game object collides with any of the obstacles added.

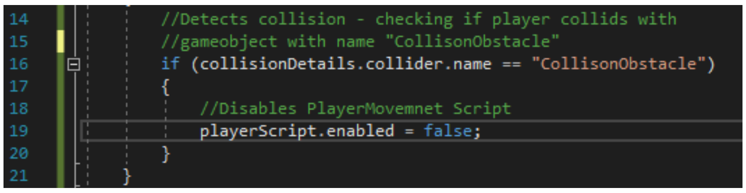
Test reference 2.3.2



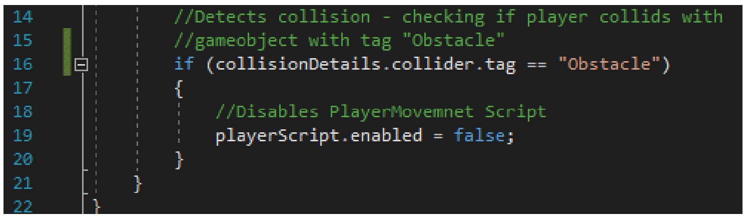
When the right arrow key is pressed, the player controlled game object does move to the right. However, when the player and the obstacle collide, the player does not stops moving. When they collide, the player is still able to use the controls. This should not happen; therefore, I will need to investigate this problem first before moving on.

The above bug that I have found is that when the player collides with the obstacle, the player continues to move. Also, players can use the left and right arrow keys to control the object after the collision. Due to still being able to be controlled the player object, I suspected that the "player movement" script is not being disabled. This suggest that the bug can be fixed by looking at the "collisions detect" script as this is where the player movement script is set to disable when the player and obstacle collide.

Bug Fix – test reference 2.3.2



```
        //Detects collision - checking if player collids with
        //gameobject with name "CollisonObstacle"
        if (collisionDetails.collider.name == "CollisonObstacle")
        {
            //Disables PlayerMovemnet Script
            playerScript.enabled = false;
        }
    }
```



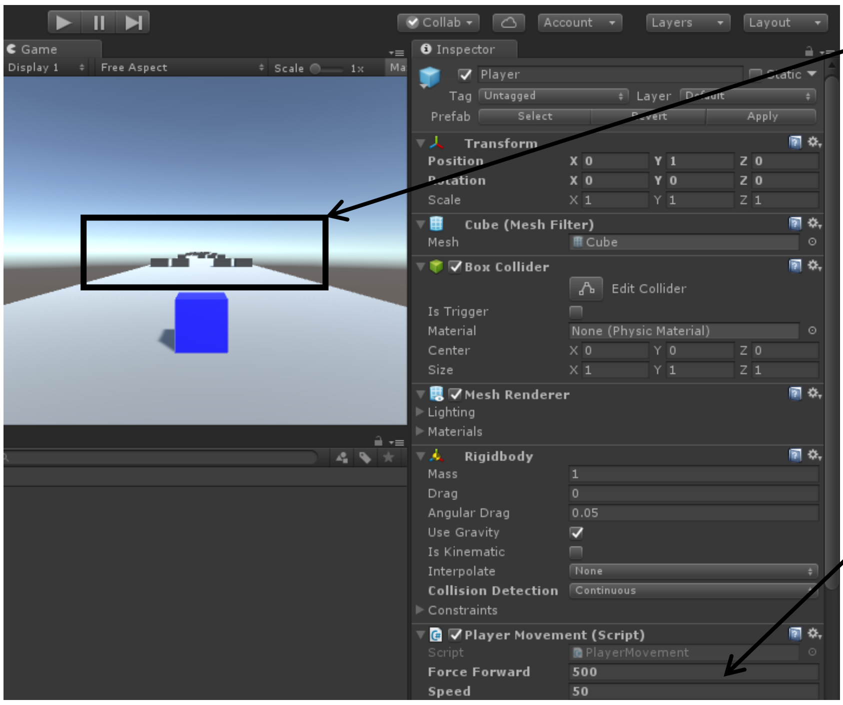
```
        //Detects collision - checking if player collids with
        //gameobject with tag "Obstacle"
        if (collisionDetails.collider.tag == "Obstacle")
        {
            //Disables PlayerMovemnet Script
            playerScript.enabled = false;
        }
    }
}
```

Fix: Checks if the player has collided with an obstacle that has been given the tag "Obstacle" rather than checking the name of the object.

When the prefabs are added to the scene the name of the object does not stay the same. This means that it is not suitable to check if the player has collided with an object with a specific name. Instead the tag of the object is checked. To fix the bug the tag "Obstacle" has been applied to all the obstacles that are currently in the scene. Then the collision detection script was edited so that it checks if the player collides with an object which has tag "Obstacle". If so, it then goes on to disabling the player movement script.

All the obstacles will be given the same tag. Checking the tag of the collision object obstacle it suitable since because if the player collides with an any obstacle in the scene then the same action is required to take place. With this bug fixed, and all other tests succeeding I can now complete mapping out Level one using the prefab.

Due to creating and positioning the obstacles, the first level will be short as this will allow players to get used to the concept of the game before moving onto the next level. After mapping out the level I tested the game myself. I found that it was hard to control the player therefore decide to change the forwards force and speed that the player moves. I will confirm these changes at the end of the sprint when carrying out usability feedback from my stakeholders.



Player
Tag Untagged Layer Default
Prefab Select Revert Apply

Transform
Position X 0 Y 1 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh Cube

Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Mesh Renderer
Lighting
Materials

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic
Interpolate None
Collision Detection Continuous
Constraints

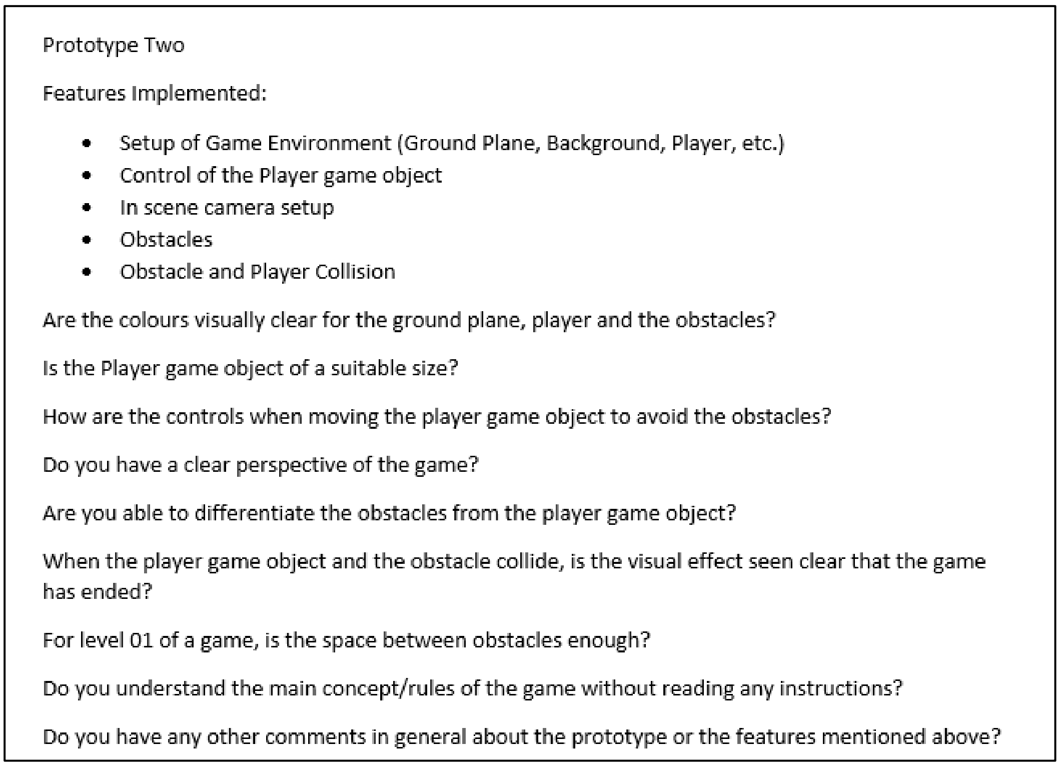
Player Movement (Script)
Script PlayerMovement
Force Forward 500
Speed 50

The level has been mapped out by hand using prefabs. The next level of the game will involve spawning the obstacles.

The forwards force and the speed the payer moves have been changed to suit this level.

Usability testing and Feedback - Prototype Two

The project is now at a point in development, where it is possible to create a prototype which can be given to stakeholders (a small group of them) to gain feedback upon and to test the usability so far. As it is only a prototype I will not be changing any of the build setting within Unity when creating the complied version of the game. To collect specific feedback from my stakeholders, I will provide them with a text document (screenshot below) which will include questions about features of the game that have been implemented. Before moving onto any further development, I will consider the feedback and the results from the usability testing.



Prototype Two

Features Implemented:

- Setup of Game Environment (Ground Plane, Background, Player, etc.)
- Control of the Player game object
- In scene camera setup
- Obstacles
- Obstacle and Player Collision

Are the colours visually clear for the ground plane, player and the obstacles?

Is the Player game object of a suitable size?

How are the controls when moving the player game object to avoid the obstacles?

Do you have a clear perspective of the game?

Are you able to differentiate the obstacles from the player game object?

When the player game object and the obstacle collide, is the visual effect seen clear that the game has ended?

For level 01 of a game, is the space between obstacles enough?

Do you understand the main concept/rules of the game without reading any instructions?

Do you have any other comments in general about the prototype or the features mentioned above?

Figure 5: Feedback Form 1

The results after gathering feedback and carrying out usability testing were generally positive with two major issues. In terms of usability testing such as asking about colour scheme, players game perspective, controls (user input to control the player game object) and game concept, the results confidently show my stakeholders are satisfied with the game so far.

One of the issues that my stakeholders found was related to controlling the player game object using the arrow keys or the WASD key. When moving using the arrow keys, they were not fond of how the player game object moved. A few on my stakeholders quoted that the movement was “too snappy”. Having further spoken to my stakeholders and tested the prototyped myself, I got an idea of how players felt about the way the game responded to user input. I will need to fix this before moving on.

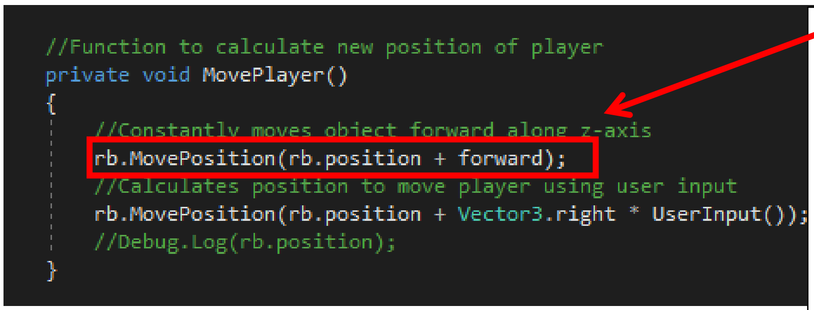
Another issue, was related to the background. I mentioned that I will be adding a solid colour to the background to the scene however, up to this point in development I have not added a background colour. From carrying out usability testing, stakeholders were finding that the ground plane did not look very attractive. I can see this being a problem for players therefore, I will act upon this feedback by adding a solid background colour. The colour that will be used will be a colour that has been use used in the game scene.

Prototype Two – Solving Issue 1

Issue one relates to the controls and the way in which the player controlled game object moves and how it responded to the user input. As mentioned on the pervious page my stakeholders quoted that the movement felt "slippery" and that the player game object would slide more than expected when trying to control it.

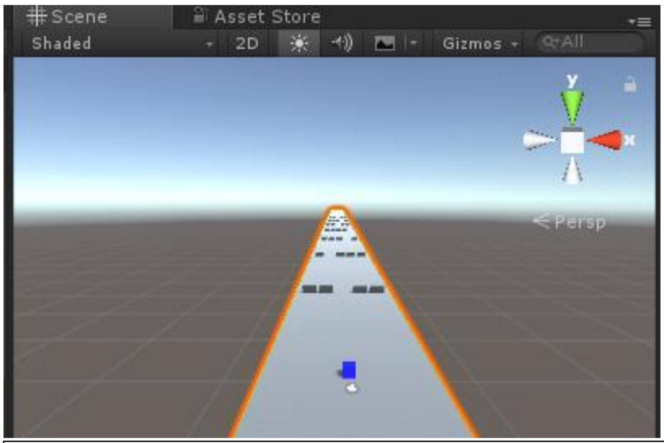
The solution to this problem would be remove the physics material that was attached to the ground plane when fixing test reference 1.3.1 – 1.3.2.1. However, as a force is being applied to move the player controlled game object forward (along the z axis), I will need to change this to make sure that the player game object remains on the ground plane when player control it.

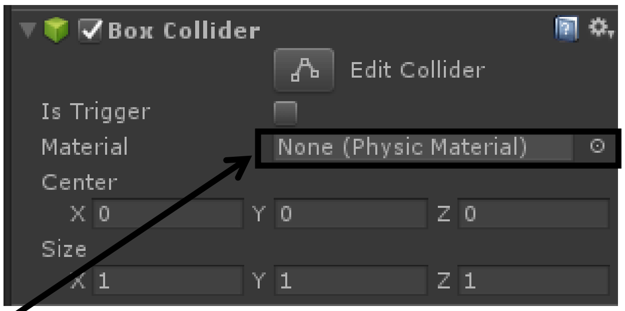
This is a suitable solution to the issue because if forces are no longer being applied to move the player game object then it will no longer rotate – this was also another issue that my stakeholders picked up when carry out usability testing. Furthermore, removing the physics material that was attached to the ground plane is a suitable solution to the issue because it will mean that there will be friction between the player game object and the ground plane therefore, when controlling the player game object, it will not fell as "slippery". The screenshots and the annotations below show, the solution to solving the issue.



```
//Function to calculate new position of player
private void MovePlayer()
{
    //Constantly moves object forward along z-axis
    rb.MovePosition(rb.position + forward);
    //Calculates position to move player using user input
    rb.MovePosition(rb.position + Vector3.right * UserInput());
    //Debug.Log(rb.position);
}
```

This line will add the vector 3 variable forward to the current position of the Rigidbody attached to the player game object, in the scene. The forward variable stores three dimensions. As I want to affect the position only along the z-axis, I will set how much will need to be added each frame to the current position of the player game object in the Unity





Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

As forces are no longer being applied to move the player game object, the physics material that was attached to the ground plane has been removed. Due to, removing the physics material there is friction between the player game object and the ground plane therefore, when controlling the player game object, it no longer feels a "slippery".

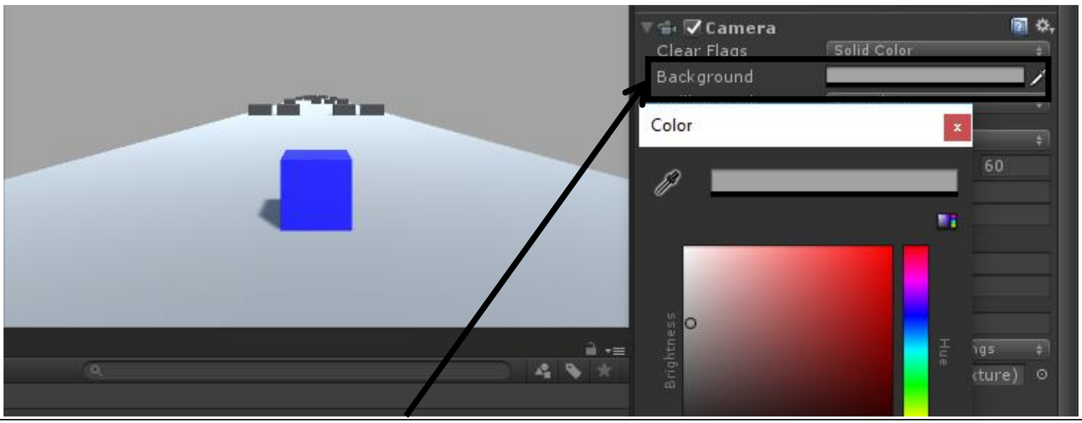
Test reference 2.4.1 – 2.4.2

Having performed both test cases specified, they succeeded. The player constantly moves forward and the response to user input in smother than it was before. Test reference 1.3.1 – 1.3.5 were also performed again having fixed this issue. These tests also succeeded.

Prototype Two – Solving Issue 2

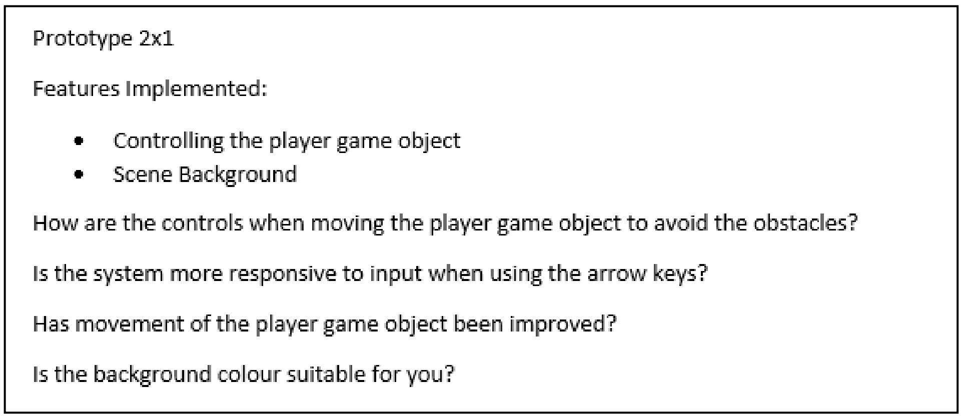
Issue two related to the background of then game. As mentioned on the pervious page my stakeholders found that the background was not “attractive” and that it did not go with the colours that are currently being used in the game scene.

The solution to this problem will be to will be to add a solid colour for the background. I will be using a solid colour because it will not distract player from focusing on the game. Also, the use of a solid colour will fit the game scene because other game elements such as the ground plane and the player game object are all solid colours. The screenshots and the annotations below show, the solution to solving the issue.


Camera
Clear Flags Solid Color
Background

The background has changed from the default (skybox) to a solid colour. The colour that I have used is a solid grey colour.

Now that the two issues that my stakeholders found when gathering feedback and carrying out usability testing have been fixed, I will create another prototype (create another complied version of the game at this point) which will be given to my stakeholders. I will be asking form them to comment on their opinion related to the solution to the two issues. The following questions were added to the prototype two feedback document:


Prototype 2x1

Features Implemented:

- Controlling the player game object
- Scene Background

How are the controls when moving the player game object to avoid the obstacles?

Is the system more responsive to input when using the arrow keys?

Has movement of the player game object been improved?

Is the background colour suitable for you?

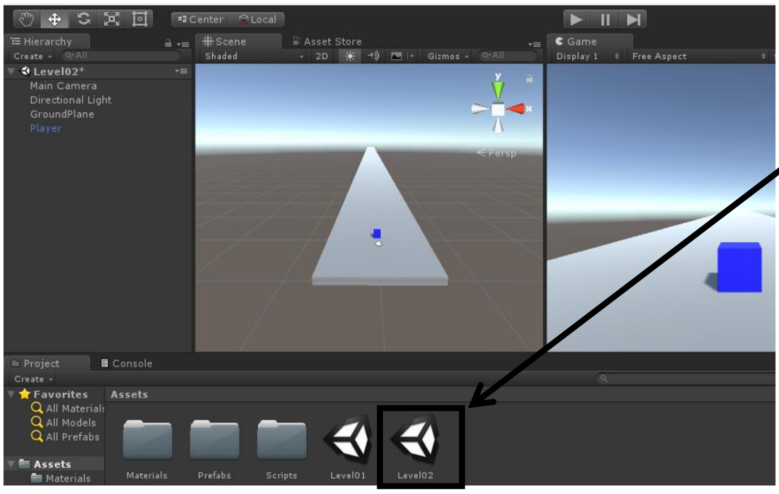
Figure 6: Feedback Form 1.1

The results after gathering feedback and carrying out usability testing from using the porotype were positive. The results confidently show my stakeholders are satisfied with the solution that have been implemented to the two issues as responses from them were that the controls feel more responsive and that the background is of a suitable colour. I will now continue with the development for this sprint.

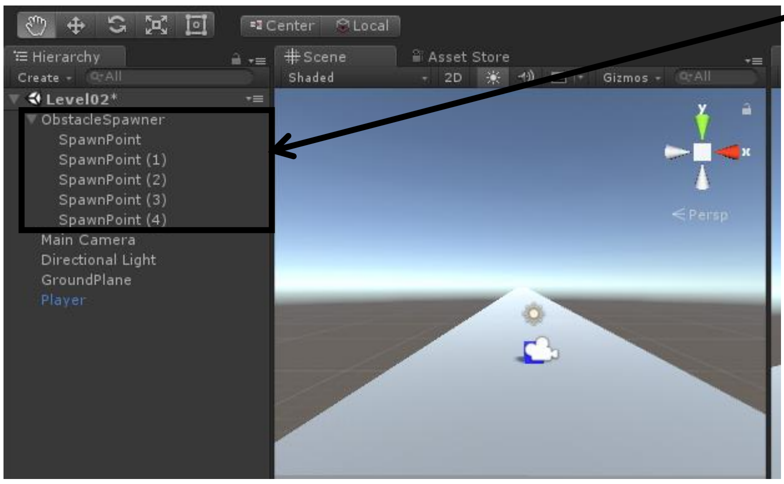
3.2.3 Spawning Obstacles

To spawn the obstacles in the game scene as mentioned in the design an array will be used. This array will store the points (positions) at which the obstacles will be spawned in the scene across the ground plane. To allow players to progress through the level and move the player game object, a random spawn point will be selected, and no block will be spawned at that position. This will be where players will have to control the player game object to pass through the gap since no obstacle will be spawned.

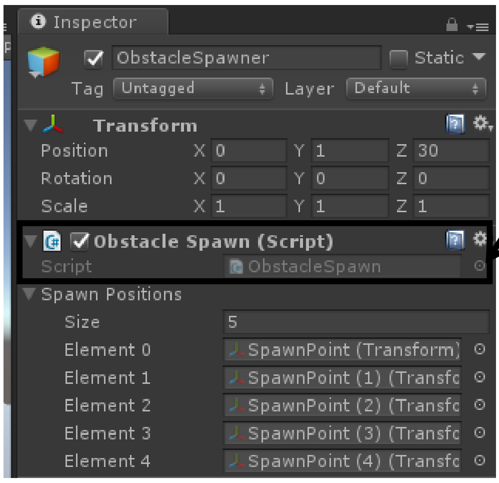
Having followed the algorithm that was planned in the design of this sprint, the screenshot below shows the code that was developed to spawn obstacles in the game scene.



As level one (the first scene) of the game has been designed, a duplicate of the scene was made to create level two. In this scene which is labelled level two, a prefab of one of the obstacles was created and then all the obstacles were removed from the game scene.

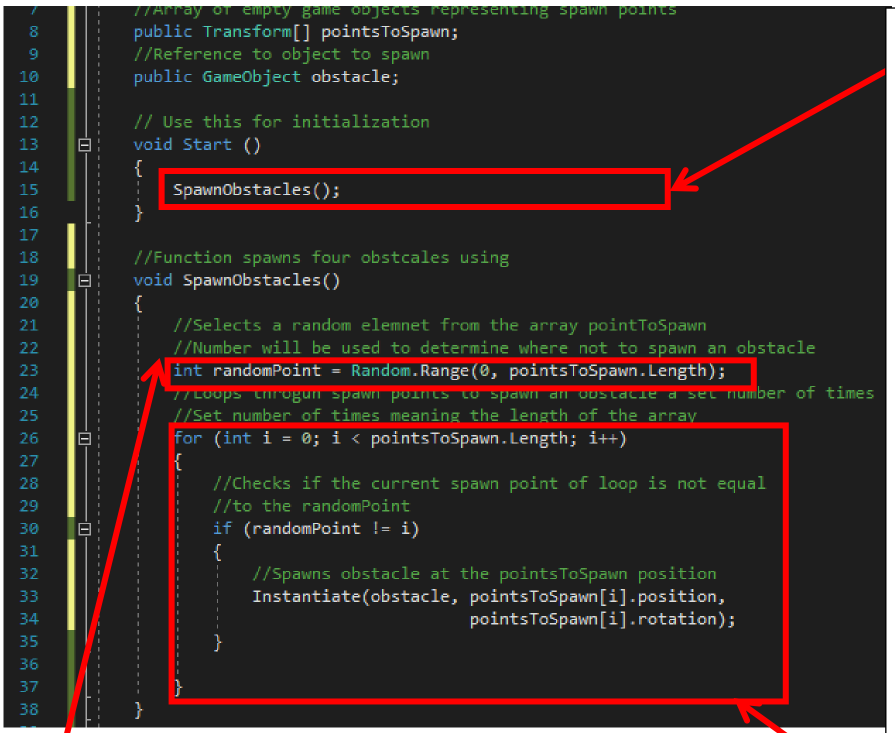


These empty game object have been created. These empty objects called SpawnPoints are a child of another empty game object called "ObstacleSpawner". Making them a child means that they will have transform values based on the parent game object (explained further later in development). The five spawn points have been set to the specific position at which obstacles should be positioned on the ground plane.



Inspector
ObstacleSpawner   Static
Tag Untagged   Layer Default
Transform
Position X 0 Y 1 Z 30
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Obstacle Spawn (Script)
Script ObstacleSpawn
Spawn Positions
Size 5
Element 0 SpawnPoint (Transform)
Element 1 SpawnPoint (1) (Transfo
Element 2 SpawnPoint (2) (Transfo
Element 3 SpawnPoint (3) (Transfo
Element 4 SpawnPoint (4) (Transfo

A C# script has been created and added as a component to the parent empty game object. Currently, an array has been defined which stores the 5 spawn points of the obstacles. The development of the code to spawn obstacles from the spawn points apart from one.



```
7       //Array of empty game objects representing spawn points
8       public Transform[] pointsToSpawn;
9       //Reference to object to spawn
10      public GameObject obstacle;
11
12      // Use this for initialization
13      void Start ()
14      {
15          SpawnObstacles();
16      }
17
18      //Function spawns four obstcales using
19      void SpawnObstacles()
20      {
21          //Selects a random elemnet from the array pointToSpawn
22          //Number will be used to determine where not to spawn an obstacle
23          int randomPoint = Random.Range(0, pointsToSpawn.Length);
24          //Loops throguh spawn points to spawn an obstacle a set number of times
25          //Set number of times meaning the length of the array
26          for (int i = 0; i < pointsToSpawn.Length; i++)
27          {
28              //Checks if the current spawn point of loop is not equal
29              //to the randomPoint
30              if (randomPoint != i)
31              {
32                  //Spawns obstacle at the pointsToSpawn position
33                  Instantiate(obstacle, pointsToSpawn[i].position,
34                                        pointsToSpawn[i].rotation);
35              }
36
37          }
38      }
```

The subroutine that is responsible for spawning the obstacles in the scene is being called in the start function. As this function is called on the first frame the script is enabled and is only called once so only one set (4 obstacles in a row) of obstacles will be spawned. To allow multiple sets of obstacles to be spawned I will have to call the SpawnObstacles subroutine in the update function. For now, it is only being called in the start function because if a bug does occur then it can result in Unity crashing as it will be using large amount of memory to store all the spawned obstacles.

Since all the spawn point are now associated with a given index, line 23 shows the code that selects a random index from the array (currently the array has a size of five elements as this is the number of spawn points). It will choose at random a spawn point which will then be used to not spawn an obstacle at that spawn point. This means that an obstacle will need to be spawned at all the other spawn points.

A for loop is being used to iterate through the array and to spawn the obstacles. An obstacle will only be spawned, if the i value of the loop is not equal to the randomPoint (which is the spawn point at which an obstacle does not need to be spawned).

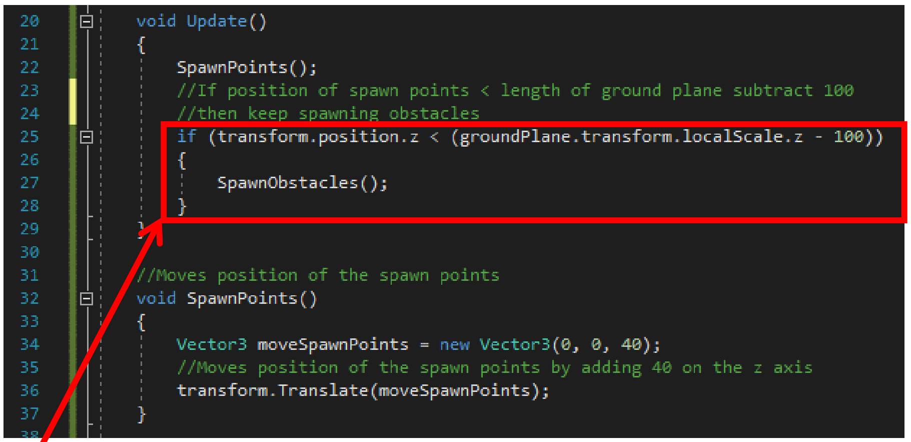
The for loop will run 5 times however will only spawn 4 obstacles in a row (one set) across the ground plane.

Line 33 shows the instantiate function being used to create a clone (spawn) of the obstacle in the scene. The parameters that the function takes are the object to spawn and information about where the object should be positioned. In this example, the second and third parameters of the function will be used to spawn an object at the spawn point position of the index at which the for loop is iterating through. This means that an object will be spawned at the position which the empty game objects called spawn points are at.

Test reference 2.5.1 – 2.5.3

Having performed the test cases specified, they succeed. The positions at which the obstacles are spawned are as expected. The obstacles spawn on the ground plane and in a row of four. Each time play mode was entered when carry out testing, the point at which no obstacles is spawned is different each time. This will make the game interesting because players will have to control the player game object to move it to different positions to avoid the obstacles. Furthermore, when the player game object collides with the obstacles the player in no long able to move. Now that the main concept of spawning obstacles has been coded and developed, I will now look at spawning the obstacles across the complete ground plane.

To spawn obstacles completely across the ground plane, the script that has been added as a component to the parent empty game object called ObstcaleSpawner, will be edited. The screenshot below shows the code that was added to the script to achieve this.



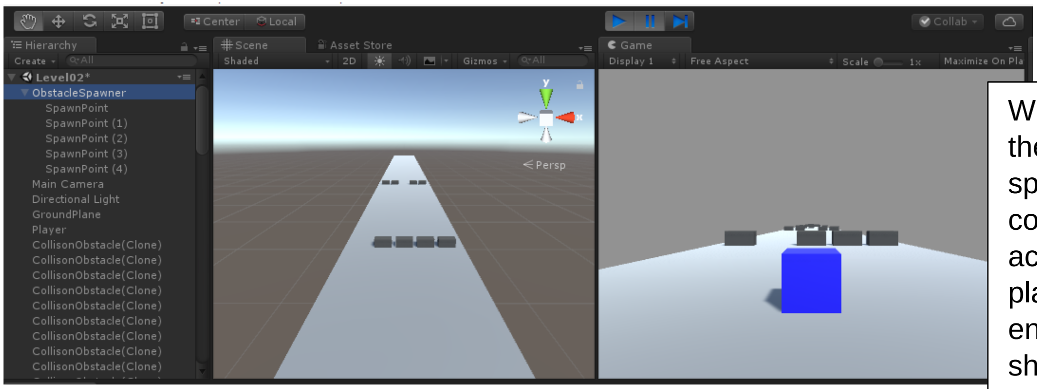
```
void Update()
{
    SpawnPoints();
    //If position of spawn points < length of ground plane subtract 100
    //then keep spawning obstacles
    if (transform.position.z < (groundPlane.transform.localScale.z - 100))
    {
        SpawnObstacles();
    }
}

//Moves position of the spawn points
void SpawnPoints()
{
    Vector3 moveSpawnPoints = new Vector3(0, 0, 40);
    //Moves position of the spawn points by adding 40 on the z axis
    transform.Translate(moveSpawnPoints);
}
```

The if statement is being used to determine if the obstacles should be spawned. If the position of the parent spawn point called ""ObstacleSpawner" is less than the length of the ground plane, then the SpawnObstacles subroutine will be called. A value of 100 is being subtracted from the length of the ground plane as the obstacles need to stop spawning before the end of the ground plane. This will also give player a clear indication that they have reached the end of the level.

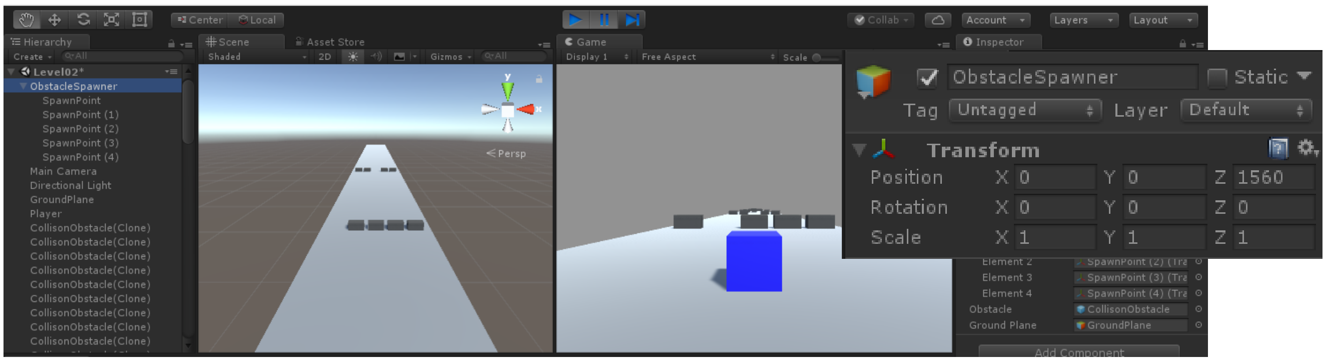
The SpawnPoints subroutine is been used to determine at what position the spawn points should be moved to once it has spawned each set of obstacles. The space between obstacles for the first level of the game was 40 along the z axis, therefore 40 is being added to the position of the spawn points. Since the spawn points are children of the "ObstacleSpawner" game object, if the transform (position) of the parent game object is affected then the transform of the children game objects will also change by the same amount. This means that is the position of the "ObstacleSpawner" game object changes by 40 on the z axis then the position of the spawn points will also change by 40 along the z axis. The transform.translate (translate is a function) is being used as it allow the spawn points position to be moved by 40 along the z axis.

The SpawnPoints subroutine is being called in the update function, because the spawn points position will need to be calculated every frame to allow the obstacles to be spawned across the ground plane.

Test reference 2.5.4



When in game mode the obstacles are spawned at the correct positions in across the ground plane. This is enough evidence to show that the initial test has succeeded.

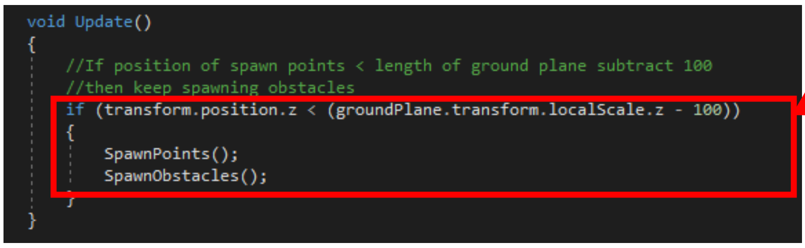

Level02*
ObstacleSpawner
SpawnPoint
SpawnPoint (1)
SpawnPoint (2)
SpawnPoint (3)
SpawnPoint (4)
Main Camera
Directional Light
GroundPlane
Player
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)

ObstacleSpawner Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 1560
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Element 2 SpawnPoint (2) (Tra
Element 3 SpawnPoint (3) (Tra
Element 4 SpawnPoint (4) (Tra
Obstacle CollisonObstacle
Ground Plane GroundPlane

The bug found from carry out this test does not affect the spawning of the obstacles but is related to calculating the position of the spawn point. The bug found is that once all the obstacles have been spawned across the ground plane, the position of the spawn point is still being calculated. As a result, the position of the spawn points along the z axis is greater than the length of the ground place (which is 1000 along the z axis) this should not be the case. Even once the player collides with an obstacle calculation are still be carried out to add to the position of the spawn points. It is important that this bug is fixed because this will result in wasted processing time when players play the game on their own computers.

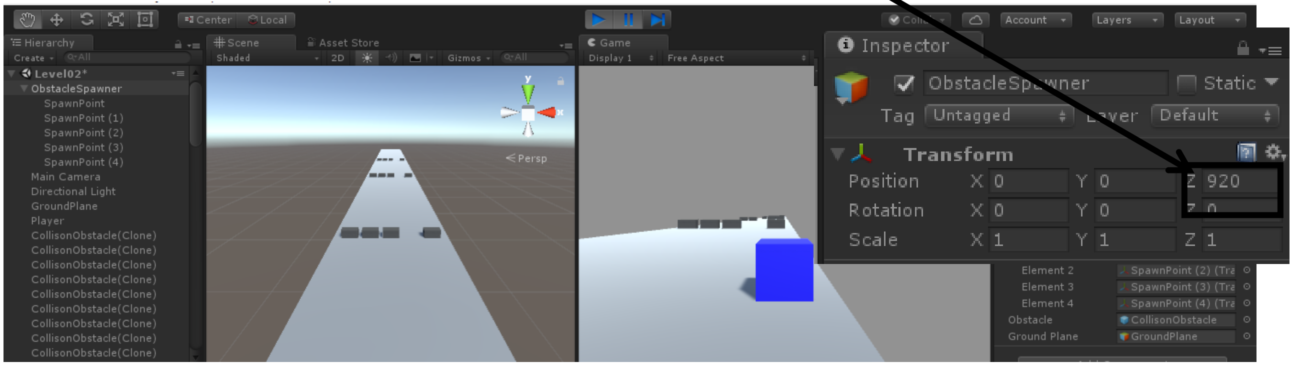
Having looked over the script, a solution to the bug would be called the SpawnPoint subroutine within the if statement. This is a suitable solution to the problem because if the position of the parent spawn point called "ObstacleSpawner" is less than the length of the ground plane, then the SpawnPoint and SpawnObstacles subroutines will be called. However, if the position of the parent spawn point called "ObstacleSpawner" is greater than the length of the ground plane then either of the subroutines will not be called and therefore no further calculations will take place.

Bug fix – Test reference 2.5.4


```
void Update()
{
    //If position of spawn points < length of ground plane subtract 100
    //then keep spawning obstacles
    if (transform.position.z < (groundPlane.transform.localScale.z - 100))
    {
        SpawnPoints();
        SpawnObstacles();
    }
}
```

Rather than calling the subroutine from outside of the if statement, it is being called within the if statement. As a result, if the position of the parent spawn point is greater than the length of the ground plane then either of the function will not be called and therefore no further calculations will take place.

The position of the spawn point now no longer exceeds the length of the ground plane which for this level is 1000 on the z axis.


Level02*
ObstacleSpawner
SpawnPoint
SpawnPoint (1)
SpawnPoint (2)
SpawnPoint (3)
SpawnPoint (4)
Main Camera
Directional Light
GroundPlane
Player
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)
CollisonObstacle(Clone)

Inspector
ObstacleSpawner Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 920
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Element 2 SpawnPoint (2) (Tra
Element 3 SpawnPoint (3) (Tra
Element 4 SpawnPoint (4) (Tra
Obstacle CollisonObstacle
Ground Plane GroundPlane

3.3 Testing

Sprint 2 has involved test references 2.1 – 2.5.4. Refer to the testing table for a summary of the tests. Unit Testing was carried out once the code was developed to: add control to the player controlled game object, detect obstacle collisions with the player game object obstacles and spawn obstacles in the scene. The test table show the tests that were carried out once the code for each feature had been developed (refer to tests 2.1 – 2.5.4).

Integration testing was carried out once each subtask of the sprint was completed. When carrying out integration testing for this sprint only test questions from 1.3.1.a were carried out. This is due to how time consuming it would have been to carry out all the test from the start and time has been identified as a limitation.

An example of when regression testing was carried out was when the bug discovered, when carrying out test with reference 2.3.2, was fixed. I carried out test references 2.2.1 – 2.3.2 again once the bug had been fixed. The results of carrying out these tests again was that there was no impact upon other features of that game that has been implemented in this sprint.

In this sprint I also carried out usability testing. The results can be seen in the feedback forms. Any issue that were found from carrying out usability testing using Prototype 2 have already be addressed at this point in development.

3.4 Sprint 2 Review

At the end of the 2nd sprint, I have created and positioned obstacles in the game for level one of the game. This fulfils success criteria 2.1. I have also developed code that detects obstacle and player game object collisions. If that player does collide with an obstacle, then the player is no longer able to control the game object. This fulfils success criteria 2.2. For level two (another game scene) of the game I have set the obstacles to spawn in the scene and at a random point no obstacle is spawned. This partially fulfils success criteria 2.4. It only partially fulfils the success criteria because all the obstacles are spawned when the game first starts not whilst the player is playing the game.

The first sprint aimed to fulfil success criteria with 1.1. – 1.3, however at the end of the sprint only success criteria 1.1 and 1.3. The start of this sprint focused on fixing the bug discovered by test reference 1.3.5. As test reference 1.3.5 has now succeeded, this fulfils success criteria 1.2. All test up to this point (up to test reference 2.5.4) have succeeded. This provides further evidence to show that success criteria 1.1 – 1.3, 2.1 and 2.3 has been fulfilled. As mentioned success criteria 2.4 has been partially fulfilled.

In this sprint I also created a prototype (CubeRun – Porotype 2) that was given to my stakeholders to carry out usability testing and provide general feedback. The results after gathering feedback and carrying out usability testing were generally positive with two major issues. Issues related to controlling the player controlled game object and to the background. Once the issues had been addressed another prototype (CubeRun – Porotype 2x1) was given to my stakeholders to carry out usability testing. The results after gathering feedback and carrying out usability testing for this prototype were also positive with no more issues.

This sprint has been successful as there are no issues with development. At this stage no modifications will need to be made to my solution (refer to the analysis section 1.5 My solution). Furthermore, using the prototypes that I have created the usability testing has been successful therefore I can carry on with fulfilling further success criteria.

4.0 3rd Sprint

4.1 Design

This sprint will further focus on spawning obstacles as I will start this sprint by fulfilling success criteria reference 2.4. Furthermore, this sprint will also focus on other game play elements and will aim to fulfil success criteria reference 2.5 – 3.2. To do this, I have broken down the sprint into individual subtasks:

1. Fulfil success criteria reference 2.4
2. Stopping the player falling from the ground plane
3. Game scoring UI
4. Implementing subtask two and three for level two

The sprint has been broken down into the following subtask because at the end of the second sprint it was identified that success criteria reference 2.4 was only partially fulfilled. This is because all the obstacles are spawned when the game first starts not whilst the player is playing the game. It would be important that this sprint starts by looking into fulfilling any success criteria that has not yet been achieved. If this sprint starts by looking at fulfilling success criteria reference 2.4 then all development related to spawning obstacles will be completed. This will mean that development can then be focused on developing solution to other features to be implemented into the game such as a scoring system.

Furthermore, the sprint has been broken down into the following subtasks because the player not falling from the ground plane and score UI are features of the game that all levels created will have. This means that it is important that a solution is implemented to add the features into the game before any more levels are created.

As there are now two scenes in the game (scenes called level one and level two) once success criteria reference 2.4 has been achieved any further development will take place back in scene one (level one). This will mean that once features have been developed in scene one, the same features will need to be replicated and tested in scene two (level two).

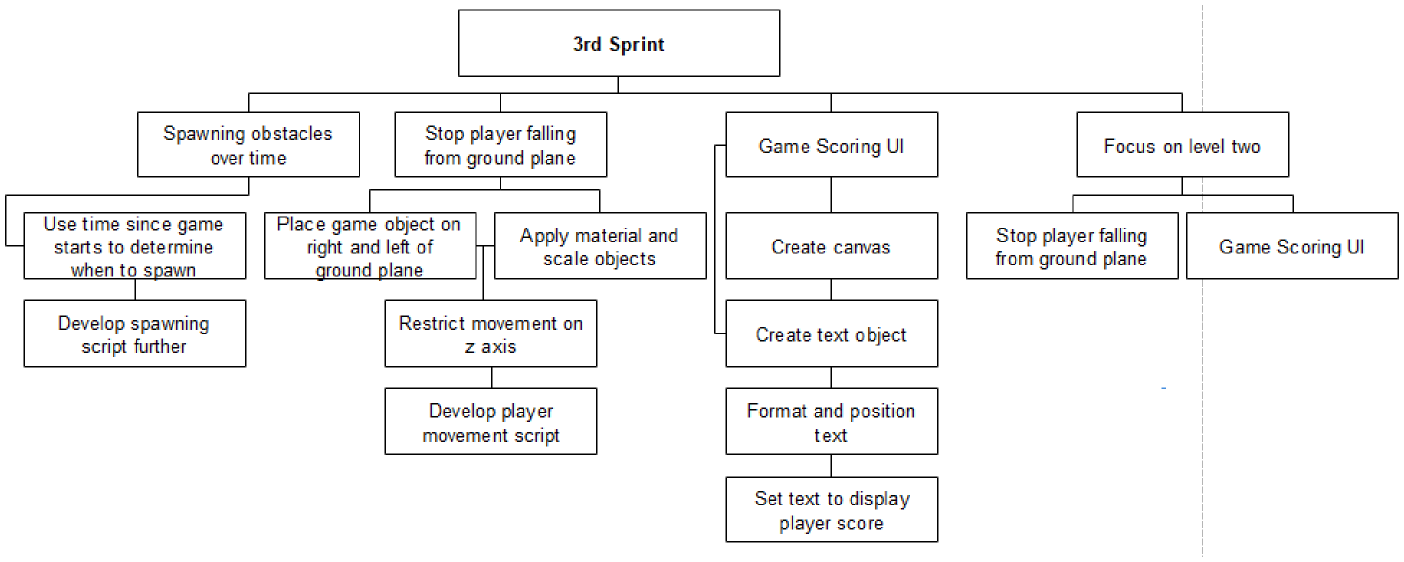

3rd Sprint
Spawning obstacles over time
Stop player falling from ground plane
Game Scoring UI
Focus on level two
Use time since game starts to determine when to spawn
Develop spawning script further
Place game object on right and left of ground plane
Apply material and scale objects
Restrict movement on z axis
Develop player movement script
Create canvas
Create text object
Format and position text
Set text to display player score
Stop player falling from ground plane
Game Scoring UI

Figure 7: Top-down design for Sprint 3

4.1.1 Spawning obstacles over time

I have developed the code required to spawn the obstacles, however all the obstacles are being spawned as soon as the game starts. This does not need to happen because player will progress through the level, therefore each set of the obstacles can be spawned after a certain amount of time. By not having all the obstacles being spawned as soon as the game starts will mean that less processing time will be required when first starting the game.

I will be redesigning the algorithm that was produced to spawn the obstacles. The algorithm below shows how the obstacles will be spawned over time (text in blue show the changes that have been made):

```
private float timeToSpawn = 0
private float timeBetweenSpawns = 1f
array pointsToSpawn [ ]
public GameObject obstacle
function Update ( )
        if (spawn point position on z axis < (length of the ground plane - 100)) THEN
                if (time since game start >= timeToSpwn)
                        SpawnPoints()
                        SpawnObstacles( )
                        timeToSpawn = time since game start + timeBetweenSpawns
                end if
        end if
end function
procedure SpawnPoints( )
        vector 3 moveSpawnPoints = NEW vector 3 (x, y, z)
        move position of spawn point on axis by amount moveSpawnPoints
end procedure
procedure SpawnObstcales( )
        int randomPoint = random number between 0 and pointsToSpawn.length
        int i = 0
        for i < pointsToSpawn.length
                if (randomPoint != i) THEN
                        Place obstacle game object at position
                        i = i + 1
                end if
end procedure
```

The changes that have been made to the algorithm are within the update function. Nested if statements are being used. If the expressions for the first if statement is true then, then it will check the next if. If the expressions for next if statement is also true, then the two procedures are called. For more details on the two procedures refer to section 3.2.2.

Nested if statements are being used because more obstacles only need to be spawned, if the position of the spawn points is less than the length of the ground plane and if the time since the game has started (in seconds), is greater than the time at which the obstacles should be spawned. If the expression of the first if statement was not being asses first, then the second is statement would result in an infinite loop since the time will keep increasing.

Variable Name	Type	Explanation
timeToSpawn	Float	Variable will store time at which the obstacles should be spawned. The value of this variable will change because the time the game has been running for will increase. Will be declared as private because this value does not need to be accessed by other scripts.
timeBetweenSpawns	Float	Variable will store the time at which the next set of obstacles should be spawned. The variable has been initialised to one because the time between each set of obstacles spawning will be one second. Will be declared as private because this value does not need to be accessed by other scripts.

4.1.2 Stopping the player from falling from the ground plane

Currently the player can use the arrow keys to control the player. However, the player game object can to controlled so that if falls from the ground plane. If a solution to this problem is not implemented and the player controlled game object did fall from the ground plane, then they will have to reopen the game window on their PC to play the game if they wanted to.

Many solutions to the problem were consider. One solution, involved allowing the player game object to fall from the ground and then respawning the player controlled game object into the scene to allow players to start from the beginning of the level. An alternative solution to the problem that was to add to two game objects (that have a box collider attached to them) to the left and right edge of the ground plane. This way, if players collide with the game objects it will stop them from falling off the ground plane and they will be able to continue playing where they collided. Another final solution was, to restrict how much players can move on the x axis when the position to move the player game object is calculated.

All these solutions are appropriate however, I will consider implementing solution two and three – using the game objects on either side of the ground plane and restricting player movement on the x axis. These are the two solutions that are being considered because, as identified in the analysis the stakeholders will be look for a quick game experience. By respawning the player into the scene to allow players to start from the beginning of the level will mean that players might not be able progress to the more challenging levels of the game as they would be spending more time on a certain level.

The two game objects placed on either side of the ground plane will act as barriers as they will warn players that they cannot exceed past them. To create the two game objects or barriers, a cube game object will be used. The dimensions that will be used will be determined by the length on the z axis of the ground plane. For scene two (level two) of the game, the ground plane has a greater length. As a result, the length of then barrier will need to be greater.

The colour of the game object will be red. An alternative option would be to use an image that will wrap around, however this will distract the players attention and will not look as appealing as using a solid colour. The red colour will be applied to the objects because players will see them are they progress through the level. Due to it being a solid colour, it will not cause any visual problems for the players and they will be able to focus on the aim of the game.

To restrict the player controlled game object form how much it can move on the x axis when the position to move the it is calculated I will be further looking at the player movement script that has been attached to the player object. I will be redesigning the algorithm that was produced to fixed bug reference 1.3.5. In practical I will be looking at the MovePlayer procedure that was designed. The algorithm below shows how the movement on the x axis will be restricted (text in blue show the changes that have been made):

```
private Rigidbody rb
public float speed
function start ()
        rb = RETURN reference to Rigidbody
end function
function fixed update ()
        UserInput ()
        MovePlayer ()
end function
function UserInput ()
        float moveAlongX == INPUT direction of movement
        return moveAlongX
end function
procedure MovePlayer()
        Move Position of rb forward on z axis
        newPosition = Current position + move right * UserInput ()
        newPosition.x = Limit to maximum and minimum on x axis
        if UserInput () > 0 THEN
                move player game object potion to the right
        else if UserInput () < 0 THEN
                move player game object position to the left
end procedure
```

The changes that have been made are the way the user input is used to determine where to move the player game object. A variable called newPosition will be used to store the calculated value for which the player will need to be moved in three dimensions (x, y, z).

The second highlighted blue line of the algorithm will solve the problem to restricting the player movement of the x axis. This is because it will limit the position relative to the x axis that is calculated by the newPosition variable. The limits (a maximum and minimum from the zero position) will be specified when developing the code since they will need to be tested to see which stop the player from falling from the ground plane. These limits will be hard coded because the length of the ground plane in the x direction will be the same for all levels that will be created.

Variable Name	Type	Explanation
newPosition	Vector 3	Calculate the position to which the player game object should be moved in the scene considering it current position. It will be initialised within the procedure as the variable will not need to be accessed by subroutines or classes. Of type vector 3 because it will store position in three dimensions. *All other variables described in section 3.1.1*

4.1.3 Game scoring UI

Before development of an algorithm that will help implement game score UI, it will be important to decide upon what will the score depend on. The score can either depend on how many sets of obstacles the player passes, or it can depend on the distance reached by the player game object. Since the aim of the game is to progress through reaching as far as possible without colliding with an obstacle it will be suitable for the score to depend on the distance reached by the player.

To allow the score to be displayed to the user a user interface game object will be used. A text UI game object will be created in the scene as this will allow text to be render onscreen. The algorithm bellow shows how the text game object will be used to show the player score.

```
public Transfrom playerPoint
public Text gameScore
function fixed update ( )
        gameScore = “SCORE” + covert player position on z axis to string
end function
```

In the algorithm the game score is being set/changed in the update function. This is because the update function is called every fame, and since the players position will constantly be changing the score text will also need to changed corresponding. As the players position on the z axis is a float value it will need to be converted to a string to allow it to be displayed as text to the players.

For future development, this algorithm will need to be developed further. As there will be levels in the game, this algorithm does not look at carrying the score from one level to another.

Variable Name	Type	Explanation
playerPoint	Object	Variable will store the position of the player in the scene. This position on the z axis of the player controlled game object will be used to determine the score achieved by the player. Variable will be declared as public to allow a reference to the object to be set in the Unity Inspector.
gameScore	Object	Variable will store a reference to the text object that needs to be updated as the player position changes. Variable will be declared as public to allow a reference the object to be set in the Unity Inspector.

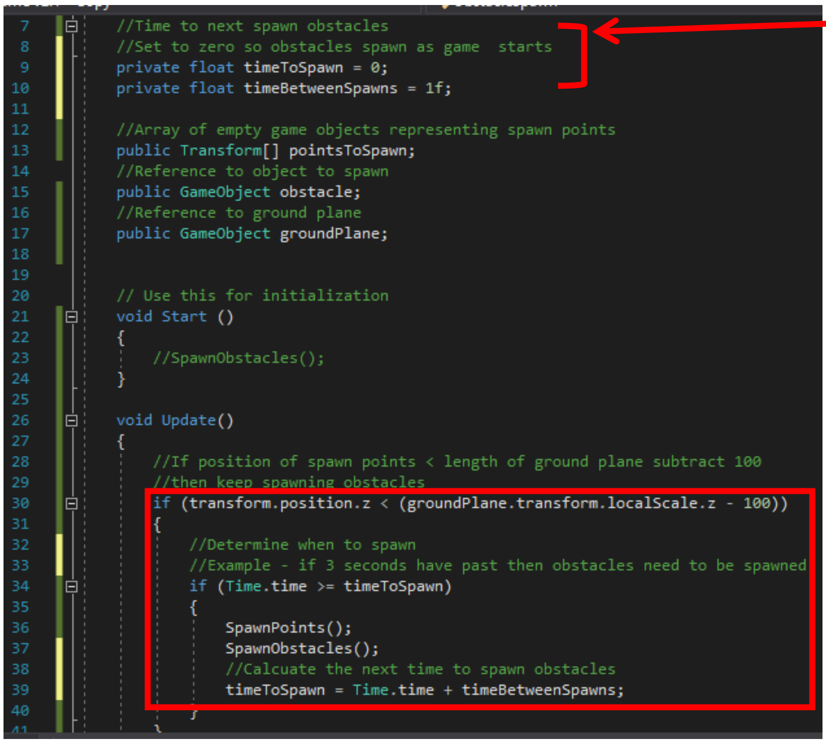
The format of the text will not be changed a lot when carry out development of this feature. Once a stage in development has reached where it is suitable to give a prototype to my stakeholders to test, questions will be asked about how they would like the test to be formatted.

4.2 Development

4.2.1 Spawning obstacles over time

As explained in the design of this section (in 4.1.1) currently in the game all the obstacles are been spawned across the ground plane for the level as soon as the game starts. This not desirable because players will progress through the level over time, therefore the obstacles do not need to be spawned all at once. Furthermore, by spawning all the obstacles at the start will means more processing time will be needed when starting the game.

Therefore, the solution to the problem will be to spawn the obstacles at a certain time. To develop the code required to do this I will continue to edit the C# script called “ObstcaleSpawn” which is attached to the parent empty game object.



```
 7      //Time to next spawn obstacles
 8      //Set to zero so obstacles spawn as game  starts
 9      private float timeToSpawn = 0;
10      private float timeBetweenSpawns = 1f;
11
12      //Array of empty game objects representing spawn points
13      public Transform[] pointsToSpawn;
14      //Reference to object to spawn
15      public GameObject obstacle;
16      //Reference to ground plane
17      public GameObject groundPlane;
18
19
20      // Use this for initialization
21      void Start ()
22      {
23          //SpawnObstacles();
24      }
25
26      void Update()
27      {
28          //If position of spawn points < length of ground plane subtract 100
29          //then keep spawning obstacles
30          if (transform.position.z < (groundPlane.transform.localScale.z - 100))
31          {
32              //Determine when to spawn
33              //Example - if 3 seconds have past then obstacles need to be spawned
34              if (Time.time >= timeToSpawn)
35              {
36                  SpawnPoints();
37                  SpawnObstacles();
38                  //Calcuate the next time to spawn obstacles
39                  timeToSpawn = Time.time + timeBetweenSpawns;
40              }
41          }
```

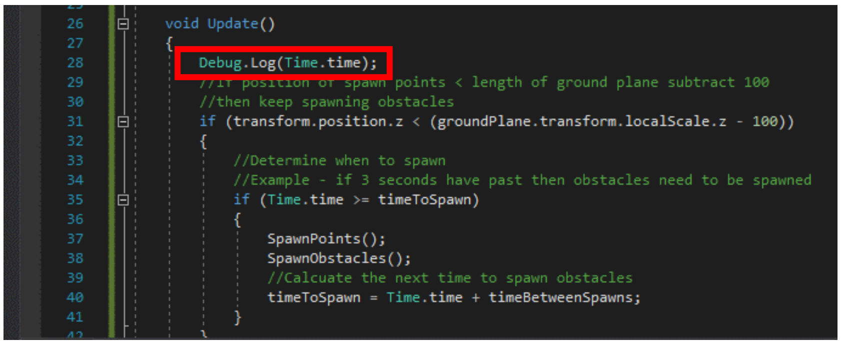
Following the algorithm for spawning obstacles over time, development started by initialising variables for the time in seconds at which the obstacles should spawn (timeToSpawn) and the time between spawning each set of four obstacles (timeBetweenSpawns). Variables have been declared as private because they do not need to be edited in the Unity inspector. The timeToSpawn variable has been set to zero because the first set of obstacles will need to spawn as soon as the game starts. This value will change, as the time since player first start to play the game increases.

A nested if statement is being used to determine when to call the subroutine that spawns the obstacles in the scene. The first if statement determines if the position of the parent spawn point called “ObstacleSpawner” is less than the length of the ground plane. If this is the case the Boolean expression is true and then the next Boolean expression will be evaluated. A nested if statements is being used because obstacles only need to be spawned across the ground plane for the size it is along the z-axis and at a certain time since the game started.

The next if statement determines if the time in seconds since the game was first run/started is greater than the time in seconds at which the obstacles should spawn. If this is the case the Boolean expression is true and therefore the subroutines SpawnPoints and SpawnObstcales will be called.

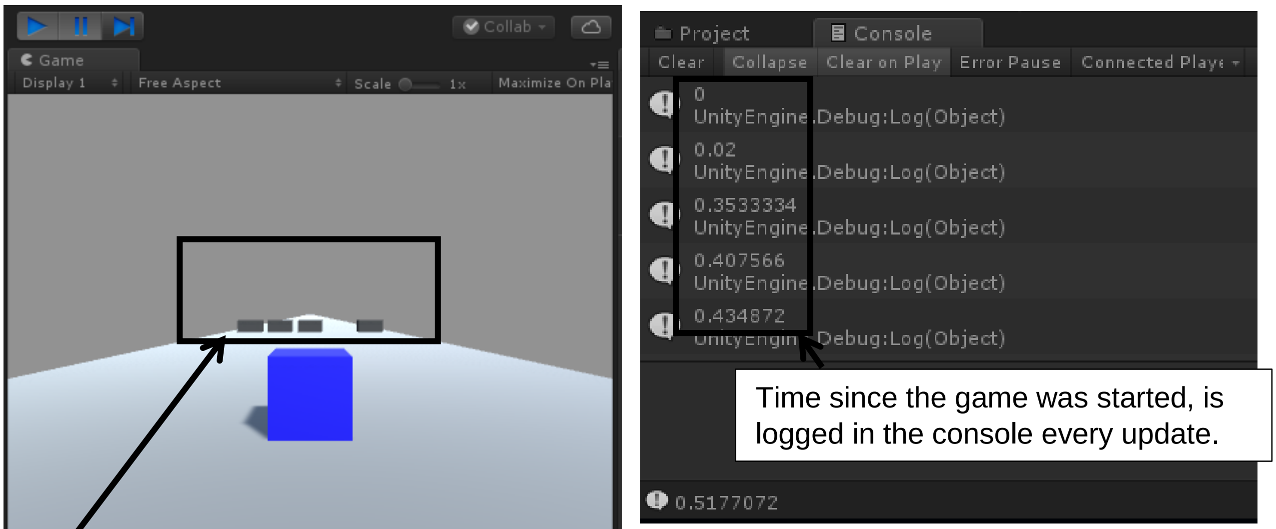
Line 39 of the code calculates the time at which the next set of four obstacles needs to be spawned and stores this value as the timeToSapwn variable. Essential, it is responsible for changing the amount of time in seconds at which the obstacles should spawn after a set obstacle have been spawned. For example, if it has been 1 seconds after the game has started, the timeToSapwn value was original 0, line 39 will set the timeToSapwn to 1 since 1 + 0 = 1. This loop will keep running until the expression of the first if statement is false.

Test reference 3.1.1 and 3.1.2



```
26        void Update()
27        {
28            Debug.Log(Time.time);
29            //if position of spawn points < length of ground plane subtract 100
30            //then keep spawning obstacles
31            if (transform.position.z < (groundPlane.transform.localScale.z - 100))
32            {
33                //Determine when to spawn
34                //Example - if 3 seconds have past then obstacles need to be spawned
35                if (Time.time >= timeToSpawn)
36                {
37                    SpawnPoints();
38                    SpawnObstacles();
39                    //Calcuate the next time to spawn obstacles
40                    timeToSpawn = Time.time + timeBetweenSpawns;
41                }
```

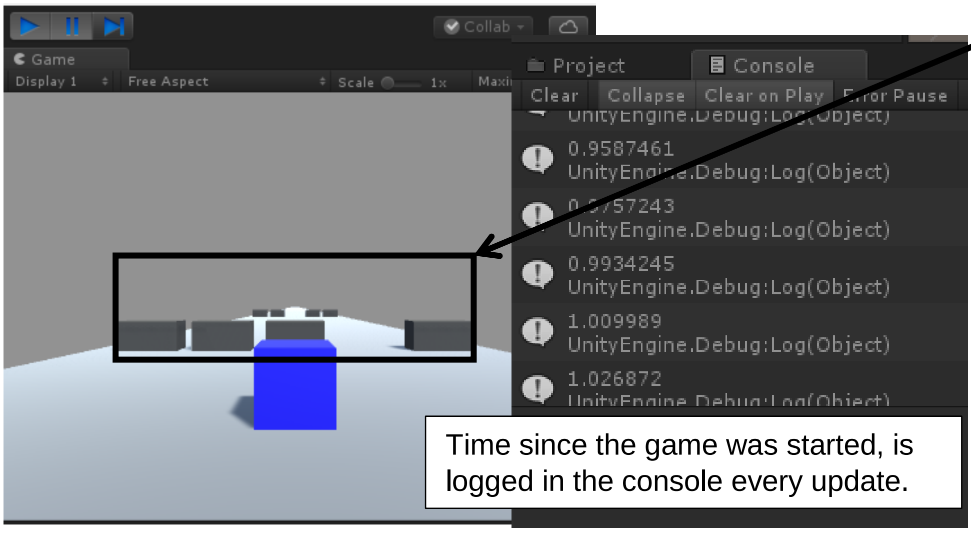
To carry out the following sets of tests I added code (Line 28) to the Update function temporally. This line of code will help carry out the tests with the following test reference 3.1.1 and 3.1.2 because its logs a message to the Unity Console. The message that will be seen in the Unity Console is the parameter that is passed to the Log function – in this case the time since the game was first run. This will help with testing because I will be able to see how the obstacles spawn over time.



```
0
UnityEngine.Debug:Log(Object)
0.02
UnityEngine.Debug:Log(Object)
0.3533334
UnityEngine.Debug:Log(Object)
0.407566
UnityEngine.Debug:Log(Object)
0.434872
UnityEngine.Debug:Log(Object)

0.5177072
```

Time since the game was started, is logged in the console every update.

When the game first starts the first set of four obstacles are spawned – the obstacles are not spawned for the whole level across the ground plane. This is enough evidence to show that the test 3.1.1 succeed since if only spawn on set of obstacles at the correct position.

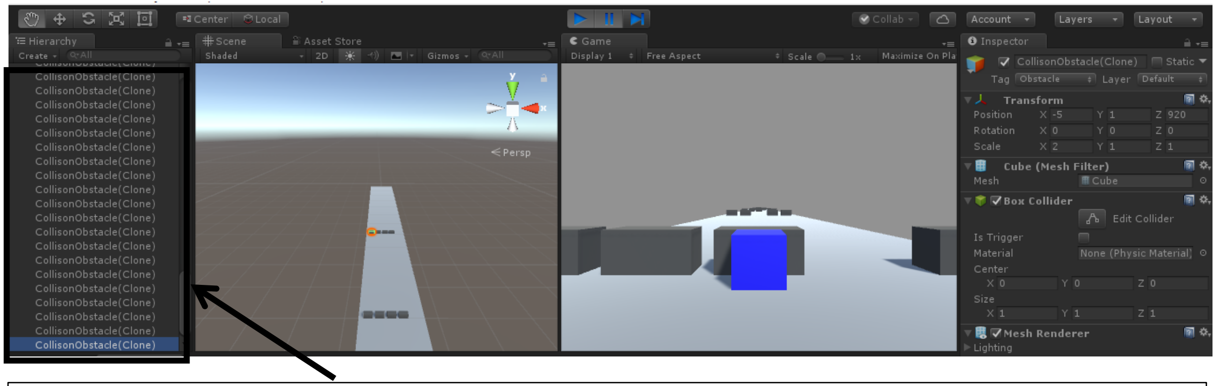


```
UnityEngine.Debug:Log(Object)
0.9587461
UnityEngine.Debug:Log(Object)
0.9757243
UnityEngine.Debug:Log(Object)
0.9934245
UnityEngine.Debug:Log(Object)
1.009989
UnityEngine.Debug:Log(Object)
1.026872
UnityEngine.Debug:Log(Object)
```

Time since the game was started, is logged in the console every update.

After one seconds of starting to play the game, the second set of obstacles are spawned. This provides further evidence that test 3.1.1 succeed. The obstacles are continually spawned with time until obstacles have been spawned for the length of the ground plane.

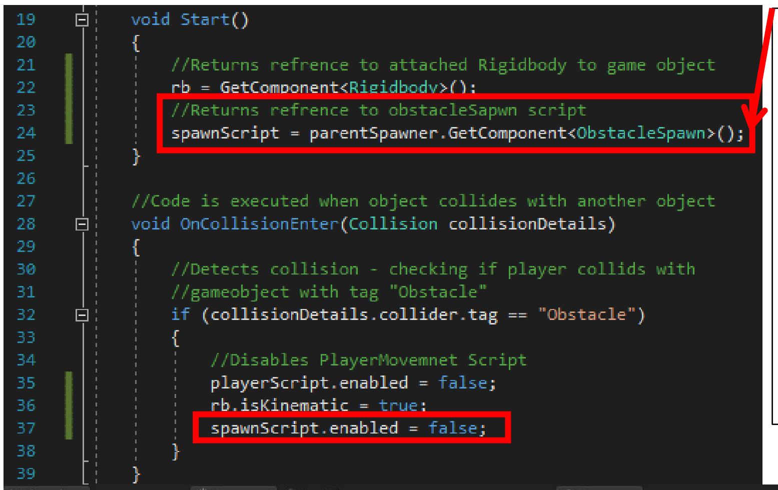
Test reference 3.1.3



When the player collides with and any of the spawned obstacles, the player is no longer able to control the game object. However, once the player and obstacles have collided the obstacles continue to spawn until they spawned for the complete length of the ground place. There is no point in spawning the remaining obstacles for the complete ground plane once the player has collided with the obstacles because the game will end.

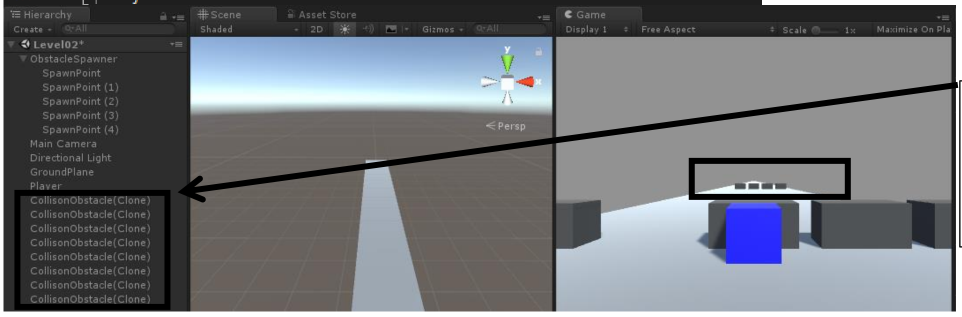
The above bug that I have found relates to continuous spawning even once the player controlled game object collides with an obstacle. This bug will need to be fixed because any obstacles that are spawned after a collision will be taking up processing time and memory on player computers. A solution to this problem would be disable the script that spawns the obstacle when the player and obstacles collides. This is a suitable solution to the problem because once players collide with an obstacle they will no longer be able to play the game and therefore no more obstacles will need to be spawned. As the script for spawning obstacles is attached to another game object I will need to create a reference to that game object in the collision Detect script.

Bug fix – test reference 3.1.3



```
void Start()
{
    //Returns refrence to attached Rigidbody to game object
    rb = GetComponent<Rigidbody>();
    //Returns refrence to obstacleSapwn script
    spawnScript = parentSpawner.GetComponent<ObstacleSpawn>();
}

//Code is executed when object collides with another object
void OnCollisionEnter(Collision collisionDetails)
{
    //Detects collision - checking if player collids with
    //gameobject with tag "Obstacle"
    if (collisionDetails.collider.tag == "Obstacle")
    {
        //Disables PlayerMovemnet Script
        playerScript.enabled = false;
        rb.isKinematic = true;
        spawnScript.enabled = false;
    }
}
```

The Collision Detect script was edited to fix this bug. Another variable called parentSpawner has been initialised to allow that "ObstacleSpawner" game object to be reference in this script. Creating this reference will allow the script called "ObstacleSpawn" to be referenced and then once the player collides with an obstacle the script will be disabled, and obstacles will no longer spawn.



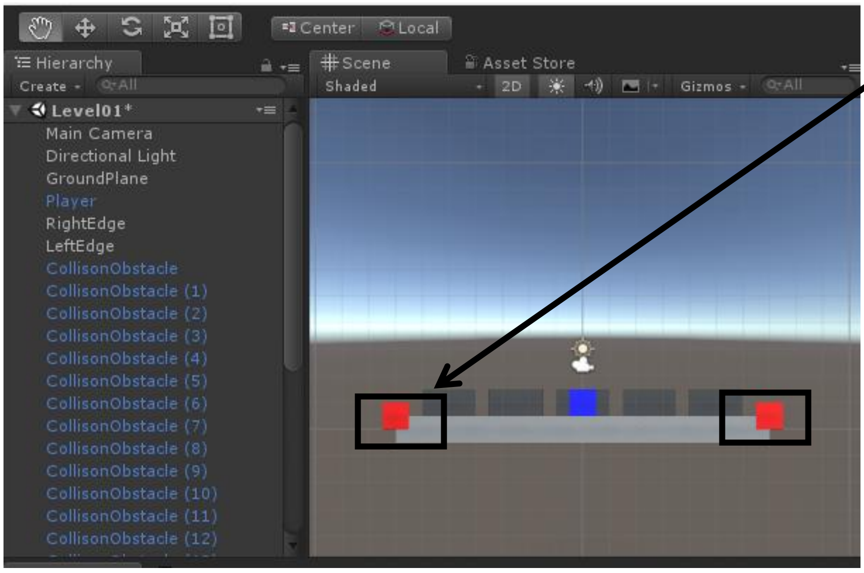
Now when the player collides with an obstacle, no more obstacles are spawned.

NOTE: Any development from this point onwards will be in scene one (level one). It will then need to be copied across to the scene two (level two).

4.2.2 Stopping the player falling from ground plane

The development of this subtask will start by implementing solution one (as described in the design section 4.1.2) to try and stop the player from falling from the ground plane. However, if solution one does not work then the algorithm designed will be followed to stop the player from falling from the ground plane.

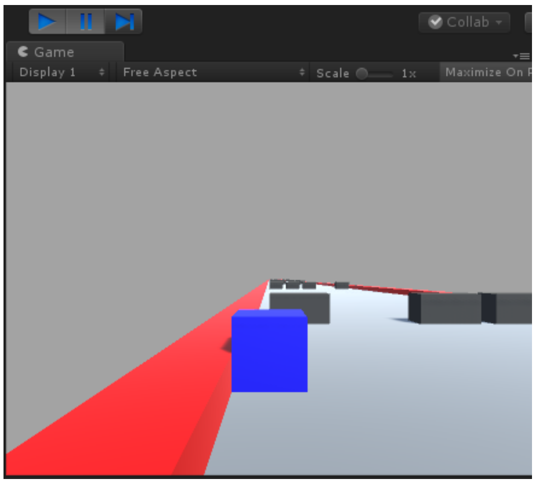
Solution one involves using two additional game objects that will be placed on either side of the ground plane. This is a possible solution to the problem because if the player collides with these game objects it should stop it from falling from the ground plane. A cube game object will be created, and it will be scaled to the size that has been specified in the design section 4.1.2. A red colour will also be applied to this game object.

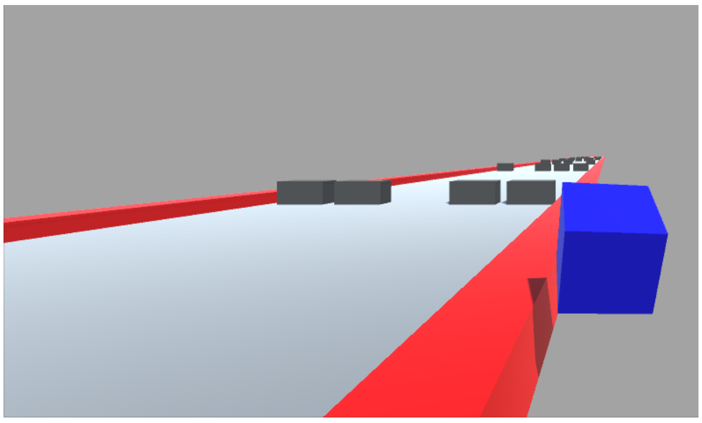


Level01*
Main Camera
Directional Light
GroundPlane
Player
RightEdge
LeftEdge
CollisonObstacle
CollisonObstacle (1)
CollisonObstacle (2)
CollisonObstacle (3)
CollisonObstacle (4)
CollisonObstacle (5)
CollisonObstacle (6)
CollisonObstacle (7)
CollisonObstacle (8)
CollisonObstacle (9)
CollisonObstacle (10)
CollisonObstacle (11)
CollisonObstacle (12)

This a 2D view of the level one scene of the game. These are the two game objects that have been added to the scene. Since these game objects have a box collider, now when the player collides with these game objects it should stop the player from falling from the ground plane.

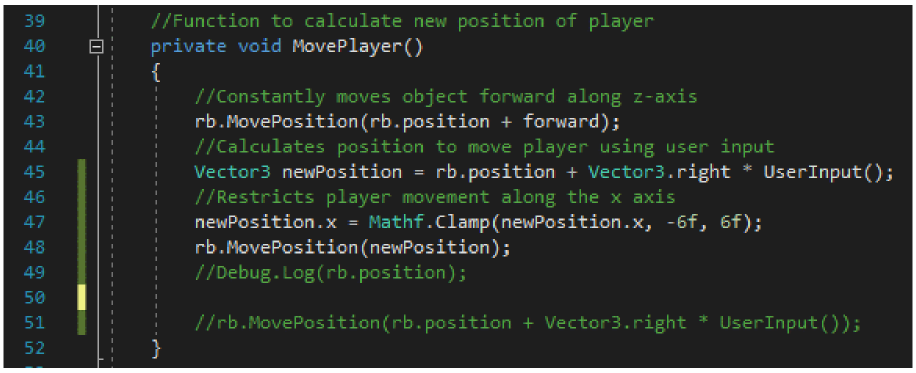
Test reference 3.2.1 and 3.2.2





When the player collides with the red game object it does stop the player from falling from the ground plane. However, if the player keeps hold of either the left or right arrow key the player controlled game object move through the red game on either side and falls of the ground plane.

As test reference 3.2.2 failed, before carrying out any further tests, the algorithm designed to stop the player from falling from the ground plane will be followed. To fix the bug described the PlayerMovemnet script that has been attached to the player controlled game object will be edited.



```
39          //Function to calculate new position of player
40          private void MovePlayer()
41          {
42              //Constantly moves object forward along z-axis
43              rb.MovePosition(rb.position + forward);
44              //Calculates position to move player using user input
45              Vector3 newPosition = rb.position + Vector3.right * UserInput();
46              //Restricts player movement along the x axis
47              newPosition.x = Mathf.Clamp(newPosition.x, -6f, 6f);
48              rb.MovePosition(newPosition);
49              //Debug.Log(rb.position);
50
51              //rb.MovePosition(rb.position + Vector3.right * UserInput());
52          }
```

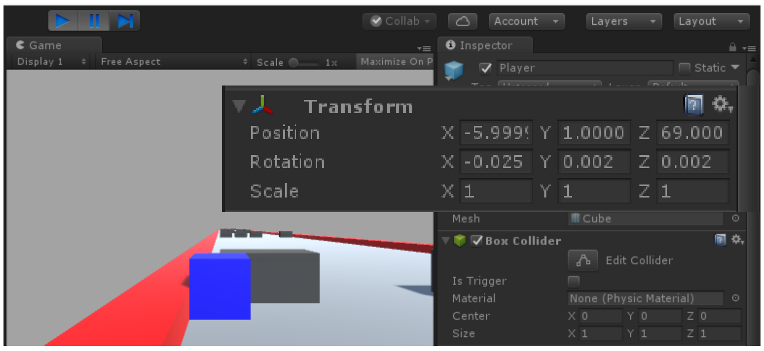
The only code that has changed is within the MovePlayer function.

Having followed the algorithm that was designed in the design section 4.1.2, the screenshot above shows the code that has been developed to add control the player controlled game object and to fix the bug described by test reference 3.2.2.

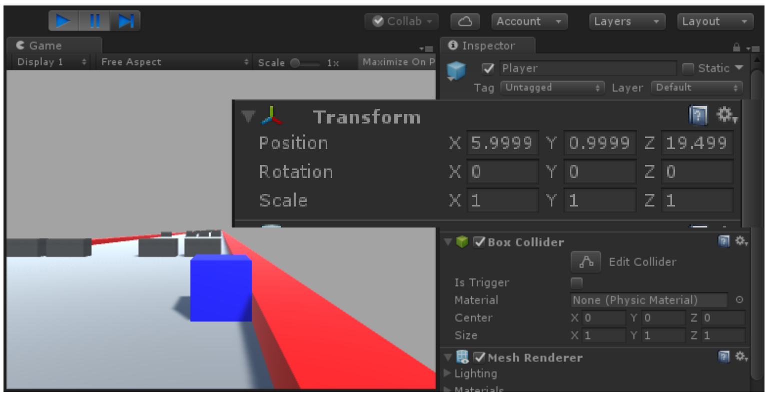
Line 45 store the calculated value for the position to which the player will be moved. It is stored as a vector 3 because it stores values of the position in three dimensions (x, y, z). For more details on this line refer to the development section 3.2.1 fixing bug described by test reference 1.3.5.

Line 47 of the code restricts the player movement along the x axis. It uses the maths function to clamp (limit) the value between a minimum value and a maximum value. In this case the calculated position in the x direction is the value that neds to be limited, therefore it is being as a parameter to the function. The other two parameters that are passed to the maths function are the minimum value and the maximum value. These values will be the limiting value because the player position along the x axis will not got below or exceed these.

Test reference 3.2.2 and 3.2.3



```
Transform
Position    X -5.999(   Y 1.0000   Z 69.000
Rotation    X -0.025    Y 0.002    Z 0.002
Scale       X 1         Y 1        Z 1
```

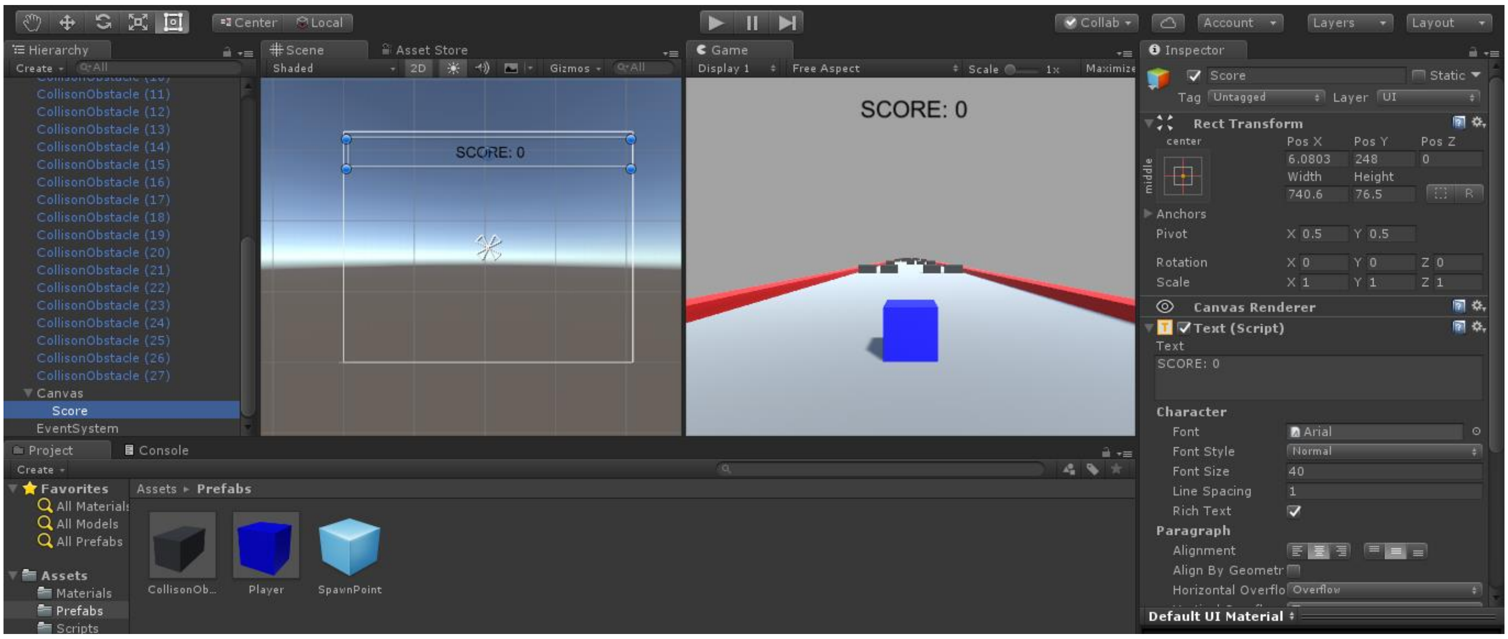


```
Transform
Position    X 5.9999    Y 0.9999   Z 19.499
Rotation    X 0         Y 0        Z 0
Scale       X 1         Y 1        Z 1
```

When the left key was held down the players position does not go below -6. This test also tests the robustness of the solution because the left key was held down even once the players position had been moved to the left and it still did not go below -6. When the right key was held down the players position did not exceed 6. This is enough evidence to show that the tests succeed since the player controlled game object does no longer fall from the ground plane.

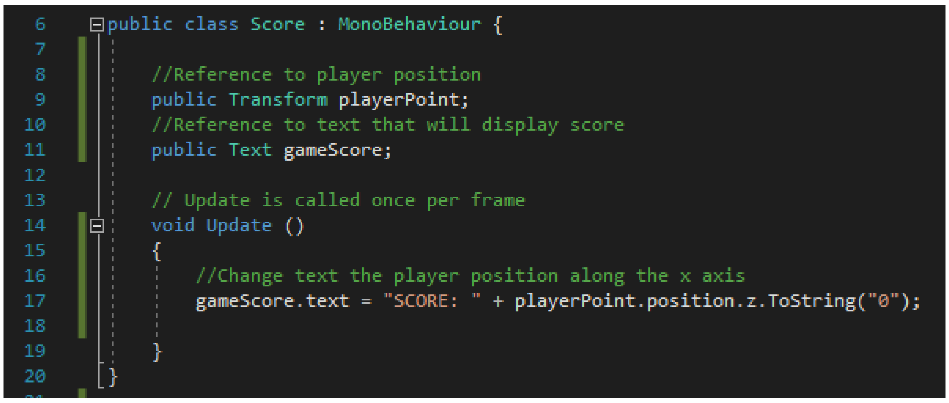
4.2.3 Game Scoring

The score that the player achieves will depend upon how far they reach each level. This means that the score will depend on the position that the player game object is along the z axis. Having followed the algorithm that was designed in the design section 4.1.3, the screenshot below shows the code that has been developed to calculate and output the score to the player.



Score
Static
Tag Untagged
Layer UI
Rect Transform
center
Pos X 6.0803
Pos Y 248
Pos Z 0
Width 740.6
Height 76.5
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas Renderer
Text (Script)
Text
SCORE: 0
Character
Font Arial
Font Style Normal
Font Size 40
Line Spacing 1
Rich Text ✓
Paragraph
Alignment
Align By Geometry
Horizontal Overflow Overflow
Default UI Material

Before any code was developed to score the game, a user interface game object was created to allow text to be displayed on the screen. The properties such as the font size, text alignment and text position, were all set before moving onto development. The text properties will need to test for usability by my stakeholders to make sure that they are suitable for them. To change the text to the show the distance that the player has reached (the score) a C# will be attached to this text object called score.

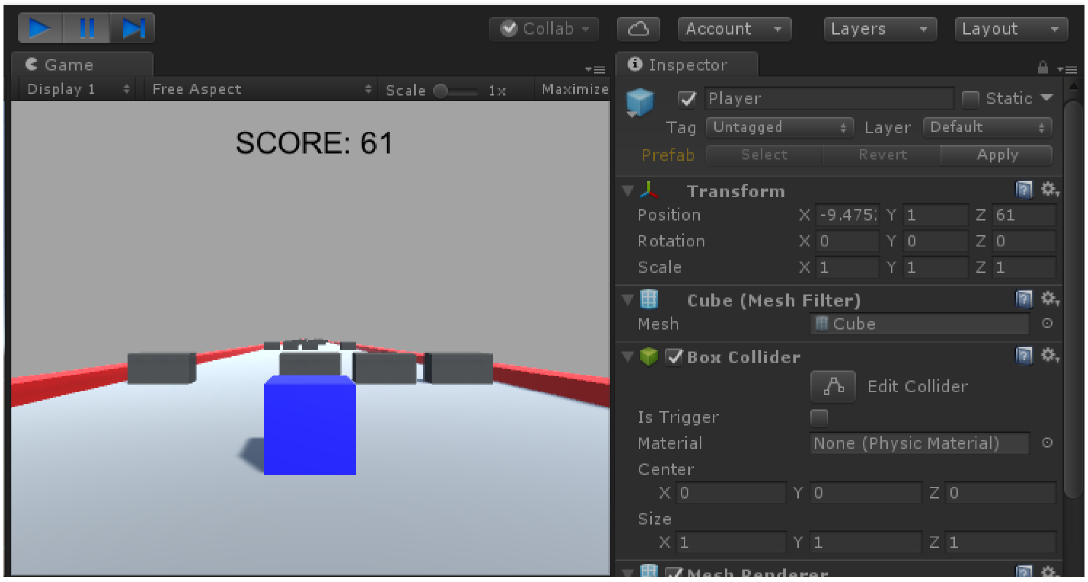


```
 6  public class Score : MonoBehaviour {
 7  
 8      //Reference to player position
 9      public Transform playerPoint;
10      //Reference to text that will display score
11      public Text gameScore;
12  
13      // Update is called once per frame
14      void Update ()
15      {
16          //Change text the player position along the x axis
17          gameScore.text = "SCORE: " + playerPoint.position.z.ToString("0");
18  
19      }
20  }
```

Following the algorithm, development started by initialling variables that will store a reference to the player position in the scene. Another variable was initialised to store the text which will be responsible for displaying the users score.

Line 17 of the code allow the text to be changed to the show the distance that the player has reached which will be an indication for the score on the game. The text is being changed to the integer value of the player controlled game object position on the z axis. It must be converted to a string because the position on the z axis is a float value however the text is a string, so the float value needs to be converted into a sting.

Test reference 3.3.1

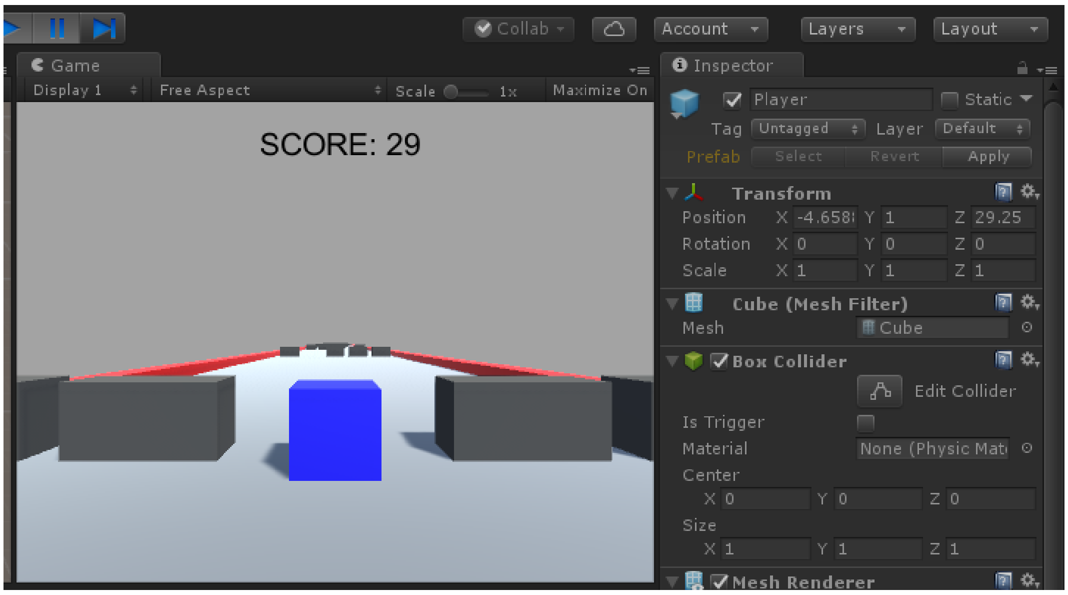


```
SCORE: 61

Player                         Static
Tag  Untagged     Layer  Default
Prefab   Select   Revert   Apply
Transform
Position   X -9.475:  Y 1   Z 61
Rotation   X 0        Y 0   Z 0
Scale      X 1        Y 1   Z 1
Cube (Mesh Filter)
Mesh       Cube
Box Collider
           Edit Collider
Is Trigger
Material   None (Physic Material)
Center
  X 0      Y 0      Z 0
Size
  X 1      Y 1      Z 1
Mesh Renderer
```

The screenshot above shows test reference 3.3.1. As the players position on the z axis increase the score also increase to the same amount. In the screenshot the position on the z axis is 61 and the score that is displayed to the players is also 61. This is enough evidence to show that the test succeed since the correct score is displayed to the player.

Currently, the scoring system has only been developed for level one. At this point in development there is no way of winning the level. If the player reaches the end of the level that player game object continues to move forwards on the z axis, this means the score continues to increase. Once development of winning a level has been completed I will need to further develop the script that was created for the game score.

Test reference 3.3.2



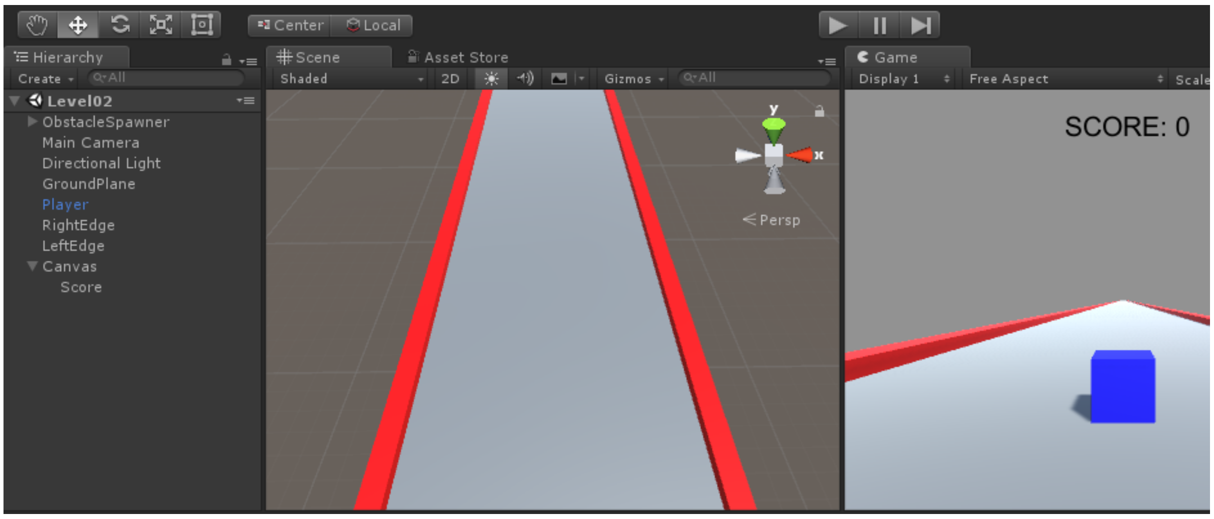
```
SCORE: 29

Player                         Static
Tag  Untagged     Layer  Default
Prefab   Select   Revert   Apply
Transform
Position   X -4.658!  Y 1   Z 29.25
Rotation   X 0        Y 0   Z 0
Scale      X 1        Y 1   Z 1
Cube (Mesh Filter)
Mesh       Cube
Box Collider
           Edit Collider
Is Trigger
Material   None (Physic Mat
Center
  X 0      Y 0      Z 0
Size
  X 1      Y 1      Z 1
Mesh Renderer
```

The screenshot above shows test reference 3.3.2. When the player position on the z axis is a non-integer value, the value is rounded either up or down to only show the score as an integer.

4.2.4 Implementing subtask two and three of this sprint for level two

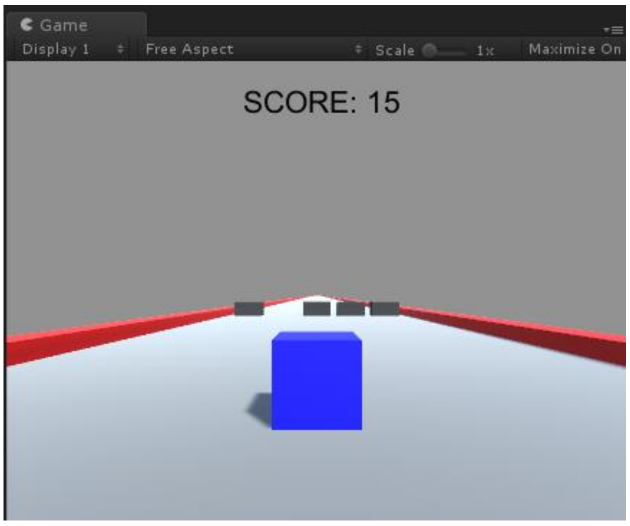
As there are now two scenes in the game that are under development, any development that takes place in scene one (level one) needs to also take place in scene two (level two). This means that I will need to also cerate the two additional game objects that will be placed on either side of the ground plane and the score canvas for scene two.

No programming needs to take place because to stop the player game object falling from this side the PlayerMovemnet script was edited. This script is attached to the player game object which is in both scenes therefore any changes made to the script will already be applied to the game object. Furthermore, no programming needs to take place for the game score because copying score canvas from scene one to scene two should copy all the objects components which includes the C# script.



The two additional game objects and the score canvas have been copied from level one to level two the implement these features in the scene. The only properties that was changed from those pervious specified was the game object scale on the z axis. Since the length of the ground plane for level two in greater the game object had to be scaled so that they appear for the complete length of the ground plane.

Test reference 3.4.3



After playing the first level, the score that was achieved, was not the score that level two started on. This is not a bug or an error that needs to be fixed but a solution needs to be implemented to allow this to happen. The score achieved in level one of the game needs to be set to the score that players start with when they have completed the first level.

4.3 Testing

Sprint 3 involved test references 3.1.1 – 3.4.3. Refer to the testing table for a summary of the tests.

Unit testing was carried out once the code was developed to: spawn the obstacles over time, restrict the player movement on this x axis and to show the players score. The test table shows the tests that were carried out when testing the code that was implemented for the feature named above (refer to tests 3.1.1 – 3.4.3).

Integration testing was carried out after each subtask of the sprint was completed. When carrying out integration testing for this sprint only test questions from 2.5.1 onwards were carried out. The results from carrying out integration testing were that there were no problems when combining the developed features from this sprint, with those that had been develop form the previous sprint. As test questions from the previous sprint (sprint 2) were only those that had been retested, when carrying out post development testing it will be important that all the test questions from the testing table are restated to make sure that all the features that have been implemented so far work together as expected.

In this sprint regression testing was carried out once a solution had been implemented to fix the bug found, when carrying out test with test reference with 3.1.3. Once the bug was fixed, test reference 2.5.1 – 3.1.3 were performed again. The results from carrying out these tests again was that the solution implemented to fix the bug had no impact upon other features of that game.

Usability testing was not carried out in this sprint. This is because, the features of the game that have been implemented as part of the development of this sprint are not very different from the previous prototype.

4.4 3rd Sprint Review

At the end of the 3rd sprint the obstacles are spawn with respect to time. This fulfils success criteria 2.4. Code has also been developed to stop the player-controlled game object falling from the ground plane since player movement is restricted on the x axis. This fulfils success criteria 2.5. A scoring system for the game has also been established since the score depends on the position of the player-controlled game object on the z axis (represents how far the player has reached). This fulfils success criteria 3.2 however only partially fulfils success criteria 3.1. It only partially fulfils the success criteria because the score total is not continues – the score from one level is not the score that the next starts on.

This sprint has been successful apart from the one issues with development. A solution to fulfil success criteria 3.1 will not be implanted. This is due to how unfamiliar I am with saving and loading data between the scenes within Unity. At this stage no further modifications will need to be made to my solution (refer to the analysis section 1.5 My solution). Furthermore, since no prototype was created to carry out usability testing, it is important that the features that have been implemented are test using the next prototype. I will now progress with fulfilling further success criteria.

5.0 4th Sprint

5.1 Design

The previous sprint was successful in fulfilling any success criteria related to spawning obstacles. Currently, up to this point there is no success criteria which I have aimed to fulfil but have not managed to fulfil, this means that it is a suitable point to move on to fulfilling success criteria reference 4.1 – 5.2 and 8.1 – 8.2.

As mentioned previously, a solution has not been implemented to allow players to win a level and progress to the next level. This sprint will start by looking at a solution to solve this problem. Furthermore, the aim of this sprint will be to focus on UI features that will be implemented to allow user to interact with the game conveniently. I have broken down the sprint into the following subtasks:

1. Loading the next level and game over UI
2. Main menu
3. Developing more levels
4. Exporting the game

The sprint has been broken down into the following subtask because, the aim of this sprint is to focus on UI elements of the game. The implementation of the level complete and game over UI will be easier to complete before focusing on implementing a main menu because this will involve creating a different scene in the game. As a result, the subtasks will be completed in this order because it will allow the simple task to be completed first before focusing on more work-based tasks.

Furthermore, the sprint is being broken down into the following subtasks because it is important that development is completed for features that will be implemented in all the levels. This way it will be easier to develop more levels since one scene of the game will be developed, can be copied and adapted to create more levels within the game (explained in more detail in section 5.1.3).

It is important that subtask one to three are completed before exporting the game because once these features have been implemented a large amount of the success criteria would have been me, making it a suitable stage to carry out testing outside of the Unity editor.

5.1.1 Loading the next level and game over GUI

At this point into development if players complete a level, there is no way of winning the level and the player-controlled game object falls from the end of the ground plane and the score continues to increase. A solution to this problem would be to place a cube game object at the end of the ground plane. The size of the cube game object (end point) will need to be the same size of the ground plane in the x directions. This will mean that players will not be able to progress beyond this point.

This is a suitable solution to the problem because it will allow the built-in Unity function OnCollisionEnter to be called when the player-controlled game object collides with the game object and as a result, the code within the function will be executed. An alternative solution would be use the player-controlled game object position on the z axis to determine when the level is completed. However, I will focus on implementing the first solution to the problem because the actions that need to before performed when the player reaches the end of level, will only be performed when the function is called. The actions that will need to be performed on completing the level are that a level complete message needs to be displayed and then the next level needs to be loaded. The algorithm below shows how this will be achieved:

```
public object levelCompleteGUI
public float loadDelay = 2
function OnCollisionEnter()
        if collier = player game object
                levelCompleteGUI = true
        end if
        loadLevel ()     //only call few second after
end function
function loadLevel ()
        Load next level
end procedure
```

The OnCollisionEnter function is a Unity specific function. This function will be will be used because when the collider attached to the player-controlled game object touches the collider attached to the end point game object, the function is called. This means that any code within it will only be executed when the two objects collide. In this example, when the player-controlled game object collides with the end point game object it will show a level complete message and then will load the next level.

The variable levelCompleteGUI will store a reference to the UI game object (the text) that will need to be shown to the when a level has been completed. The game object is being set to Boolean true when the player-controlled game object collides with the end point game object because initially it will set to false which means that when the game starts it will not be shown. However, upon colliding it will need to show the message.

Furthermore, the loadLevel procedure, will be responsible for loading the next level. It will also be called when the player-controlled game object collides with the end point game object, however there will be a delay in calling the procedure. This is because the next level does not need to load straight away but there will need to be enough time so that the player is able to see the level complete message and prepare to play the next level.

Variable Name	Type	Explanation
levelCompleteGUI	Object	Variable will store a reference to the text object that needs to be shown when the player-controlled game object collides with the end point game object. Variable will be declared as public to allow a reference to the object to be set in the Unity Inspector.
loadDelay	float	Variable will store the amount of time on seconds between the level complete message being shown and the next level of the game loading. Will be declared as public to allow it to set in the unity inspector as unsure of how much time will need to be elapsed between two events.

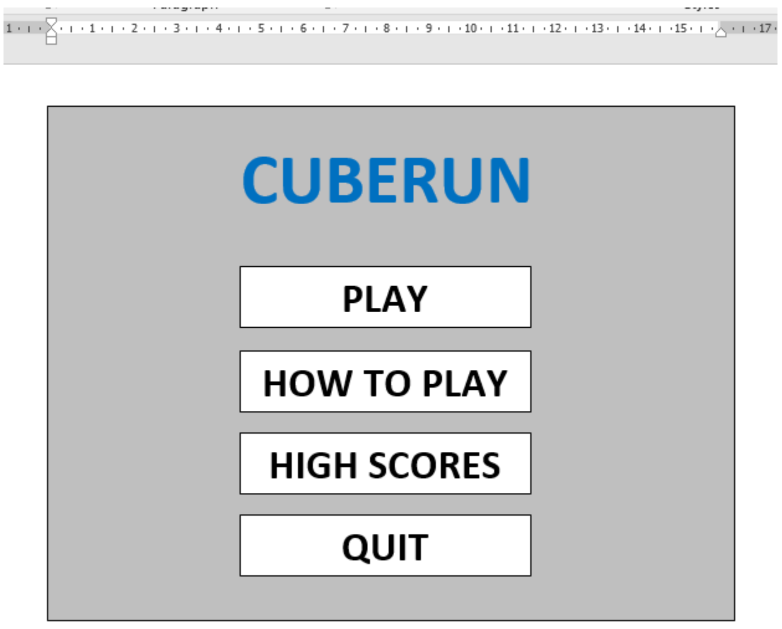
The game over GUI will be shown when the player controlled game object collides with a player controlled game object. As the player game object already has a script attached to it that detects when it collides an obstacle, code willed added to this script. From the algorithm on the previous page, line 09 will be implemented as code (but for the game over GUI) within the OnCollisionEnter function that is in the CollisionDetect script.

5.1.2 Main Menu

A main menu will be implemented to allows players to interact with the game conveniently and to navigate to certain parts of the game. It is important that the main menu is simple to use, and functions ad expected because if players are unable to use the main menu they will not be able to access the game.

Below an image of the design of the GUI for the main menu can be seen. During development the GUI will be created within the Unity Editor. The main menu GUI will be very simple and easy to navigate therefore it does need to be created in photoshop or any other editing software.

A grey background is being used for the main menu to fit the colour scheme of the game. Also, it will not cause any visual problem for players are it does not conflict with other colours that are used on the main menu. When the cursor is over the button the colour of the buttons will be set to change to a blue. This will be done because it will give user a further indication to what button they will be about to press on. The text of the button will be of a bold format and large in font size (not sure on exact font size) because this will make them easy to read and they will stand out.



The algorithm bellow shows how the GUI elements of the main menu will be programmed to carry out a specific action:

```
If button pressed == play
        StartGame()
end if
If button pressed == How to play
        Show instruction        //no longer show menu, show instructions
end if
If button pressed == high scores
        Print ("Not available at this point")
end if
If button pressed == quit
        QuitGame ()
end if

procedure StartGame()
        Load next scene
end procedure
procedure HighScores ()
        Print ("Not available at this point")
end procedure
procedure QuitGame ()
        Close window
end procedure
```

In the algorithm, each task carried out by each button on the main menu, will be created into a sperate subroutine. This is because it will be easier to maintain the code as it keeps it organised. Furthermore, the scope of the subroutines will be public so that they can be called from outside of the class because they perform common tasks such as loading the next scene and close the application window.

The StartGame procedure will be called when the play button on the main menu is pressed. This is because the common task performed by the StartGame procedure is that the next scene is loaded. When players first open the final executable final, they will first see the main menu therefore, if players press the player button it will need to load the next scene which is the first level of the game that players will play.

The HighScores procedure will be called when the high scores button on the main menu is pressed. Since there is an issue with the scoring for the game a solution has not yet been implemented to save the high scores of players. Therefore, if this button is pressed it will need to show the message "Not available at this point". A separate procedure is not need, however, it has been used so when in future development if the button does come available then code from this subroutine will be executed.

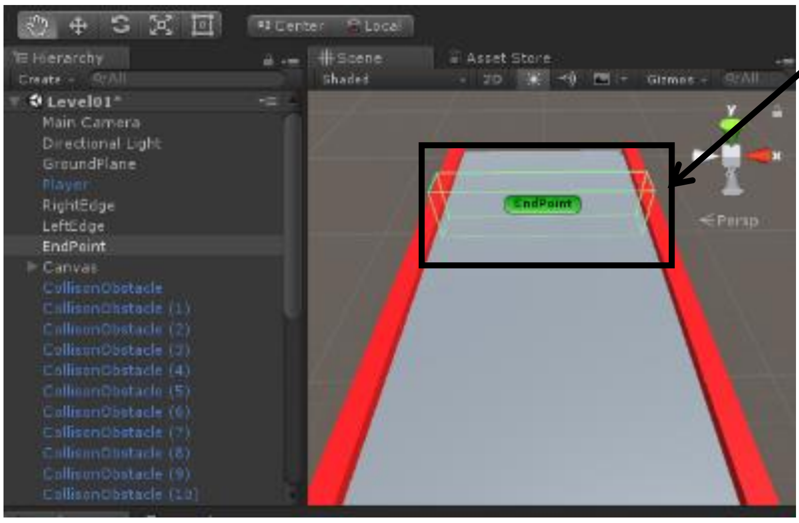
Finally, the quit game procedure will be called when the quit button on the main menu is pressed. The is because the common task performed by this subroutine is that it will close the window of the executable file.

5.2 Development

5.2.1 Loading the next level and game over GUI

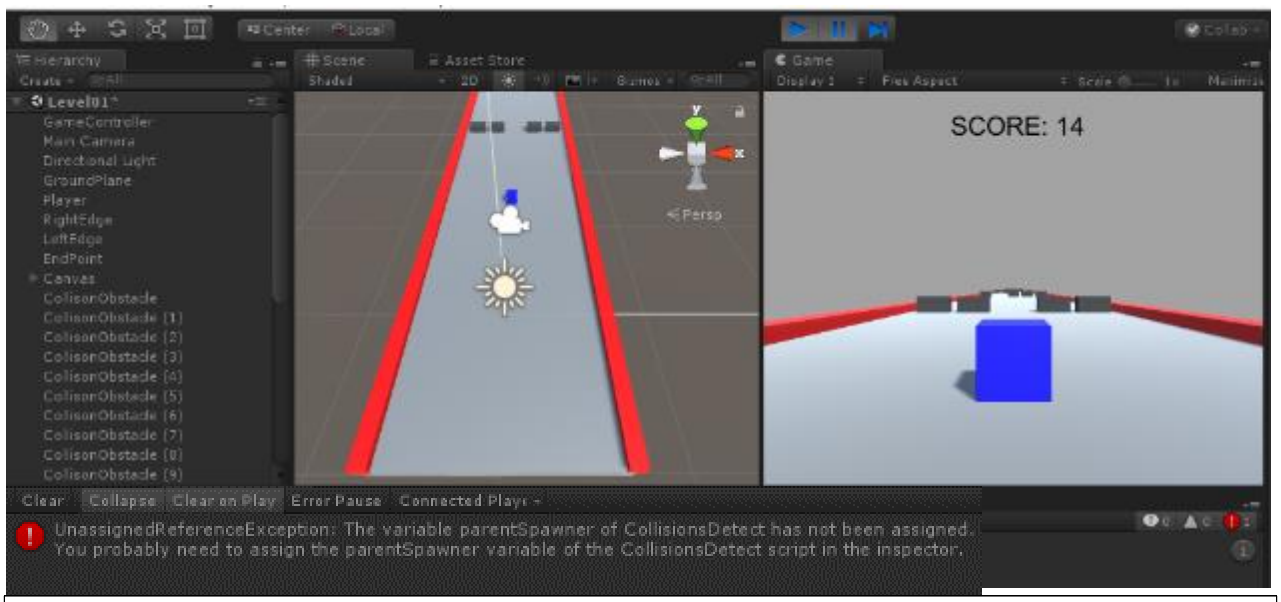
To allow the next level to load and display the game over GUI, canvas and text objects will be used. These objects will then need to be programmed so that once a player has completed a level it displays the level complete GUI and the loads the next level. Also, once the player has collided with an obstacle the game over GUI needs to be displayed to the player.

At this point in development there is no way of winning the level. If the player reaches the end of the level that player game object continues to move forwards on the z axis, this means the score continues to increase. As a solution to this, a game object will be positioned on the end of the ground plane.



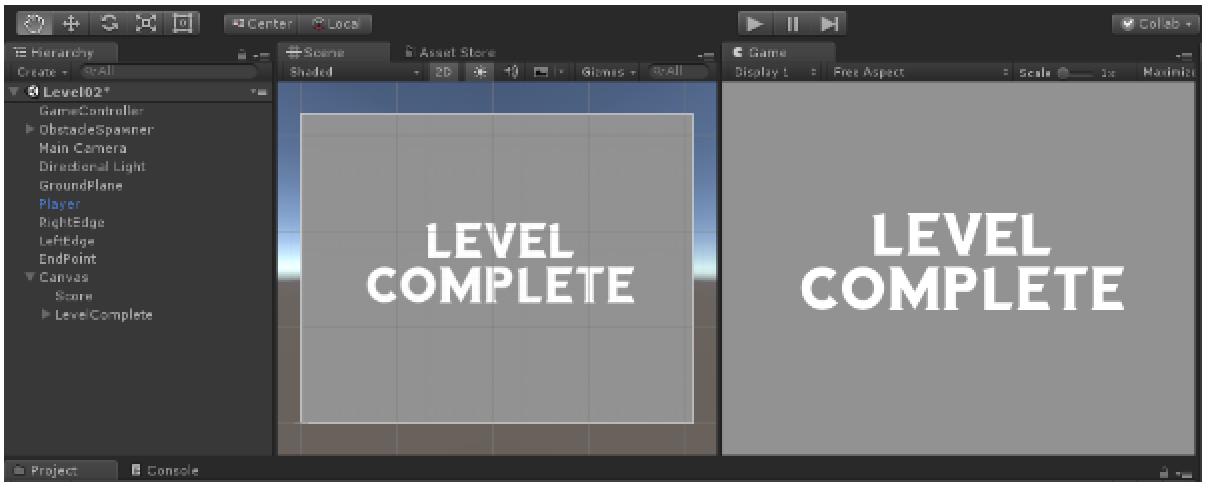
This is the game object that has positioned on the end of the ground plane. This object will not be seen by the players as it will be used to determine if the player has reached the end of the level. As a result, the mesh rendered has been disabled and no material has been applied to allow its colour to be changed.

The object has been given an icon so that during development the I know that the object exists.



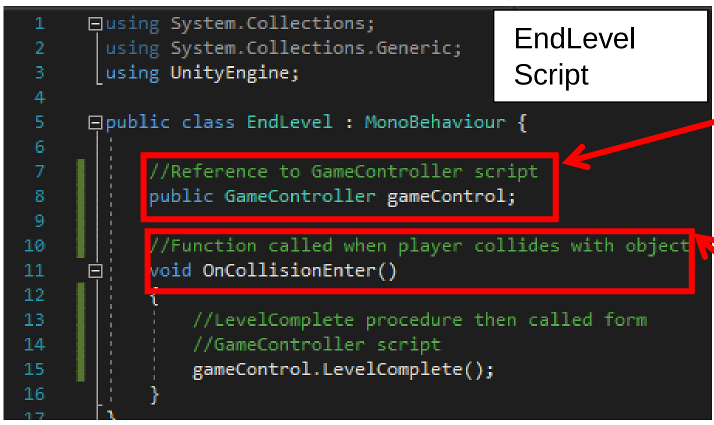
```
UnassignedReferenceException: The variable parentSpawner of CollisionsDetect has not been assigned.
You probably need to assign the parentSpawner variable of the CollisionsDetect script in the inspector.
```

This error message shows when trying to run level one of the game to test it. Since the "spawn" script has been created this error message has been appearing in the console. It will be ignored because it relates to assigning objects to variables that are not in scene one, however are used in scene two with the "spawn" script. I will look at fixing this error later in development. For this reason, I will continue development in scene two (level two).



This panel (image that fits the screen) was created so that when the player game object collides with the end point game object it will need to be displayed to the users. As the game does not need to start by showing the level complete GUI, the panel will be disabled, and code will be developed to enable it when required to do so (can be seen below).

Now that in the game there is a way to determine if the player has reached the end of the level and the panel has been designed, code will be developed to show the level complete GUI and then to load the next level. Two next scripts will be created. One of these scripts will be responsible for any scene management. The other script will be responsible for detecting if the player has reached the end of the level, if this is the case then a specific action will be performed.



EndLevel Script

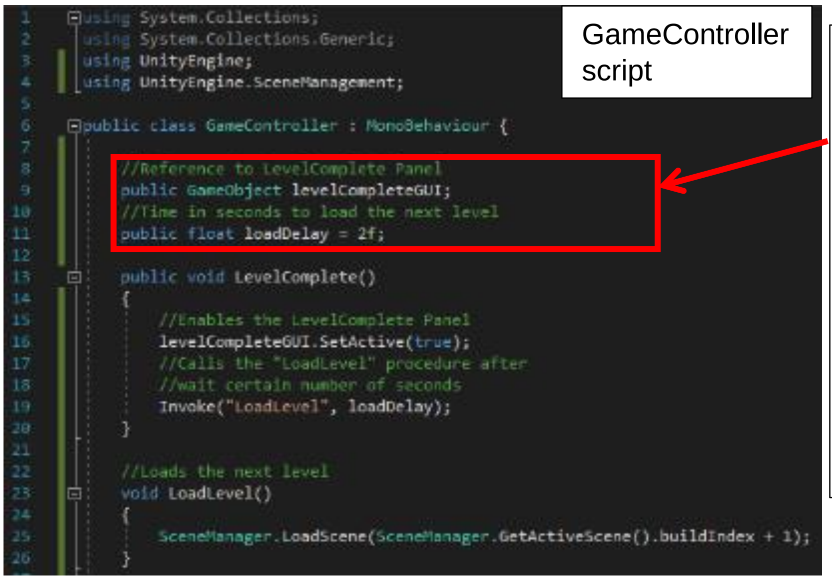
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EndLevel : MonoBehaviour {
6
7       //Reference to GameController script
8       public GameController gameControl;
9
10      //Function called when player collides with object
11      void OnCollisionEnter()
12      {
13          //LevelComplete procedure then called form
14          //GameController script
15          gameControl.LevelComplete();
16      }
17  }
```

Following the algorithm designed, development started by declaring a variable to create a reference to another script to allow its subroutines to be accessed.

Function “OnCollisonEnter” called when the colliders of the two game objects collide.

Line 08 of the code use a function called OnCollisionEnter. The function will be called when the collider attached to the player controlled game object touches the collider attached to the end point game object. This time, when using the function, no parameters are being passed to the function because it does not need to check the name of the collider that has been hit. Since the end point game object will remain in a fixed position the OnCollisionEnter function will need to be called regardless.

When the player controlled game, object touches the collider attached to the end point game object the subroutine LevelComplete is called from the GameController script. This contains the code that will be executed when the player and the end point game object collide. The code can be seen on the next page. Two separate scripts are being used to keep the code organised. This way any game management code (i.e. that is responsible for loading the next level) will be coded in the GameController script.



```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class GameController : MonoBehaviour {
7
8       //Reference to LevelComplete Panel
9       public GameObject levelCompleteGUI;
10      //Time in seconds to load the next level
11      public float loadDelay = 2f;
12
13      public void LevelComplete()
14      {
15          //Enables the LevelComplete Panel
16          levelCompleteGUI.SetActive(true);
17          //Calls the "LoadLevel" procedure after
18          //wait certain number of seconds
19          Invoke("LoadLevel", loadDelay);
20      }
21
22      //Loads the next level
23      void LoadLevel()
24      {
25          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
26      }
```

Following the algorithm designed, development started by declaring a variable to create a reference to the panel game object that was created (as seen on the previous page). Another variable called LoadDelay was declared as this the time in seconds that will wait until the subroutine is called. *For more detail refer to the paragraphs below*.

When the player controlled game object touches the collider attached to the end point game object the subroutine LevelComplete is called from the GameController script. The scope of the subroutine LevelComplete, has been declared as public to allow it to be accessed by the other script. Furthermore, the scope of the LoadLevel subroutine has been declared as private because it will not need to be accessed outside of the class and therefore not by any other script.

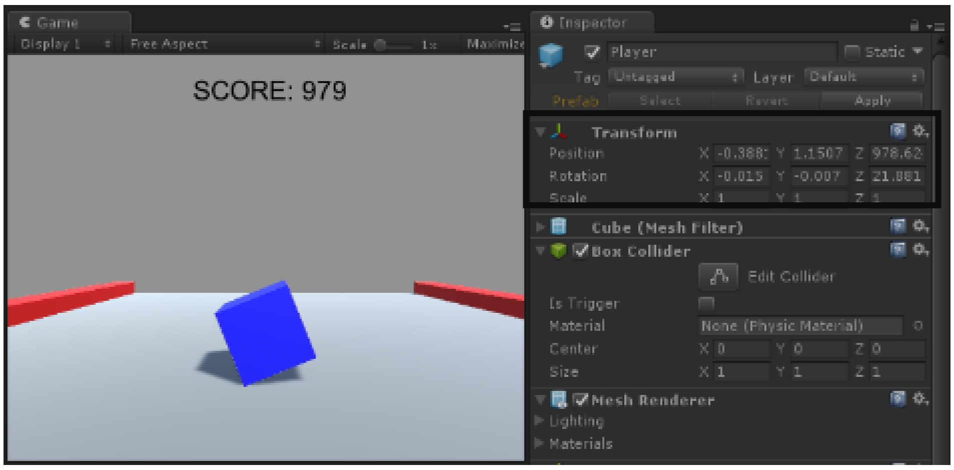
Line 16 shows the SetActive function being used/called. The parameter that is passed to the function is Boolean true because currently the panel has been deactivated, therefore since the panel game object needs to be displayed it will need be activated.

Once the level complete panel game object has been displayed to the user, the LoadLevel subroutine will be called. Line 25 shows that code that is used to load the next level. The parameter that is being passed to the SceneManger.LoadScene function in the index (which is assigned in the build setting) of the currently load scene and one is being added to it. One is being added to the index of the currently scene because this will to allow the next scene to be loaded.

Line 19 shows the code that calls the LoadLevel subroutine. It involves using the Invoke function as this allow the subroutine to be called at a later time. The string “LoadLevel” is being passed to the function because that is the name of the subroutine to be called. The variable loadDelay is also passed to the Invoke function since that is time in seconds until the LoadLevel subroutine should be called. Once the panel game object has been activated it will wait 2 seconds and then called the LoadLevel subroutine which is responsible for loading the next level. There need to be a delay between activating the panel and loading the next level because if there was no delay the players will not see the level complete GUI and the next level will load instantly.

Since development is taking place in scene two I will create a copy of this scene and this will be scene 3 (level three). All the scene will then be added to the build setting before testing is carried. However due to the unfixed error and this development taking place in scene two (level two), scene one will not be added to the build setting just yet.

Test reference 4.1.1



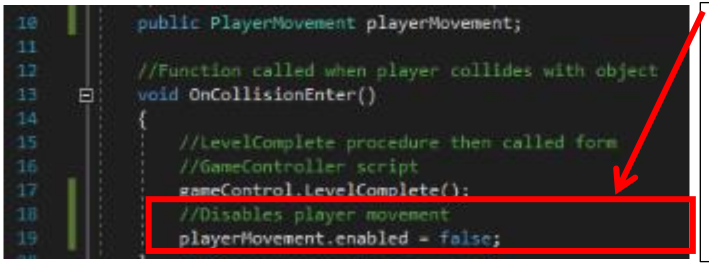
```
Player                                   Static
Tag Untagged      Layer Default
Prefab  Select  Revert  Apply
Transform
Position    X -0.388:  Y 1.1507   Z 978.62
Rotation    X -0.015   Y -0.007   Z 21.881
Scale       X 1        Y 1        Z 1
Cube (Mesh Filter)
Box Collider
            Edit Collider
Is Trigger
Material    None (Physic Material)
Center      X 0        Y 0        Z 0
Size        X 1        Y 1        Z 1
Mesh Renderer
Lighting
Materials
```

Before the player controlled game object reaches the end of the level, it does collide with the end point game object. However, once the two objects collide, the is still able to control the object to move in the left and right direction.

To carry out this a few lines of code were commented, to allow them to not be a part of the script temporarily. This test should have been carried out before these lines were added to the script.

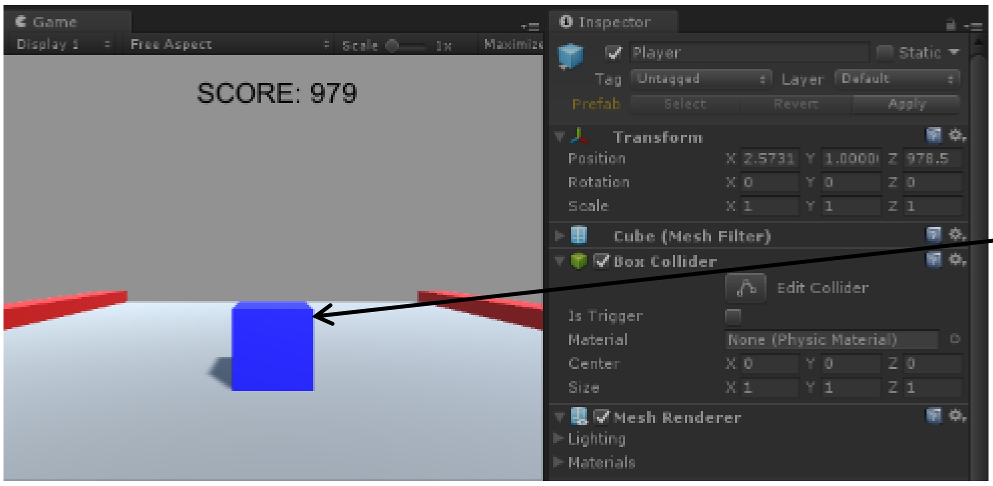
The above bug that has been found is that once the player game object collides with the end point game object, its transform values are still being updated. I suspect that this is due to the player movement script still being enabled. To fix this bug I will set the player movement script to be disabled once the player collides with end point game object. This will be a suitable fix to the solution because once the level has been completed the player game object will not need to be controlled by the players since the next will have loaded.

Bug fix – test reference 4.1.1



```
10      public PlayerMovement playerMovement;
11
12      //Function called when player collides with object
13      void OnCollisionEnter()
14      {
15          //LevelComplete procedure then called form
16          //GameController script
17          gameControl.LevelComplete();
18          //Disables player movement
19          playerMovement.enabled = false;
```

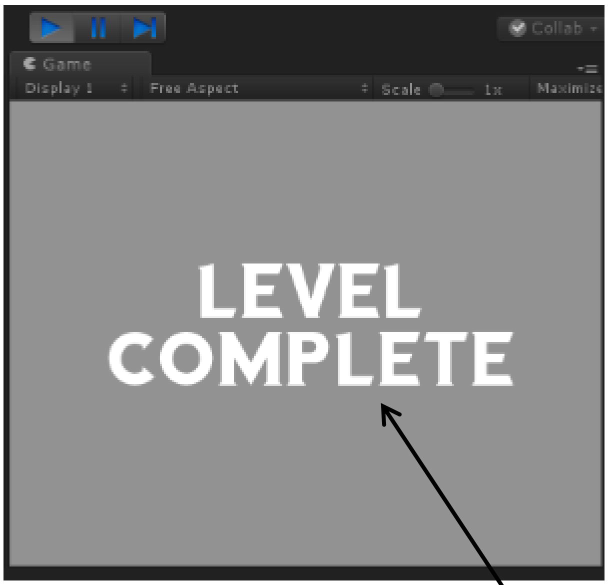
Line 10 and 19 were added to the EndLevel script. For more detail on these refer to section 3.2.1 where the code is explained. Now when the player game object collides with the end point game object, the player object no longer moves.

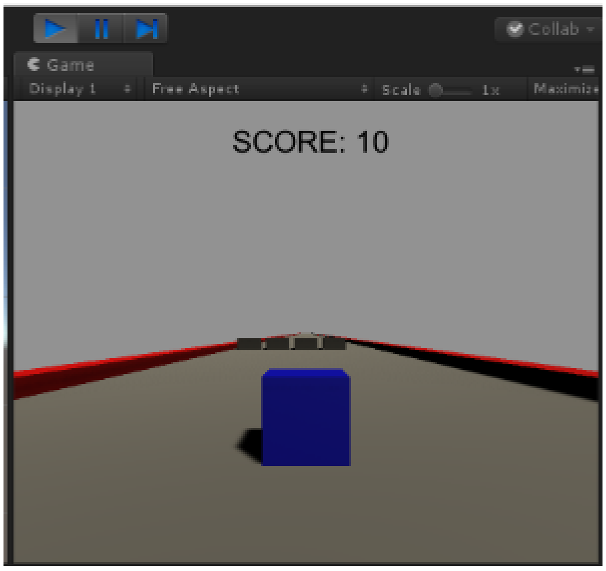


```
Player                                   Static
Tag Untagged      Layer Default
Prefab  Select  Revert  Apply
Transform
Position    X 2.5731   Y 1.0000(  Z 978.5
Rotation    X 0        Y 0        Z 0
Scale       X 1        Y 1        Z 1
Cube (Mesh Filter)
Box Collider
            Edit Collider
Is Trigger
Material    None (Physic Material)
Center      X 0        Y 0        Z 0
Size        X 1        Y 1        Z 1
Mesh Renderer
Lighting
Materials
```

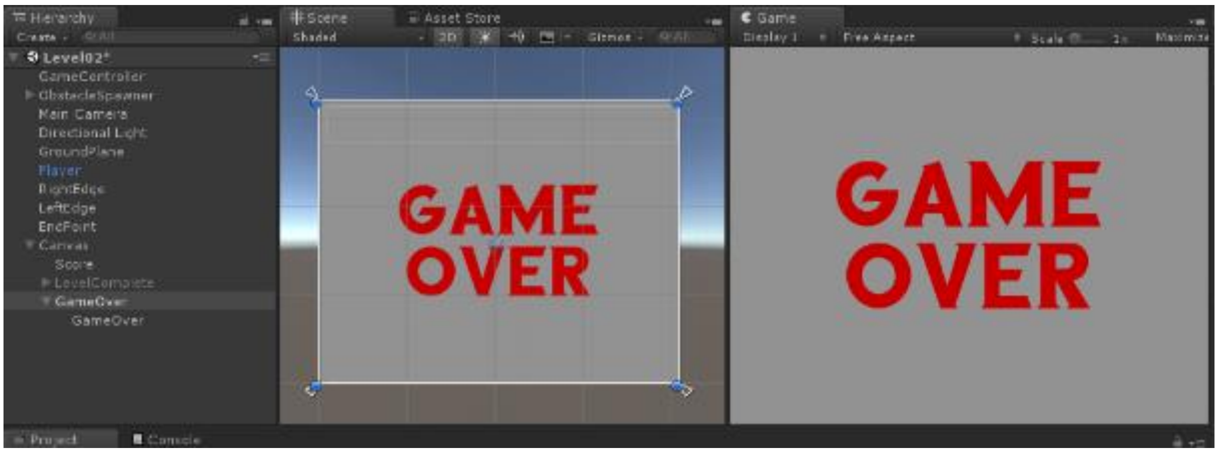
Once the player collides with the end point game object the player is not able to able to move in the left or right direction. Also, when the player collides with the end point game object the it remains on the ground plane and does not rotate and an angle.

Test reference 4.1.2



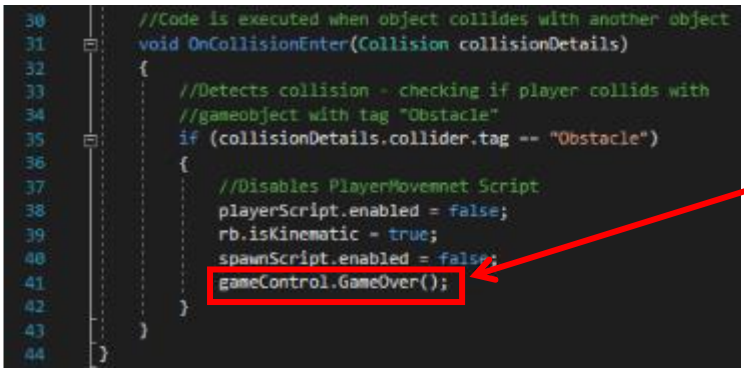


When the player controlled game object collides with the end point game object the level complete GUI is displayed to the player. Once the level complete GUI is displayed the next level is not loaded instantly but it is loaded after waiting for a few seconds. This is enough evidence to show that the test has succeeded.



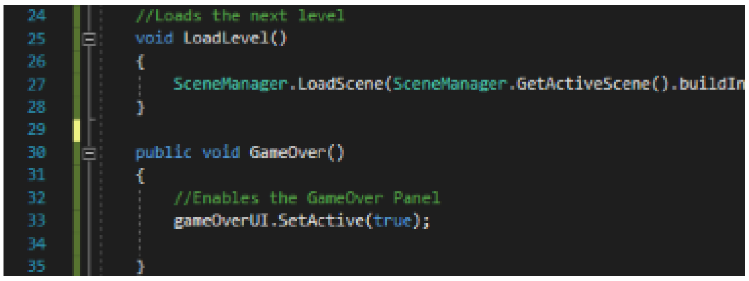
This panel (image that fits the screen) was created so that when the player game object collides with and obstacle it will need to be displayed to the players. As the game does not need to start by showing the game over GUI, this panel will also be disabled, and code will be developed to enable it when required to do so (can be seen below).

Since the game over GUI panel will only need to be displayed to the users when they collide with an object, the CollisionDetect script will be edited. Also, the GameController script will be edited since that is responsible for any scene management.



```
30      //Code is executed when object collides with another object
31      void OnCollisionEnter(Collision collisionDetails)
32      {
33          //Detects collision - checking if player collids with
34          //gameobject with tag "Obstacle"
35          if (collisionDetails.collider.tag == "Obstacle")
36          {
37              //Disables PlayerMovemnet Script
38              playerScript.enabled = false;
39              rb.isKinematic = true;
40              spawnScript.enabled = false;
41              gameControl.GameOver();
42          }
43      }
44  }
```

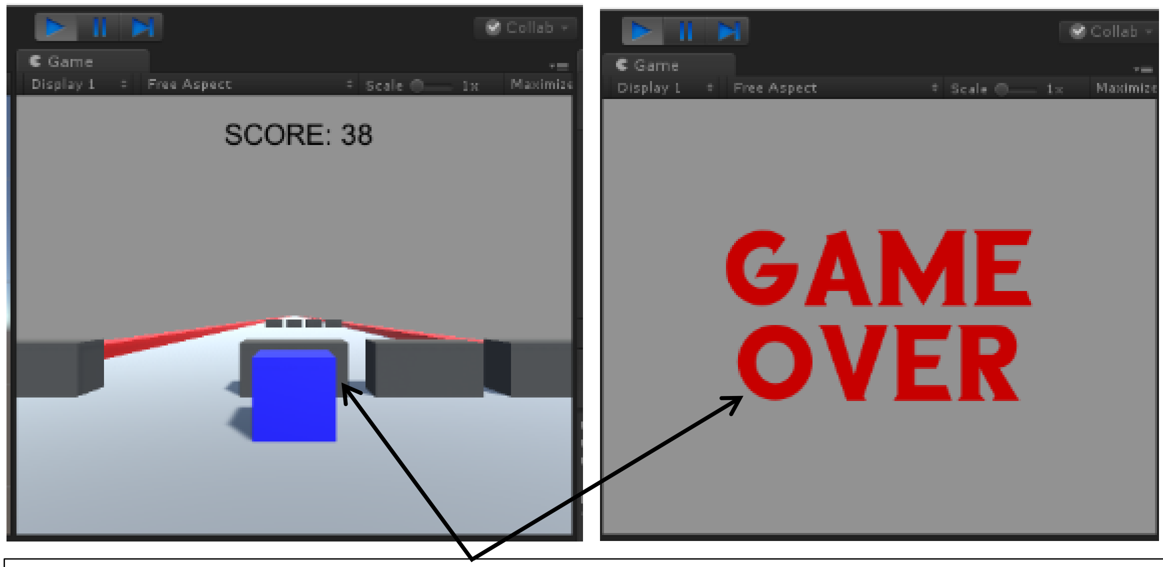
Line 42 was added to the CollisionDetect script. When the player controlled game, object collides with an obstacle the subroutine GameOver is called from the GameController script. This contains the code (see below) that will be executed when the player game object collides with an obstacle.



```
24      //Loads the next level
25      void LoadLevel()
26      {
27          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIn
28      }
29
30      public void GameOver()
31      {
32          //Enables the GameOver Panel
33          gameOverUI.SetActive(true);
34
35      }
```

When the player controlled game object collides with an obstacle the subroutine GameOver is called from the GameController script. The scope of the subroutine GameOver, has been declared as public to allow it to be accessed by the other script. Line 33 of the GameController script shows the SetActive being used/called. The parameter that is passed to the function is Boolean true because currently the panel has been deactivated, therefore since the panel game object needs to be displayed it will need be activated.

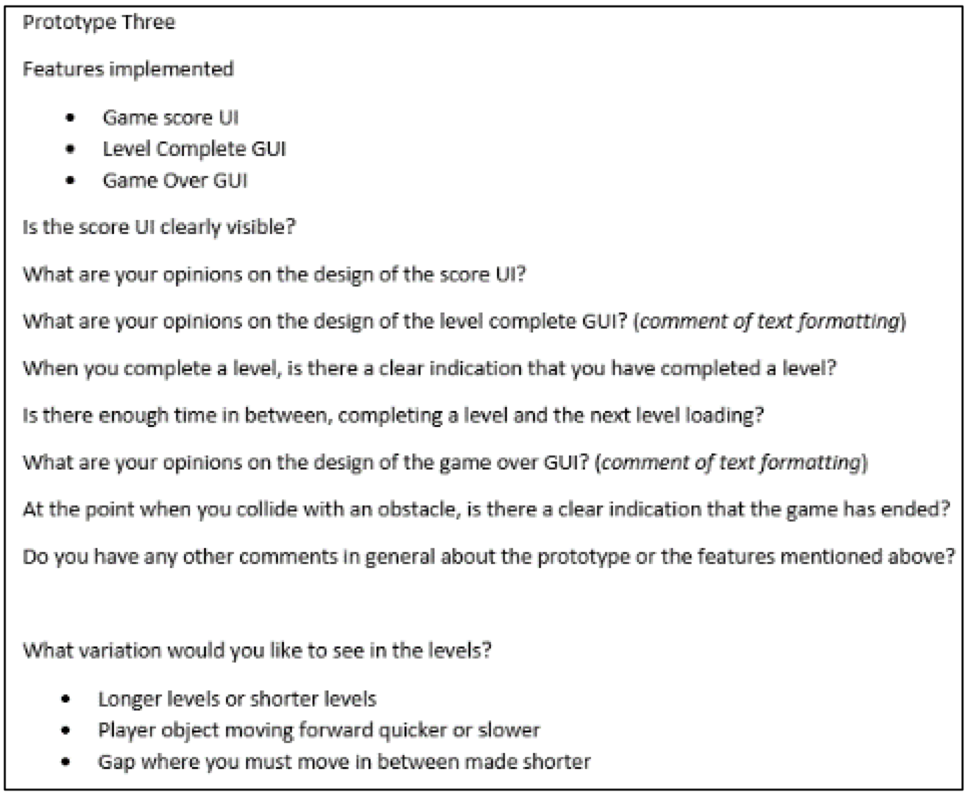
Test reference 4.1.5



When the player controlled game object collides and obstacles the game over GUI is displayed to the player. Any GUI, will need to be tested in full screen to make sure the text scales to the correct size depending on the size of the window of the final executable program. When carrying out post development testing it is important that these tests are carried out. Refer to the post development test spreadsheet in the testing table (appendix A).

Usability testing and Feedback – Prototype Three

The project is now at a point in development, where it is possible to create a prototype which can be given to stakeholders to gain feedback upon and to test the usability so far. As it is only a prototype I will not be changing any of the build setting within Unity when creating the complied version of the game. This prototype will only focus on collecting feedback on the GUI elements that have been implemented. To collect specific feedback from my stakeholders, I will provide them with a text document (screenshot below) which will include questions about features of the game that have been implemented. Before moving onto any further development, I will consider the feedback and the results from the usability testing.



Prototype Three

Features implemented

- Game score UI
- Level Complete GUI
- Game Over GUI

Is the score UI clearly visible?

What are your opinions on the design of the score UI?

What are your opinions on the design of the level complete GUI? (*comment of text formatting*)

When you complete a level, is there a clear indication that you have completed a level?

Is there enough time in between, completing a level and the next level loading?

What are your opinions on the design of the game over GUI? (*comment of text formatting*)

At the point when you collide with an obstacle, is there a clear indication that the game has ended?

Do you have any other comments in general about the prototype or the features mentioned above?

What variation would you like to see in the levels?

- Longer levels or shorter levels
- Player object moving forward quicker or slower
- Gap where you must move in between made shorter

Figure 8: Usability testing and feedback from

After, colleting feedback from my stakeholders about the current build (prototype there) and carrying out usability testing, I have complied a list of the results:

- When the next level loads the score start from zero
- Level complete and game over screen are too similar (*suggested by one person*)
- Unable to see the score after each level
- Don't know what the final score is

The positive comments from this prototype were that the score was clearly visible, the fonts were suitable as they looked appealing and the time between loading ending a level to loading the next level was enough. The issues that will be addressed with this prototype are that the player is unable to see the score after each level and the player is not able to see the final score. It is important that players are able to see their score in between levels to see how they are progressing through the game. Furthermore, it is important that players are able to see their final score to increase the competitive nature of the game and to encourage players to play the game again. The other two issues will not be addressed due to time limitations.

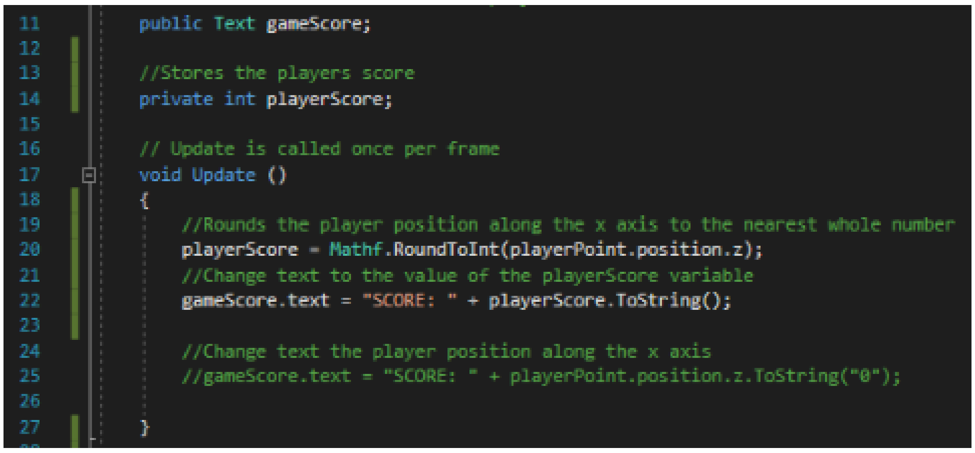
Prototype Three – Solving Issues

The issues that will be solved related to the player score. Specifically I will be fixing the issues related to the player not being able to see their score when the level complete and game over GUI is displayed to the player. It is important the issues are fixed as they would increase the competitive nature of the game.

While the player is playing the game the score text is what gives players and indication of their score. This solution to the issues would be to enable the score text to be displayed once the level complete and game over panels have been enabled. This is a suitable solution to the problem because it will show the score at the end of each level. Also, since the Score script is attached to the score text object, this will means that the correct score will be shown when each GUI elements is displayed to the players.

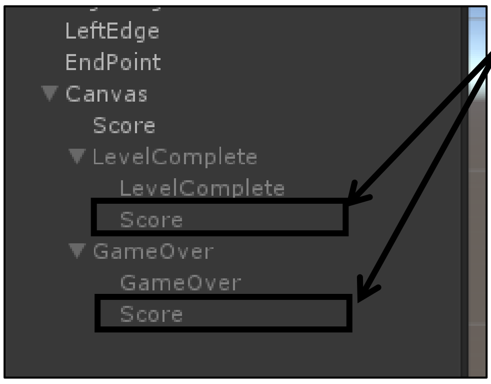
If the issue related to the score not starting from the score of the previous, was to be fixed then the solution that is being considered to showing the score after each level would show the cumulative score after each level.

After looking over the Score script a few changes were made. These changes can be seen bellow. The reason for why these changes were made are because no variable was being used to store the player score.



```
11      public Text gameScore;
12
13      //Stores the players score
14      private int playerScore;
15
16      // Update is called once per frame
17      void Update ()
18      {
19          //Rounds the player position along the x axis to the nearest whole number
20          playerScore = Mathf.RoundToInt(playerPoint.position.z);
21          //Change text to the value of the playerScore variable
22          gameScore.text = "SCORE: " + playerScore.ToString();
23
24          //Change text the player position along the x axis
25          //gameScore.text = "SCORE: " + playerPoint.position.z.ToString("0");
26
27      }
```

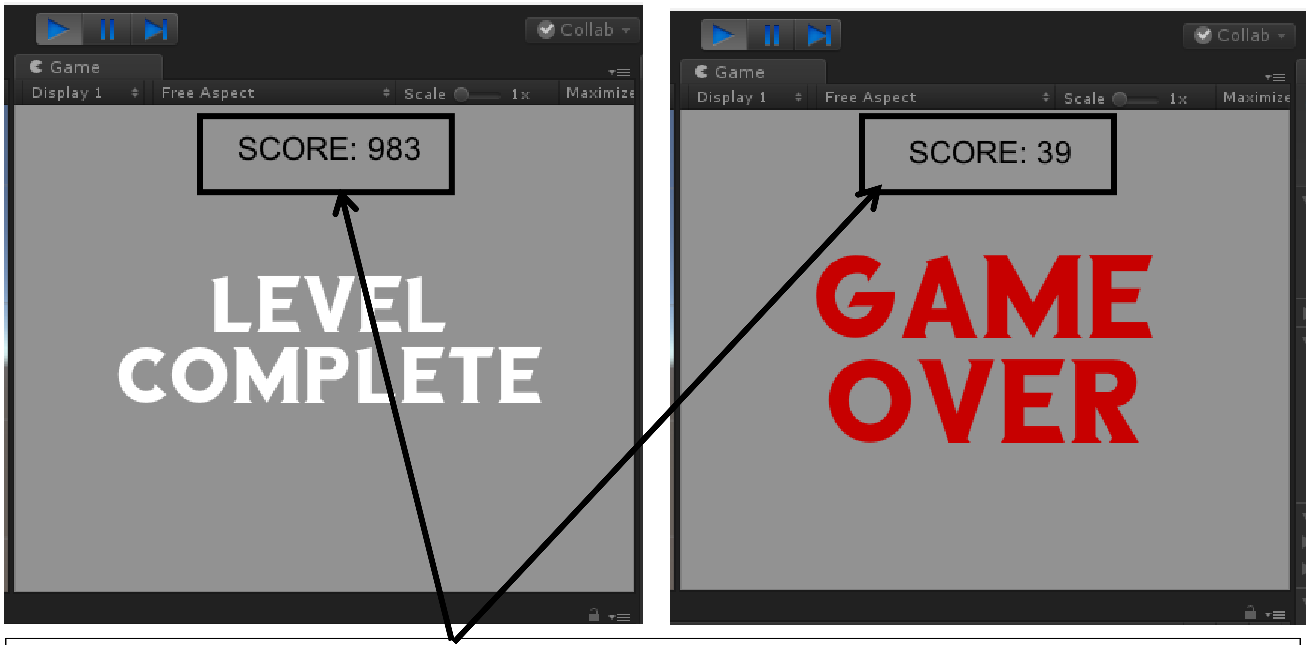
Line 14 show the playerScore variable being declared. It is declared as a private variable as it does not need to be accessed directly by other scripts and the value does not need to be set in the Unity inspector. Line 20 show the maths function being used. It is being used because the players position on the z axis is a float value and the score only needs to be an integer value.



```
LeftEdge
EndPoint
▼ Canvas
    Score
  ▼ LevelComplete
      LevelComplete
      Score
  ▼ GameOver
      GameOver
      Score
```

To solve the issue, related to the player not being able to see their score when the level complete and game over GUI the following solution has been put in place. The score text game object was copied to also be a part of the level complete and game over panel. Now when the level complete and game over panel will be activated the player should also be able to see their score before they progress to the next level.

Test reference 4.2.1 – 4.2.2

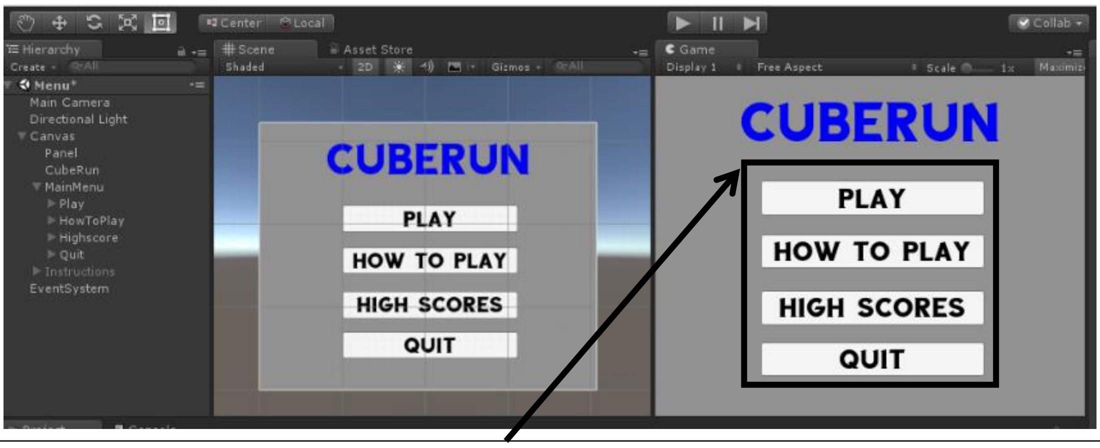


Once the player completes the level the level complete GUI is displayed as well as the score, however the score is only that score for that specific level (it is not a running score for all the levels). Also, when the player controlled game object collides with an obstacle the game over GUI is displayed as well as the score.

5.2.2 Main menu

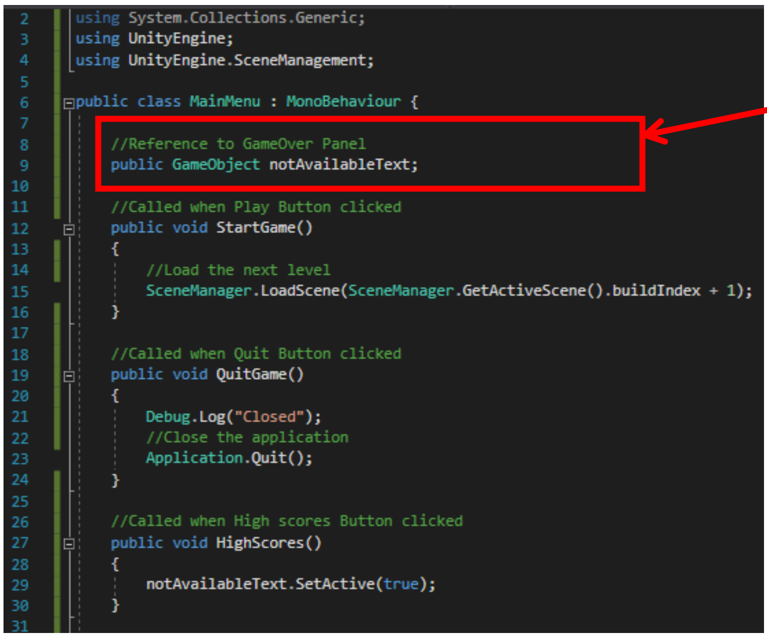
To create a main menu which will be the GUI that player interact with to begin playing the game, development will start by creating a new scene. In this scene game object such as buttons will be used to allow player to interact with the main menu. If the player clicks on the button a specific action will then be performed. Each button will also have a text element to allow player to know what will be performed when the button is clicked.

Before, development of any code takes place the design on the menu will be mapped (see below) and then each button will be programmed to perform its specific action.



Each of these are button GUI elements that players will be able to able to press. The design of these buttons involved changing the text, formatting the text and changing the transition that takes place when the player will interact with the button. For example, if the player hovers over the button with their cursor then the colour of the button will change to blue.

Before the development pf any script was started the buttons were all grouped as one game object by creating an empty game object. This was done because when the main menu is displayed to the user, all these buttons will be displayed. Also, it will allow all the buttons to have the same C# attached to them even if each button will perform a different action when clicked on.



```
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class MainMenu : MonoBehaviour {
7
8       //Reference to GameOver Panel
9       public GameObject notAvailableText;
10
11      //Called when Play Button clicked
12      public void StartGame()
13      {
14          //Load the next level
15          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
16      }
17
18      //Called when Quit Button clicked
19      public void QuitGame()
20      {
21          Debug.Log("Closed");
22          //Close the application
23          Application.Quit();
24      }
25
26      //Called when High scores Button clicked
27      public void HighScores()
28      {
29          notAvailableText.SetActive(true);
30      }
31
```

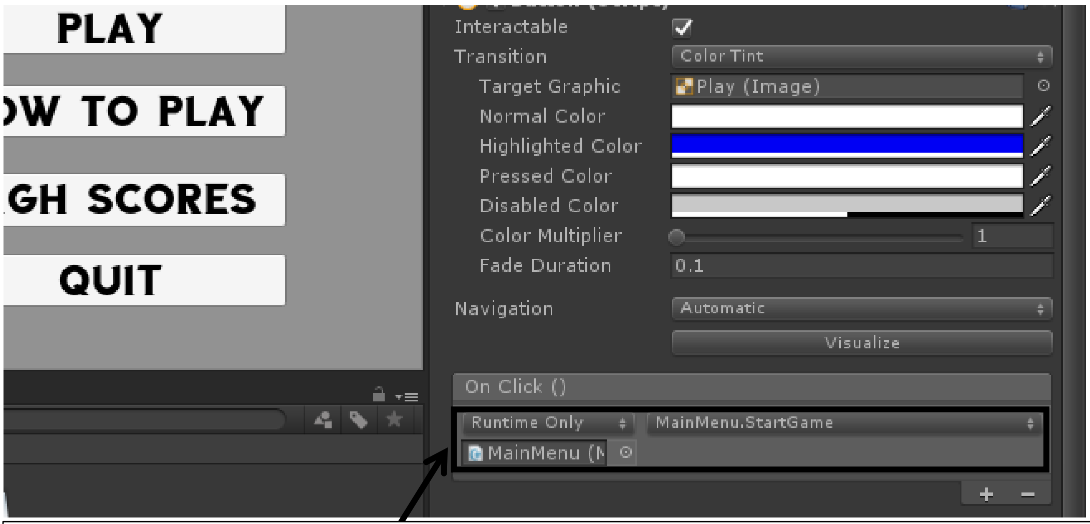
Following the algorithm designed, development started by declaring a variable to create a reference to the text game object. Currently this text is not active however it will need to activated when player click on the high score button as this features is not available.

All the subroutines that will be called when the specific button is pressed, have been declared as public. The return type has been assigned to void as the subroutine does not return a specific value or string. A separate subroutine has been created for each button because it will allow them to be assigned to each button GUI element in the Unity inspector. More details about this can be on the next page.

Line 15 shows that code that will load the next level once the “Play” button has been pressed on the main menu screen. The parameter being passed to the SceneManger.LoadScene function is the index (which is assigned in the build setting) of the currently load scene and one is being added to it. One is being added to the index of the currently scene because this will to allow the next scene to be loaded. In this example, the main menu will have index 0 and level two will have index 1. So, when the player presses the pay button one will be added to 0 to load level two.

When the players press the quit button, it will need to close the application. Line 23 shows the code that is responsible for this. The Application.Quit function is being used as this quits the application window that will be open when the players are playing the game. As the final application is not created this button can’t be tested in Unity. Line 21 is the code that is being used to display a message to the console, to make sure that the button works when the players click on it.

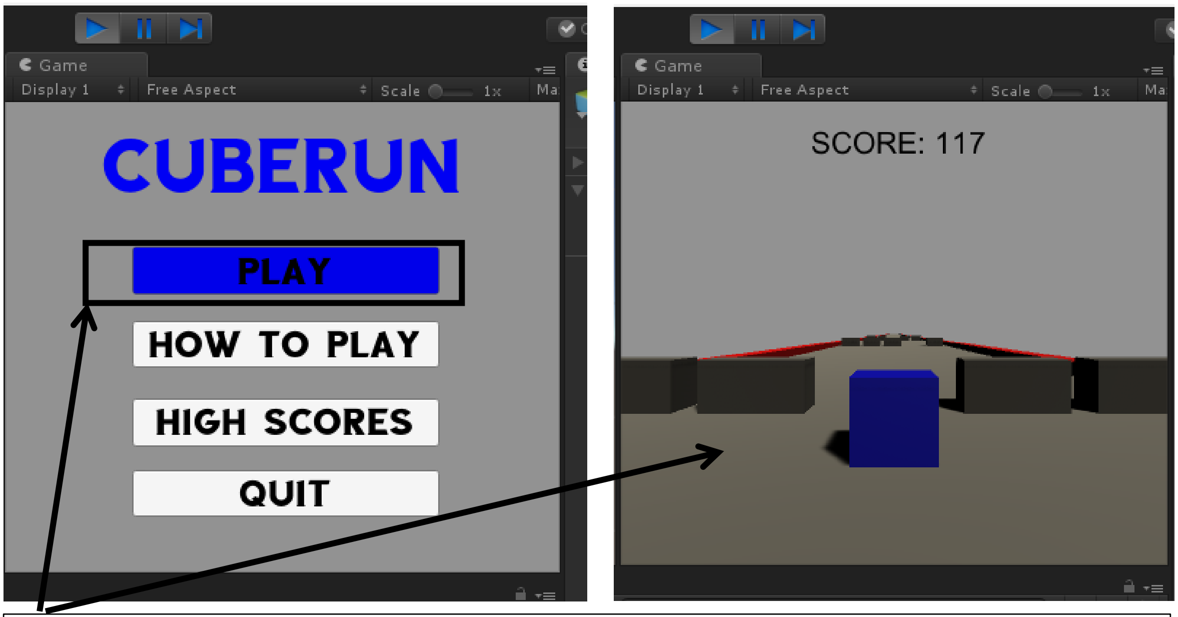
When the players try pressed the “High score” button it will need to display a message saying that the button is not available. Line 29 shows the SetActive function being called. The parameter that is passed to the function is Boolean true because currently the text has been deactivated, therefore since the text game object needs to be displayed it will need be activated.



Interactable ✓
Transition Color Tint
Target Graphic Play (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
Visualize
On Click ()
Runtime Only MainMenu.StartGame
MainMenu

This is how the button is set to call the subroutine that will carry out a specific action when it is pressed. In this example when the players press the play button the StartGame subroutine will be called. This subroutine is responsible for loading the next scene which in this example should be scene two. The play, high scores and quit buttons have all been assigned to the subroutines that should be called when the button is pressed. Testing for these buttons will be carried out before setting up the "how to play" button.

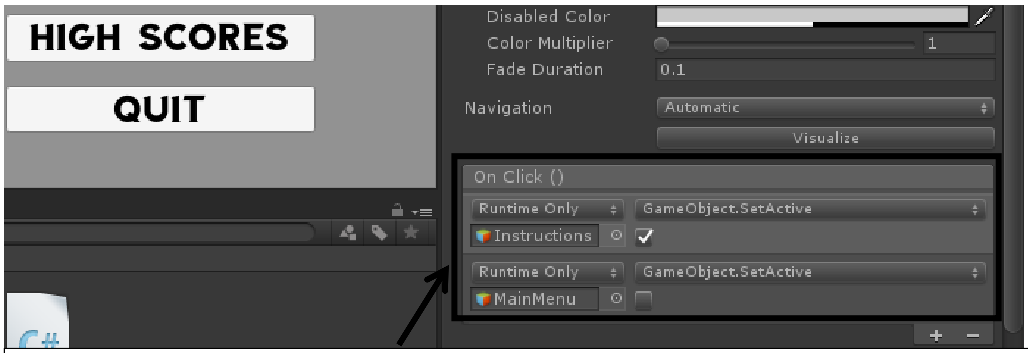
Test reference 4.3.3



When the play button is pressed, the next scene (level two) does load, however the colours are unlike the colours that were original applied to then game object. As all the colours seems dark I suspect that this is due to the lighting in the scene.

The above bug that I have found is that the colours that were original applied to then game object seem darker when the next level is loaded. I suspect that this is due to the lighting in the scene. This bug will be fixed later in development when exporting the game to a final executable file.

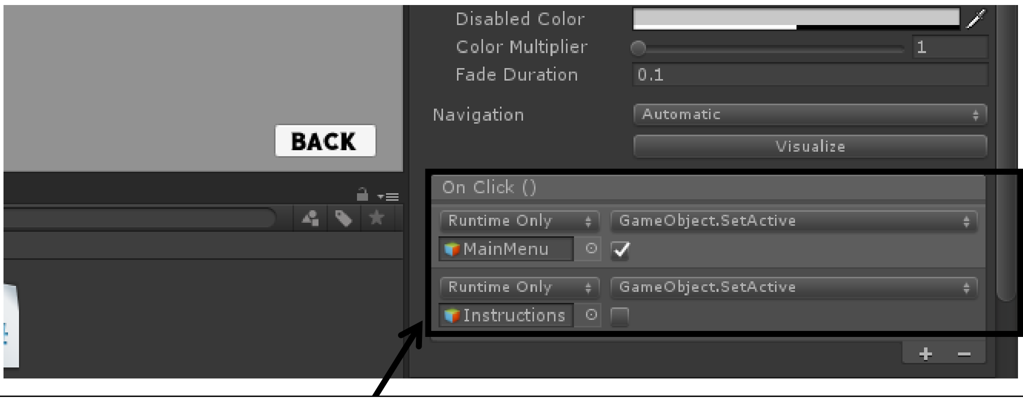
Code for the how to play button does not need to be developed since it does not need to load another scene or quit the application, but it need to simply show another menu that will give player the option to read the instructions and then return to the main menu.



Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize

On Click ()
Runtime Only | GameObject.SetActive
Instructions | ✓
Runtime Only | GameObject.SetActive
MainMenu | ☐

These are the actions that will be performed when the players click on the "how to play" button. When players do click on the button it will active the Instruction menu (hence why the box has been ticked) however at the same it will disable the main menu.

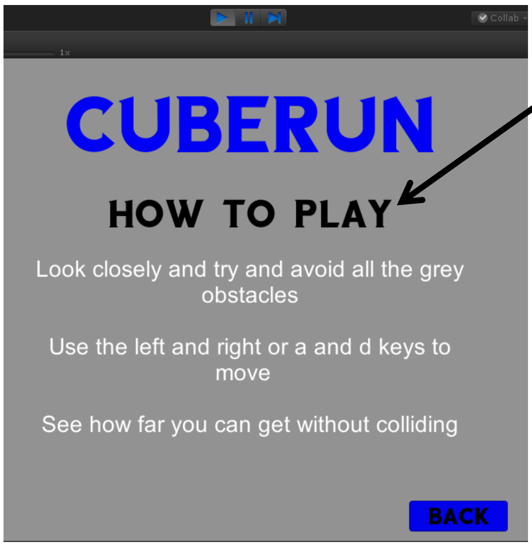


Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize

On Click ()
Runtime Only | GameObject.SetActive
MainMenu | ✓
Runtime Only | GameObject.SetActive
Instructions | ☐

These are the actions that will be performed when the players click on the "back" button on the instruction menu. When players do click on the button it will active the main menu (hence why the box has been ticked) however at the same it will disable the Instruction menu.

Test reference 4.3.7 and 4.3.8



Look closely and try and avoid all the grey obstacles

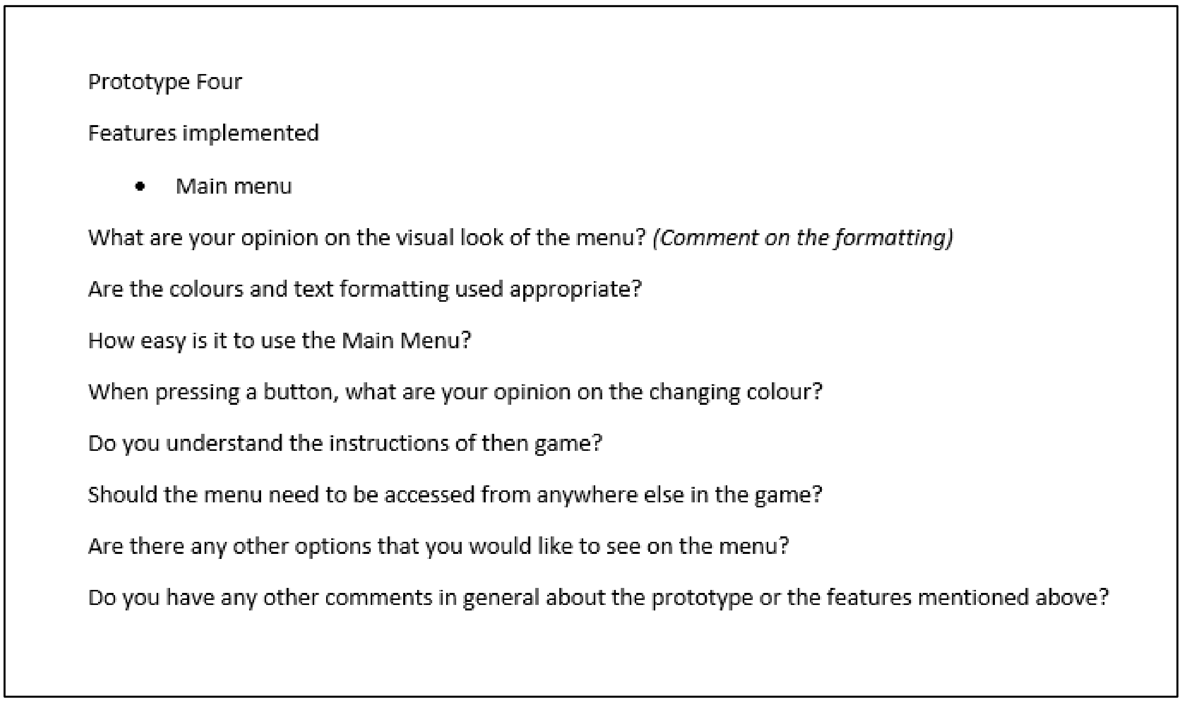
Use the left and right or a and d keys to move

See how far you can get without colliding

When the "how to play" button is pressed the instruction menu is shown and the main menu is no longer seen. Also, when on the instruction menu if the cursor hovers over the "back" button it changes to a blue colour. If the back button is pressed the instruction menu is no longer visible, however the main is visible. This is enough evidence to show that the tests have succeed.

Usability testing and Feedback – Prototype Four

The project is now at a point in development, where it is possible to create a prototype which can be given to stakeholders to gain feedback upon and to test the usability so far. As it is only a prototype I will not be changing any of the build setting within Unity when creating the complied version of the game. This prototype will only focus on collecting feedback on the main menu and button elements that have been implemented. To collect specific feedback from my stakeholders, I will provide them with a text document (screenshot below) which will include questions about features of the game that have been implemented. Before moving onto any further development, I will consider the feedback and the results from the usability testing.


Prototype Four

Features implemented

- Main menu

What are your opinion on the visual look of the menu? *(Comment on the formatting)*

Are the colours and text formatting used appropriate?

How easy is it to use the Main Menu?

When pressing a button, what are your opinion on the changing colour?

Do you understand the instructions of then game?

Should the menu need to be accessed from anywhere else in the game?

Are there any other options that you would like to see on the menu?

Do you have any other comments in general about the prototype or the features mentioned above?

Figure 9: Usability testing and feedback from

After, colleting feedback from my stakeholders about the current build (prototype four) and carrying out usability testing, I have complied a list of the results:

- When game over is displayed there is no way to quit the application
- At the end of the next level, when the game is complete there is no way to quit the application

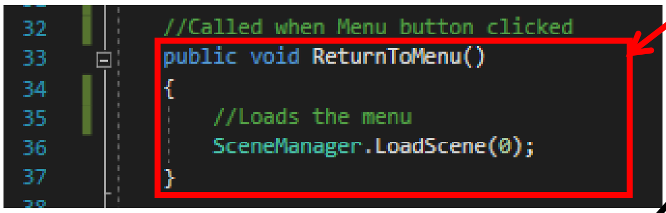
The positive comments from this prototype were that the main menu was easy to navigate, and all the buttons functioned as they were expected. Furthermore, comments on the visual look of the main menu were also positive. The issues that will be addressed with this prototype are that the player is unable to quit the game once they have completed all the levels or once they have collided with an obstacle and the game over GUI is displayed. It is important that a solution is put into place to address these issues because my stakeholders commented that if they completed the game or collided with an obstacle then they would have to close the complete application and then open it up again.

Prototype Four – Solving Issue 1 and Issue 2

Issue one relates to the game over panel and when it is displayed. When the game over panel is displayed, there is no way to end the game or restart the game unless if the user closes the application and then open it again.

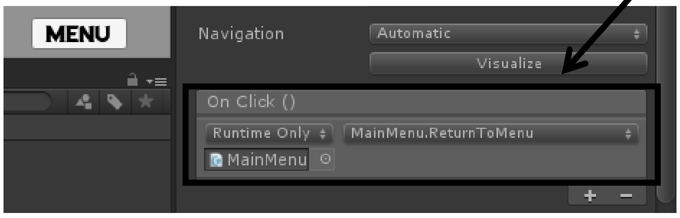
The solution to this problem would be adding a button to the game over panel in scene two so that when the player clicks on the button it will load the main menu again. Since the main menu will be loaded again player will have to option to click on any other button that is one the main menu. Furthermore, by placing this button here when the player collides with an obstacle they will have more of an option rather than just ending the game.

This is a suitable solution to the problem because when the player collides with an obstacle they will be able to click on the button to return to the menu. When returning to the menu players can press the quit button to close the application. The screenshots and the annotations below show, the solution to solving the issue.



```
//Called when Menu button clicked
public void ReturnToMenu()
{
    //Loads the menu
    SceneManager.LoadScene(0);
}
```

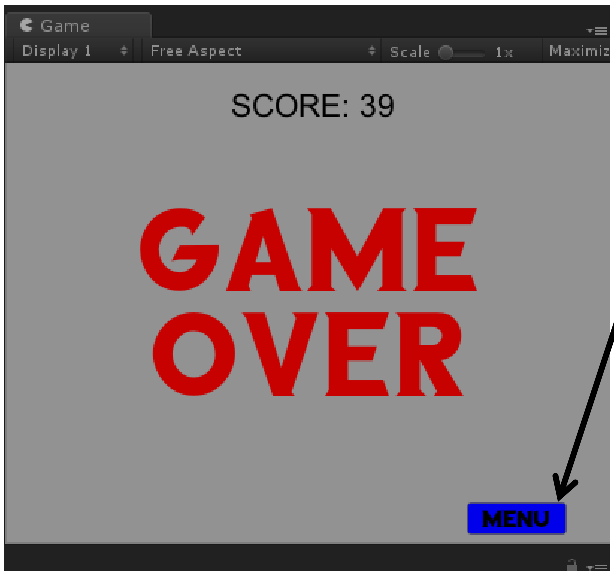
This is the subroutine that was added to the MainMenu script. It will be called when the button “Menu” on the game over panel is pressed.



On Click ()
Runtime Only
MainMenu.ReturnToMenu
MainMenu

This is how the button is set to call the subroutine that will carry out a specific action when it is pressed. In this example when the players press the play button the ReturnToMenu subroutine will be called. This subroutine is responsible for loading the main menu.

Test reference 4.4.1



When the player controlled game object collides with an obstacles, the game over panel is activated. Also, players can press the “Menu” where they can return to the main menu to press any of the other buttons on the main menu.

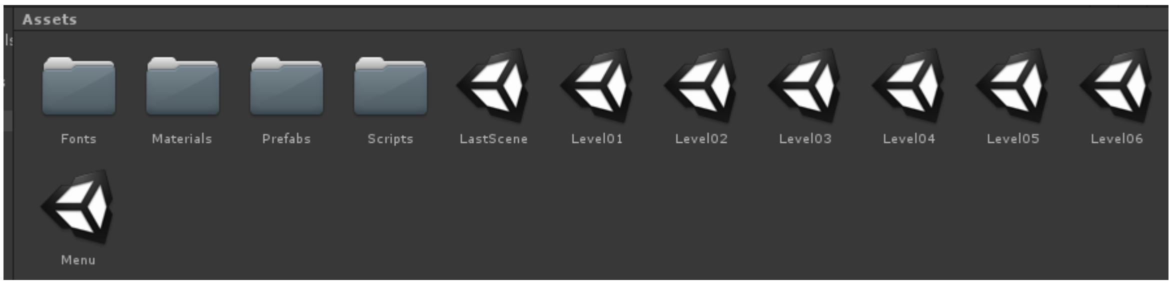
A similar solution was put into place for issue two, however a whole new scene was created with the main menu button added to it. A new scene was created because it will show a message saying that they have completed the game. From this scene player will be able to return the main menu.

5.2.3 Developing move levels

To create move levels, a copy of scene two (level two) will be created. This way of creating more levels is not the best way however, since many big changes will not be made it Is a suitable solution to the problem. Another reason for why copies of the level two will be used is because this level has been fully tested for its functionality during development and therefore the other level should also function the same way.

I will not be documenting how the changes that will be made to each copy of level two, however I will record in a table what changes have been made for future reference. Furthermore, any more code will not need to be developed when creating these new levels.

Level 01	This level will not be included as a part of the final build because due the error that has not been fixed. The error can be seen at the start of this development section (page 74).
Level 02	This is the level that is being copied to create all the other levels.
Level 03	Scale and position of the ground plane was changed. Scale was changed to increase the size of the ground plane on the z axis. The position was changed so that the player-controlled game object is placed at the beginning of the ground plane
	Scale and position of the left and right game object changed. Scale and position was changed so the objects are the same length of the ground plane and start from the beginning of the ground plane.
	Position of the end point game object was changed so that the player controlled game object only collides with this object at the end of the level. It was positioned at the end of the ground plane.
Level 04	The changes that were made to level three were also applied to this level.
	The amount that players position moves forwards on the x axis was changed from the 0.3 to 0.5. This change has been made because it will affect how fast the player controlled game object move forwards.
Level 05	The changes that were made to level four were also applied to this level.
	The amount that players position moves forwards on the x axis was changed from the 0.5 to 0.6. This change has been made because it will affect how fast the player controlled game object move forwards.
Level 06	The changes that were made to level five were also applied to this level.
	The amount that players position moves forwards on the x axis was changed from the 0.6 to 0.7. This change has been made because it will affect how fast the player controlled game object move forwards.

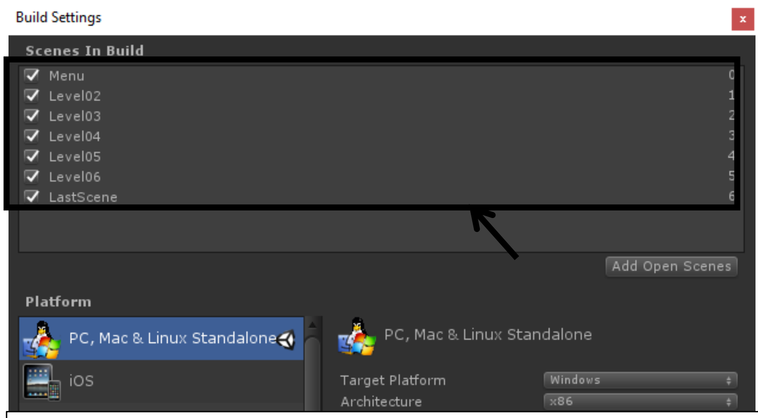


These are all the scene that had been created. As the scenes are a copy of level 02, which has been fully tested, each individual scene will not need to be tested right now. However, I will be carrying out testing of all the levels when performing post development testing.

5.2.4 Exporting the game

Exporting the game will allow a single executable file to be created. This single executable file has been created when building prototypes of the game however, none of the build setting have been changed. It is important that these build setting are configured so that the executable file runs as expected on the desired platform. As identified in the analysis section, my game will be built to run on PCs, so it is important the build settings are adjusted to support this platform.

Before exporting the game test reference 4.3.3 will be investigated further. It was suspected that the issue was to do with the lighting. After carrying out some research I found that when loading scenes in the Unity editor there is not enough time to carry out the lighting calculations. As a result, the colours appeared darker when loading the next scene. This means that the bug will not need to be fixed, because once the final exactable file is created there will be no problem with the lighting.



Build Settings

Scenes In Build

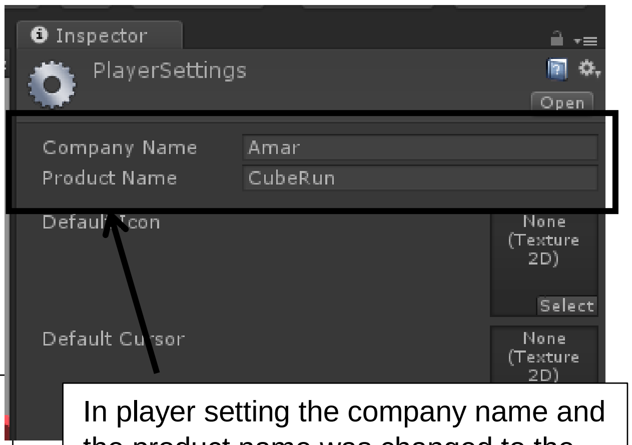
✓ Menu	0
✓ Level02	1
✓ Level03	2
✓ Level04	3
✓ Level05	4
✓ Level06	5
✓ LastScene	6

Platform: PC, Mac & Linux Standalone

Target Platform: Windows

Architecture: x86

The first part of the export phase involved adding all the required scenes to the builds settings and selecting the platform for which the game is being built for. The architecture 32-bit was chosen since the game does not have high graphical calculations. By selecting 32-bit the game will also be run on higher architecture such as a 64-bit system.



Inspector – PlayerSettings

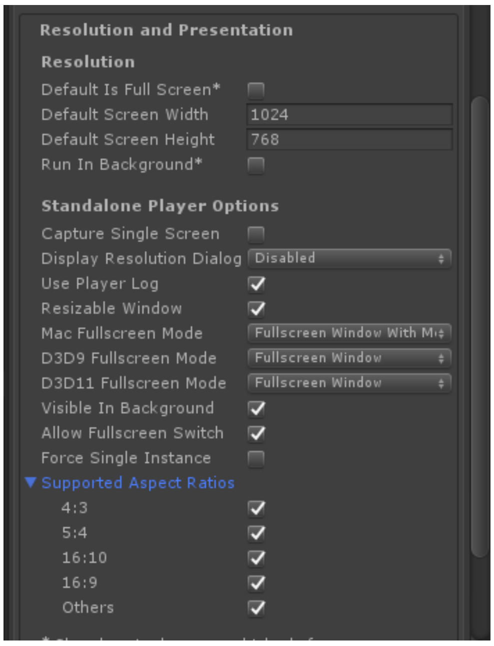
Company Name: Amar

Product Name: CubeRun

Default Icon: None (Texture 2D)

Default Cursor: None (Texture 2D)

In player setting the company name and the product name was changed to the following. An icon could have been added but, since one was not created it was not added.



Resolution and Presentation

Resolution

Default Is Full Screen*	☐
Default Screen Width	1024
Default Screen Height	768
Run In Background*	☐

Standalone Player Options

Capture Single Screen	☐
Display Resolution Dialog	Disabled
Use Player Log	✓
Resizable Window	✓
Mac Fullscreen Mode	Fullscreen Window With M
D3D9 Fullscreen Mode	Fullscreen Window
D3D11 Fullscreen Mode	Fullscreen Window
Visible In Background	✓
Allow Fullscreen Switch	✓
Force Single Instance	☐

Supported Aspect Ratios

4:3	✓
5:4	✓
16:10	✓
16:9	✓
Others	✓

The "default is full screen" box was unticked as the game will not need to star in full screen – player will be able to change the size of the screen to suit them.

The resizable window option was ticked because this will allow the game screen size to be adjust by the player depending on what size screen they want. Since "default is full screen" has been unticked this option will allow player to also play in full screen if they wish.

All aspect ratios were ticked. This will allow the game to be run on many different size monitors.

5.3 Testing

Sprint 4 involved test reference 4.1.1 – 4.5.1. Refer to the testing table for a summary of the tests that were carried out during development of this sprint.

Unit testing was carried out once the code was developed to load the next level (display messages to the player) and to allow the users to control the menu. The test table shows the tests that were carried out when testing the code that was implemented for the feature (refer to tests 3.1.1 – 3.4.3).

Integration testing was carried after each subtask of the sprint was completed. When carrying out integration testing for this sprint only test questions from 2.5.1 onwards were carried out each time a subtask was completed. When carrying out integration testing once the first subtask of this sprint was completed, there was an error with the scene one (level 01) of the game. This error was not fixed and as a result, scene one has not been included as part of the final game.

Regression testing was carried out once a solution had been implemented to fix the bug found when carrying out test reference 4.1.1. Once this bug was fixed, test reference 1.3.1 – 1.3.5 and 2.2.1 – 2.2.6 were performed again to make sure that the player movement was not affected in any other way. In this sprint, no further regression testing needed to be carried out.

Usability testing was carried out in this sprint twice in this sprint. It was carried out once the UI features such as the game over text and level complete text were added. Furthermore, usability testing was carried out again after the main menu was implemented. The results from carry out usability testing have been summarised and the relevant actions have been taken to solve the issues that occurred.

5.4 Sprint 4 Review

At the end of the 4th sprint, level complete and game over GUI have been implemented. This fulfils success criteria reference 2.3 and 8.2 because when the player-controlled age object reaches the end of the level, the level completed GUI/message is shown. Furthermore, when the player-controlled game object collides with an obstacle during game play, the game over GUI/message is shown. This fulfils success criteria reference 8.1. Furthermore, a main menu has been implemented which allows players to interact with the game conveniently and to navigate to certain parts of the game. This fulfils success criteria 5.1 and 5.2. The game includes many levels, with each level varying in difficulty. This fulfils success criteria 4.1 and 4.2 since each level completed by the player, increases the speed of the player-controlled game object moves forwards.

This sprint has been successful in fulfilling the success criteria it aimed to fulfil. Furthermore, using the prototypes that I have created the usability testing has been successful and the relevant actions have been taken to solve the issues that occurred. Due to not having enough time to fulfil any further success criteria, the final executable file has been created. I will now move onto carrying out post development testing and evaluating the solution.

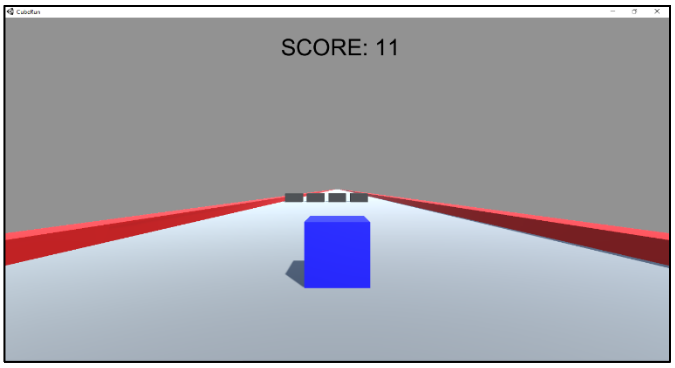
6.0 Evaluation

6.1 Post development testing for functionality and robustness

Before I proceed with any further testing, I have identified a few post development tests that need to be carried out. Once I have carried out these post development test I will be carrying out further test that test the robustness of the game and a final usability testing will be carried by a large group of my stakeholders. For any errors or bugs that I find I will not be fixing them however, might describe potential solutions to fix the errors.

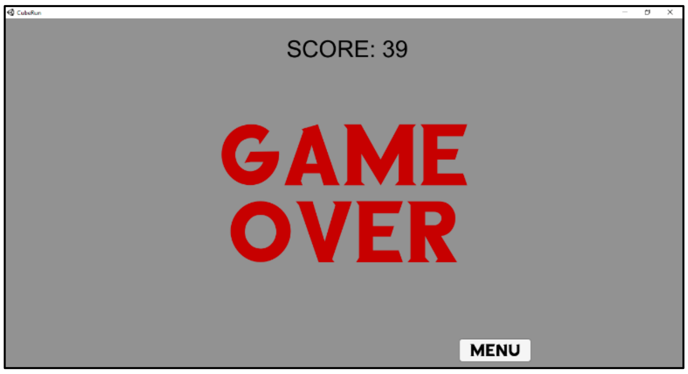
6.1.1 Post development tests

Test reference 1.4.3



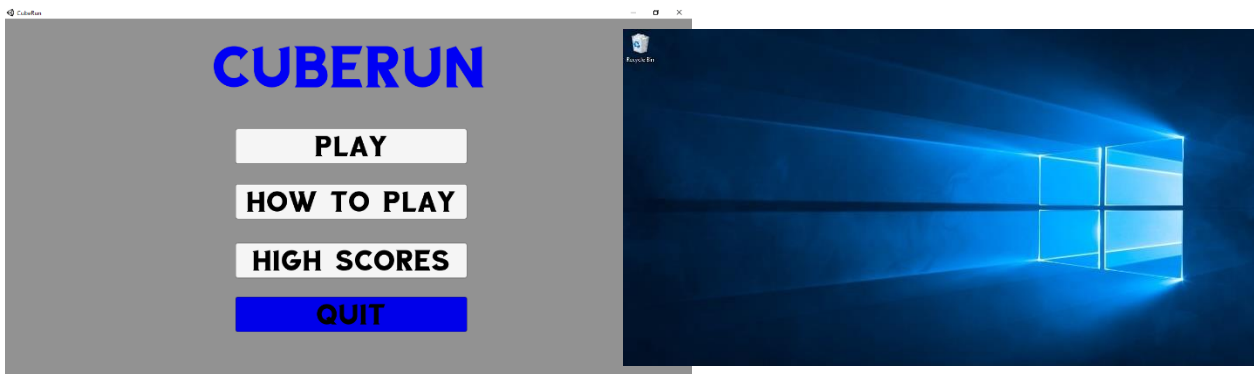
The screenshot on the right shows that even when the game is being player in full screen, the player game object is clearly visible and the compete width of the ground plane can be seen. Furthermore, players can see the obstacle clear in the distance. This is enough evidence to show that the test has succeed.

Test reference 2.3.4



The screenshot on the right shows that when the player controlled game object collides any obstacles part from the first set the player does stop moving and the game over GUI is displayed to the player. From this point the player can select the main menu and this takes them back to the menu. This is the expected action that should be performed. This is enough evidence to show that the test has succeed.

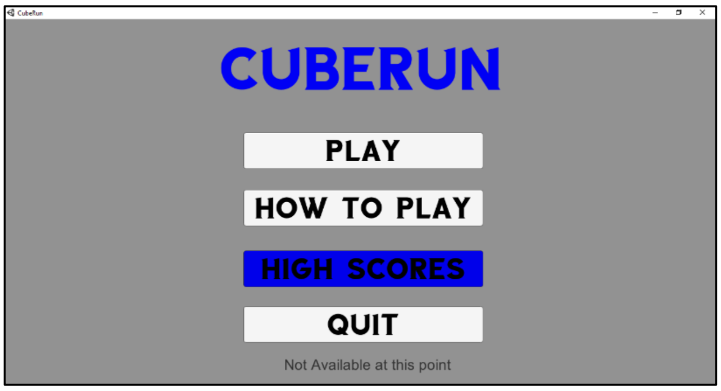
Test reference 4.3.6



The screenshot shows that when players press the quit button on the main menu the application closes. This is enough evidence to show that the test has succeed.

Testing for function and robustness

Main menu

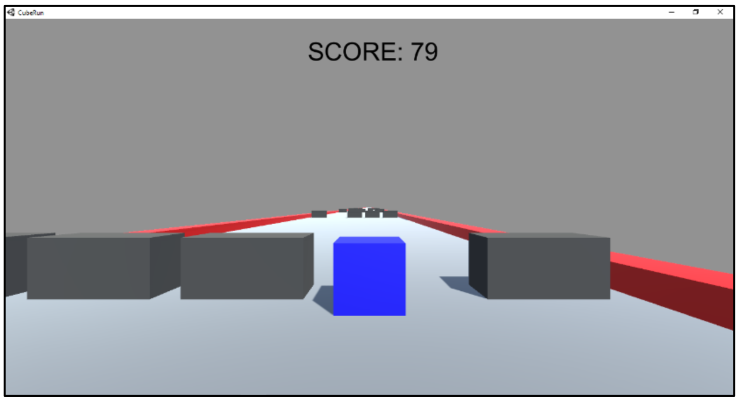


The main menu can be navigated using the mouse. The correct action is performed when the buttons are pressed using the mouse from the main menu. For features that are not available such as the high scores a message is displayed saying the feature is not available. One issues with the main menu is that when the high scores button is clicked it shows the relevel text however it keeps the text displayed until another button is pressed. A potential solution to this problem would be to only show the text when the cursor hovers over the button.

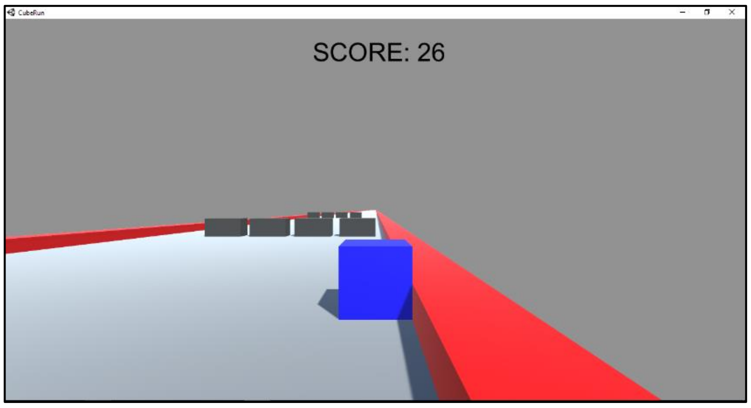
Game Mode

When pressing the play button from the main menu the first level successfully is loaded. The player game object can clearly be seen, and the player score starts from zero. The score increases as the player constantly moves forwards. The player can move left and right either using the arrow keys or the a and d keys.

Player controls



The player-controlled game object can only move left or right using the arrow keys or the a and d keys. If any other keys such as the spacebar or other letter keys are pressed, then the player object only constantly move forward. Furthermore, if the right arrow key is pressed along with the a key (which should move the object left) then the player object only constantly move forward. As the expected actions are performed when invalid keys are used this confirms that the player controls function as expected.

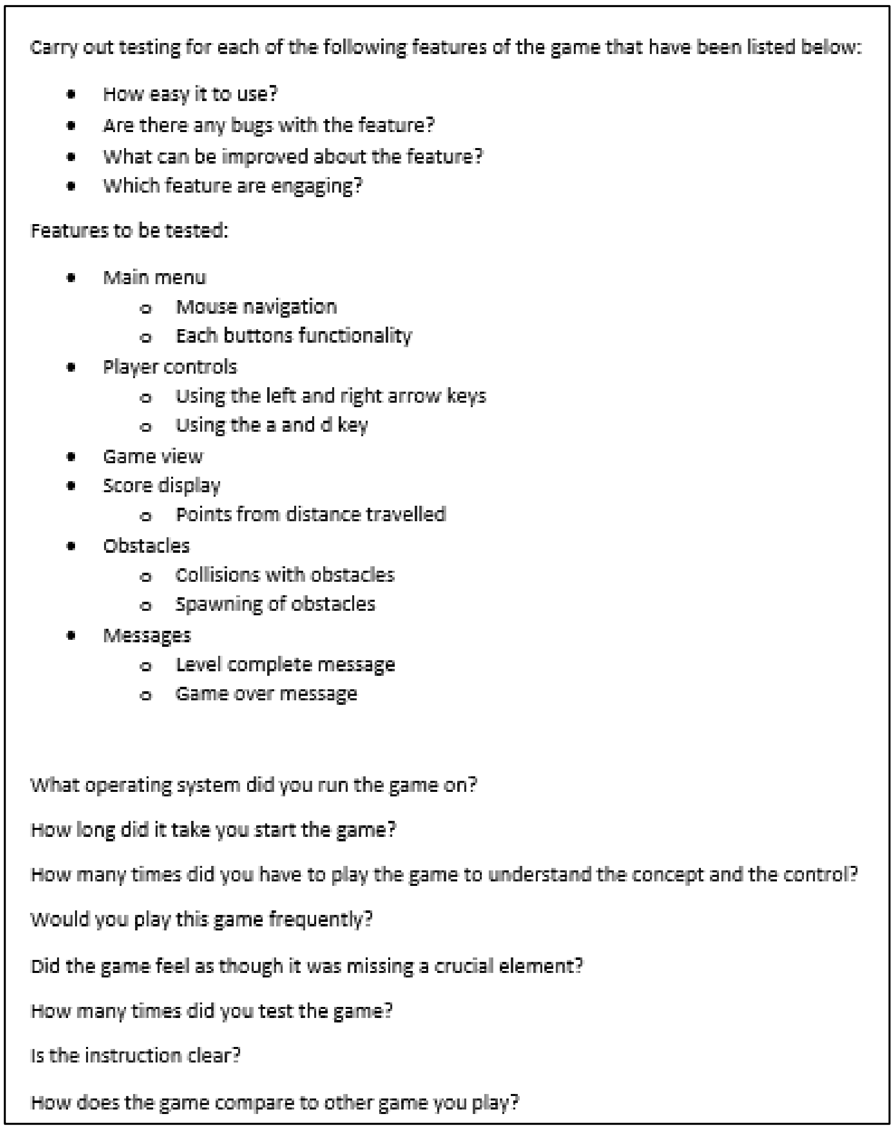


The screenshot on the right shows that once the player-controlled game object tries to move to the, its position does not exceed the maximum. As a result, the player controlled game object does not fall from the ground plane. This is also true for the left direction. When the player controlled game object moves to the left direction, it cannot exceed the width of the ground plan in the x direction.

For more details on testing for robustness refer to the Appendix C – Post development testing.

Usability testing

The project is now at a point in development, where it is possible to give my stakeholders a copy of the executable game file to carry out usability testing. To collect specific feedback from my stakeholders, I will provide them with a text document (screenshot below) which will include questions about features of the game that have been implemented.



Carry out testing for each of the following features of the game that have been listed below:

- How easy it to use?
- Are there any bugs with the feature?
- What can be improved about the feature?
- Which feature are engaging?

Features to be tested:

- Main menu
  - Mouse navigation
  - Each buttons functionality
- Player controls
  - Using the left and right arrow keys
  - Using the a and d key
- Game view
- Score display
  - Points from distance travelled
- Obstacles
  - Collisions with obstacles
  - Spawning of obstacles
- Messages
  - Level complete message
  - Game over message

What operating system did you run the game on?

How long did it take you start the game?

How many times did you have to play the game to understand the concept and the control?

Would you play this game frequently?

Did the game feel as though it was missing a crucial element?

How many times did you test the game?

Is the instruction clear?

How does the game compare to other game you play?

The results from carry out usability testing have been summarised below (discussed in more detail within in the evaluation):

- Some found the first level too easy
- Player controls were not what was expected by some – took more goes than expected to get a grasp of the controls
- Score was not a continuous count across the levels
- No pause menu – no way to access the main menu while playing the game
- No effects when player collides with an obstacle
- Game over GUI shown too quickly

6.2 Evaluation of solution

The solution that has been devised will be compared with the original success criteria. This will allow an overall conclusion to be decided upon whether the solution is suitable and if it will help solve problem (identified in the analysis section) that it aimed to solve, or if further development needs to take place before it does.

Success criteria 1.1

All the level that are a part of the game have the basic game features. For example, each level has a ground plane and an object that the player is represented by. The criterion has been fully met because test reference 1.1 and 1.2 shows that when the game was being tested a grey platform and a blue cube appeared in the game scene. Furthermore, post development testing shows that the criterion has been met since each level starts with these basic game features in the scene.

Success criteria 1.2

The player-controlled game object can be controlled by the player to move it to the left or right on the ground plane. Players can use either the left and right arrow keys or the a and d keys to control the player. The test references 1.3.1 – 1.3.5 and 1.3.1a – 1.3.5a show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

If the players inputs valid keys (such as the left and right or a and d keys) the player moves in the corresponding direction (shown by test reference 1.3.1a and 1.3.2a), however if invalid keys are used such as the spacebar or the up arrow, the player controlled game object does not move in either left or right directions (shown by test reference 1.3.3a and 1.3.4a). Furthermore, validations test for user input were carried out to make sure that the system only responded to the correct input and these passed, which provide further evidence showing that the criterion has been met.

However, when carrying out usability testing during the post development phase, some people found that the player-controlled game object did not respond to user input to well. They found that it did not response to user input smoothly. Also, comments included that it took my stakeholders a few times playing the game before they felt comfortable with the controls. Even if the criterion has been met, there are still issues that my stakeholders were not pleased about. The issues with the user input not feeling smooth enough can be addressed in future development by redeveloping the player movement script. The position to which the player-controlled game object should be moved can be calculated with higher precision. This will mean the game object will move to a more accurate position in the scene.

Success criteria 1.3

The camera in the game is set to a suitable position to allow the player to see the object that represents them in the game. The position of the camera has been set to a suitable position since players have a clear view of the obstacles that they need to avoid. Since the camera in the scene is always the same distance from the players-controlled game object, players have a clear game view for all the levels of the game. The test references 1.4 – 1.4.3 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

The criterion has been fully met because the tests show that when the player controlled game object moves to the left, the camera in the scene follows the player (shown by test reference 1.4.1) This is also true for when the player moves to the right (shown by test reference 1.4.2). Furthermore, since player have a clear view of the game the criterion has been fully met with no issues that need to be addressed in future development.

Success criteria 2.1

Success criterion focus on obstacles that players must avoid. This criterion is achieved if the obstacle game objects are placed on the ground plane and when the player-controlled game object collides with them then it stops the player-controlled game object from moving. If the player-controlled game object collides with an obstacle, it stops player from controlling the game object. However, none of the obstacles are pre-placed (added to the game scene manually during the design of the level).

Scene one (level one) that was developed did include pre-placed obstacles however, level one was not includes as part of the final solution. Furthermore, each level of the game starts differently. The criterion has not been met because pre-placed obstacles were supposed to be at the start of each level to allow the player to adjust to the level before experiencing the spawned obstacles. This issue can be addressed in future development as the error from scene one can be fixed so that this scene can be included as part of the final solution. Furthermore, to allow pre-placed obstacles to be at the start of each level, the time at which obstacles are spawned can be set to spawn a certain amount of time after each level is start. This way pre-placed obstacles can be added to the start of each level.

Success criteria 2.2 and 8.1

When the player-controlled game object collides with an obstacle, the player is no longer able to control the game object and the game over GUI is shown to the user. When the game over GUI is shown to the player, there is the option to return to the Menu, where players have a number options to select from. The test references 2.2.1 – 2.3.4 and 4.1.6 – 4.1.7 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

If the player-controlled game object collides with an obstacle, the player game object is no longer able to be controlled by the player (shown by test reference 2.2.1 and 2.2.3). Once the player-controlled game object has collided with an obstacle the game ends showing the game over GUI to the player (shown by test reference 4.1.6 and 4.1.7). When carrying out post development testing, this was further testing by colliding the player-controlled game object with different obstacles in the scene. The results were that the player-controlled game object is no longer able to move and the game over GUI was shown. This provide further evidence showing that the criterion has been met.

The results from carrying out usability testing during the post development phase shown that my stakeholders expected an effect to be seen when the player collided with an obstacle, rather than the game over GUI loading straight away. The issue that the player does not see a certain effect when the player-controlled game object collides with an obstacle can be addressed as a delay by a small amount can implemented so the game over GUI does not load straight after a collision. Furthermore, an effect such as the obstacle being destroyed when the player-controlled game object collides with it can be implemented.

Success criteria 2.4

While the player is playing the game, the obstacles are spawned across the ground plane. The obstacles are spawned in rows as sets of four. The obstacles are spawned in a uniform order with an obstacle not being at a random point from the row. The test references 2.5.1 – 2.5.4 and 3.1.1 and 3.1.4 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

Since there is no obstacle spawned in at a random point from the row (shown by test reference 2.5.2), this makes the game more interesting and means that my stakeholders can play the game repeatedly due to it being a different challenge each time. The obstacles are spawned whilst the game is being played (shown by test reference 3.1.1).

Currently, the game has set distance between spawned obstacles. However, to make it more entertaining the gap between spawned obstacles can either vary each level or vary between each set of four obstacles. This can be addressed in future development as random number can be chosen between 1 to 5 and that will be the distance the obstacles will be spawned, from the previously spawned obstacles.

Success criteria 2.5

The player-controlled game object is not able to fall off the ground plane. The player movement is restricted in the left and right direction. The test references 3.4.1 - 3.4.3 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

If the player keeps holds of either the left or the right arrow key the player-controlled game object does not exceed a certain position (show by test reference 3.2.3 and 3.2.4). This is an example of and extreme input that attempts to “break” the program. Since the left and right arrow keys were held down when carrying out these tests, the solution that has been implemented (to restrict player movement) does meet the success criteria. As valid and extreme test data were used when testing and there are no unsuccessful tests, this further shows that the criterion has been fully met.

Success criteria 3.1 and 3.2

As the players are playing the game the score that is achieved by the players does depend on the distance travelled. Also, as the game is being played the score is updated during play. After carrying out usability testing, the format and display of the score is clear for player to see. The test references 3.3.1 - 3.3.3 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion 3.2 has been fully met. The test reference 3.4.3 show test that were carried out to assess if criterion 3.1 has been met. Since the tests were unsuccessful, it shows that the criterion 3.1 has not met.

As the players position on the z axis increases, the players score is updated in real time to show this positions as an integer to the player (shown by test reference 3.3.1 and 3.3.2). This shows that the criterion 3.2 has been fully met. However, criterion 3.1 has not met because the score obtained from one level is the score that the next start on. The score count is not continues across the levels (shown by test reference 3.4.3). This issue can be addressed in future development buy the use of variables. A variable could be used to save the score for each level. These variables can then all be added together, to give the player a final score. However, this will mean that the continuous players score for each level will not be shown to the player until they have completed all the levels or collide with an obstacle.

Success criteria 4.1 and 4.2

The game has a total number of five levels. The levels load automatically (after waiting a short time) once the player has completed each level. As players progress through the levels, the player-controlled game object moves forwards at a faster speed and levels require a longer amount of time to play.

To create all the additional levels, a copy of level two was used and changes were made to vary the levels. When carrying out development no tests for each level were carried out. However, level tests were carried out during post development (can be seen in the post development section). Both criterion have been fully met because when the player completes a level the next level is loaded automatically, and the gameplay varies between level.

Success criteria 5.1 and 5.2

The main menu has a simple graphical user interface. Players can access the main menu from the game over screen and once they have completed the game. The mouse is used to press the buttons on the main menu (no other input method is supported). The test references 4.3.1 – 4.3.8, 4.4.1 – 4.4.3 and 4.4.4 – 4.4.6 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

If players, press the buttons on the main menu then the correct action is performed. When the play button is pressed the first level of the game is loaded (shown by test reference 4.3.3). When the how to play button is pressed, the instructions are shown on how to play the game (shown by test reference 4.3.7). Also, when the quit button is pressed the windows application close (shown by test reference 4.3.6).

When carrying out usability testing questions about the GUI were asked. The responses were that the GUI made the main menu easy to use. This provides further evidence showing that the criterion 5.2 has been met as player are able to use the GUI to access other parts of the game when they wanted. However, one issue that will need to be addressed in future development will be the ability to access the main menu when playing the game. The issue can be addressed by the implementation of a pause menu. When players access the pause menu the game will be paused and from this menu they will have the option to return to the main menu.

Success criteria 8.2

When the player completes a level, the player, the player is no longer able to move, the level completed GUI is shown. Once the level complete GUI is shown after waiting for a few seconds the next levels loads. The test references 4.1.1 – 4.15 show the tests that were carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

Once the player completes a level the, the level completed GUI is displayed (shown by test reference 4.1.2). This is only shown when a player completes a level. After a few seconds of showing the level complete GUI, the next level of the game is loaded (shown by test reference 4.1.5). If a player completes all the levels then a message is shown the player, and the option to return to the main is given.

Success criteria 10

To play the game no additional or any special software is required. The solution that was developed in Unity has been executed to a final executable file which can be run on many PCs. The test reference 4.5.1 show the test that was carried out to assess if the criterion has been met. Since none of the tests were unsuccessful, it shows that the criterion has been fully met.

When carrying out usability testing, it was found that when the executable file was run on PCs lower than windows 10, the player controls did not work smoothly. The player-controlled game object did not to the left and right, however it lagged. Player movement was the only feature of the game that was affected when running the final executable file on a PC with an operating system lower than windows 10. This issue can be addressed in future development by making sure that the game frame rate is independent. This will mean that rather than the player movement depending on the frame rate of a player's PCs, it will depend more on time.

Overall, the solution has fulfilled a large amount of success criteria however, many issues remain with usability features and there are features missing from the solution. Further development will be needed to be taken place before the solution aims to solve the problem my solution aims to solve as identified in the analysis section.

6.3 Maintenance and limitations

A limitation of the solution relates to the spawned obstacles in the game. Currently, once the players have a been faced with a set of obstacles and they have passed them, the obstacles game object are not destroyed (they still exist in the scene until the next level is loaded). If in future development the levels were made longer then this would mean that there will be many of the same obstacles in the scene. This no efficient on the computer since it stores more in memory than required. A possible solution that could be implemented would be to destroy the obstacles from the scene in relation the players-controlled game objects positions. For example, if the player game object was a certain distance on the x-axis then it should delete all obstacles with a position less than that of the player game object.

Another limitation of the solution relates to how the player controlled game object moves. From carrying out usability testing a few of my stakeholders that were approached to carry out testing has issues with the way the system responds to user input. It is important that, issues with user input are addressed since this is how player interact with the game. A solution that could be implemented would be to redesign the player movement script. This redesign will need to involve higher prevision calculation to which position the player should be moved in the scene.

Another limitation of the solution is the way in which more levels were created. A copy of scene two (level two) was made multiple times during development. Changes were then made to the copies of the level to create variation in gameplay. However, this was not a very efficient method because it limited how much the features for a specific level can be changed. A solution that could be implemented to make gameplay more interesting would be to have more of a randomised approach to levels. For example, the distance between spawned obstacles could be different each time. Also, the gap that players must pass through can be of a different size each time.

From the success criteria there were many planned features that could not be implemented into the game due to time limitation. These features include:

- Saving players high scores (criteria reference 6.1 and 6.2)
- Screen effect that make the game challenging ((criteria reference 7)
- Video tutorial on how to play the game (criteria reference 9.1)

Before, implementing player score saving, it would be important that a solution is implemented to issue with the player score starting from zero for each level. As mentioned, a variable could be used to save the score for each level. These variables can then all be added together, to give the player a final score. However, this will mean that the continuous players score for each level will not be shown to the player until they have completed all the levels or collide with an obstacle. Therefore, an alternative solution would to add the score from the before the start of the level. This could be achieved by the use a static variable that will be used to save the player score.

Once the solution, for an overall score has been implemented, then player high score can be saved. To implement this, feature some research would be needing to be carried out, since I do not know how the save data with a game developed by using Unity. A solution, that could be used to implement score saving would to use a text file that save the players score. The score in this text file can then be compared to determine which of the scores are the highest top five. These scores can then be accessed by player from the main menu of the game.

To implement screen effects, different types of obstacles could be spawned in the game at random times. If players collide with a certain type of obstacle then this would cause for example, the scene camera to shake and as a result it would look like the screen is shaking. Also, if player collided with another type of obstacles then other effects such as paint being splatted onto the screen could take place. Bu implementing screen effects this would make the game more entertaining and challenging at the same time.

Furthermore, the easiest feature to implement would be to the video tutorial on how to play the game. To implement this feature, I would have to create and edit the video footage playing the game. This video file will then need to be embedded into the game so that when the players click on the “how to play” button on the main menu the video file should play rather than them having to read instructions.