Analysis section

Introduction

Since the 90s, there has been a stigma against video games due to many people believing that video games are a waste of time and money and nothing useful comes out of it for young generations. Many youths spend their time away to play video games with a smaller focus on education to them, mostly due to video games being more fun than sitting in a classroom hour at a time. Many parents and teachers believe it may not benefit them in the future as it does not improve their academic knowledge. With this belief in mind, I have taken the opportunity to bridge the gap between videogames and education with my new element platformer.

The basis is simple enough. You will be placed in a side-scroller-esque game where you are an element cube going sideways, jumping on platforms while avoiding cubes. The stationary cubes are of different elements and depending on the element you encounter; you may react with it (game over) or just pass right through it. If you react with an element or fall off a platform, it is game over. If it is a reaction, a text will show what the product of the reaction was. There may be a 2-minute timer, where the player must survive. They can always return and have a fun time in trying to survive until the timer stops. There will be tubes to collect which will help in gaining points for their score. Their name and score will be stored on a leader board. C# will be used to create this idea and it will be powered by Unity.

This interactive program will be a very helpful game for students who want to learn more about the chemistry of elements while not being considered boring to them.

Problem identification and features

Problem Recognition:

The problem which I decided to overcome is children playing games without it benefiting them in an educational aspect. Parents or teachers may often get frustrated with a child when they prioritize games over their education because of the fun-factor that classrooms are missing. The overall problem is finding a way to bridge the gap between their education and entertainment, whereas the actual underlying problem is finding a game to entice students into playing it. This gave me the idea of a side-scrolling platformer. Side-Scrolling games have been around since the ‘80s the formula has rarely changed from travelling from point A to B. Adding variable obstacles to the game (the elements of the stationary cubes) gives my game a unique and educational aspect. This combines education in a subtle yet interesting way as students can apply what they learnt in chemistry while they play. This will help in overcoming my problem. In summary, I want to create a fun side-scrolling game that players can play while having some sort of learning curve related to their education.

Computational Methods:

The problem is suited to a computational approach for many reasons. The solution will be a game that uses a keyboard and mouse to navigate through the menu and control the character in the game. The solution will need to run on a computer as creating virtual controls on a smartphone, as

well as the assets, will consume too much space on the screen. Having the solution run on a computer will provide an environment where the controls are separate from the screen.

Decomposition: The problem can be decomposed into simpler steps.

1. The player will press play and will be able to start the game.
2. The user will jump on platforms and avoid certain cubes.
3. If the player is still alive once the timer has reached the end, they win. If they end up falling off the platform or reacting to a cube, it is game over and shows the product of the reaction.
4. After the game is over, their name and time-taken will be stored on a database and will be displayed on a leader board.

These steps outline each key stage that the program must have to work correctly. The end-result should have no problems in transitioning from one stage of the program to another.

Divide-and-Conquer: Divide-and Conquer will be of significant help on my project as I can divide my task into several sub-tasks, allowing me to make the problem more manageable to solve. Focusing on developing each stage at a time allows me to focus on the task-at-hand and not interfere with other stages of the solution.

Abstraction:

Abstraction removes or hides these details from the view of the user to make the process appear simple.

The graphics will be restricted to 2d as 3d graphics are not necessary for a platformer game nor does it add any extra use. The colour scheme may change but will remain simple, with duller colours for the background and brighter colours for the sprites. There would be much more detail such as realistic textures for each element (such as gold or aluminium) but a realistic graphics is not required for my solution.

Movement will use the traditional controls for pc games: ‘W’, ’A’, ’S’, ’D’ or the directional arrow keys. These keys will individually correspond to the direction of movement of the player. Pressing ‘D’ or the right arrow key will result in the character moving to the right. Pressing ‘W’ or the up-arrow key will result in the player jumping. The cubes of the elements are abstractions of real elements (such as magnesium, gold sulphur). The movement is an abstraction of how a liquid element would move since they cannot ‘jump’ in real life rather they can flow.

The game will have a timer placed on one corner of the screen.

Simple visual effects will be added such as when the player reacts to another cube of a different element. This is an abstraction of how real reactions occur as real reactions will last for longer periods of time or be loud and flashy.

Sound effects will be added such as when the player jumps and lands, reacting with a cube and falling off a platform. This is an abstraction of the sounds that an element would make when they fall onto something or react with another element.

All usernames will have 2-10-character limit lock on the leader boards due to the possibility that a player’s full name may go over 15 characters which may affect the formatting over the leader board.

Thinking ahead:

Thinking ahead is the process of the user identifying what the preconditions, for a project, are and the inputs and outputs required ahead of creating the solution. Using this will give me a rough idea of the problems I might face and the input and output devices I'll need prior to creating my solution.

•The inputs for my solution will require input devices such as a mouse and keyboard to navigate through the menu and play the game.

•The output of my solution will require a monitor as a necessity so that the player is able to see the game and speakers would be needed if they would like to hear the game.

•If the player does not wish to complete the game, the pause menu should provide a 'quit' option to go to the main menu

•The scores of the leader board will be stored in an external file so that they are not lost when the program is closed. The precondition is that the scores are stored in order from highest to lowest so that a sorting algorithm can use it to display the scores in-game.

Pattern/Problem Recognition:

Recognising patterns in the coding of my program: When there is a pattern in my program I will have to identify it and find a solution to increase the efficiency of my code. Loops such as while loops have the potential to fix this. A repeated section of my program can be converted into a method, as they can make your code more efficient and much easier to manage/read due to a reduction in lines.

Problem recognition in my program: Recognising and acknowledging issues, usually brought up from testing the program, and being able to determine what is causing the problems and addressing it. For example, if a player has not reacted with a cube despite the element being appropriate, I will have to identify and resolve that problem. It is common for problems to arise simply due to a syntax error.

Stakeholders

The demographic for this program is quite varied. It ranges from children to parents who want to educate their kids and also to students who want to learn more about space while having fun.

Key Stakeholders:

Students: Students are key stakeholders in this quiz. My classmates, Hamid Chowdhury and Azim Khanazada are students who both usually like to spend their free time playing games or engaging in leisure activities and would rather not engage in more work after spending their entire day in school. This often leads them, and to a lot of other students, to not learn at home due to the thought of boring themselves for potentially hours at home, just to maintain their education.
The side-scroller allows them to have a fun and engaging game while subtly learning about the "chemistry" of the elements in the game.

Parents are stakeholders for this game, parents like Lenard Herney and Netta Mill. Who are hard-working parents that wish their children to be successful in life (Like many other parents would). Lenard and Netta, as well as many other parents across the UK, have expressed their concerns that their children are struggling to find free time for themselves as they are caught up by the pressure of their exams. Both parents, as well as many other parents across the UK, have expressed their concerns that their children are spending too much time on games over education/ productivity. My game will be their solution to that problem as it will enable students to enjoy themselves in a

traditionally styled game with unique elements while also learning, which in-turn, satisfies both parties.

Research the problem.

Interview with Hamid Choudhury and Azim Khanazada:

I have considered asking my classmates to participate in an interview to outline what key aspects that this platformer must have. This also allows me to remove anything that may not be favourable to them and their feedback will be used as a guide for creating a game that meets their standards.

Introduction questions:
•Have you ever played a side-scrolling platformer before?

•Do you like platformers?

•What are your opinions on games that also have some element of education?

The make-up of a good game:
•Are sound effects necessary in your games?

• Should I feature a detailed GUI?

•Should I feature visual effects?

Final Thoughts:
•Would you like animations in the menu?
•Are realistic models and graphics necessary?

<u>Interview with Hamid Chowdhury and Azim Khanazada 25/06/2020</u>

Introduction questions:
1) Have you ever played a side-scrolling platformer before?

Hamid: Yes, I have played some on some old gaming websites.
Azim: Yeah, I'm familiar with them. I have played Mario and some flash-based games before.

2)Do you like platformers?

Hamid: Yes, as I find them easy to pick up and play.
Azim: They are alright since they do not require a steep learning curve.

3)What are your opinions on games that also have some element of education?

Hamid: It is great for me as it keeps me engaged with the game.
Azim: I've seen games that have educational elements, but I don't think that those types of games are even fun.

The make-up of a good game:
4) Are sound effects necessary in your games?

Hamid: I find them necessary as it keeps me immersed in the game.
Azim: Of course, otherwise the game would feel unprofessional.

5) Should I feature a detailed GUI?

Hamid: I would rather have a simple-looking GUI.
Azim: I don't mind as long as it is easy to understand.

6)Should I feature visual effects?

Hamid: They are not a necessity, but I would not mind.
Azim: You can add them but if they are not too distracting or jarring.

7)what is your opinion on avoiding enemies based on your academic knowledge

Hamid: I feel that would be something refreshing for an educational game.
Azim: That seems interesting.

Final Thoughts:

8)Would you like animations in the menu?
Hamid: They are not that important because I do not really notice any animations in the menu.
Azim: Yeah, it adds detail.

9)Are realistic models and graphics necessary?

Hamid: Not really.
Azim: Not necessarily if the gameplay is fine.

Overview of the answers:

My goal of interviewing them was to understand what students would want in the game. Both students have played platformers before, so they are familiar with those types of games and do have a liking towards it. What I can take from that is that the solutions should have controls like that of other platformers so that it's easy to pick up and play. The scepticism that Azim has expressed over educational games tell me that they may have not been engaging. In terms of the make-up of the game, it seems that sound effects are heavily desired in the game whereas a detailed Graphical User Interface does not seem important as Hamid dislikes it while Azim wants a simple-to-understand interface. Visual effects can be added if they are not distracting the player. The idea of applying academic knowledge when avoiding enemies seems welcomed for my solution. Adding menu animations, to the solution, will decided in the interviews with the 2 parents as Hamid and Azim have different feelings on menu animations. My solution does not require to have realistic graphics and models.

<u>Interview with Lenard Herney and Netta Mill 26/06/2020</u>

1)What are your opinions on games that also have some element of education?

Lenard: I think they are good for students however there aren't many available and.
Netta: I believe that games with education can be really good as long as the education don't overshadow the gameplay.

2) Should I feature a detailed GUI?

Lenard: A detailed GUI is not needed if the game works.
Netta: It does not need to be detailed; I would rather have one that is more straightforward.

3)Should I feature visual effects?

Lenard: Yes, students could enjoy the game more but do not make it flashy.
Netta: Visual effects should be added.

4)Would you like animations in the menu?

Lenard: No, that is unnecessary.
Netta: I would not mind but they usually go unnoticed.

5)Are realistic models and graphics necessary?

Lenard: No.
Netta: The graphics should not be realistic. They should be simple so that they can run on ordinary computers.

The goal of this interview was to compare the other stakeholders', the parents', views to those of the students. I reduced the number of questions due to the students being the main targets of this game, so I chose the questions that would be relevant to the parents. From the answers to the first question, I have concluded that the games with some element of education would be favoured in my solution. The consensus is that a simple GUI is preferred. Visual effects should not be distracting in my solution, based on the feedback. Menu animations are unlikely to appear in the solution as they have been found to be unnoticed by 2 of my stakeholders.

From these 2 interviews I have understood that detail-in-design was not a big interest from both parties (Parents and Students). The solution should have a good implementation of chemistry and will need to have a good balance between education and entertainment.

Researching Similar Solutions:

New Super Mario Bros: Mario is known to many as one of the oldest platformers that have existed. 'New super Mario Bros' has similar features to the original. The player is placed in pre-built levels where they must avoid enemies and not fall off the map while trying to reach the end goal in a certain amount of time. If the player does not reach the end before the timer reaches 0, it is game over. There are different types of enemies: some walk on the platforms while some are stationary. The game has an item system where it can grant the player power ups that help them traverse through the level.

Menu:

This game has a big title so that it is eye-grabbing. The start screen is slightly animated with a moving background. The menu is intuitive as shows the options of the game in a simple format. The menu shows the characters, but I could only capture a frame. There is background music playing.

What I can apply to my solution:

I like that the menu makes the game seem welcoming with its picture moving in the background and has an animation for the players to look at if they have not selected an option, I could add an image of the game with the character and enemies instead of a blank background with options. I would like to replicate how the menu keeps the options simple and accessible for students.

Experience in-game:

I have taken 2 screenshots in-game and have placed them side by side. The player moves the character to the right where he can jump over pits, like in the first image. If the player falls into the pit, it's game over. The second image shows the player encountering a moving enemy and using a power-up to avoid losing. There is a counter on the top right corner which indicates how much time the player has left to complete the mission. The item system is what the player can use to activate a power-up item that they have found. The bottom of both screenshots shows the player's progression though the map. There is a coin system in which the player can collect and gives the player an extra life if 100 coins are collected.

What I can apply to my solution:

The timer in my solution will be placed in the corner of the screen, like the one shown in the game above so that it does not take too much space. I will be using the 'fall' feature where the play loses when they fall off the platform. I am considering adding a power-up option which grants the user an ability to not react with cubes for a limited amount of time. Just before the level started there was a small gesture which told me to "get ready", I will be adding a "get ready" and "start" message in my solution. The idea of moving enemies could make my game more engaging as well as adding a point system such as the coin system which is used in the Mario game. My game will not feature a progress bar as it'll be designed as an endless platformer until the timer runs out.

Firebox and Water girl in The Forest Temple: One of the stakeholder's, Azim Khanzada, recommended to me to check out some of the gameplay mechanics of 'Fireboy and Watergirl in The Forest Temple'.

Menu:

The Menu has a large eye-grabbing title with a design that complements the game, the balance between using Fireboy and Watergirl. The Menu is kept simple with three options to choose from and having a large play button to direct the player immediately. The characters placed on both sides with a slightly animated background makes the game menu memorable, as it does not have a bland design.

What I can apply to my solution:

Based on the intuitive design of the first game analysis (New Super Mario Bros) and this game, I have decided to create a simple menu with a nice background which shows the cubes of different elements in the background. The play button will be bigger than the other buttons. Background music may be added as it was a common factor in this game and the Mario game.

Experience in-game:

The game starts off by showing the controls of the characters, along the background of the game, with their respective colours. It shows collectible diamonds for each player and has messages to guide the player such as "Never mix fire and water". This is not shown in the screenshot, but it gives a rank to when the level is completed.

Game Over menu:

If a character mixes with a different element, there is a puff of steam and the game-over screen pops up. It shows the two options to retry and go back to the menu.

What I can apply to my solution:

I like the idea of adding instructions to the start of the game as it will be helpful for students who wanted to play the game straight away. I can use the points the player accumulates in the game to determine their rank e.g., a player who gained 10 points gets an A rank. I would like to have my menus have a constant theme like how FireBoy and Watergirl constantly have vines and moss in their forest themed game. Navigating the menus will be simple and clear. The game over screen will give the options to return to the menu or quit the game. The product screen is a feature that shows the product of a reaction in my game (given that a reaction has taken place).

<u>An extra meeting with my key stakeholders:</u>

This is an email that I have sent to my key stakeholders (Hamid Chowdhury and Azim Khanazada):
"Hello,
I've been looking at features that can help me create my game, including the one you recommended to me Azim, and I've been wondering if you could add your input to it. When the game starts it'll display a menu with the title of the game and three options: play, access options or quit. When you choose to play, your cube(character) will be of a certain element and you will be in the play-state. Once you are in, you can move across platforms using the 'W','A','S','D' keys or the directional arrow keys can avoid cubes of different elements(enemies) with your movement. There is a 2-minute timer which shows you how long the game will be, and you can collect 'tubes' which serve as a point system which increases your score. The higher your score, the higher the probability of a rank(A,B,C,D etc.).If you fall off a platform the game-over screen will appear and you will see your points and rank, you'll be able to enter your name which will be displayed on the leader board if it's in the top 5.If you lose by reacting to another cube, a screen will show up showing you the product of the reaction between the two elements. If you manage to survive the 2 minutes, the game will compare your score to the other scores and determine your place in the leader board. The game will be designed to look nice while being easy to use. As far as system requirements go, an average computer is recommended to be used as well as a capable operating system."

Sincerely,

Ehsan"

<u>The responses I have received form my stakeholders:</u>

Hamid:

"This is the first time I've heard of platformer related to a field of science. It sounds like a good idea. Using points to contribute to your score for a leader board sounds like a good way to entice other students like me to try the game. The idea for the menu seems to be going well, I like the sound of a straight-forward menu. My computer is fairly capable so I'm certain that is should be able to run this game."

Azim: "You should add the leader board feature to the menu so it's accessible from the get-go. The simple options for the menu seem like a good idea but I menus look nice with pictures or an animation playing instead of a bland background. The alphabetical rank idea sounds nice because it is something that players can understand straight away. In terms of requirements, I have a gaming PC, so I won't have any troubles with running this application."

Features of the proposed solution

After examining both the previous game's features, I have an idea of what I should and should not do in my own solution, which may result in my stakeholders taking a liking to my game.

Menu

A feature that is common in both platformers is that they have a friendly user-interface. The menu will be easily accessible to students as it will consist of 2-4 buttons on each menu. The themes of the menu will be designed on photoshop. Menus should be effortless in navigating through as they are the first thing to show up on the user's screen and branches into the other features of the game.

Scoring: The score will be based on the number of tubes collected throughout the run of the game and will contribute towards the rank they get at the end.

Score Counter: This will be displayed on one of the corners of the screen during gameplay and allows the player to see their current score while playing.

Enemy elemental cubes: The other cubes are of different elements and act as obstacles for the player. If the player encounters a cube of a reactive element, the game is over.

Power-ups: A special power-up may appear where it grants you invincibility to other cubes for a limited amount of time**.**

Lives: The player will have one life as default and may be able to get one extra life during the game if they acquire a certain number of tubes.

Sounds: When the player clicks on an option or reacts with cubes in game, a sound will be made. Music may be playing in the background.

Pause: It is one of the game states which lets the user pause the game.

Product Screen: If the player's cube comes into contact with another cube of a different element, they will react, and the game is over. A screen will appear showing the product of the reaction and may provide a fact about the product, this enables the user to learn while playing.

Game-Over screen: This will be displayed with the message "game over" and the player will be asked to a name to go alongside the score and rank that was achieved.

Options (in menu): A game state that allows the user to toggle with settings such as turning the music off.

Re-using tubes: Re-using sprites instead of constantly creating new sprites during gameplay, Reduces the processing power required to run the game.

Re-using tubes: Re-using sprites instead of constantly creating new sprites during gameplay, Reduces the processing power required to run the game.

Rank system: Depending on the score achieved, the user will be given a rank such as an A or B etc.

Visual effects: When a player reacts with a cube, a small explosion and smoke effect may appear.

Player animation: This will bring my player's sprite to life.

Platforms generated: Platforms will be endlessly generated until the timer runs out.

Timer: The timer gives the player an incentive to progress in the game and avoid enemies while collecting tubes for their score in 2 minutes.

Leader board: This will display the top 5 players with the highest scores.

Limitations

Moving enemies: Enemies that move on platforms adds a dynamic to the game but programming the enemy's behaviour is something that I may not be able to do, given the time limitation.

2-player mode: Having 2 players playing on the game locally will make the game enjoyable but may be confusing with more sprites on screen as well. Issues with screen space may occur as one player may be significantly ahead of the other.

An endless runner: An endless runner generates platforms and enemies endlessly until the player loses. A common issue in developing endless runners is that the assets used over time causes the game to use up more RAM which eventually leads to the game crashing.

Online play:

A sizable limitation is that my proposed solution will not have online multiplayer available. Setting up a server to take in many inputs, and outputting scores from many people may be overwhelming and it may not send data back to each client within an acceptable time.

Limited time per session:

Constantly using the assets in an endless side-scroller tends to take up memory which could result in the game crashing. To prevent this, I have added a timing aspect so that there is a 2-minute timer when you start playing. This will end the game and prevent it from crashing.

Requirements including hardware and software.

Software:

Users will need to have a desktop computer or laptop with either the Windows or Mac operating systems (OS) installed. These operating systems are the industry standard OS for Computers and Laptops. My game will be made on C# and Unity which supports making and playing games on these platforms. These operating systems will also allow the user to install the game on to their PC to run it and play.

Software Requirements	Reasonings
Windows 7 SP1+	This is the minimum Windows OS required to run and play Unity applications.
MacOS 10.12+	This is the minimum Mac OS required to run and play Unity applications.

Hardware:

After researching similar games and their required hardware I have gathered that the processor will likely need to be a minimum of 1.1 GHz to be able to run the game. This has enough base clock speed to run the program without fps drops and lag. 2GB ram will also be required as a minimum to store the game in main memory without needing virtual memory from the storage which is much slower. Furthermore at least 1GB of HDD or SSD storage will be needed for the user to install the game on their computer.

Hardware Requirements	Reasonings
Peripherals [Mouse, Keyboard, Monitor…]	These basic peripherals are required as the player will use the keyboard as an input device for the character's movement. The use is used to navigate through the menus. A monitor will be used as an output device for displaying the game.
CPU Minimum: [1.1 GHz +] Recommended [2 GHz +]	Base CPU clock speed to ensure that the quiz runs smoothly.
RAM Minimum: [1 GB+] Recommended [2 GB+]	2 GB of RAM is Recommended to store games in main memory.
HDD/SSD/USB	Required for actually storing the game.
Speakers:	Recommended to use as the player will be able to hear the game.

Success criteria

My projects success can be measured by each of these criteria's:

REQUIREMENT	JUSTIFICATION	REFERENCE	EVIDENCE REQUIRED TO BE COMPLETED
Main Menu.	The menu will need to function properly as it acts as a point where the game can branch out to its other features such as "play", "options", "quit" etc. The player will be able to select what they want to do here.	FireBoy and WaterGirl.	Screen shot of the main menu (also a video).
Pause menu.	Pausing allows the player to take a quick break from the game or leave the game entirely.	FireBoy and WaterGirl.	Screen shot of the pause menu (also a video).
Game-over screen.	This screen should display when the player has fallen. It should give the players the option to restart or leave.	Standard across all games.	Screen shot of the game-over menu (also a video).
Score-system.	Implementing a functional score system will give the player an incentive to carry on and play the game. Collecting tubes (test tubes in chemistry) will give the player and number of points which can be added to their score. The game also displays the high score.	New Super Mario Bros.	Screen shot of the Score-system (also a video).

Platforms being re-used by a function.	Having platforms being activated and de-activated in the game, reduces CPU usage. This makes the game more stable when running on non-gaming PCs.	Original Idea.	Screen shot of the platforms being re-used by the game (also a video).
Speed-up.	The game speeds up the longer the player is running. This adds a difficulty curve to the game.	New Super Mario Bros.	Screen shot of the player speeding up, the longer they play (also a video).
Random Platform Spaces.	Makes the gameplay feel less uniform and more engaging for the player.	Original Idea.	Screen shot of the platforms with a varied distance between each other (also a video).
Random Platform sizes.	Makes the gameplay feel less uniform and more engaging for the player.	New Super Mario Bros.	Screen shot of the random platform sizes (also a video).
Platforms of varied height.	Makes the gameplay feel less uniform and more engaging for the player and makes more use of the jump mechanic.	New Super Mario Bros.	Screen shot of the random platform heights (also a video).
Restarting.	Allows the player to restart gameplay if they wish.	New Super Mario Bros.	Screen shot of the game restarting (also a video).
Tubes (collectible points).	Gives extra points to the player when they are picked up.	New Super Mario Bros. FireBoy and WaterGirl.	Screen shot of the tube giving points (also a video).
Tubes on random platforms.	Keeps the payer engaged as tubes will not be spawning on every platform.	New Super Mario Bros.	Screen shot of tubes spawned on random platforms (also a video).
Score Counter.	Allows the player to see their score in-game.	New Super Mario Bros.	Screen shot of the score counter (also a video).
Power-ups.	This will help my solution have a game-like atmosphere and let the player have more fun.	New Super Mario Bros.	Screen shot of the Score-system (also a video).
Visual effects.	Visual effects like animation keeps the game engaging to my stakeholders but	New Super Mario Bros.	Screen shot of the visual effects/animation (also a video).

	should not take up a lot of screen space so that it's not distracting.		
Leader board.	The leader board will display 5 players who have the highest scores. The Scores will be stored in a file which is used in the game to collect data for the leader board. This makes the game seem arcade-esque.	Standard across all games.	Screen shot of the leader board (also a video).
Sound.	This will bring life to the game. Sound effects should be present when the player jumps or lands on a platform, reacts with a cube or clicks an option in a menu. Background music will be added into the background of the menu.	New Super Mario Bros. FireBoy and WaterGirl.	A video showing the sound-effects feature.
Options.	If the user does not wish to hear the music or the sound effects, they can toggle it off in settings.	New Super Mario Bros.	Screen shot of the options menu (also a video).
Controls.	Controls will allow the player to be able to move across platforms using 'W','A','S','D' or the directional arrow keys.	New Super Mario Bros. FireBoy and WaterGirl.	Screen shot of the controls responding (also a video).
Enemies.	The elementary cubes that the player will face acts as enemies as they provide the stakeholder's a challenge as well as a threat of losing. The elements of the cubes act as the gateway to the educational aspects of the game.	New Super Mario Bros.	Screen shot of the enemies on platforms (also a video).

Game messages.	Messages such as “Ready and “go” may be added at the start so that it lets the user prepare to play the game.	FireBoy and WaterGirl.	Screen shot of the game messages (also a video).
Product screen displays the product of a reaction.	The educational aspect of the game is within the product screen. The game giving feedback on what the kind of element the user has reacted to. This lets the player’s knowledge expand.	Original idea.	Screen shots of the product screen displays the product of a reaction

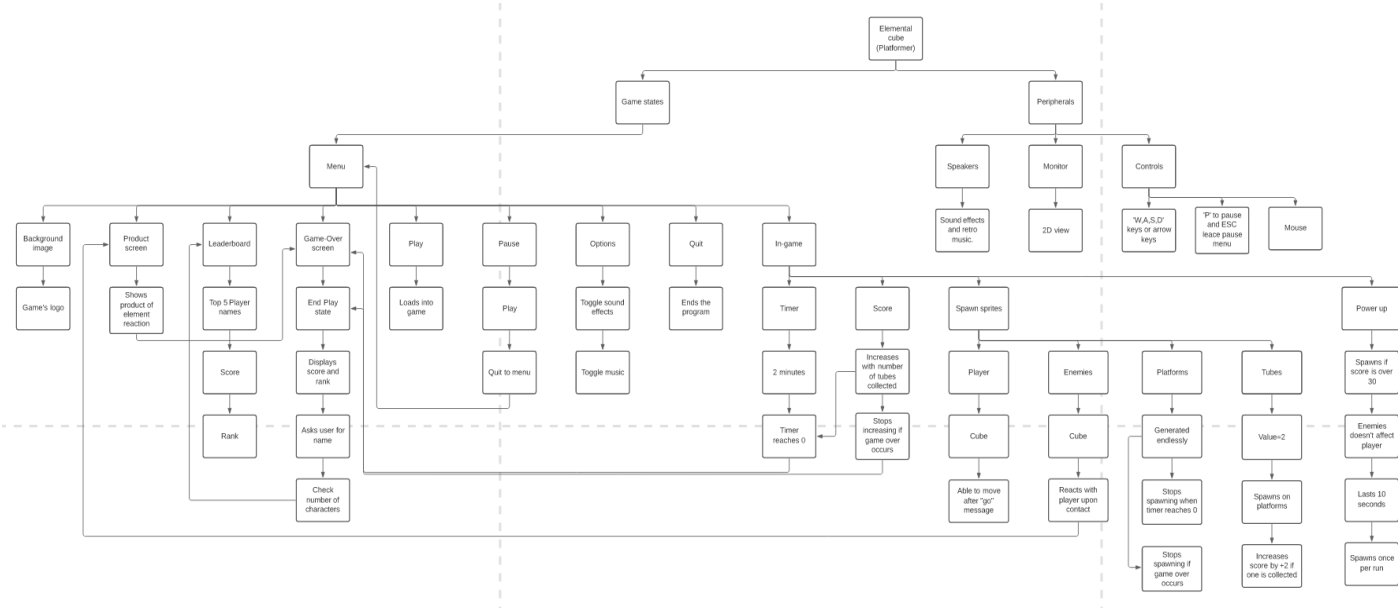
My game needs to have a name to be memorable and have its own identity. After considering the features of my game, I have decided to name the game “Elemental Cube”.

Design section

Total marks 15

Decomposition of the problem

First rendition of the hierarchy model:



Why am I using a hierarchy model to break down my problem?

Decomposing my problem from the top-down makes it easier for me to see every function I need to implement into my program. This simplifies my approach to the solution as I can see my modules with their respective sub-modules. This approach enables me to include computational thinking skills such as “thinking ahead” to identify the inputs and outputs of my solution, in advance. This will let me program things which I will need in the future early on and it will let me implement them later very easily. “Thinking logically” can also be applied here to determine what are the most important branching points and decisions within my game.

Explanation of each section

Menu

This is one of the major components of the game as it can branch off to all the other game states.

Background picture.

The picture in the background of the menu makes the game seem more fun and eye-grabbing.

Play

One of the game-states which will load the sprites and start the game after the “Go message”.

Leader board

One of the game-states that will load up the leader board and will display the top 5 players with the highest scores and rank.

Options

This gives the player the option to turn off the sound effects or music.

Quit

Allows the player to exit the game entirely.

Scoring

A score counter:

Having a score counter, available to see during gameplay, allows the player to easily view their current score while playing.

Leader board

High score name validation

This asks the user for their name which will be shown on the leader boards. If the input is less than 2 characters or greater than 10 characters, it will count as invalid.

Rank

If the player's score enters a certain boundary (e.g., 10<x<)

New score being inputted into the board.

This runs the function which will update the leader board in the game.

Display the score on the display.

Displays the updated leader board on the players display.

Pause menu.

Pressing 'p' or clicking a pause button activates the game-state which pauses the game and will list the two options available.

Un-pause option

When the "ESC" key is pressed, it will quit the pause menu and the game continues.

Quit option.

A button allows the player to return to the main menu.

Game re-using resources (tubes and platforms).

Allowing platforms and tubes to be re-used reduces the requirements needed by the stakeholder's computers. I see this as a necessary component to add to the game as tubes and platforms will be appearing near-constantly throughout gameplay.

Game over screen

Display the game-over screen.

If a player loses from a fall or reaction with another element, the game-over screen on the display.

End-play state
This will lead to the end-play state where the score and rank will be displayed, and the user will be asked to input a name between 2-10 characters.

Product screen

If the player has ended up in a reaction with another cube, the game will end, and the product screen will be displayed to the player. The product screen shows the product of the element (e.g., if the player's cube is aluminium and reacts with a water cube, the product screen will show the equation and the product: Magnesium Oxide (MgO)).

Playing the game

Sprites
Once the sprites have spawned (Player's cube, enemy cubes, tubes, and platforms), then the game can start.

Visual effects

There will be 4 main animations: The player moving left and right, the player jumping, the player landing and the player reacting with an enemy cube. Other animations will include idle for enemies, collecting tubes (for their score) and for when the power-up is activated. Using animations will add fluidity in the game instead of using static sprites.

SFX

There will be sound effects for when a player jumps or lands, collecting tubes and colliding with other cubes. There will be a sound effect for when a player clicks on an option in a menu. This act as auditory feedback for the player.

Scoring

When a player collects a tube, their score increases by +2.

Timer

The timer will begin once the game has started and will last for 2 minutes.

Movement

The player cube's movement will be either WASD or the directional arrow keys.

Spawning and de-spawning

Spawning and de-spawning sprites such as platforms and tubes allow me to decrease the amount of RAM and processing power needed for the game.

Power up.

The player achieving a score of >= 30 grants them the ability to be able to go through enemies for 10 seconds

Game Messages

Just before the game starts, the message “Ready...” will appear, shortly followed by the message “Go!”.

Peripherals

View

Two-dimensional

The game will be in a two-dimensional view as that is the perspective side-scrolling platformers have kept.

Windowed

Algorithms

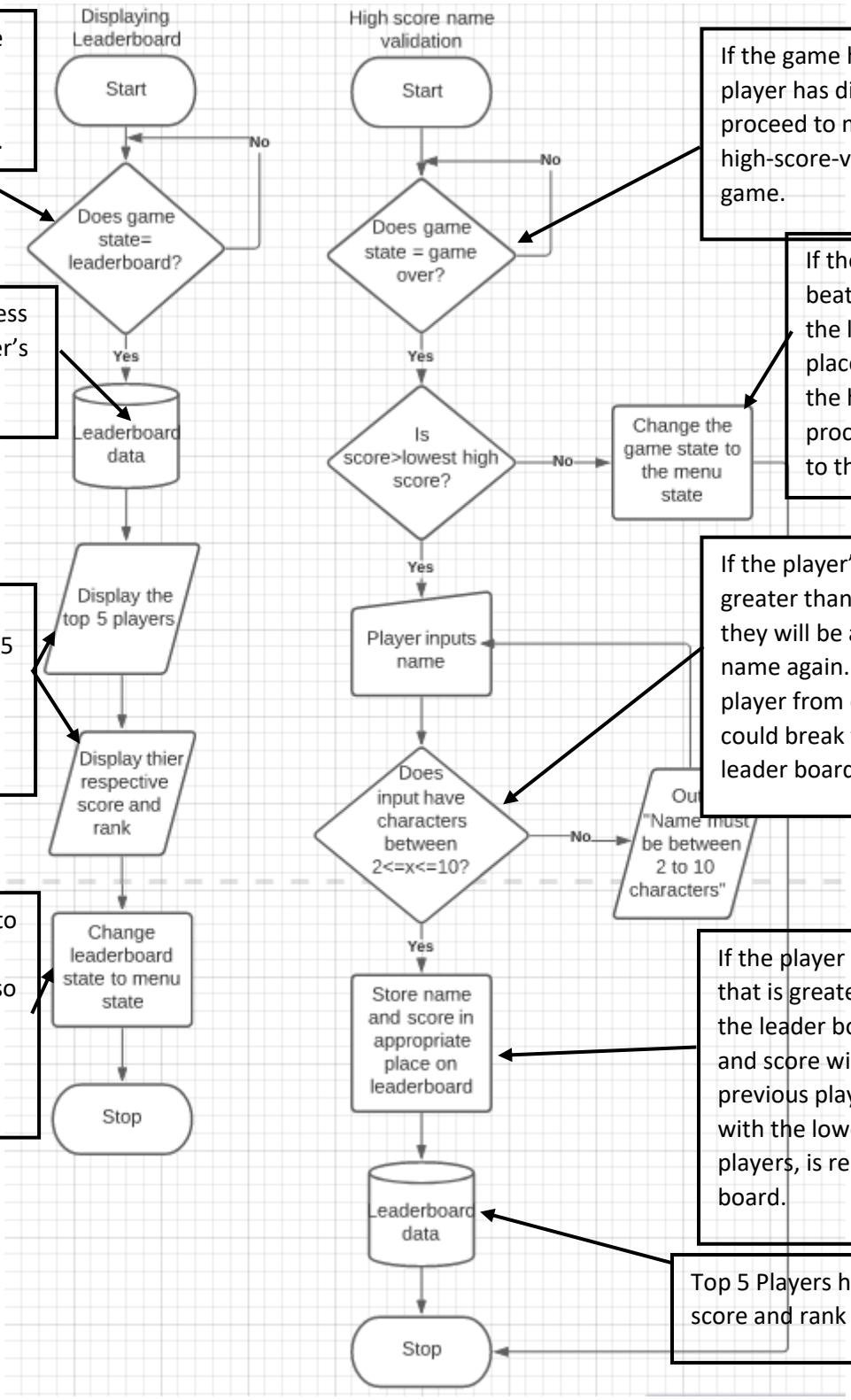
Leader Board

If the current game state is not the leader board state, the game will not display the leader board.

The game will access the previous player's data from the leaderboard.

The leader board will display the top 5 players in order according to their score and rank.

The game transitions to the main menu after displaying the scores so that the player has other options to explore.



If the game has ended (e.g., the player has died), the game will proceed to move forward to the high-score-validation section of the game.

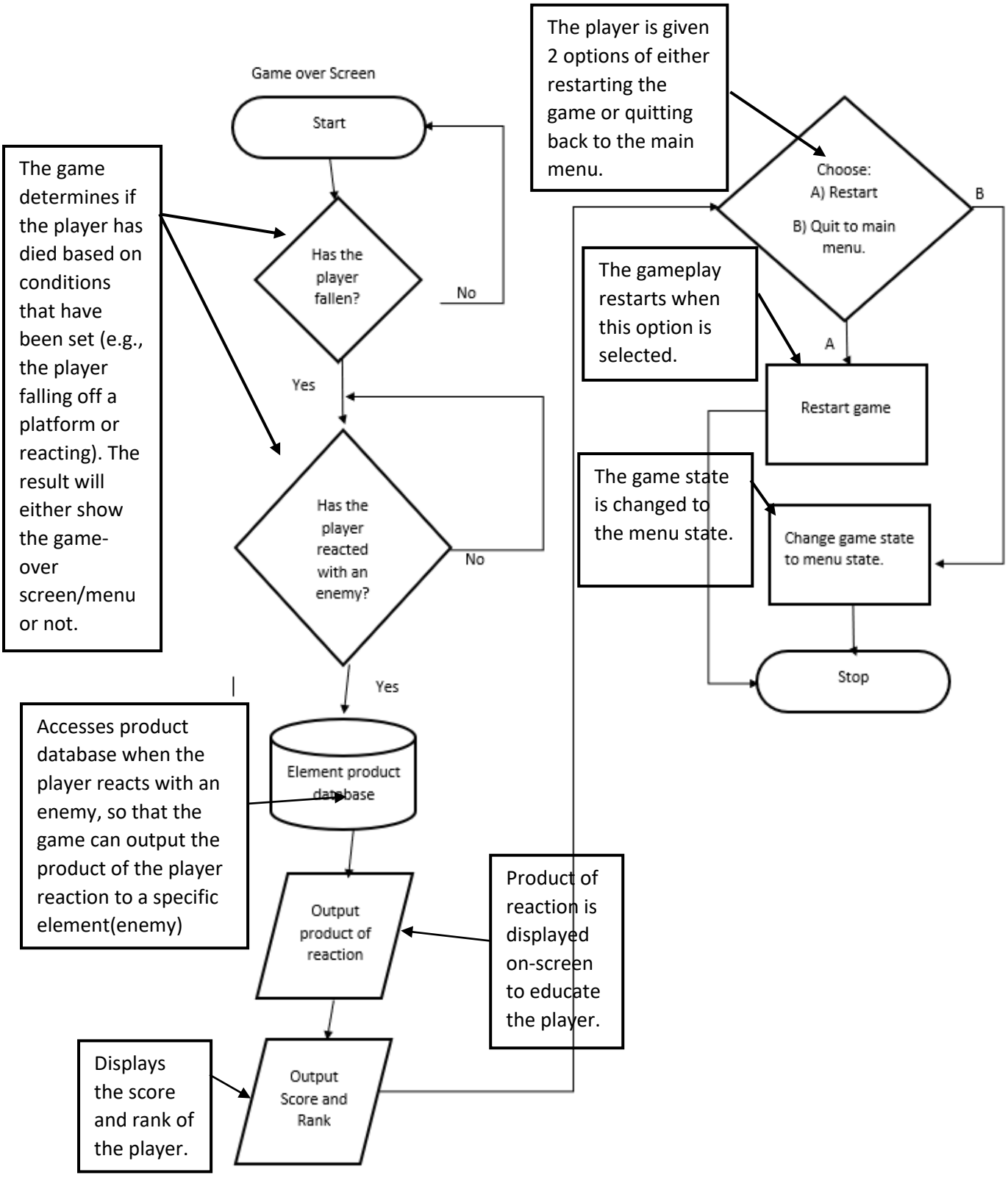
If the player has not beaten the lowest score on the leader board (5th place), the game will skip the high-score validation process and will transition to the menu.

If the player's name is lesser or greater than the character limit, they will be asked to enter the name again. This is to prevent the player from entering names that could break formatting of the leader board.

If the player has achieved a score that is greater than of the scores on the leader board, the player's name and score will be saved in the previous player's place. The player with the lowest score out of the 6 players, is removed from the leader board.

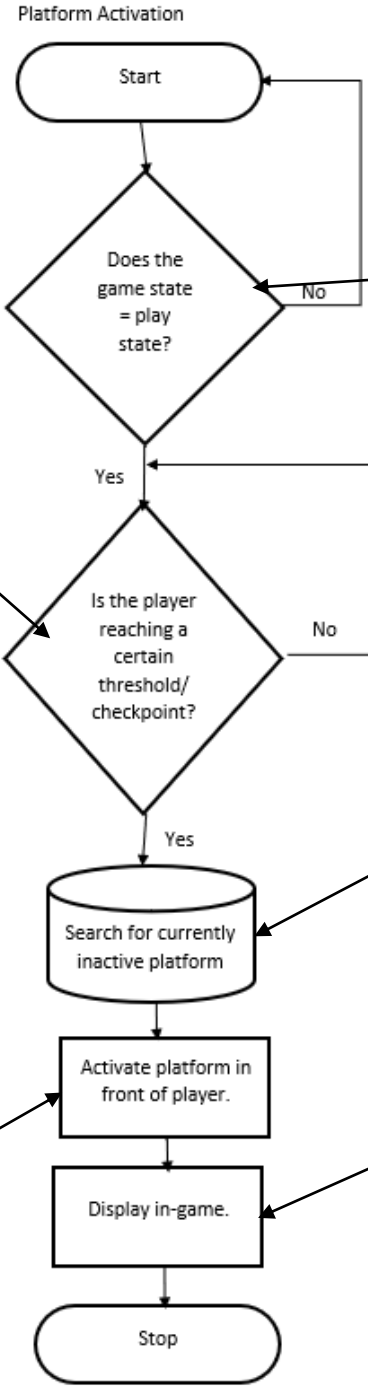
Top 5 Players have their name, score and rank stored.

Game Over screen



Reusing Platforms.

Platform activation:



Determines if the game is in the play-state so that the game can activate platforms. Without checking this, the game may activate platforms in the menus which may affect performance.

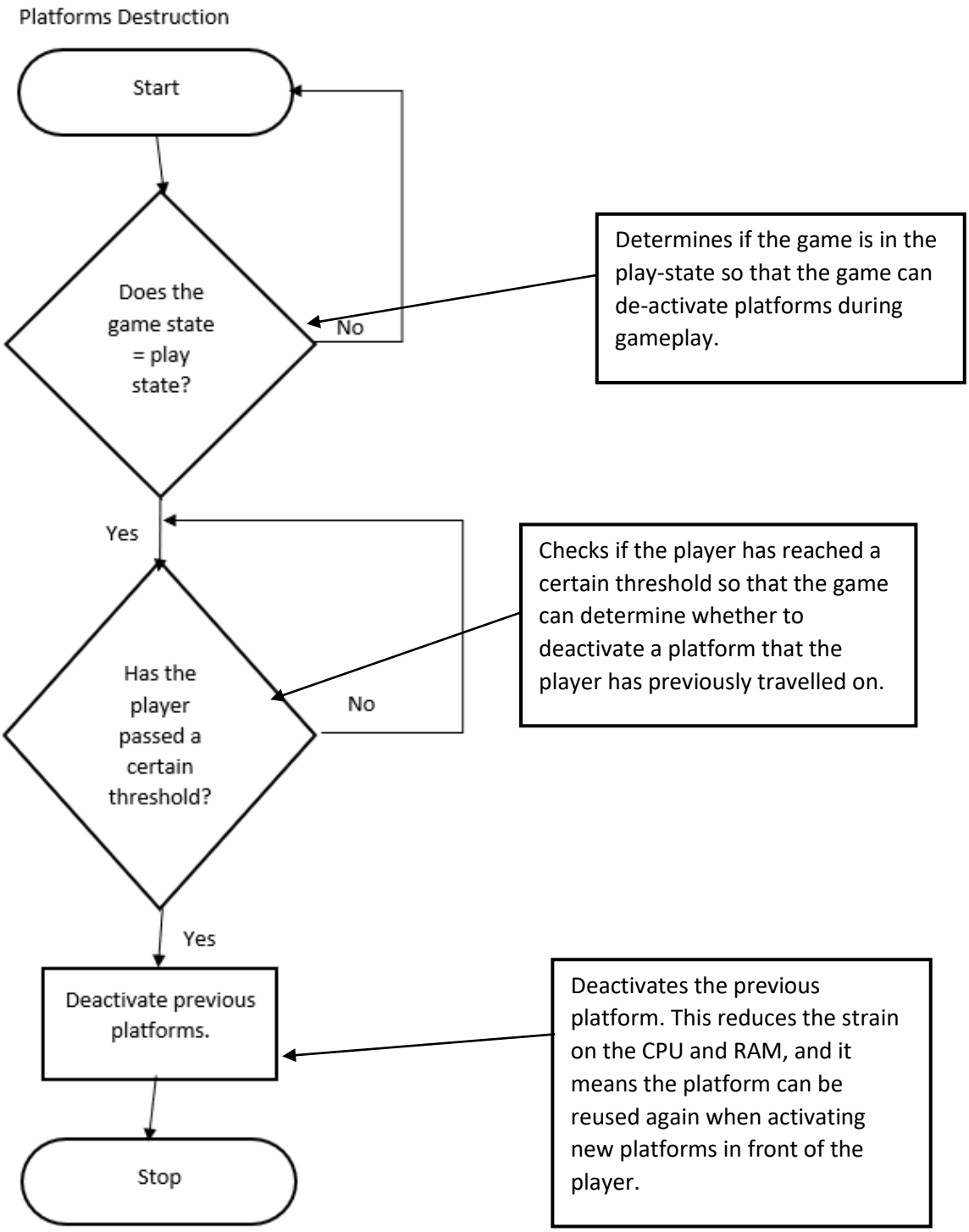
Checks if the player has reached a certain threshold so that the game can determine whether to activate a platform in front of the player.

Access in-game database for a platform that is inactive so that It can reuse it in-game. Re-using platforms instead of spawning new ones every time reduces processing power required by the CPU to run the game.

Platform is activated in front of the player so that the player can land on it. This will allow for endless gameplay to continue.

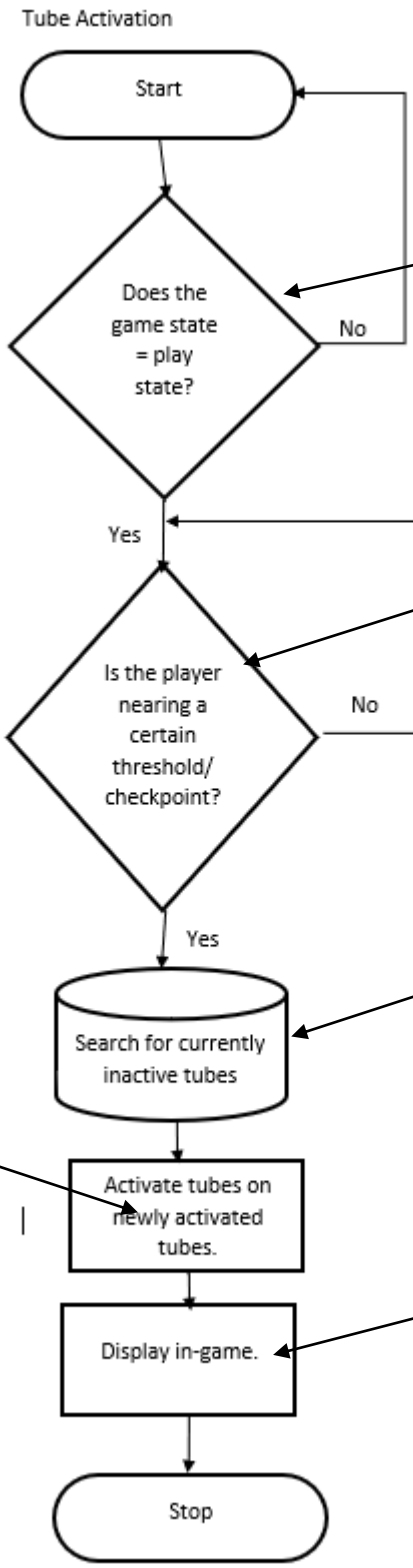
Platform is visible to the player so that they know where they can land.

Platform destruction:



Reusing Platforms.

Tube Activation:



Determines if the game is in the play-state so that the game can activate tubes. Without checking this, the game may activate tubes in the menus which may affect performance.

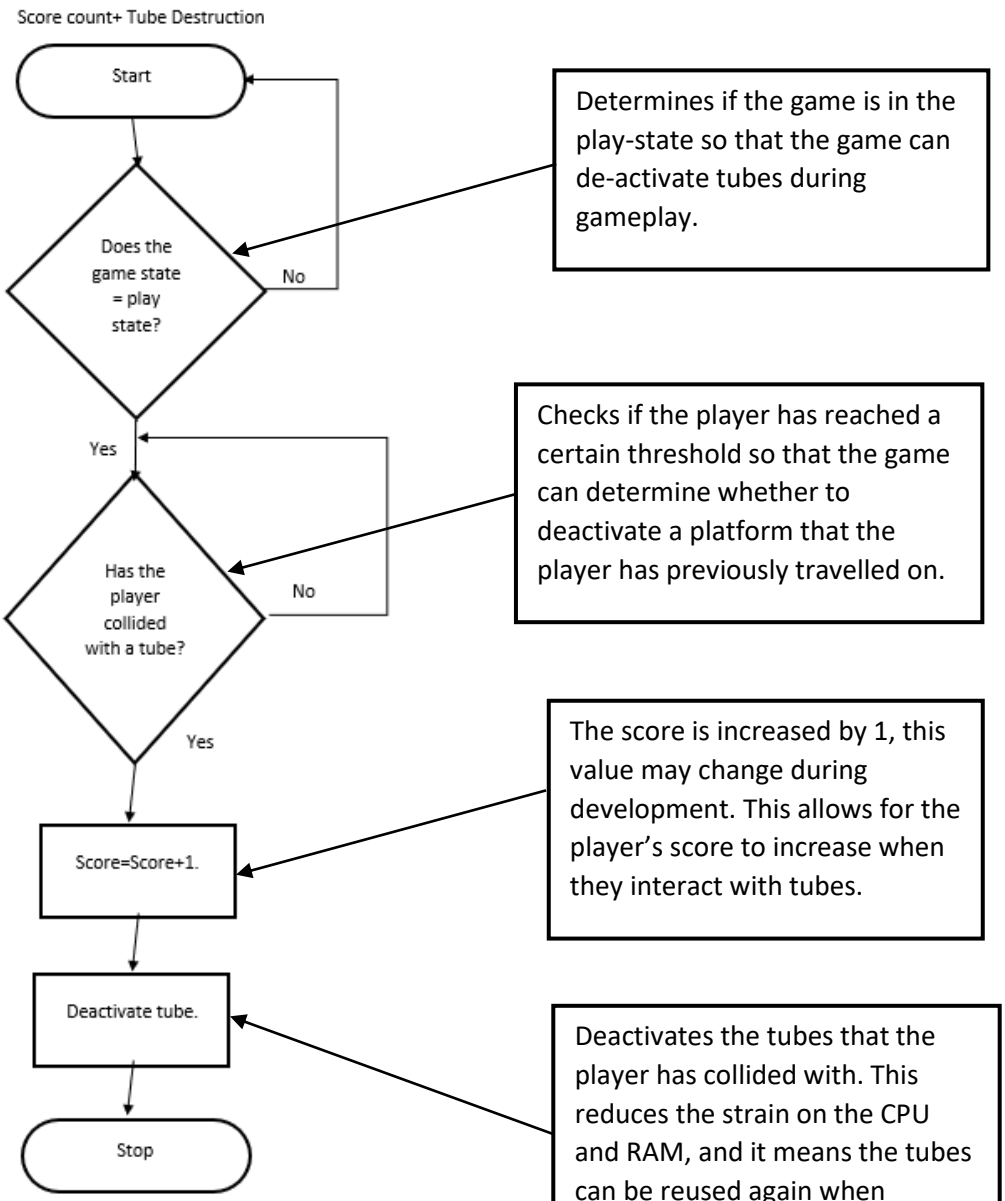
Checks if the player has reached a certain threshold so that the game can determine whether to activate tubes in front of the player.

Access in-game database for tubes that are inactive so that It can reuse it in-game. Re-using tubes instead of spawning new ones every time reduces processing power required by the CPU to run the game.

Platform is activated in front of the player so that the player can land on it. This will allow for endless gameplay to continue.

Tubes are visible to the player so that they know where to collect points.

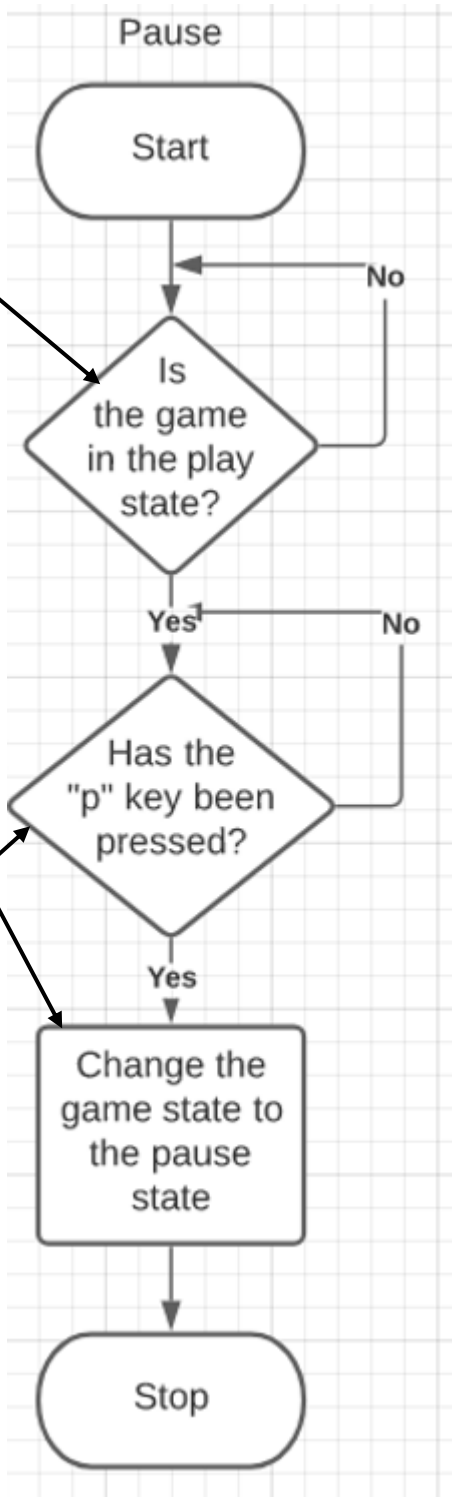
Score count+ Tube destruction:



Pause Menu.

Pause

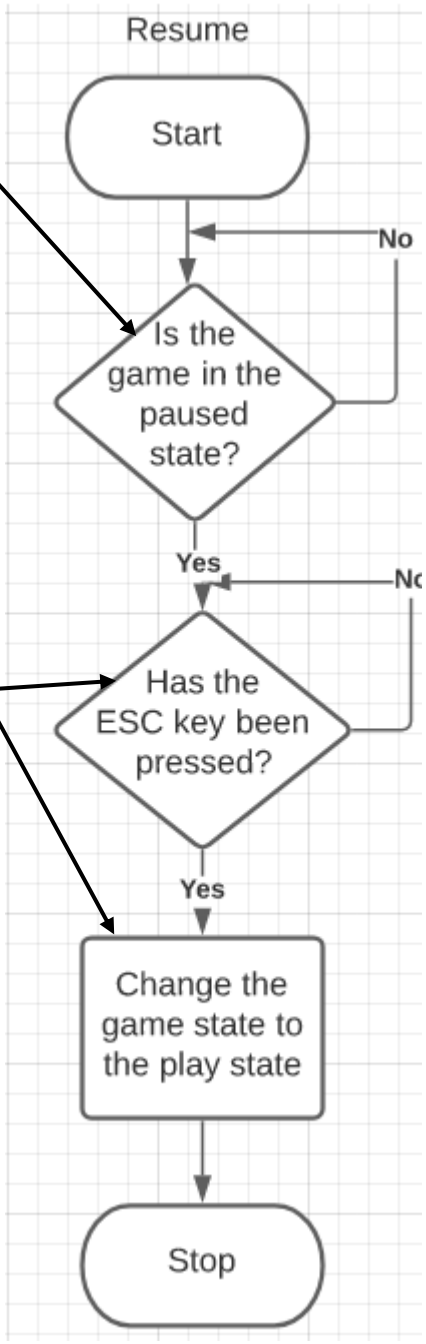
Determines if the player is in the play-state so that the game can determine whether to allow the pause option to be available.



If either the ‘p’ key or pause button has not been pressed or the game will continue as normal. If either of those buttons have been pressed, the game will change game-states.

Resume

Determines if the player is in the paused state so that the game can determine whether to allow the resume option to be available on



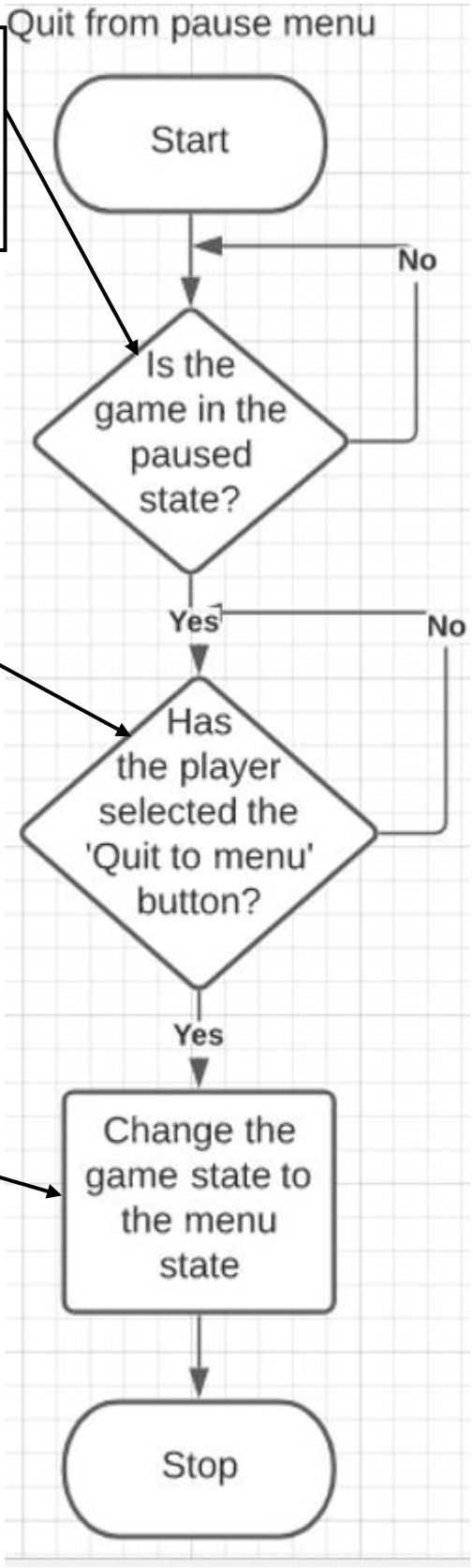
Depending on if the ESC key or resume button has been pressed, the game state will change.

Quit to main menu

Determines if the player is in the paused state so that the game can determine whether to allow the quit to main menu option to be available on the pause menu.

Depending on if the quit to the key or resume button has been pressed, the game state will change.

Game state changes from pause menu to main menu.



Menu

Checks if the game is in the menu state so that the game can determine whether to start the main menu or not.

Outputs the background of the main menu to show the player what game-state they are at.

Outputs the options of the menu are a drop-down format. Displays 'Play' 'Leader board' 'Options' 'Quit'.

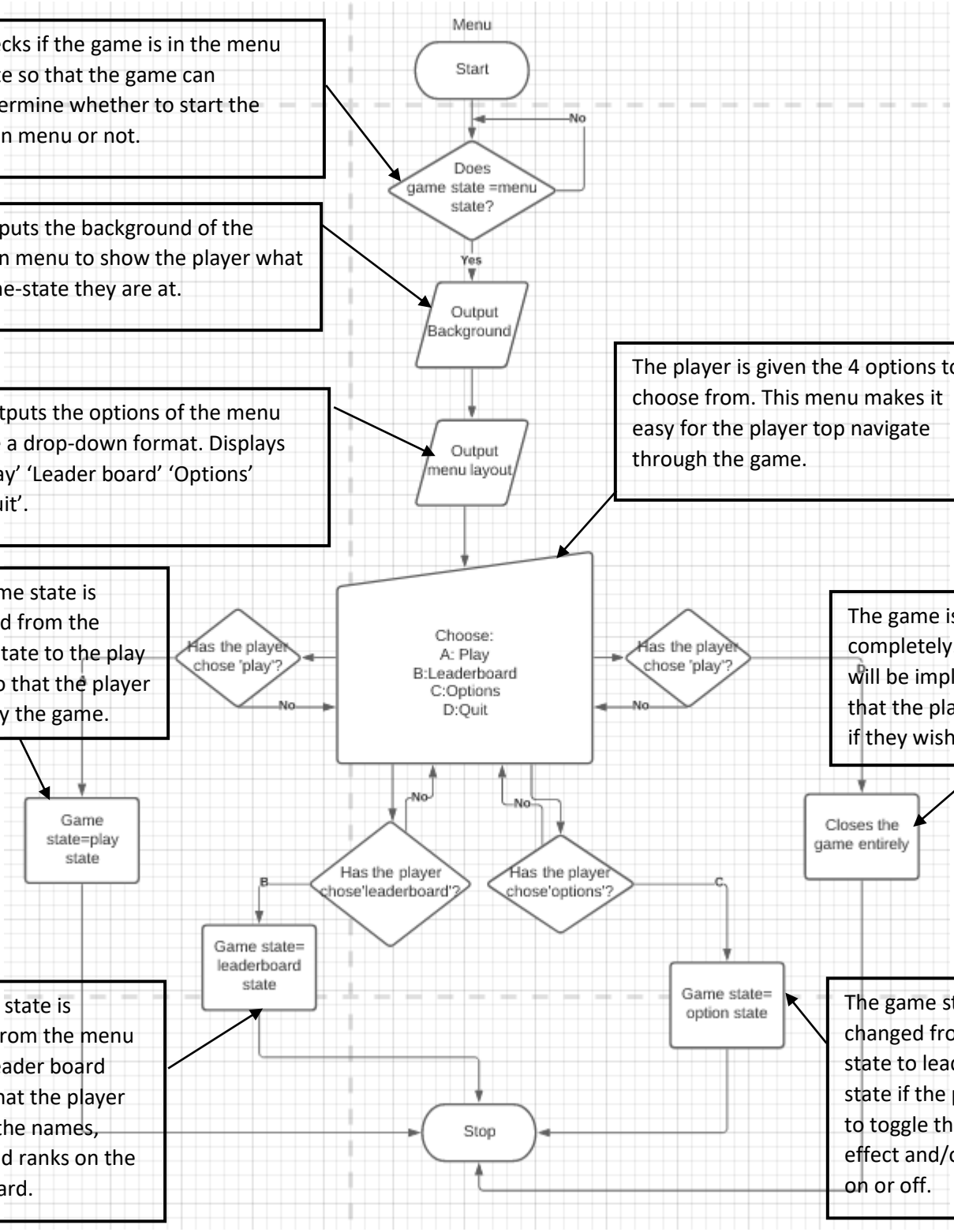
The player is given the 4 options to choose from. This menu makes it easy for the player top navigate through the game.

The game state is changed from the menu state to the play state so that the player can play the game.

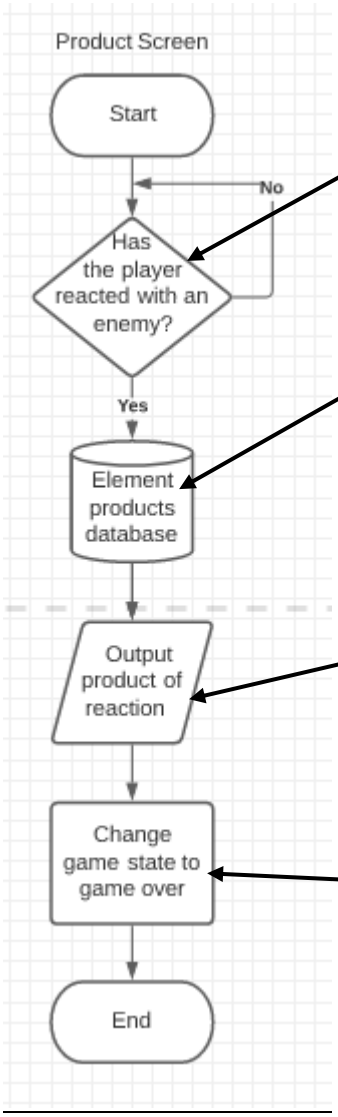
The game is closed completely. This option will be implemented so that the player can leave if they wish to.

The game state is changed from the menu state to leader board state so that the player can view the names, scores, and ranks on the leader board.

The game state is changed from the menu state to leader board state if the player wishes to toggle the sound effect and/or the music on or off.



Product Screen



The game checks if the player has collided with an enemy. If the player has not collided with an enemy, the gameplay continues.

If the player has reacted with an enemy, the game accesses the element product database and searches for the product associated with that enemy.

The product of the reaction is displayed on the product screen. This educates the player on what kind of element is created from a certain

Game state changes from product screen to the game over screen/menu.

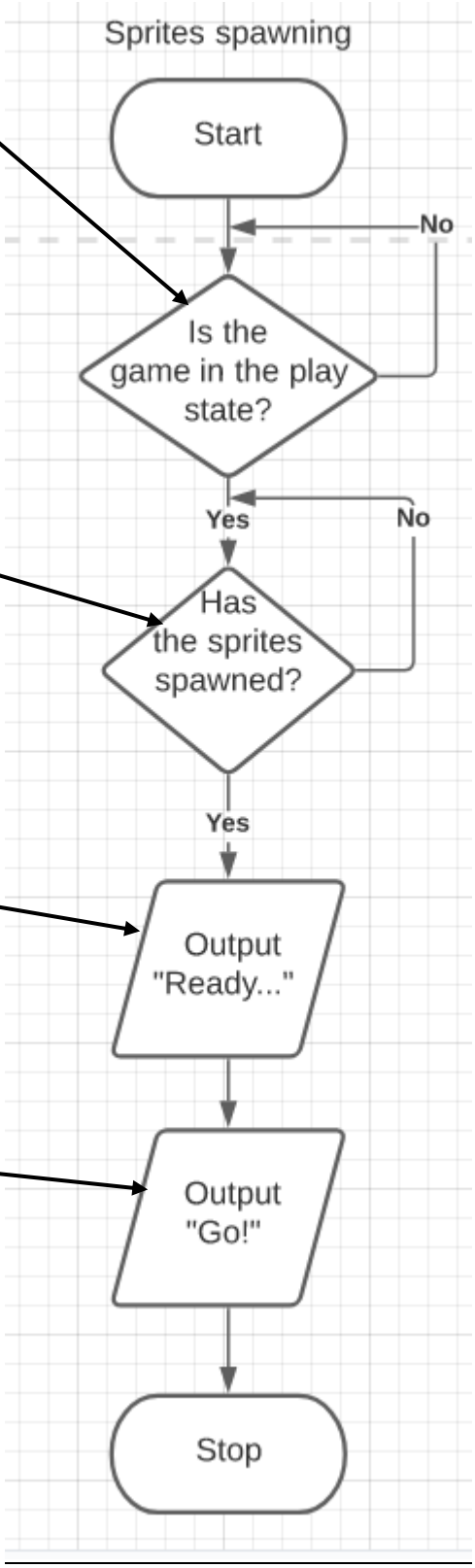
Sprites and game messages

Determines if the player is in the play-state so that the game can determine whether spawn the sprites for the game (player, tubes, platforms etc.).

Depending on if the sprites have spawned, the game state will output the game messages.

The message “Ready” is displayed on-screen to tell the player to get ready.

The message “GO!” is displayed on-screen which tells the player that they can move.



Timer

Checks if the player is in the play-state and that if the game messages have been displayed so that the game start the timer.

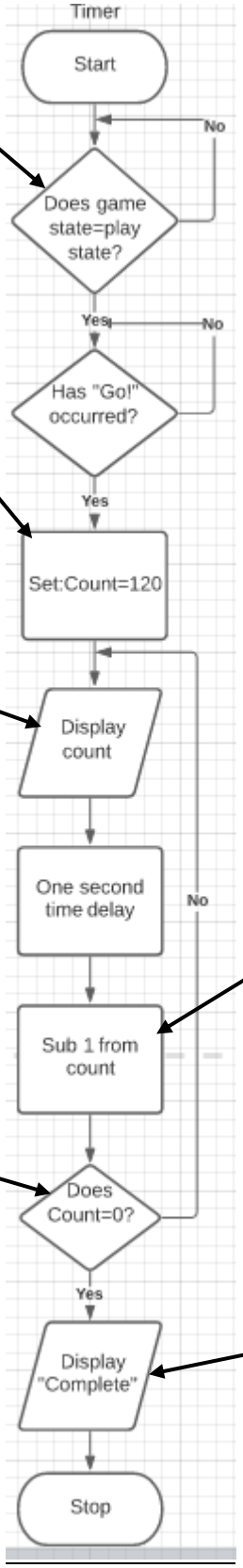
Timer is to 120 seconds (2 minutes) which will be the span of the game.

The timer is displayed so that the player can view the amount of time that they have left.

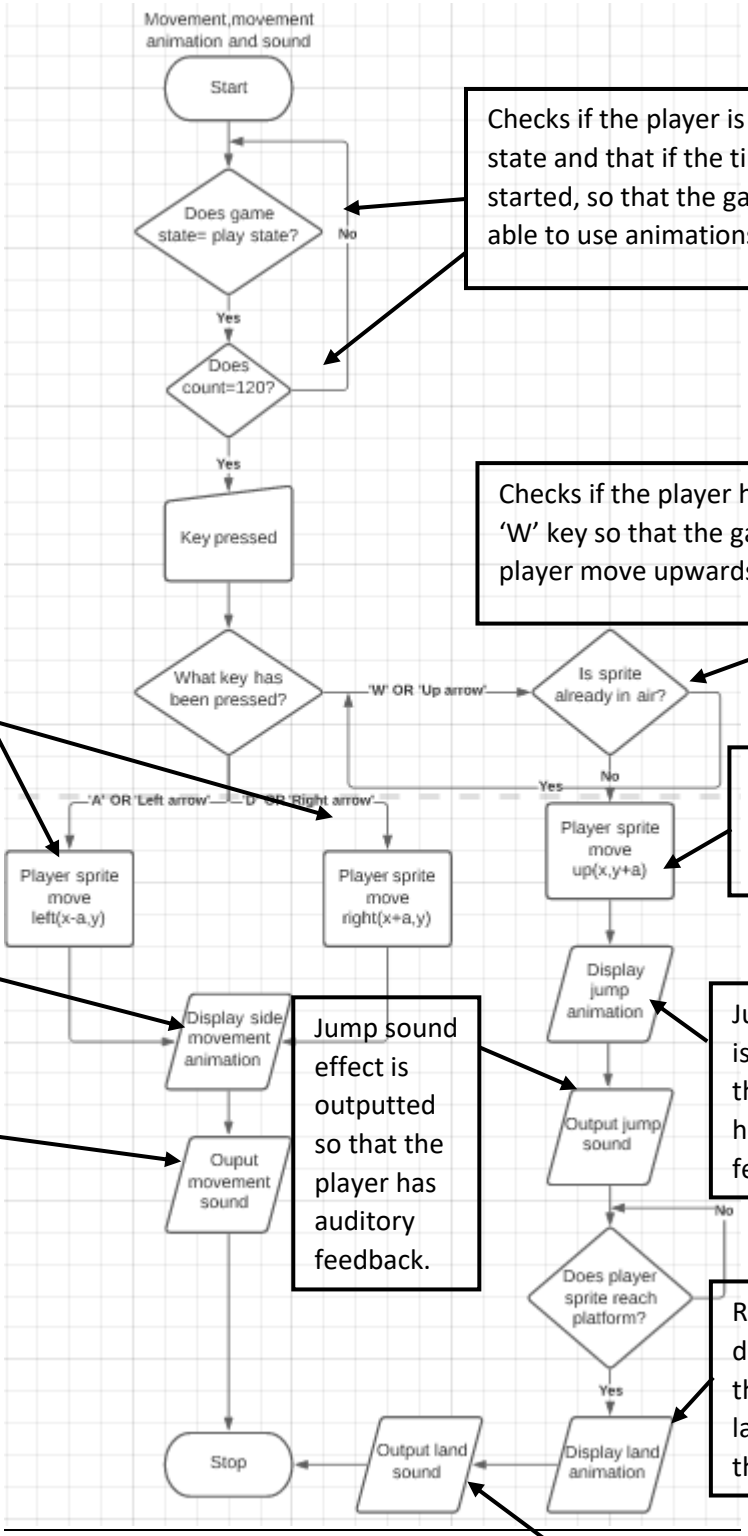
The value displayed on the timer will decrease by 1 after one second has elapsed.

If the value of the timer has reached 0, the game will proceed to display the message “complete” to the player.

The game proceeds to display “Complete” based on the result of previous decision.



Movement, movement animations and sound



Checks if the player is in the play-state and that if the timer has started, so that the game will be able to use animations.

Checks if the player has pressed the 'W' key so that the game can let the player move upwards.

Checks if the player has pressed the 'A' key or 'D' key so that the game can let the player move horizontally backwards or forwards.

Player moves up in the air.

Jump animation is displayed so that the player has visual feedback.

Jump sound effect is outputted so that the player has auditory feedback.

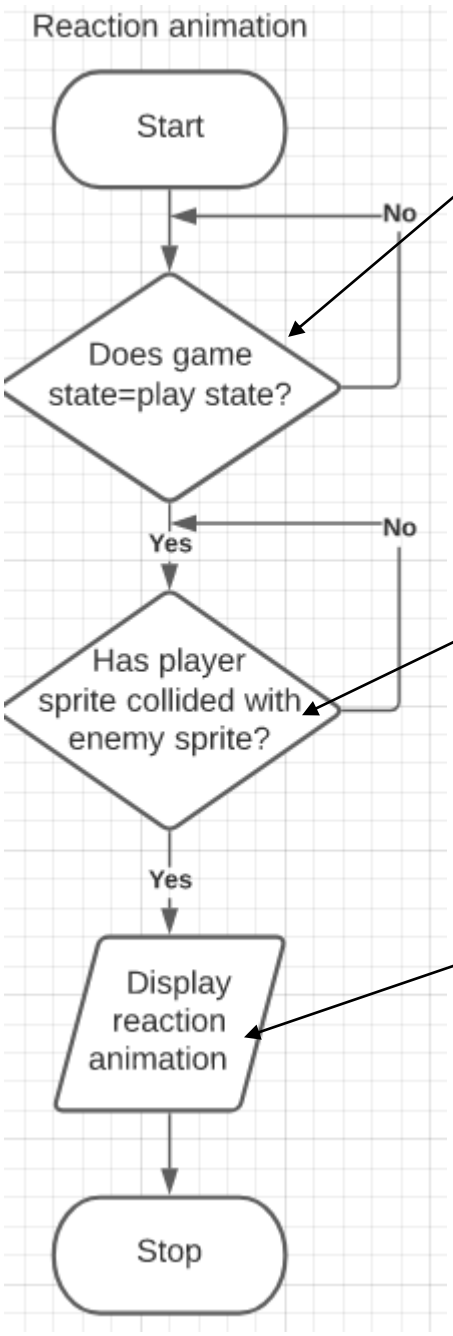
Jump animation is displayed so that the player has visual feedback.

Jump sound effect is outputted so that the player has auditory feedback.

Run animation is displayed when the player has landed back on the ground.

Landing sound effect is outputted so that the player has auditory feedback.

Reaction animation

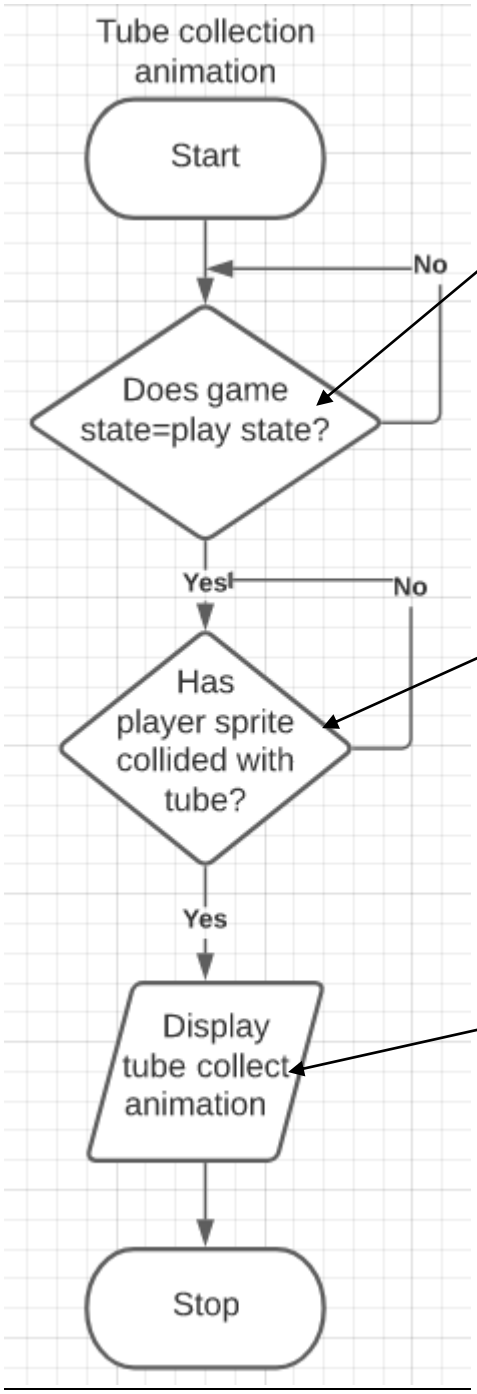


Checks if the player is in the play-state so that the game can allow for reaction animations to occur.

Checks if the player has collided with an enemy. Depending on the answer, the reaction animation will be displayed.

Reaction animation is displayed if the player has collided with an enemy.

Tube collection animation

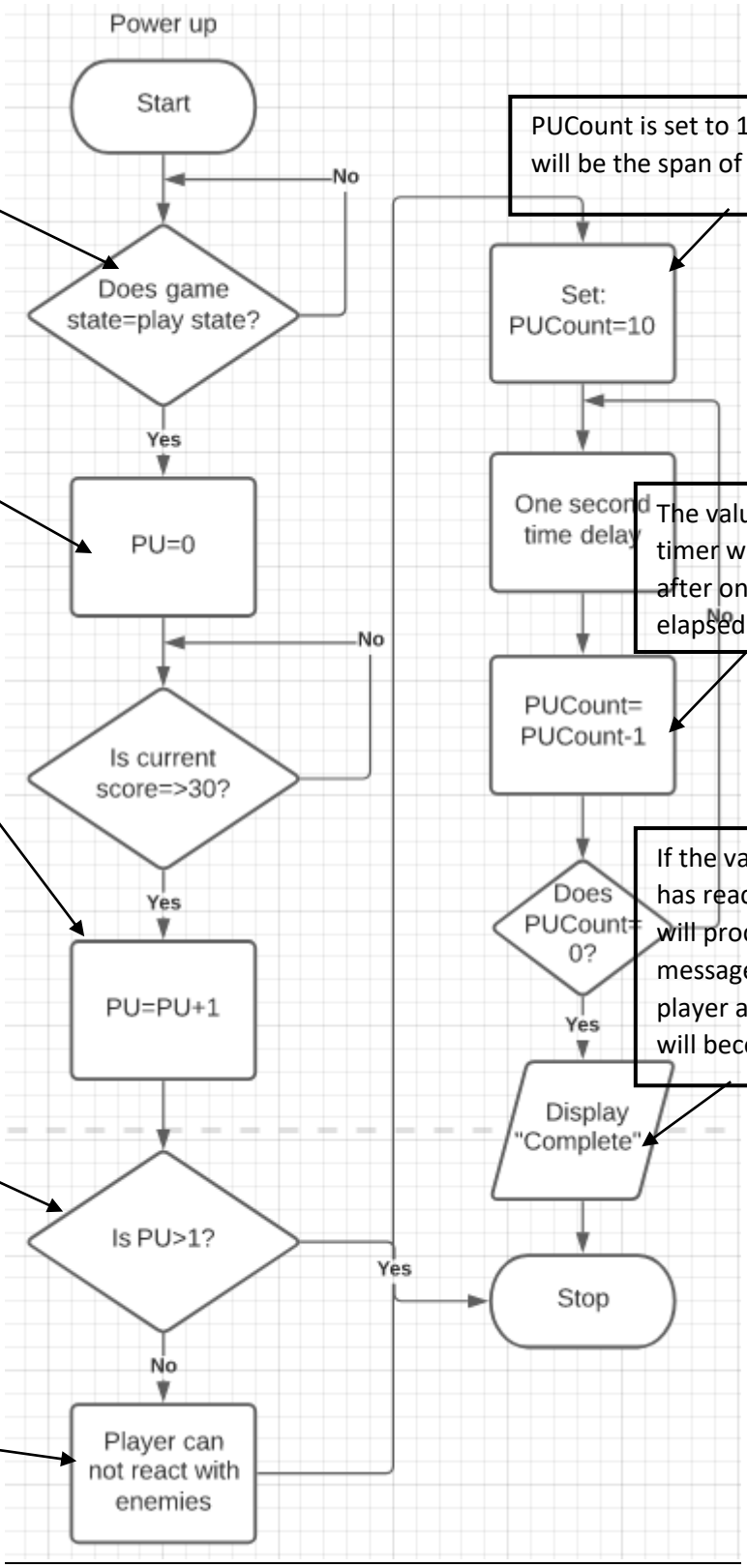


Checks if the player is in the play-state so that the game can allow for reaction animations to occur.

Checks if the player has collided with a tube. Depending on the answer, the tube collection animation will be displayed.

Reaction animation is displayed if the player has collided with an enemy.

Power up animation



Checks if the player is in the play-state so that the game can allow for power-up animations to be present.

PUCount is set to 10 seconds which will be the span of the power up.

The value of PowerUp is set to 0. This variable will be called later.

The value displayed on the timer will decrease by 1 after one second has elapsed.

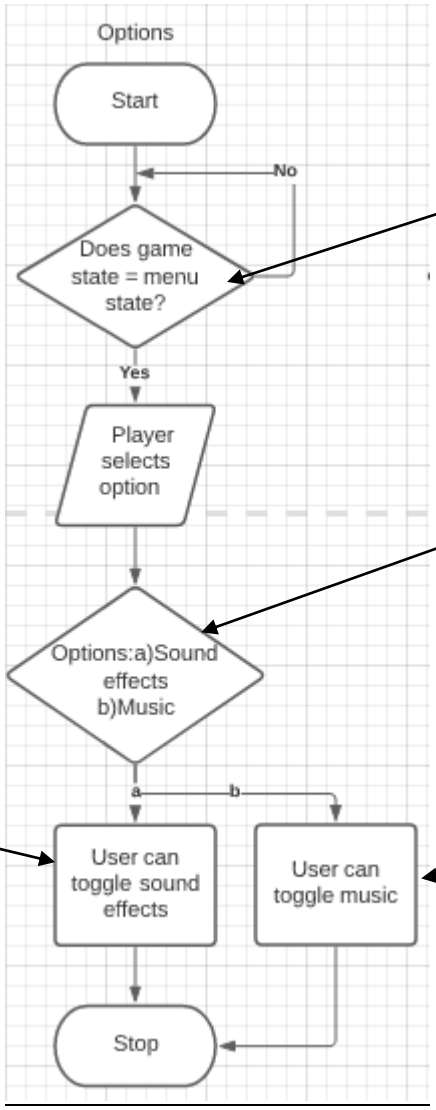
When the score is greater than 30, the value of PowerUp is increased by +1.

If the value of the PUCount has reached 0, the game will proceed to display the message “complete” to the player and the power up will become inactive.

If the value of PowerUp score is greater than 1, the power up will not be available for the player to use. This is to prevent power ups from stacking up and making the game too easy.

If the value of PowerUp score is not greater than 1, the power up will be available for the player to use.

Options from menu



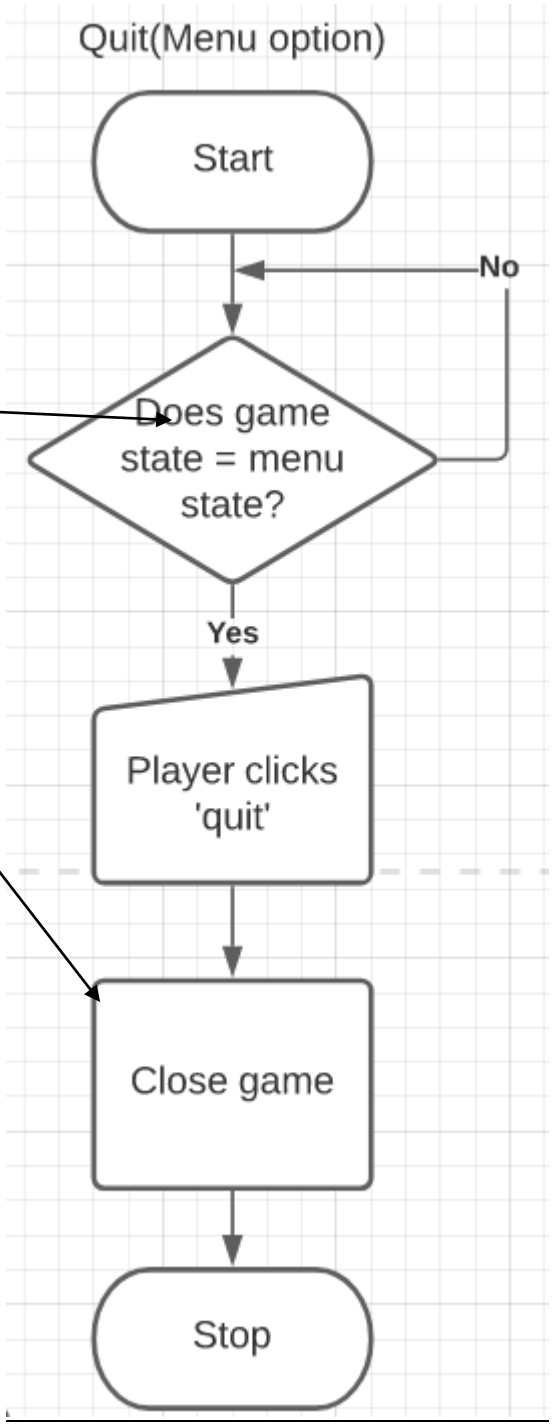
Checks if the player is in the menu-state so that the game can display the menu options to the player.

The options menu gives the player the options to toggle either the sound effects or the background music.

Sound effects can be toggled on or off, for the player's preferences.

Music can be toggled on or off, for the player's preferences.

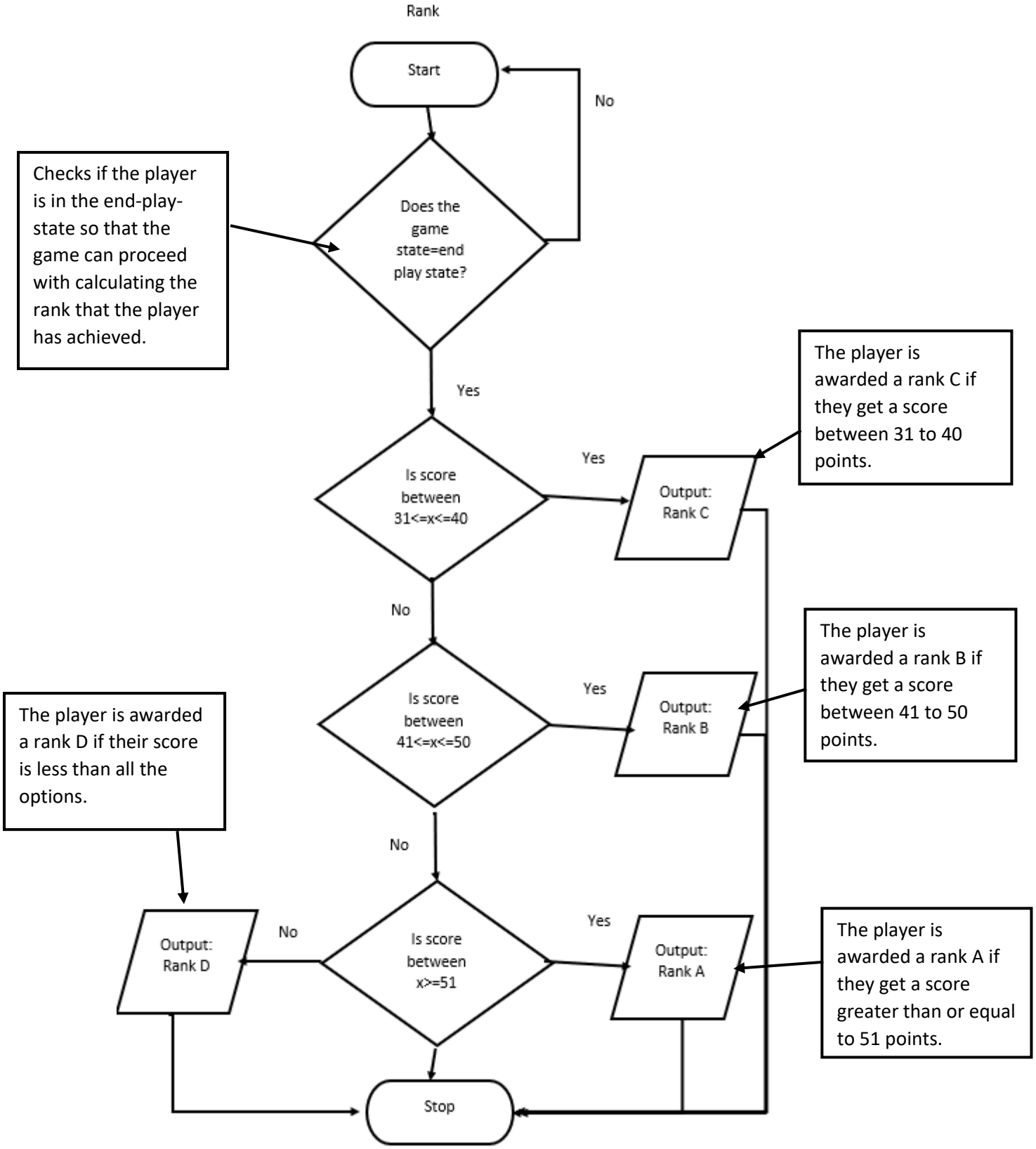
Quit (menu option)



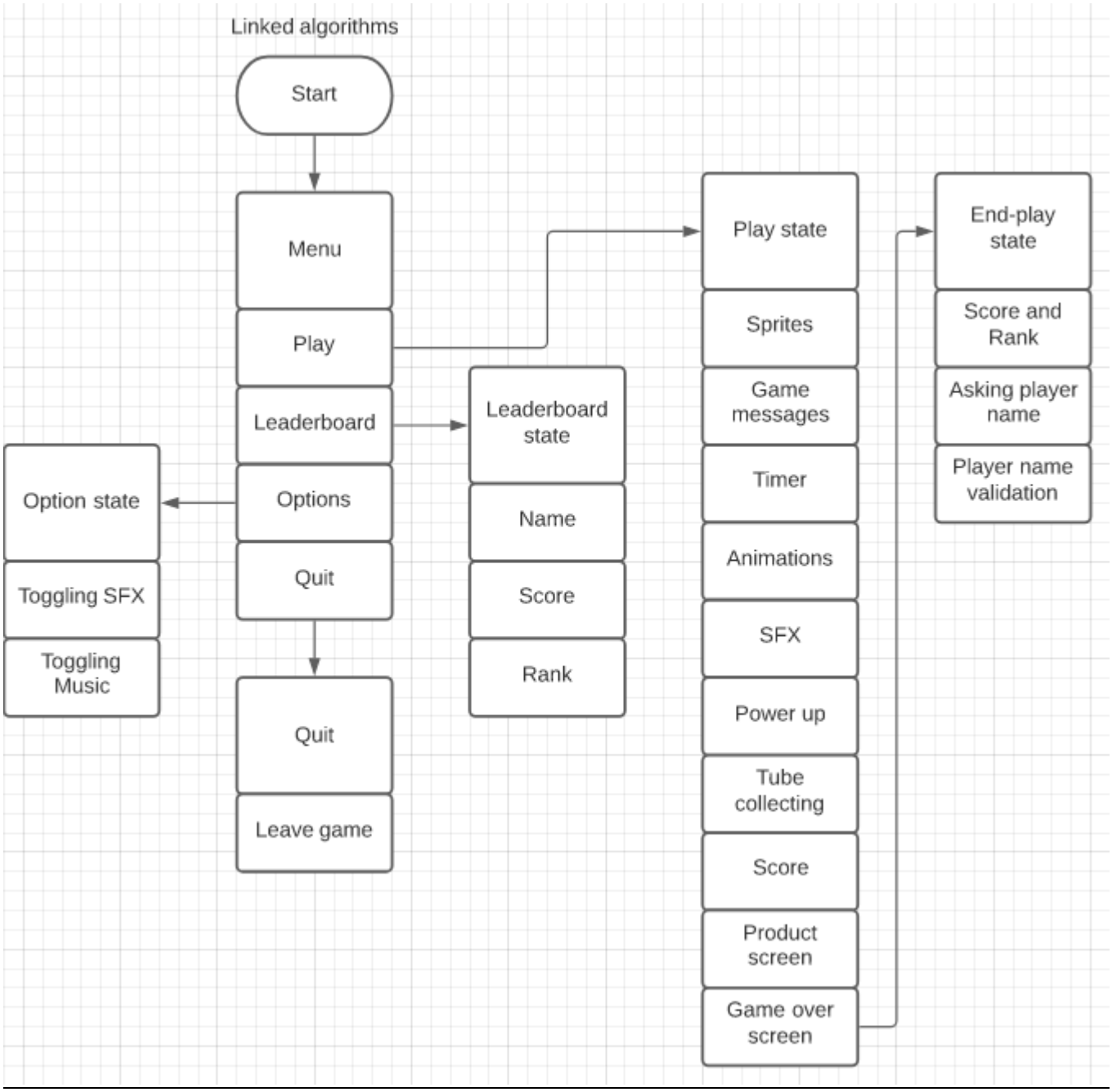
Checks if the player is in the menu-state so that the game can display the menu options to the player.

If the player clicks the 'quit' option in the menu, the game stops running and closes. This is a standard feature across all well-known games so that the player can quit.

Rank



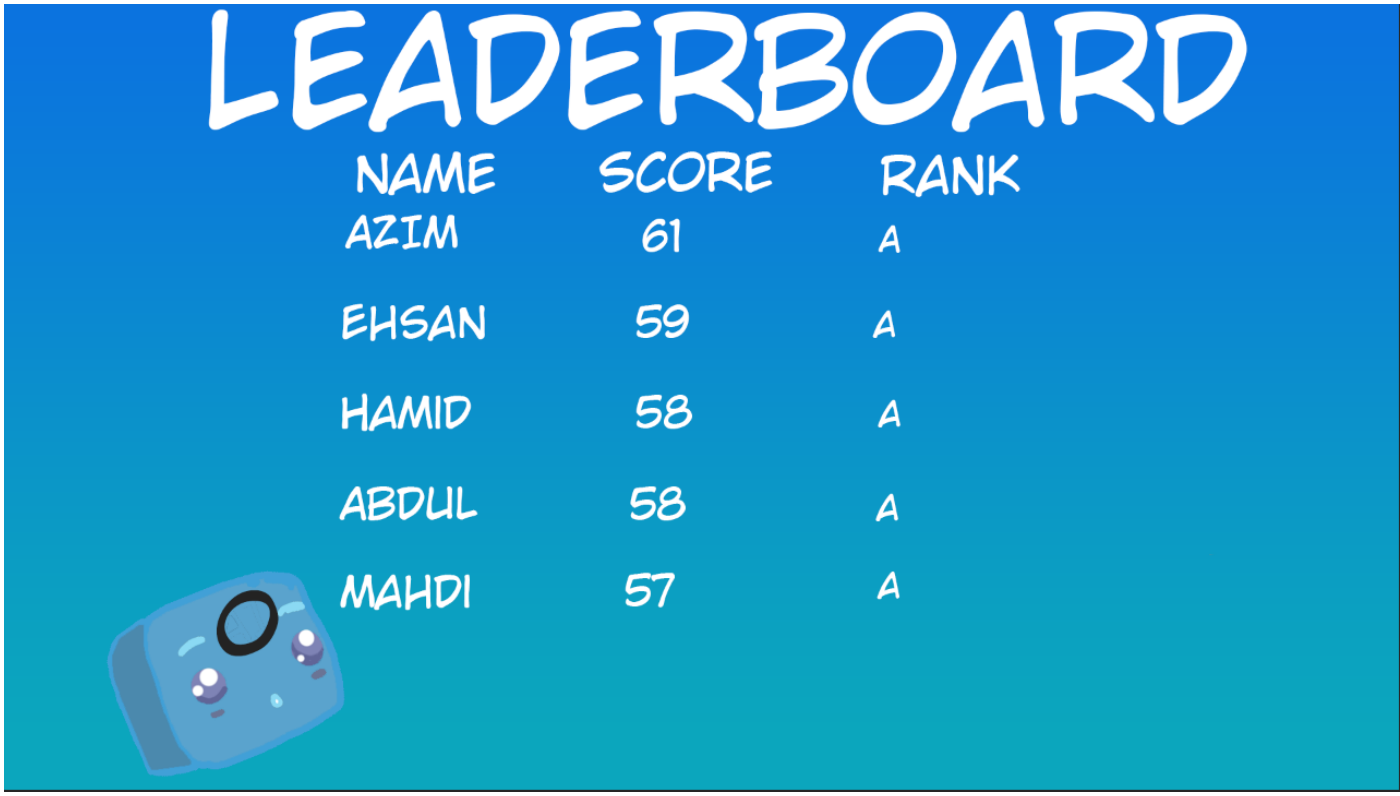
All Algorithms linked



Usability features

For usability features, the goal is that it needs to be easy to understand and intuitive for students. The designs that I have used, represent the frame of the final structure that I am going to use for each game state, visuals are bound to change in the actual game due to the tools available on Photoshop and Unity.

Leader board

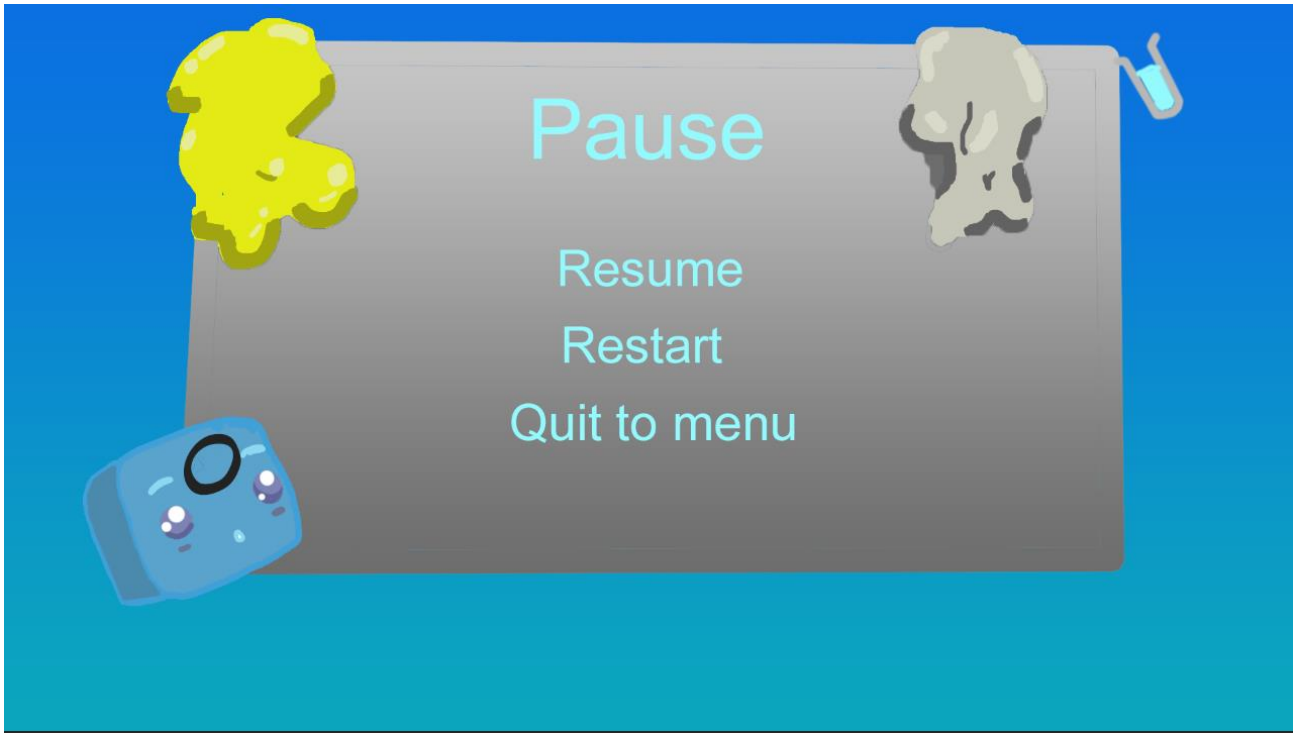


The title of the image is large and at the top of the screen which informs the user what stage of the game they are at. Also, the places of the leader board are placed in the middle of the screen and are easy to see. I changed the leader board to 5 from 10 as it is easier to see the scores and easier to read with only 5 placements.

Game Over Menu

If a player reacts with another cube or falls, the game will finish, and the game-over screen will show up. This shows a visual of the player sprite, oxygen cube, but sad with tubes littered around it. The player will be given two options to either restart and play again or they can quit to the main menu if they wish to do something else.

Pause menu.

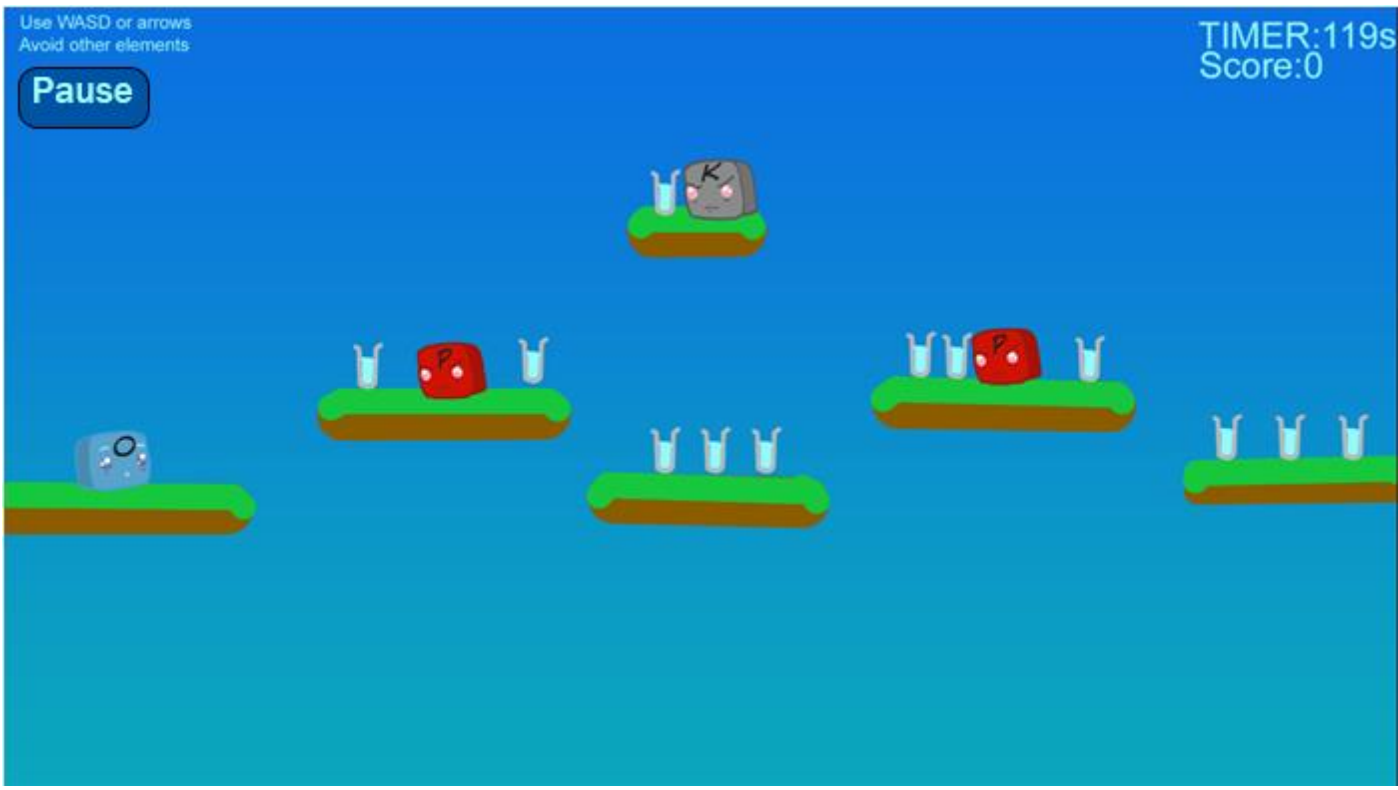


The pause menu is a game state that contains 3 buttons. The 'Resume' button can allow the play to return to the game, the 'restart' game allows the player to restart the game and the 'quit to menu' which lets the player quit to the menu. The theme of the pause menu has different elements along the pause screen.

Game menu

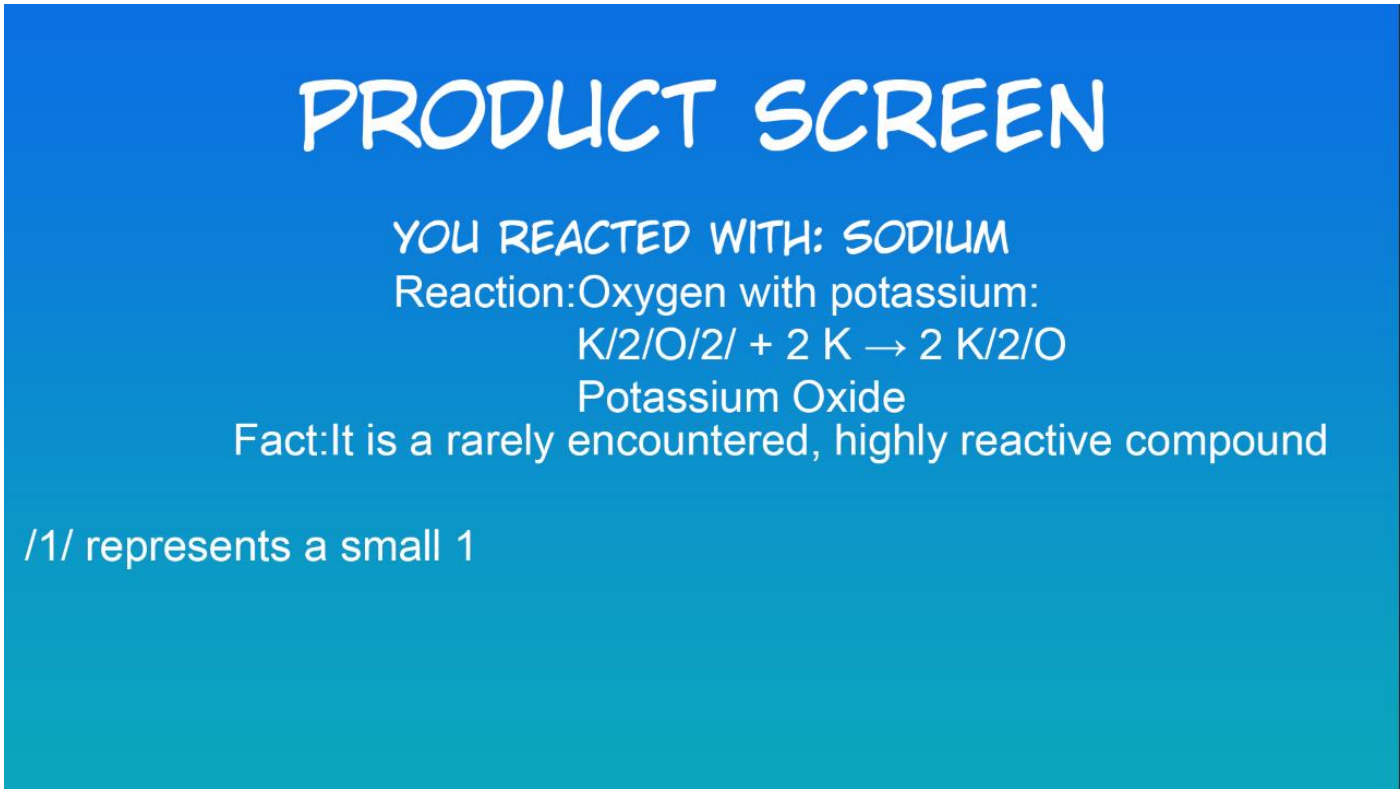
The menu has 4 buttons to choose from. The buttons are kept simple and distinct due to the different colour compared to the background. The 'PLAY' button is larger compared to the other buttons so that the player can choose to play straight away. The title of the game is clearly shown at the top of the screen in different colours. To make the menu seem interesting I added sprites from the game.

Play-state



This is the play state, where the player is playing the game. The timer and score are placed on the top right corner while the controls and objective are placed on the top left corner of the screen. The Pause button is pressed under the controls which allows the player to pause the game if they wish to. The enemy sprites have an angry face to signify that they are enemies and are facing to the left whereas the player's sprite is facing to the right. The cubes are placed on the platforms which the player can collect and increase their score with. The enemies have a symbol on them which represents the element that they are e.g. "O" on the player sprite represents "oxygen", "K" represents potassium

Product screen

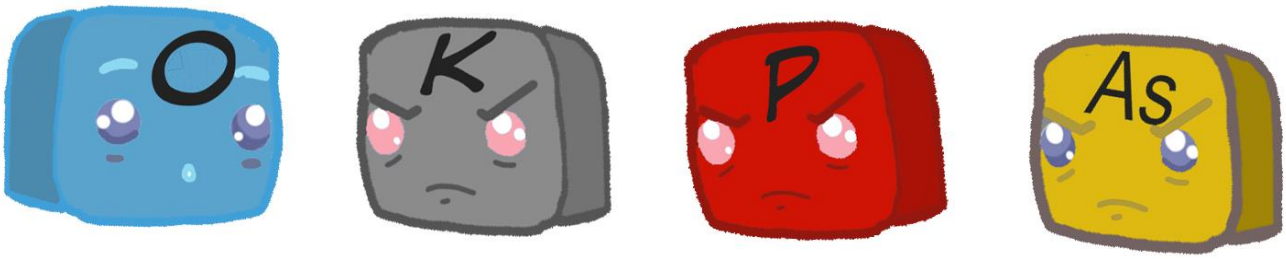


The product screen is shows before the game over screen and occurs if the player has encountered an enemy cube which causes a reaction between elements. The product screen informs what kind of element the player has reacted with the equation underneath. A fact is also provided about the element.

Game over screen results

Once the game is over, the leader board is loaded along with your score for comparison. The player can enter their name.

Assets



The sprites for the elemental cubes will be made in photoshop. Different sprites for each element.

There are also various platform assets available online that I can use to create my side scroller/platformer for my own taste. Here are some of the assets I can use for the platform that the player moves on.

Variables and data structures

Menu

Method	Name	Data type	Explanation	Justification
Procedure	Menu	N/A	This is where the game can branch to different parts that are accessible.	The menu is required for the game to operate well.
Image	menu background	.png	Serves as a background for the menu.	This game makes the game look more appealing for my stakeholders.
Procedure	play	N/A	The game state will switch from the menu state to the play-state.	This is required as it serves as a transition from the menu state to the play state/This leads to the play-state.
Procedure	leader board	N/A	The game state will switch from the	This is required as it serves as a

			menu state to the leader board state.	transition from the menu state to the leader board state/This leads to the leader board state.
Procedure	options menu	N/A	The game state will switch from the menu state to the option state.	This is required as it serves as a transition from the menu state to the option state/This leads to the option state.
Procedure	QuitGame	N/A	Closes the game	This is required as it allows the player to leave the game.
Sound file	ButtonClick_sfx	.mp3	This audio file will be played when the player clicks on a button.	Auditory feedback for the player to signal that they have clicked an option.
Sound file	BG Music	.mp3	This music will play when the player is on the menu	Make the game seem livelier instead of a static silence

Leader Board

Method	Name	Data Type	Explanation	Justification
Procedure	Leader board	N/A	This is where the player can access the leader board from the menu.	This is required as it allows the player to see the names, scores and ranks of the top 5 players.
Procedure	New hs score	N/A	A new high score is added to the scoreboard.	This is required as it allows the leader board to be updated.
Variable	FScore v	integer	This is the user's score.	This will allow the value of the users score to be compared to values of the other score currently on the

				leader board.
Procedure	hs name and validation	N/A	This will only allow high score names that are 2<x<10 to be inputted	Limiting the number of characters inputted by the player, allows the appearance of the leader board to be unaffected by long names.
Array	hs array	N/A	This is the structure used to organize the scoreboard data	This will allow the new high scores to be rearranged
Text File	hs file	.txt	This is to store the high scores	This will allow the high scores to be saved

Paused

Method	Name	Data type	Explanation	Justification
Procedure	pause menu	N/A	The pause menu occurs when the player pressed the 'p' key or pause button is pressed.	This allows the user to pause the game if they wish to take a break or quit to menu
Variable	p pause	Boolean	This variable will determine whether the Pause procedure will occur.	Gives the player a quick way to pause. the game
Variable	ESC Un-pause	Boolean	The player resumes the game when they press ESC.	This will allow the user to resume playing the game.

Options

Method	name	data type	explanation	justification
Procedure	Options	N/A	This is where the game can access the options from the menu.	This is required as it allows the player to access options and toggle music and SFX.
Variable	sfx toggle	Boolean	SFX can be turned on	If the player does

			or off when clicking the SFX button.	not wish to listen to the sound effects, they can toggle it off.
Variable	music toggle	Boolean	Music can be turned on or off when clicking the SFX button.	If the player does not wish to listen to the background music, they can toggle it off.

Re-using platforms.

Method	name	data type	explanation	justification
Procedure	PlatformReuse	N/A	This is where the game can re-use tubes instead of deleting and creating new ones while the game is running.	This reduces the stress on the CPU and RAM which means that the game will run smoother.
Variable	isPlatformAheadDeactivatePoint	Boolean	The game deactivates platforms that have passed a certain point.	Deactivating platforms instead of deleting them means that they can be re-used which reduces CPU power required.
Variable	isPlayerBehindActivatePoint	Boolean	The game activates platforms that are inactive, in front of the player.	Activating platforms instead of creating new ones means that they can be re-used which reduces CPU power required.

Re-using tubes.

Method	name	data type	explanation	justification
Procedure	TubesReuse	N/A	This is where the game can re-use tubes instead of deleting and creating new ones while the game is running.	This reduces the stress on the CPU and RAM which means that the game will run smoother.
Variable	playerCollideWithTu	Boolean	The game	Deactivating tubes

	bes		deactivates tubes that have collided with the player.	instead of deleting them means that they can be re-used which reduces CPU power required.
Variable	isTubesInactive	Boolean	The game activates tubes that are inactive, on recently activated platforms.	Activating tubes instead of creating new ones means that they can be re-used which reduces CPU power required.

Score

method	name	data type	explanation	justification
Procedure	Score	N/A	This keeps track of the player's score.	This will need to be included to give the Leader board its function.
Variable	Game State	String	The value of the current game state is held in this variable. If the game state is = to 'play state' for the score procedure to be started.	This is needed so that the score procedure will not occur when the game is out of the play state
Variable	tube pickup	Integer	This keeps track of the tubes collected by the player	This is required as it affects the user's score
Variable	ingame_score	integer	The score that the player is achieving is = no. of tubes collected x2.	This is how the user's score is calculated as tubes are worth 2 points each.
Variable	Go message	Boolean	The score procedure will start once this variable is true.	This is required for the timer so that it does not start before the player is able to move.
Variable	Countdown	Integer	The timer for the run. Once it reaches 0, the game will end.	This is not to drag out the game.

Product screen

method	name	data type	explanation	justification
Procedure	product screen	N/A	Player sees a product of reaction and a fact of the reaction's product.	This enables the player to learn more of chemistry whenever they lose.
Procedure	reaction	string	A product of the chemical reaction between the 2 cubes are shown here	This informs the player what the product of the chemical reaction is, letting the player learn more of chemistry.
Procedure	product fact	String	For each possible product in the game, there will be a fact of the product.	This gives the player more info about the chemical element.

Game over

method	name	data type	explanation	justification
Procedure	game over	N/A	The game-over screen is shown once the player has fallen or reacted with another cube.	This is to show the player that they have lost and that the game is over.
Procedure	game over background	N/A	A background image is loaded with the game over screen.	Makes the game-over screen look pleasing for the player.
Procedure	leader board results	N/A	The player is shown their results in comparison with those of the leader board.	This lets the player know their results and how it is compared to the top 5 players.

Playing

method	name	data type	explanation	justification
Procedure	play state	N/A	This procedure is linked to the ability to play the game. All respective procedures are started e.g., Timer, score etc.	This procedure that links all the procedures relating to the playing state.
Image	player sprite	.png	This will be the sprite for the player	This acts as a visual to where the player is in the game.
Image	platform sprite	.png	Sprites for the platform.	This acts as a visual to where the player can move without falling.
Image	enemy sprite	.png	This will be the sprite for the enemies. Slight variations in sprites for each element.	This acts as a visual to where the enemies are in the game.
Image	tube sprite	.png	This will be the sprite for the tubes.	This acts as a visual to where the player can collect tubes for points.
Procedure	Powerup	N/A	Once the player reaches a certain score, they are given a powerup.	This makes the game more engaging for the player.
Variable	key pressed	char	This variable holds the character press and compares it to the movement controls and changes the player's position.	This enables the player to move.
Procedure	power up limit	N/A	If the player has already activated their power-up once, they can't use the power-up again for the rest of the run.	This prevents the game from being too easy for the player.
Sound file	tube collection sound	.mp3	This sound is played when the player collects a tube	This acts as auditory feedback for the player when

				they collect a tube.
Variable	timer value	integer	the value the timer is set to 120 and decreases by one every second	This acts as the countdown function of the game.

Test data

Navigating the menu

Test Data: Main menu.

Test Data	Type
Left Mice Click (LMC).	Valid.
1	Invalid.
w	Invalid.
#	Invalid.

Buttons click sound effect:

Test data	Type
'Play is selected'	Valid
'Play is not selected'	Invalid
'Leaderboard is selected'	Valid
'Leaderboard is not selected'	Invalid
'Options is selected'	Valid
'Options is not selected'	Invalid
'Quit is selected'	Valid
'Quit is not selected'	Invalid

Scoring

Test Data: Scoring system.

Test Data	Type
Score accumulates over time.	Valid.
Score rapidly increases.	Invalid.
Score is displayed.	Valid.
Score is saved on Leader board.	Valid.
Score is 0 at restart/reset.	Valid

Leader board

New high score

Test data	Type
(new score)> (scores on Leader board)	valid
(new score) < (scores on Leader board)	invalid

High score name and validation:

Test data	Type
EhsanAhmed	valid
SJ	valid
iqh3wehn23n	invalid
89	invalid

Options

Test data	Type
Left click SFX button	Valid
Left click music button	Valid
Right click SFX button	Invalid
Right click music button	Invalid

Quit

Test data	Type
Left click quit button	Valid
Right click quit button	Invalid

Play

Game state: Play-state.

Test data	Type
'Menu select: 'Play'	Valid
'Menu select: 'Leaderboard'	Invalid
'Menu select: Options'	Invalid
'Menu select: Quit'	Invalid

Player Death

Test Data: Player restarts after death.

Test Data	Type
Player is set at beginning after reset.	Valid.
Player is spawned on platforms.	Valid.

Test Data: Game over menu.

Test Data	Type
Left Mice Click (LMC) on Restart button.	Valid.
Left Mice Click (LMC) on Quit button.	Valid.
w	Invalid.
#	Invalid.

Timer

Test data	Type
'Game message: "Go!" has been displayed'	Valid
'Game message: "Go!" has not been displayed'	Invalid

Pausing

Test Data: Pause menu.

Test Data	Type
Left Mice Click (LMC) on Pause button.	Valid.
'ESC' key	Valid.
w	Invalid.
#	Invalid.

Un-pause.

Test Data	Type
Left Mice Click (LMC) on Resume button.	Valid.
'ESC' key	Valid.
w	Invalid.
#	Invalid.

Controls

Test Data: The player's movement.

Test Data	Type
'A' key	Valid
'D' key	Valid
'W' key	Valid
'1' key	Invalid

‘Space bar’	Invalid

Test Data: The player jumping.

Test data	Type
Player responds to ‘W’ key while on ground.	Valid.
Player responds to ‘W’ key while not on ground.	Invalid.
Left mice click on player.	Invalid.

Test Data: Player’s movement animations.

Test data	Type
‘A’ key transitions player to running animation.	Valid.
‘D’ key transitions player to running animation.	Valid.
‘W’ key transitions player to jumping animation.	Valid
‘1’ key.	Invalid.
‘Space bar’.	Invalid.

Camera

Test Data: Camera following player.

Test data	Type
Camera moves with ‘A’ key.	Valid.
Camera moves with ‘W’ key.	Valid.
Camera moves with ‘D’	Valid.

key.	
Camera moves with ‘Q’ key.	Invalid.
Camera moves with ‘3’ key.	Invalid.

Use of Platforms

Test Data: Endless platform generation.

Test Data	Type
‘A’ key generates platforms endlessly.	Valid.
‘D’ key generates platforms endlessly.	Valid.
‘W’ key generates platforms endlessly.	Invalid.

Test Data: Platform destruction.

Test Data	Type
Platform destroys soon after disappearing from screen.	Valid.
Platform remains after disappearing from screen.	Invalid.

Test Data: Platform space randomization.

Test Data	Type
Platforms obey distance Between platforms values.	Valid.
Platforms use random value for spaces.	Invalid.

Test Data: Platform re-used by game.

Test Data	Type
Platforms deactivated off screen.	Valid.
Platforms are re-used.	Valid.
Platforms remain active after being off-screen.	Invalid.

Test Data: Random platform sizes.

Test Data	Type
Different sized platforms on screen.	Valid.
Different sized platforms together.	Invalid.
Platforms obey space randomization	Valid.

Test Data: Random platform height.

Test Data	Type
Platforms are loaded at different heights.	Valid.
Platforms displayed at different heights on-screen.	Valid.
Platforms displayed off-screen.	Invalid.

Difficulty Curve

Test Data: Player speeding up.

Test Data	Type
Game speeds up gradually with time.	Valid.
Game speeds quickly over a short time frame.	Invalid.

Test Data: Random enemies spawning.

Test Data	Expected
Enemies do not appear on every platform.	Valid.
Enemies spawn anywhere.	Invalid.

Tubes

Test Data: Tube interaction.

Test Data	Expected
Tubes increase points.	Valid.
Tubes de-spawn on contact.	Valid.
Tubes remain after contact.	Valid.
Player can click on tubes (Left mice click).	Invalid.

Test Data: Tube destruction.

Test Data	Expected
Tubes de-spawn on contact.	Valid.
Tubes remain after contact.	Invalid.

Test Data: Random Tube spawning.

Test Data	Expected
Tubes do not appear on every platform.	Valid.
Tubes spawn anywhere.	Invalid.

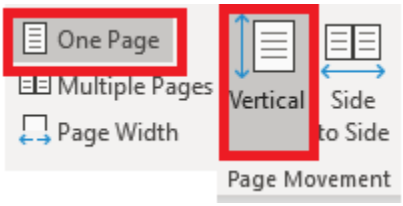
Iterative development of coded solution

Total marks 15

Testing to inform development

Date:01/01/2021

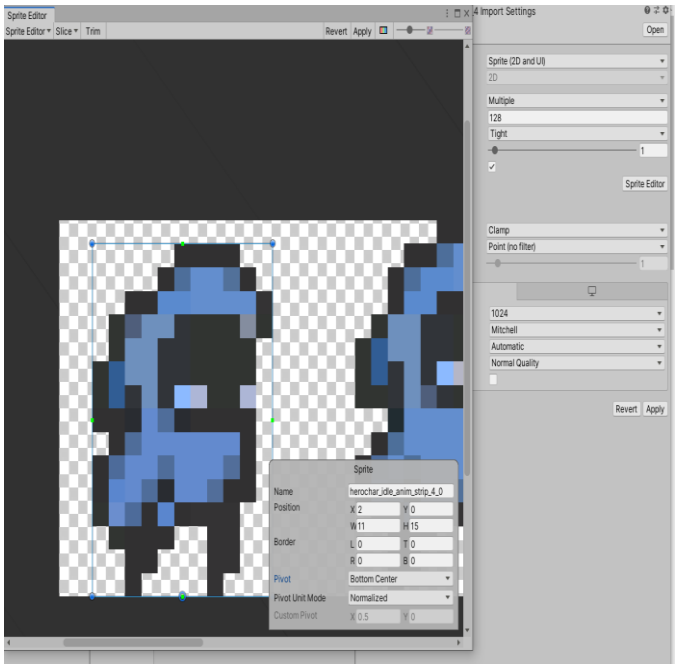
Note: Please read in vertical view, not horizontal view. This will prevent formatting errors between images and text.



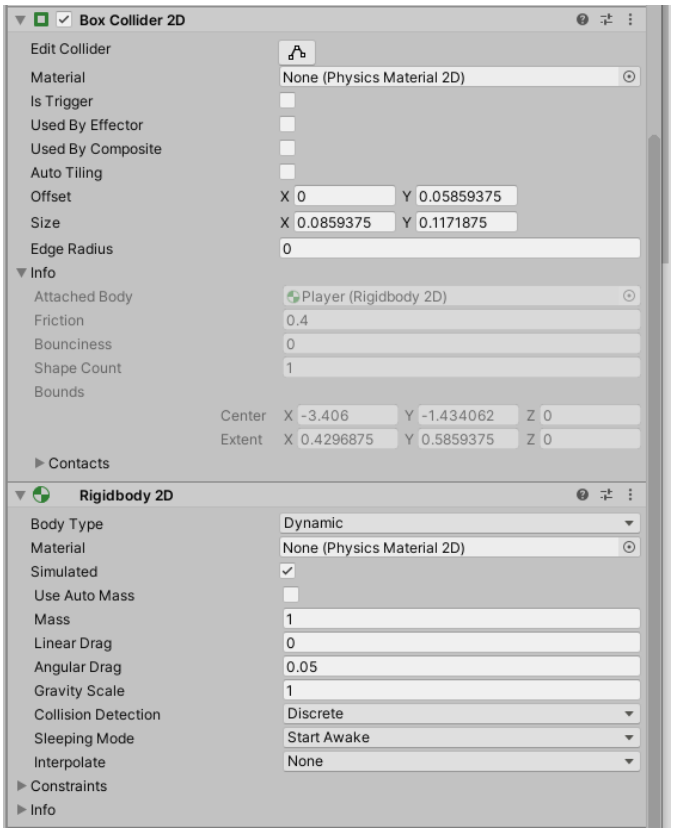
If you are viewing this document on Word, please select the One Page option and Vertical option in the view tab.
Google Docs and Adobe PDFs already have vertical view as the default view.

Creating Player Movement.

For my endless sides scroller I want to get my game mechanics out of the way first before I get into the looks of the game, therefore I decided to use a free resource pack for sprites before I create my own. My game will most likely contain pixel art. In my analysis and design, I stated the player being able to move in the game.



I opened a character sheet in Unity's sprite editor and have selected the sprite I will temporarily be using. I set the pivot at the bottom as that is where the sprite will interact with other objects such as a platform (e.g., standing on a platform).



I added a Box Collider2D and a RigidBody 2d to my sprite.

The Box Collider2D lets the player sprite detect when it has collided with another object (e.g., Platform) and the RigidBody allows the player sprite to obey physics in the game rather than just being suspended in the air.

Player Controller script

Unity lets me use C# scripts, where I can program my game and add properties to sprites etc.

Here I have set my player's speed and the force of his jump as a float value which allows me to tinker with his speed and jump during development. The player's RigidBody has been set to private so that no external script alters the player's physics. This is shown below:

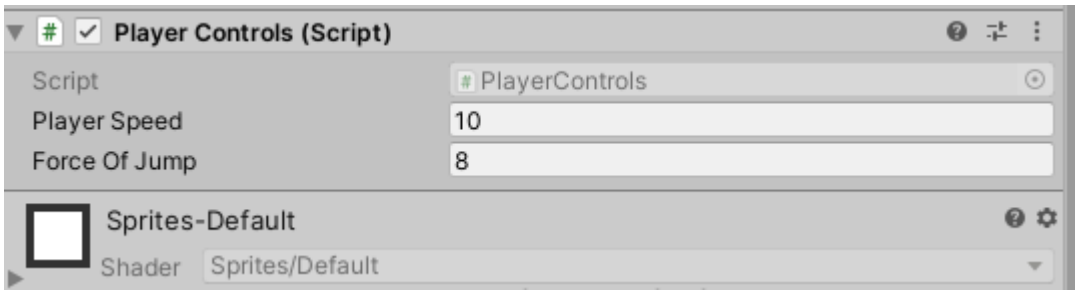
```
public float playerSpeed;
public float forceOfJump;

private Rigidbody2D playerRigidBody;

// Start is called before the first frame update
0 references
void Start()
{
    playerRigidBody = GetComponent<Rigidbody2D>();
}
```

These are my controls for the character which allows him to move in the game if D, W or A is pressed. Since the game is a 2d endless side-scroller, I have assigned the x-coordinate to the player's speed and the y-coordinate to the player jump.

These are quite simple statement as the game checks if a key is being pressed and depending on the value of player speed, it'll move forward by that amount (e.g. player's speed is 10, pressing 'D' will result in the player sprite moving by (+10,0) on the x-axis)

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

    if (Input.GetKeyDown(KeyCode.D))
    {
        playerRigidBody.velocity = new Vector2(playerSpeed, playerRigidBody.velocity.y);
    }

    if (Input.GetKeyDown(KeyCode.A))
    {
        playerRigidBody.velocity = new Vector2(-playerSpeed, playerRigidBody.velocity.y);
    }

    if (Input.GetKeyDown(KeyCode.W))
    {
          playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, forceOfJump);
    }

}
```

The player's movement can be altered on the script it's attached to here:



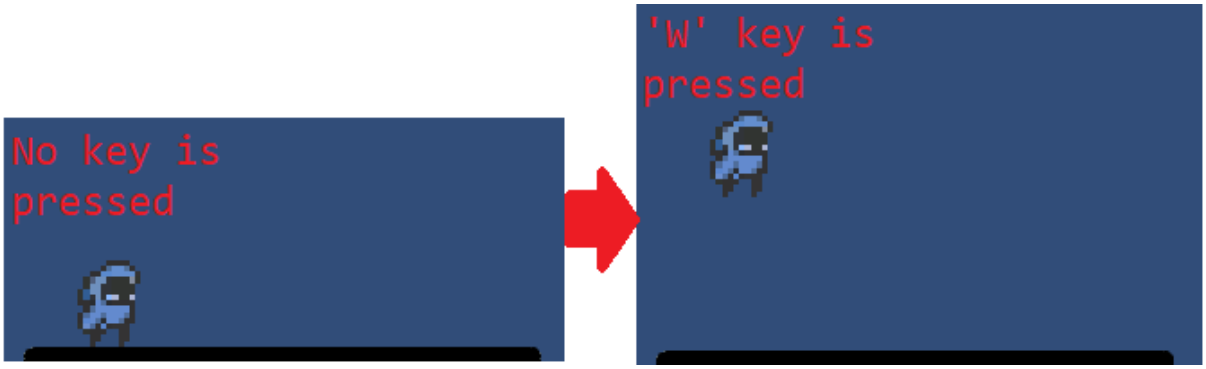
I quickly ran into a problem where if I hold down onto a key, the player will not continue moving. However, I realised that when I changed the method from 'GetKeyDown' to 'GetKey', the player's sprite was able to continue moving. If the key is pressed, the player's coordinates keep updating. The GetKeyDown method will remain for jumping as the player could jump away from the screen, if 'W' is held for too long.

```
if (Input.GetKey(KeyCode.D))
{
    playerRigidBody.velocity = new Vector2(playerSpeed, playerRigidBody.velocity.y);
}

if (Input.GetKey(KeyCode.A))
{
    playerRigidBody.velocity = new Vector2(-playerSpeed, playerRigidBody.velocity.y);
}

if (Input.GetKeyDown(KeyCode.W))
{
      playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, forceOfJump);
}
```

If the 'D' key is being held down the x-coordinate value continually increases from the current x-coordinate value of the player. Similarly, if the 'A' key is being held down the x-coordinate value continually decreases from the current x-coordinate. When the 'W' key is being pressed (or held down) the player's y-coordinates are increased momentarily before returning to its initial value.

As you can see below, when the ‘D’ or ‘A’ is pressed, the player is able to move left or right. Which is what I was expecting to happen as these will serve as the one component of the player’s controls.

The jumping function was also a success as the player jumps when the ‘W’ key is being pressed. The player being able to jump will serve as a way for the player to traverse across platforms.



In my analysis and design stage, I stated that the ‘W’,’A’,’S’ and ‘D’ keys would be used for the controls that the player can use. The ‘W’,’A’ and ‘D’ keys have been used. The ‘D’ key being used as a way for the player to go down, will not needed in the game as the ‘RigidBody2D’ will being the player back to the ground due to physics.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Whether the sprites move along the platform if one of the control buttons are pressed.	‘W’,’A’,’D’ keys for controls.	Player’s controls are a mandatory party of the gameplay, without it the player will not be able to move or progress in the game.	The player moves up, left or right depending on the key being pressed.	The player moves up, left or right depending on the key being pressed.

Whether the player returns to the ground after 1 jump and cannot double jump.	The 'W' key for jumping.	If the player can double jump or more, then the player will be able to cheat the game.	The player returns to the floor and is unable to jump further while in the air.	The player can infinitely jump off and away from the screen. They can spam jump.

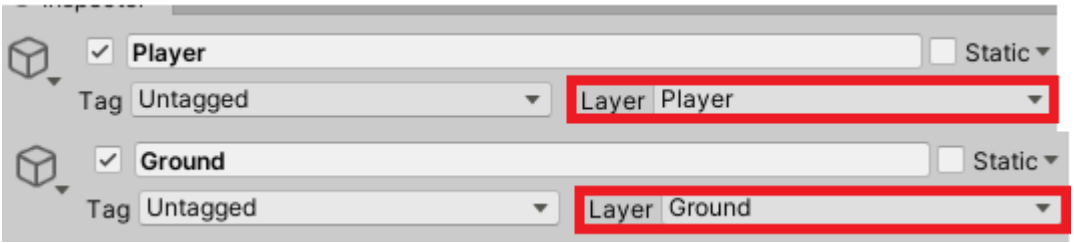
Test Data: The player's movement.

Test Data	Expected	Actual
'A' key	Valid	Valid
'D' key	Valid	Valid
'W' key	Valid	Valid
'1'	Invalid	Invalid
'Space bar'	Invalid	Invalid

Letting the player recognise the ground and prevent spam jumping.

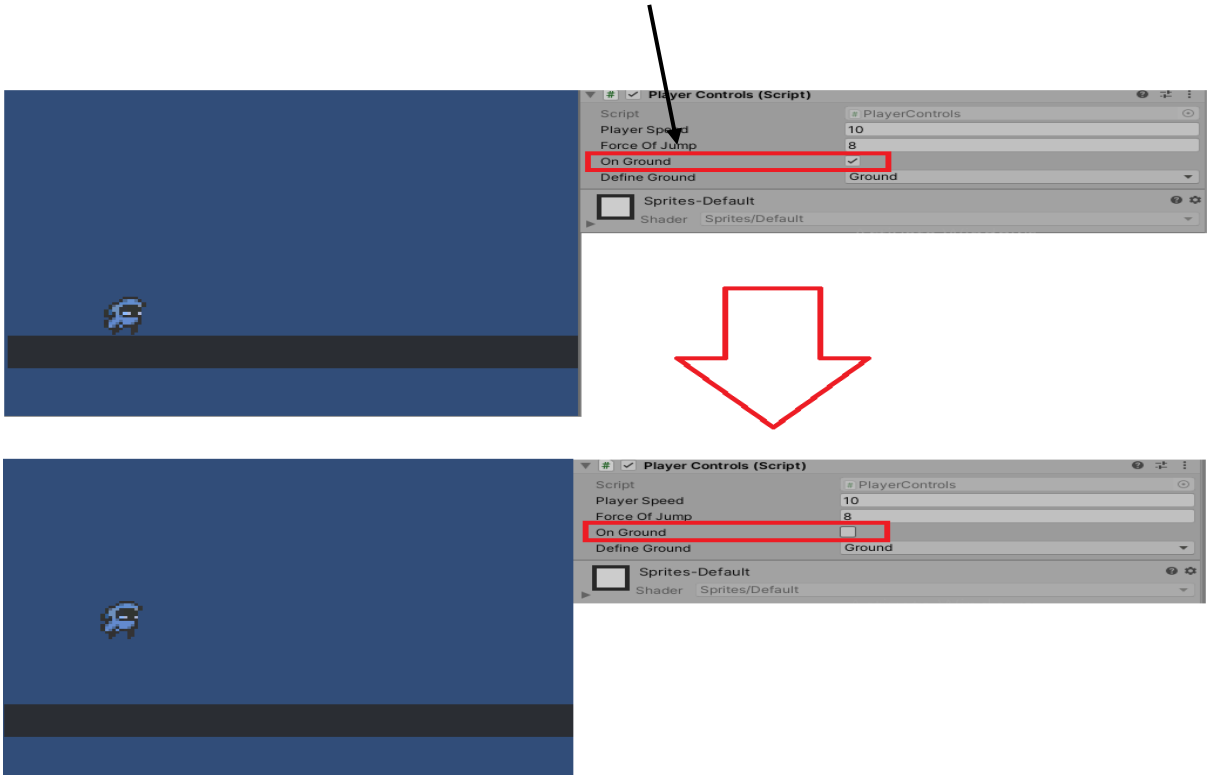
While testing the movement in the game I noticed that the player could spam the 'W' key and continuously jump up and off the screen where they cannot be seen, shown in my testing table above. To combat this, I will be using a ground verifier which checks if the player's collider and the grounds collider components are in contact with one another.

The player and the ground platform will be assigned their own layers on Unity which allows them to be identified in the game. 'onGround' is a Boolean variable which will return true if the player is in contact with the ground. The Player and the ground have been given their own layers: 'Player' and 'Ground' respectively which allows me to create a collision between these two game objects. This is shown here:



```
public bool onGround;
public LayerMask defineGround;

private Collider2D playerCollider;


// Start is called before the first frame update
0 references
void Start()
{
    playerRigidBody = GetComponent<Rigidbody2D>();
    playerCollider = GetComponent<Collider2D>();

}
```

I have created an 'if statement' so that whenever the player is on the ground, 'onGround' is true. If 'onGround' is true, then the player can jump. This prevents the player from continuously jumping upwards. This can be shown when I am testing the game.



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Whether the player returns to the ground after 1 jump and can't double jump, when they're not grounded.	The 'W' key for jumping.	If the player can double jump or more, then the player will be able to cheat the game.	The player returns to the floor and is unable to jump further while in the air.	The player cannot jump again when they are already in the air. They can only jump if they are touching the

				ground.

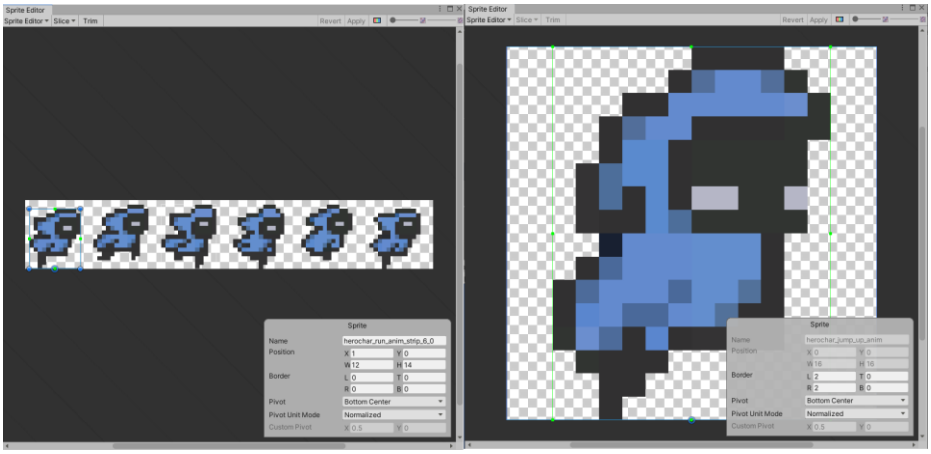
Test Data: The player jumping.

Test data	Expected	Actual
Player responds to 'W' key while on ground.	Valid.	Valid.
Player responds to 'W' key while not on ground.	Invalid.	Invalid.
Left mice click on player.	Invalid.	Invalid.

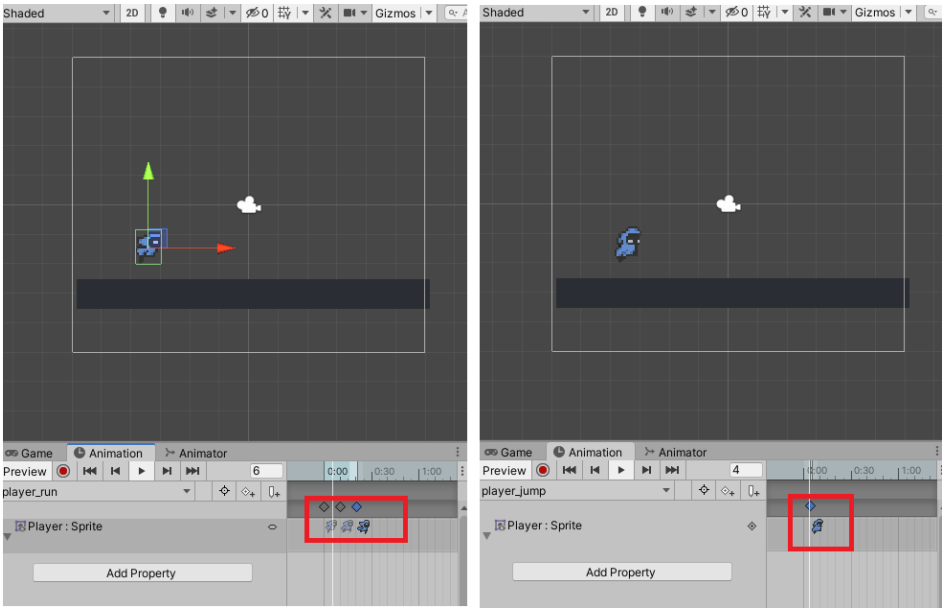
Movement Animations

Animation was something desired by my stakeholders for my game, shown in my design stage. I am going to use the sprite sheets of my temporary sprite before I create and implement my own so that I can practice how animation works in Unity.

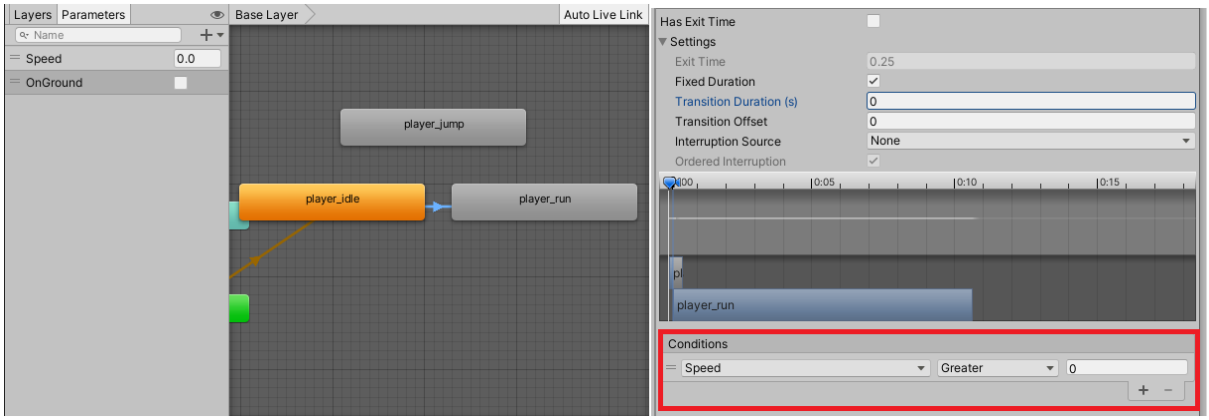
These are the sprites that I will be using for running and jumping, in the sprite editor. These can be selected individually and be used together, such as the multiple sprites in the running sprite sheet, but I will be using 3 different 'running 'frames for now.



Near the bottom right of the first image (shown below), there is 3 sprites that are spaced out and depending on the spacing, the animation could be quick or long. I spaced the 3 sprites out a little bit and saved it. The jumping animation will be one frame as that player sprite cannot really do much in the air.



After creating the animations, I must create transitions so that when the player is moving an animation will be activated. I have set a condition for when the running animation starts, when the speed is greater than 0, the idle animation will transition into the running animation. This is shown below:



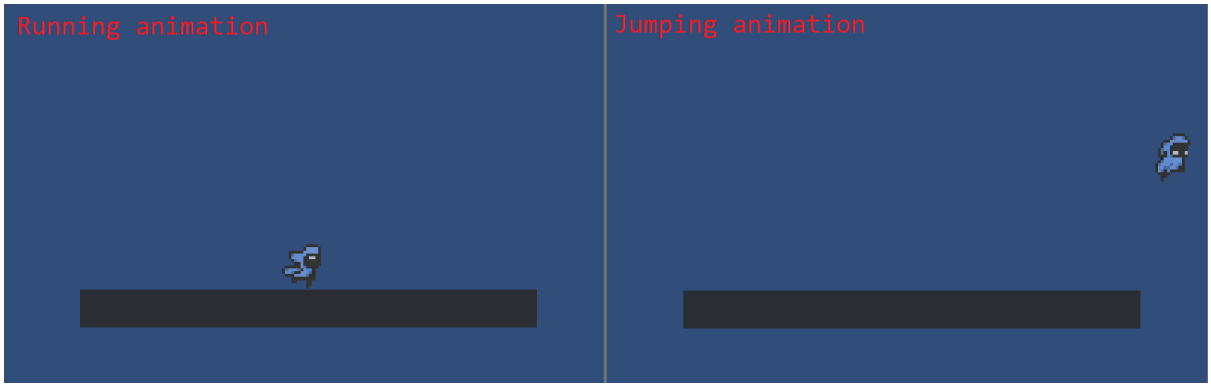
After setting the conditions, I added 2 statements in my script for the player controller. I used the set float method, for my running animation, as it allows the game to send float values to the animator. If the conditions are fulfilled, the player sprite will transition into the running animation. Same principle is applied for the jumping animation, the game will send a true or false value to the animator and depending on the value, the player will transition to-or-from the jump animation.

Test Data: Player's movement animation.

Test data	Expected	Actual
'A' key transitions player to running animation.	Valid.	Valid.
'D' key transitions player to running animation.	Valid.	Valid.
'W' key transitions player to jumping animation.	Valid	Valid.
'1' key	Invalid.	Invalid.
'Space bar'	Invalid.	Invalid.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
When the Player moves left or right, the player's sprite will transition into the running animation.	'A' or 'D' key.	Animation brings life into a game and makes it look less static and boring. It acts as a clear visual indicator that the player is moving.	The player sprite will transition into a running or jumping animation depending what key, for the controls, is being pressed.	The player sprite transitions into a running or jumping animation depending what key, for the controls, is being pressed.
When the Player moves up and down, the player's sprite will transition into the jumping animation.	'W' key.	Animation brings life into a game and makes it look less static and boring. It acts as a clear visual indicator that the player is moving.	The player sprite will transition into a Jumping animation if the 'W' key is being pressed.	The player sprite transitions into the jumping animation if the 'W' key is being pressed.

When the 'D' or 'A' key is pressed, the player will transition into the running animation. When the 'W' key is pressed, the player will transition into the jumping animation. Both are shown below:



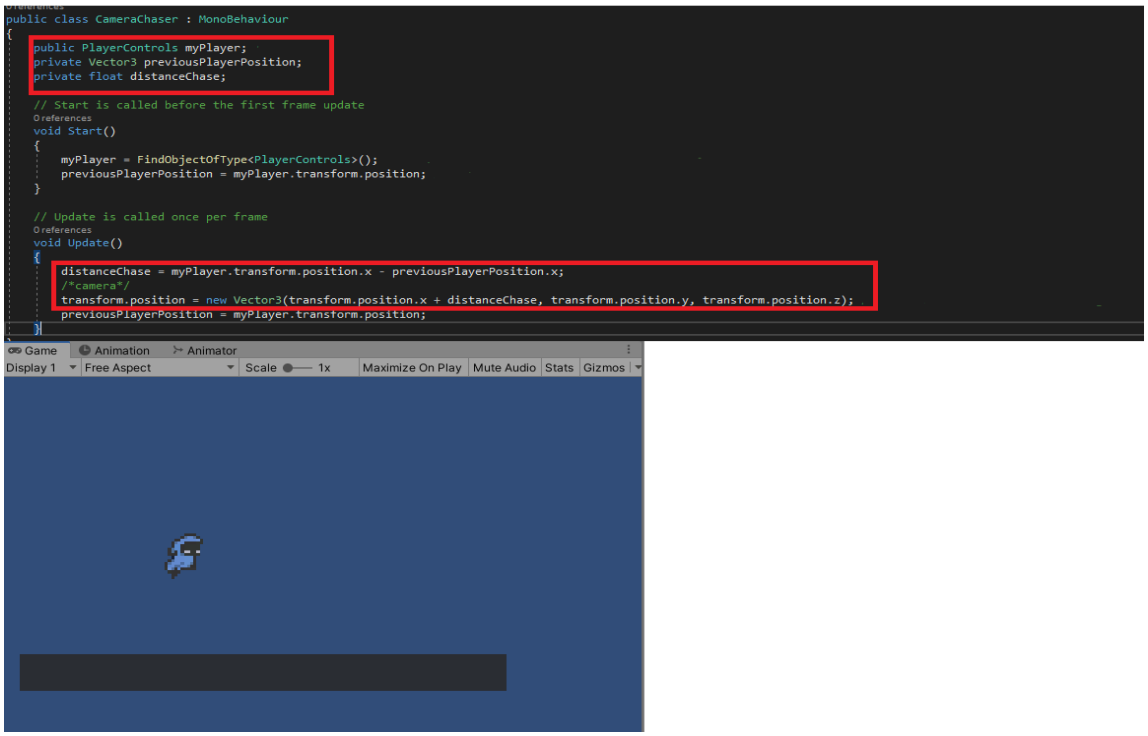
Making the camera follow the player.

Originally, I attached the camera to the player which I believed would have resulted in letting the user always see the player sprite in the game, however that created another problem. That being the player may not be able to see what is below or in front of them when they are playing. This is shown below:

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The camera following the player when the player moves and lets the	The camera moves in-game so it should be based on the keys for the	The camera should follow the player so that the player can see the	The camera follows the playing during the game and allows them to	Making the main camera a child of the player resulted in the camera

player see platforms.	player movement controls.	platforms and obstacles up ahead.	see under their player sprite and what is up ahead.	following the player, but not at a fixed position. This caused platforms to be blocked up ahead

In the image below,3 new variables have been created. The first one refers to the position of the player from the player controls script, while the other 2 are going to be used to let the camera chase the player. The second red box highlights that the difference between the camera's initial position and the player's sprites current position, is the distance the camera needs to travel to keep up with the player. This ended up relieving the issue of the camera hiding the platforms from sight as shown under the script, in the image.



Test Data: Camera following player.

Test data	Expected	Actual
Camera moves with 'A' key.	Valid.	Valid
Camera moves with 'W' key.	Valid.	Valid

Camera moves with 'D' key.	Valid.	Valid
Camera moves with 'Q' key.	Invalid.	Invalid
Camera moves with '3' key.	Invalid.	Invalid.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The camera following the player when the player moves and lets the player see platforms.	The camera moves in-game so it should be based on the keys for the player movement controls.	The camera should follow the player so that the player can see the platforms and obstacles up ahead.	The camera follows the playing during the game and allows them to see under their player sprite and what is up ahead.	The camera followed the player at a fixed position which allows the player to see what platform or obstacles may be up ahead

Date:02/01/2021

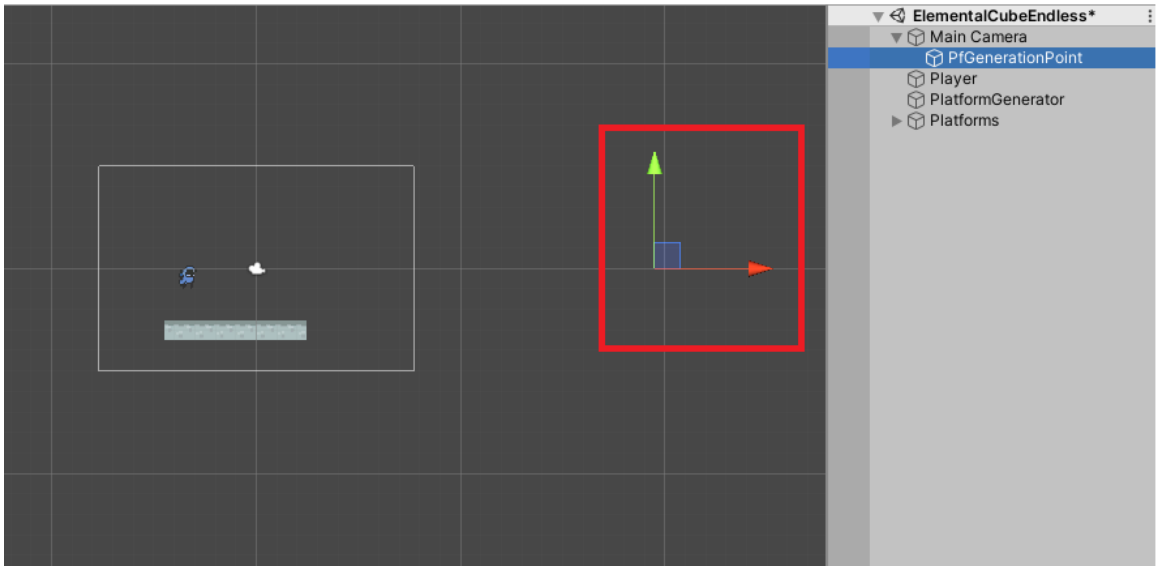
Generating infinite platforms

One of the main mechanics that my project should have achieved is generating an infinite number of platforms before the timer runs out. Right now, I will focus on generating an infinite number of platforms before I go into changing the spacing, heights and lengths of the platforms.

For now, I will be using royalty-free sprites again, this time for my platforms. Platform design is not noticed that much so it will be unlikely that will I change the sprites. The platform is shown below:

In the image below, I created a new game object (the highlighted one) and set it way ahead of the player. It will act as a point where new platforms are generated, and I put it as a child

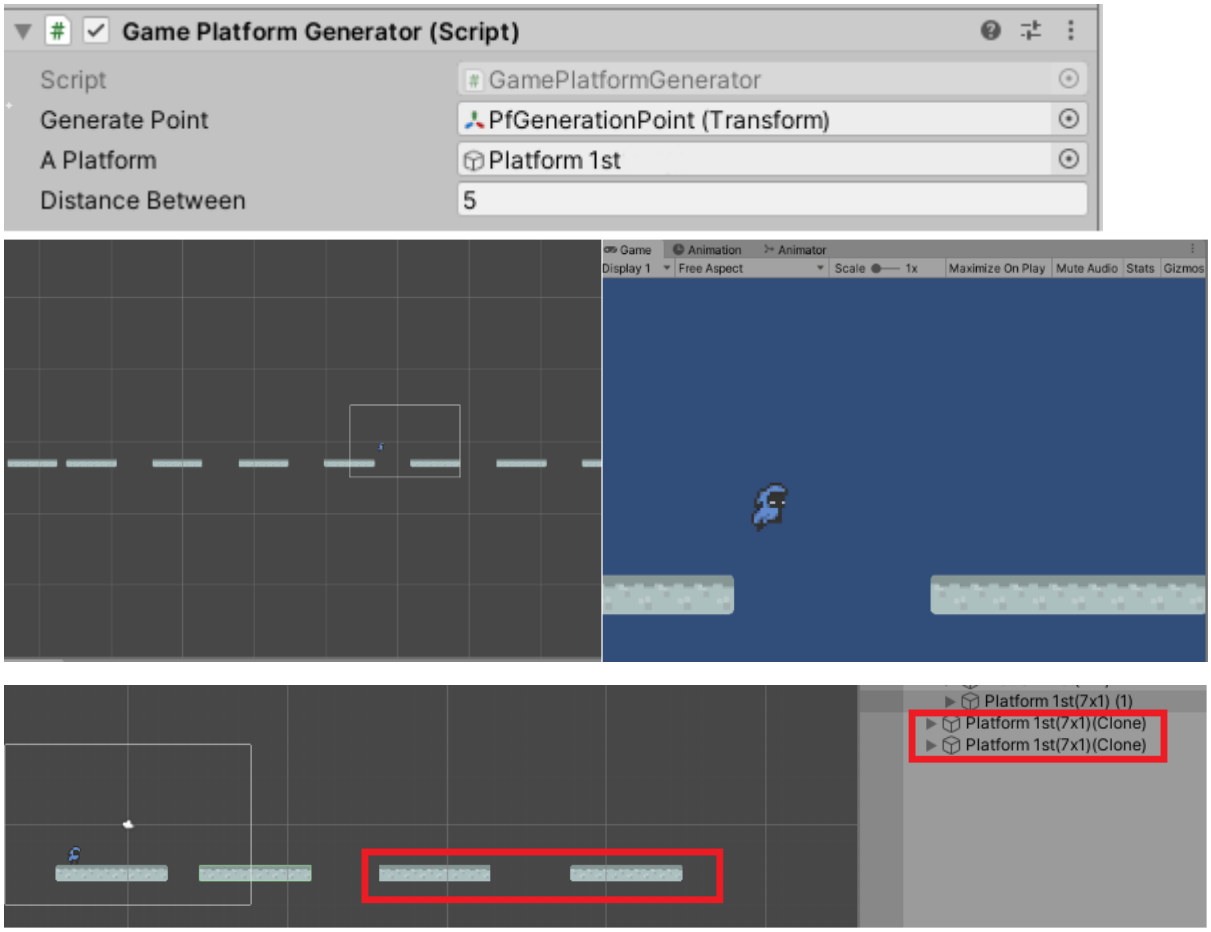
of the main camera. This means that when the camera is moving, the platform generation point will always be a set distance ahead.



I added a few float variables to the script that will help to generate the platforms at a certain length. Determining the distance between the platforms can be altered in Unity's interface and I can use that value in an 'if' statement to new platforms. To generate the platforms ahead was relatively simple. The platform moves ahead a little bit, based on the length of the platform and value of the distance in-between the platforms. Once this is done, a copy of a platform will be created using 'Instantiate'. This is shown below:

```
ublic class GamePlatformGenerator : MonoBehaviour


    public Transform generatePoint;  //Point where platforms need to be generated
    public GameObject aPlatform;  //The platform that'll generate ahead of the player
    public float distanceBetween;
    private float widthOfPlatform;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        widthOfPlatform = aPlatform.GetComponent<BoxCollider2D>().size.x;

    }

    // Update is called once per frame
    0 references
    void Update()
    {
        if (transform.position.x < generatePoint.position.x)
        {
            transform.position = new Vector3(transform.position.x + widthOfPlatform + distanceBetween, transform.position.y, transform.position.z);
            Instantiate(aPlatform, transform.position, transform.rotation);
        }
    }
```

I added the distance in between the platforms here and the platform that the script will be duplicating when creating new platforms (Platform 1st). This is shown below:



As shown above, platforms are being generated at a set length continuously if I am behind the generation point. However, I realised that I ran into the problem of memory management, which is caused by platforms remaining in the game even after I passed them. This could provide problems for my stakeholders as they may run out of memory relatively quickly. Like the platform generation point, I will need to add a point behind the camera which will destroy platforms after they pass through it. This will conserve memory.

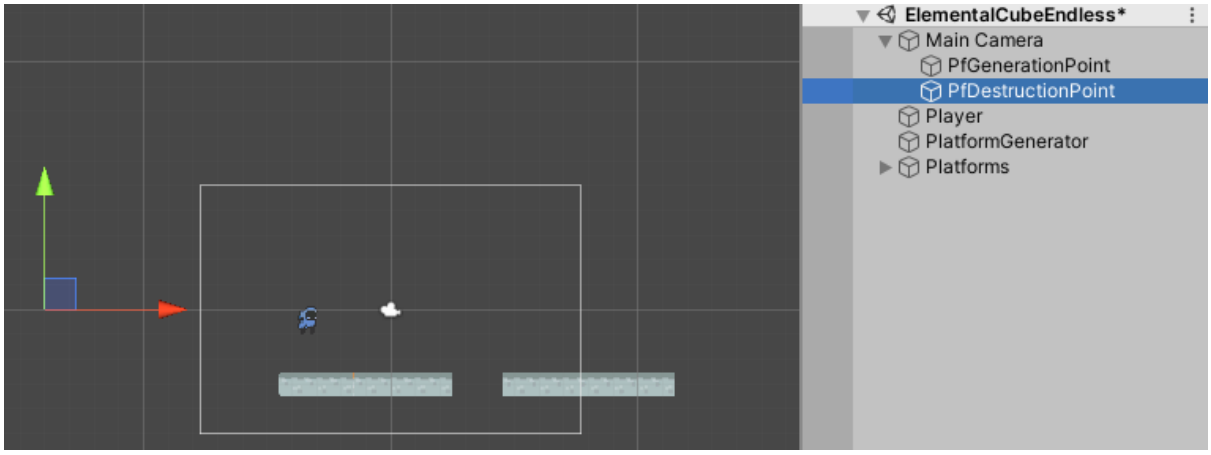
Test Data: Endless platform generation.

Test Data	Expected	Outcome
'A' key generates platforms endlessly.	Valid.	Valid
'D' key generates platforms endlessly.	Valid.	Valid.
'W' key generates platforms endlessly.	Invalid.	Invalid.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The game continuously generating platforms for when the player is moving	The player will not be able to control the platform spawn. but the platforms only spawn depending on if the player is moving (Player movement keys)	Generating infinite platforms keeps the player engaged as they can only lose from falling, running into enemies, or running out of time.	The game generates platforms continuously if the player is moving.	The game generates platforms continuously if the player is moving along the platforms.

Creating a platform destroyer

This is my follow up to my problem mentioned earlier. I created a new game object in Unity, that will also be under the Main Camera object, that will be set behind the player. Platforms passed this point will be deleted.



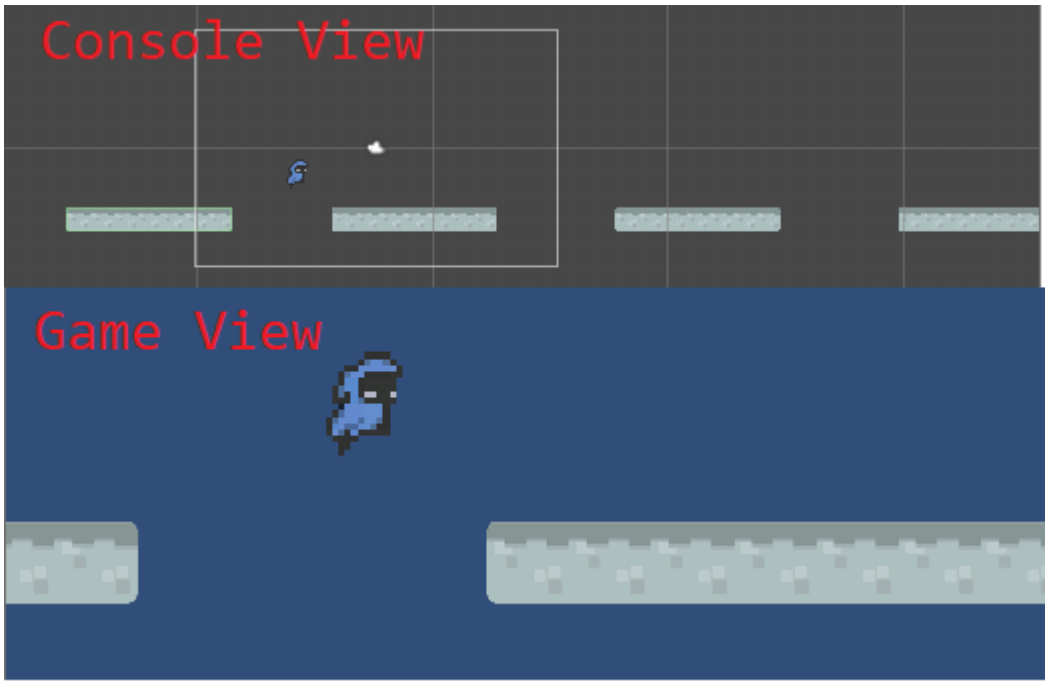
In the platform destroyer script, I made a game object which is equal to what I called the game object mentioned earlier, 'PfDestructionPoint', through using the 'find' method which searches for any object with that exact name.

```
public GameObject destructPoint;

// Start is called before the first frame update
0 references
void Start()
{
    destructPoint = GameObject.Find ("PfDestructionPoint"); //Finds the object called platform destruction point
}
```

The condition to delete objects passing the destruction point was relatively simple. When a game object, to which this script is attached to, has coordinates that are less than that of the destruction point's, they are destroyed. They have been attached to the platforms.

```
if (transform.position.x < destructPoint.transform.position.x)
{
    Destroy (gameObject);
}
```

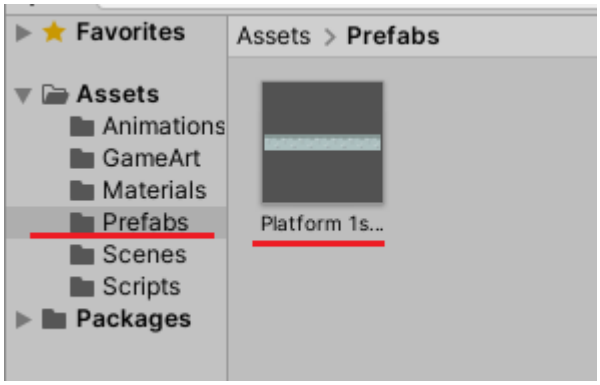
When I tested the game, the platforms behind me were destroyed which is what I wanted (Shown below). However, I ran into a problem where no more platforms were being generated a few seconds into playing the game. I quickly realised that the script(program) which I created to generate the platforms, was making copies of the first platform. This means that once the first platform is destroyed, the script cannot generate copies of that platform.



I realised what the problem was once this message was shown below my game:

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The game destroys the platforms after a certain point when the player has passed it.	No input from the player as it is an in-game object.	Not deleting game objects will result in more memory (RAM) being occupied. Deleting objects	The game continuously deletes platforms behind the player when the player is moving	The game only destroyed the first 2 platforms, and no more platforms were generated due

		conserves memory and improves performance.	forward.	to the platform generator having no platform to refer to when generating a new object.

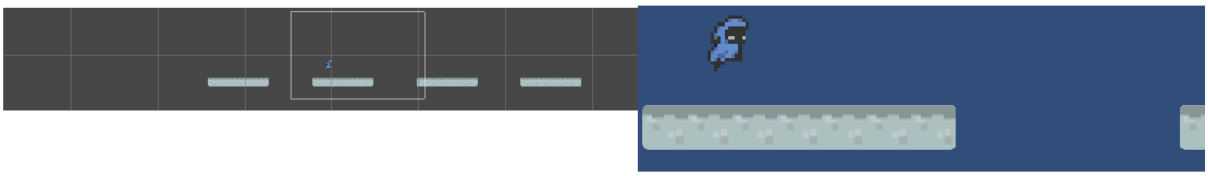


I was able to solve my problem by creating a duplicate of my original platform and dragging my 1st platform away from the scene and placing it as a prefab. After that I created a new folder called 'Prefabs' (reusable object in Unity) where I can use that platform as a reusable asset in my game. Shown on the left.

The platform generation and destruction is working normally now in the game(above).

Start

Current



What is being tested?	Input	Justification	Expected outcome	Actual outcome
The game destroys the platforms after a certain point, when the player has passed it.	No input from the player as it is an in-game object.	Not deleting game objects will result in more memory (RAM) being occupied. Deleting objects conserves memory and	The game continuously deletes platforms behind the player when the player is moving forward.	The game continuously deletes platforms behind the player when the player is moving forward.

		improves performance.		

Test Data: Platform destruction.

Test Data	Expected	Outcome
Platform destroys soon after disappearing from screen.	Valid.	Valid
Platform remains after disappearing from screen.	Invalid.	Invalid.

Random Spaces between platforms

I need to have varying spaces between each platform other the game would feel boring to my stakeholders. Currently my platforms have a uniform space between each other; now that I have gotten platform destroying out of the way, I will need to focus on changing the space between platforms while the player is running.

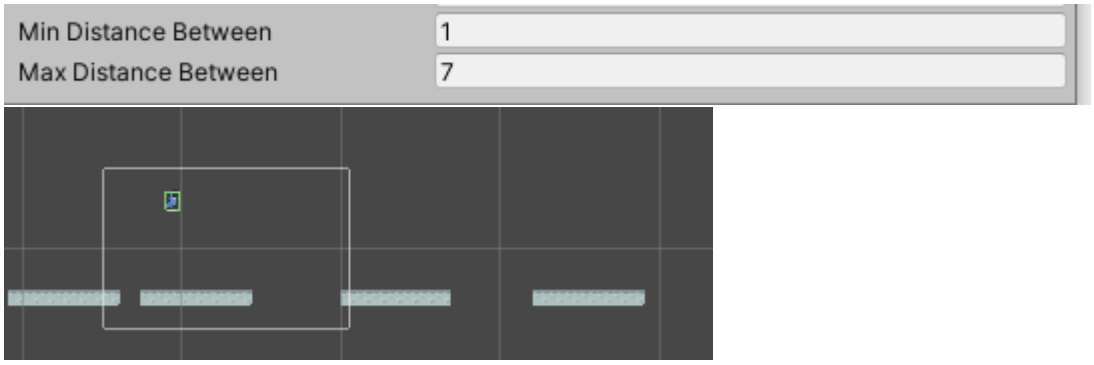
```
public float minDistanceBetween;
public float maxDistanceBetween;
```

In the platform generator script, I set 2 new float variables that I'll be able to tinker with inside Unity. These represent the minimum and maximum possible distance that the platforms will have between each other.

In the 'if' statement I used the 'Random. Range' method for my platform spacing, this means that platforms can have a random space between with a distance ranging from the minimum (e.g., 1) to the maximum (10) distance.

```
if (transform.position.x < generatePoint.position.x)
{
    distanceBetween = Random.Range(minDistanceBetween, maxDistanceBetween);

    transform.position = new Vector3(transform.position.x + widthOfPlatform + distanceBetween, transform.position.y, transform.position.z);
    Instantiate(aPlatform, transform.position, transform.rotation);//Creates a copy of an existing object
}
```

As shown on below, there is no longer a uniform space between each platform. This resulted in my game being more engaging and random which is good for my stakeholders.



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Randomised distance between platforms.	No input from the player as it is an in-game object.	Makes the game more enjoyable to play and less uniform	The game produces gaps of varying size between new platforms.	The game produces gaps of varying size between new platforms.

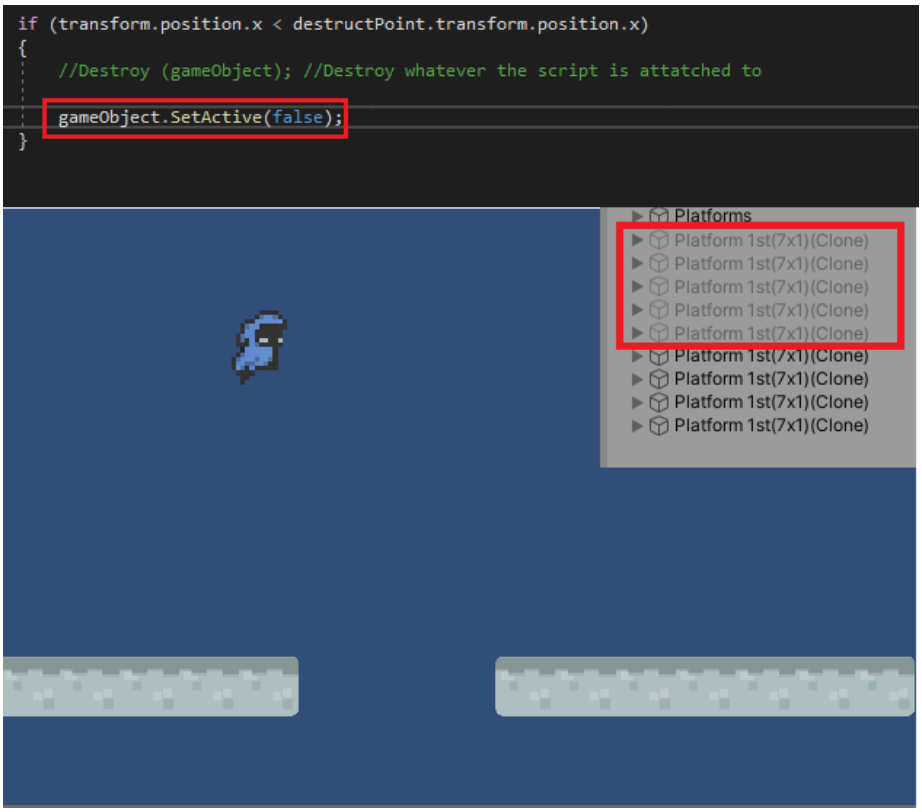
Test Data: Platform space randomization.

Test Data	Expected.	Actual
Platforms obey distance Between platforms values.	Valid.	Valid.
Platforms use random value for spaces.	Invalid.	Invalid.

Date:03/01/2021

Using object pooling

In my analysis' success criteria, I wanted a function which re-uses platforms to relieve performance issues. After some research, I found out that I can use a common method used in games which is called object pooling. By using object pooling, all the objects I will need at any specific moment before playing is pre-instantiated. This saves time because instead of my game creating new objects (e.g., platforms) and destroying previous objects while playing, my game will reuse objects from a "pool". This makes the game run smoother and reduces processing time.

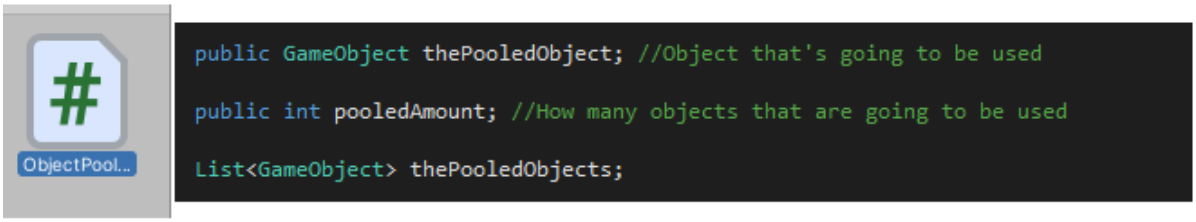


In my platform destroyer script, I changed my 'if' statement so that instead of deleting the objects the game deactivates it.

However, when I tested the game, the objects were still present (though hidden) after they had deactivated. This is shown on the left, the platforms that have been deactivated are greyed out. This is a problem because while the game can re-use the inactive platforms that are available, it chooses to create brand new ones instead. Which isn't efficient object pooling and can cause issues with memory usage and decreases CPU efficiency.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Object Pooling on generating platforms.	No input from the player as it is a method.	Using object pooling in my game reduces processing power required and reduces the amount of RAM needed.	The game reuses game objects that are inactive instead of creating new objects.	The game did not reuse inactive game objects but instead made new ones which reduces CPU efficiency.

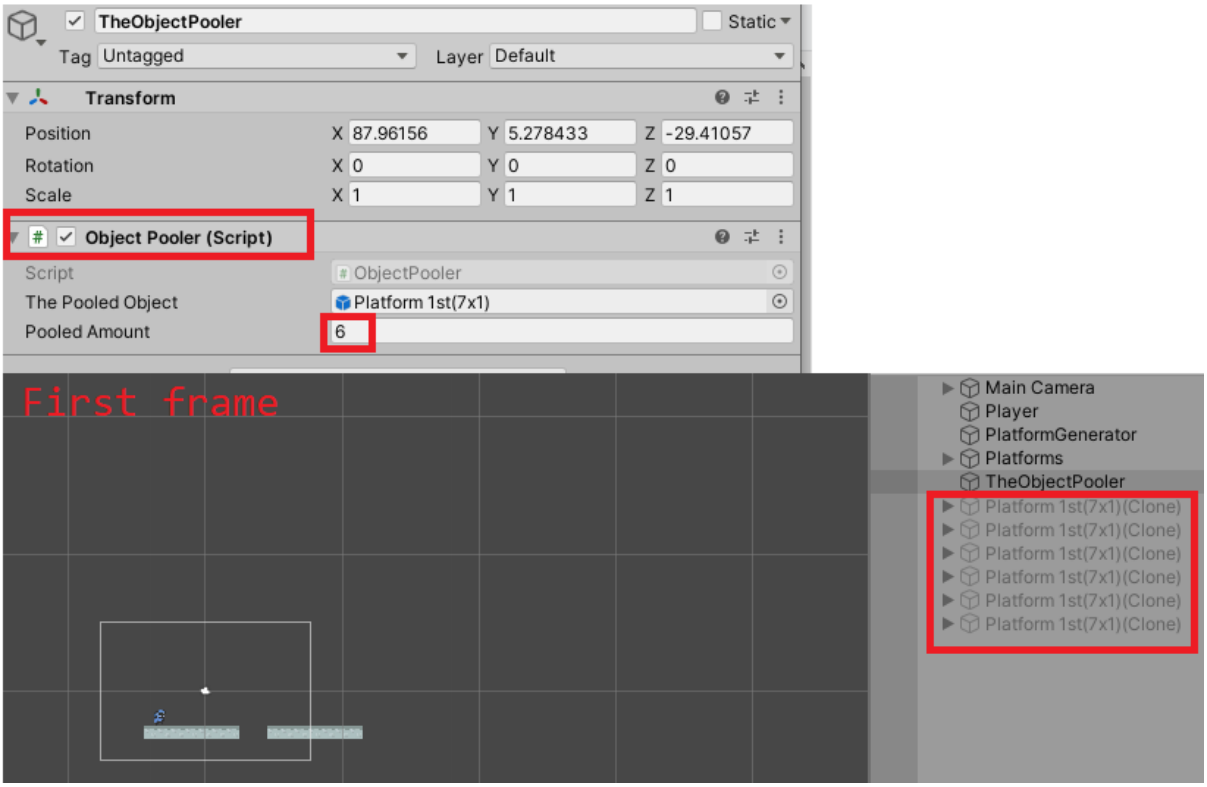
To tackle this, I created a new C# script for my new game object which is called 'object pooler'. In the script, I set up a list for a 'for' loop that I will be using to make the game reuse objects, such as platforms in this case.



At that start of the game, I would like to have some platforms ready beforehand, so in my 'for' statement I set a counter. If the value of 'i' is less than the number of objects I would like at the start, the game pre-instantiates the specific number of objects I'd like.

```
0references
void Start()
{

    thePooledObjects = new List<GameObject>();

    for (int i=0; i<pooledAmount;i++)
    {
        GameObject myObject = (GameObject)Instantiate(thePooledObject); //used gameobject in brackets to cast whateber is behind it,inot a game object
        myObject.SetActive(false); //created new object but don't want  it set active in the scene,deactivated
        thePooledObjects.Add(myObject); //adds one to i
    }

}
```

Before continuing further, I decided to see if the 'for' loop would work. I saved the script and added it as a component to my previously mentioned Object pooler. I inputted the number of platforms I would like to have pre-instantiated and ran the first frame of the game. They were not visible, but they were ready in the game, as shown below:



To keep reusing platforms from my object pool, I created another 'for' loop. If there is an inactive object available, it'll be returned (i.e., visible) in the game. If there is none that are available, the script below the 'if' statement creates a new inactive platform to use. This improves CPU efficiency as the computer does not have to process a new game object every

single time.

```
public GameObject GetPooledObject() //Looking for a game object to get
{
    for (int i = 0; i <thePooledObjects.Count; i++)
    {
        if (!thePooledObjects[i].activeInHierarchy) //Goes to position 0 in the list. activeInHierarchy:is it inactive in the scene at the moment
        {
            return thePooledObjects[i];  //returns an object that's not currently active ,the script won't loop afterwards
                                         //if an object is active, 'id' statement is false
        }
    }
    GameObject myObject = (GameObject)Instantiate(thePooledObject); //Adds a new inactive object to the list
    myObject.SetActive(false);
    thePooledObjects.Add(myObject);
    return myObject; //returns the object that has just been instantiated
}
```

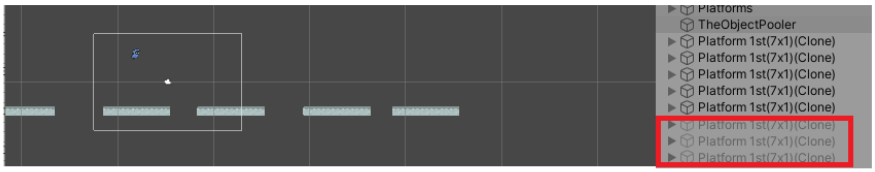
I referenced my Object pooler in my platform generator script, under the name ‘myObjectPool’. The platform copies the position and the alignment of the platform generation point, this means that I will not have platforms that are angled. The platform is set to ‘true’ so that the player can interact with it.

```
public ObjectPooler myObjectPool;
```

```
if (transform.position.x < generatePoint.position.x)
{
    distanceBetween = Random.Range(minDistanceBetween, maxDistanceBetween); //will pick random value for the space between

    transform.position = new Vector3(transform.position.x + widthOfPlatform + distanceBetween, transform.position.y, transform.position.z);
    //Instantiate(aPlatform, transform.position, transform.rotation);//Creates a copy of an existing
    GameObject newPlatform = myObjectPool.GetPooledObject(); // Get an object is one of the available objects that are inisde my object pool

    newPlatform.transform.position = transform.position;

    newPlatform.transform.rotation = transform.rotation;
    newPlatform.SetActive(true);
}
```

As shown below, the game is running with the new script. I set the game to have 8 platforms available at-most for testing. The greyed-out objects are the ones that are not currently being used at the start. A few seconds later, one of the previous inactive platforms has become active while a previous active platform has become inactive during gameplay. This solved my problem of the inefficient object pooling I was facing. This means that processing power is reduced which is a huge benefit for my stakeholders who would want a smooth experience with this game on their computers.

Started playing:



A few seconds later:

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Object Pooling	No input from	Using object	The game	The game

on generating platforms.	the player as it is a method.	pooling in my game reduces processing power required and reduces the amount of RAM needed.	reuses game objects that are inactive instead of creating new objects.	reuses platforms that are not currently active instead of creating a new platform.

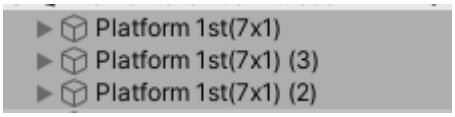
Test Data: Platform re-used by game.

Test Data	Expected	Outcome
Platforms deactivated off screen.	Valid.	Valid.
Platforms are re-used.	Valid.	Valid.
Platforms remain active after being off screen.	Invalid.	Invalid.

04/01/2021

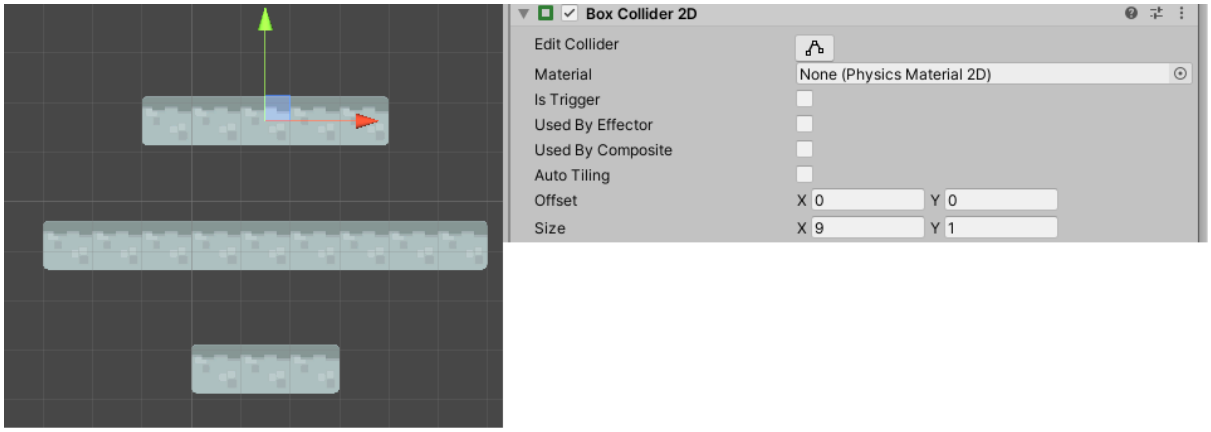
Randomising platform lengths/sizes.

Random platform lengths are a part of my success criteria, in my analysis, to make the game more engaging and to add some difficulty. So, I will be creating platforms of different length and implementing it into my game.



To start, I made duplicates of my original platform and highlighted them. I selected 'unpack prefab completely' which means that if I change a platform, the other platforms are not affected.

After creating the duplicates, I rearranged some blocks of the platform to vary the lengths. After changing the lengths of each platform, I had to change the length of the Box Collider attached to it. If The platform was 9 'blocks' long, the length of the box collider has to be 9 'blocks' long along the x-axis. This is shown below:




```
//public ObjectPooler myObjectPool;
```

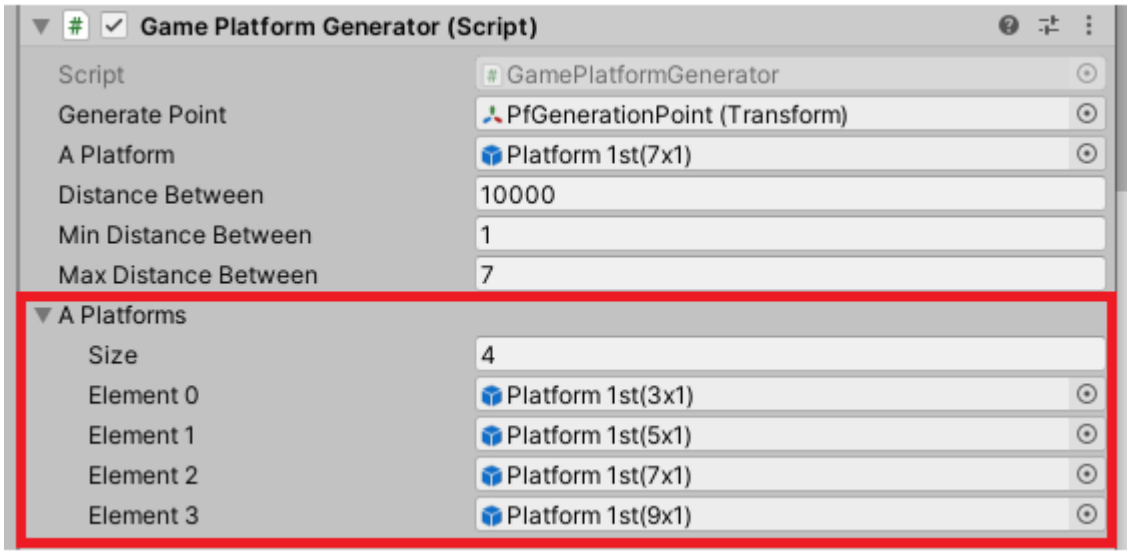
To just test the random platform spawning, I temporarily disabled the object pooling for the time being.

Previously in my platform generator script, I had the game generate a specific platform(the first one I used(7x1)).This time I set an array for the unique platforms I created and the game will randomly select one from the array, to spawn in the game.

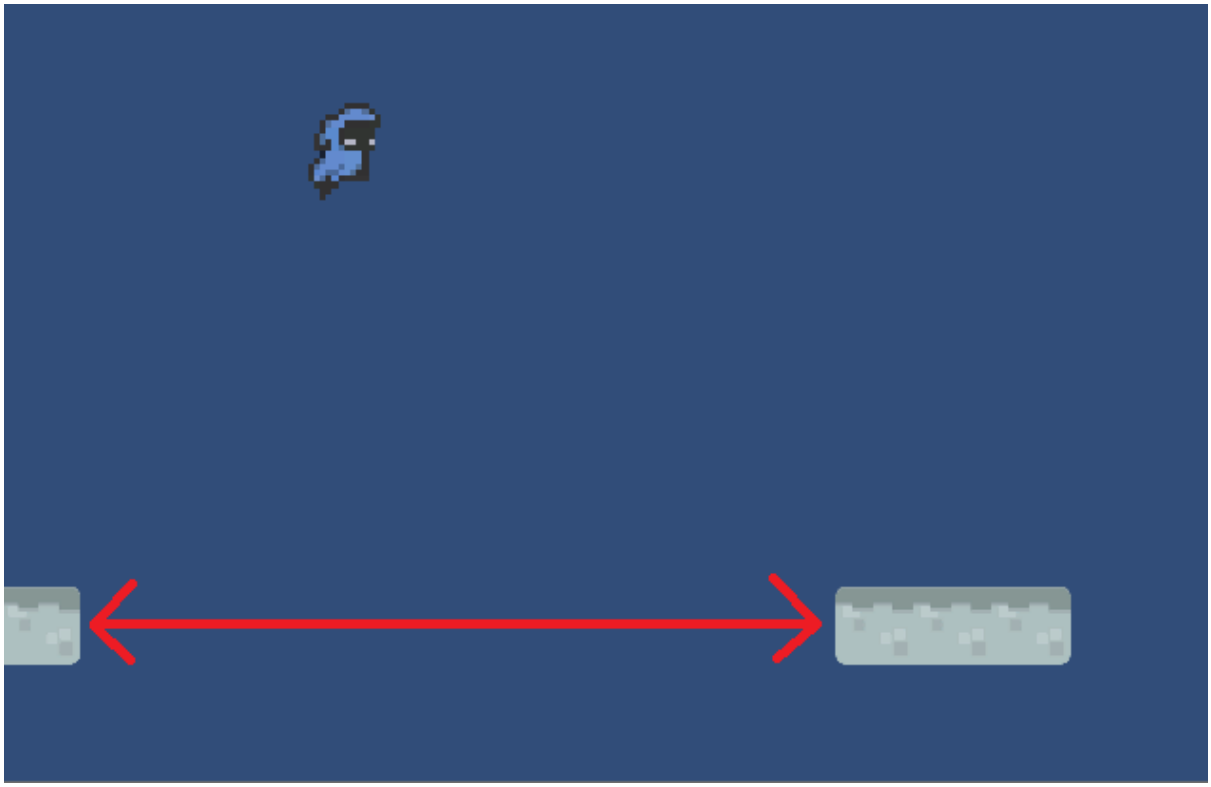
```
public GameObject[] aPlatforms; //Array of platforms
private int thePlatformSelector;
```

```
thePlatformSelector = Random.Range(0, aPlatforms.Length); //Selects a platform between 0 and the length of the array
Instantiate( aPlatforms[thePlatformSelector], transform.position, transform.rotation);//Creates a copy of an existing
```

After that, I added the different platforms in the array I created. Since the array is always indexed at 0, the length of my array goes up to 3 elements. This is shown below:



When running the game, I ran into a problem which was that there were large spaces in between each platform. This could affect gameplay as the player may not be able to successfully jump onto each platform with such large gaps. This is shown below:



Test Data: Random platform sizes.

Test Data	Expected	Outcome
Different sized platforms on screen.	Valid.	Valid.
Different sized platforms together.	Invalid.	Invalid.
Platforms obey space randomization	Valid.	Invalid.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Platforms of varying length are being spawned in the game.	No input from the player as it is a method.	Varied platform length makes the game more engaging to the player and to add some difficulty.	The game spawns platforms of varying length during gameplay.	The game spawns platforms of varying length during gameplay.

Appropriate spaces between Platforms of varying length, that are being spawned in the game.	No input from the player as it is a method.	Without appropriate spaces for the player to jump across, the player loses instantly. This would ruin gameplay.	The game produces gaps of varying size between new platforms being generated, that the player can jump across.	The game produces gaps that may be impossible for the player to jump across.

When looking through my script I realised that the gaps between my platforms included the width of the platform itself. Since my script was only considering the width of the original platform I used, my game was not able to create appropriate gaps based on the new platforms I made earlier.

```
widthOfPlatform = aPlatform.GetComponent<BoxCollider2D>().size.x;
```

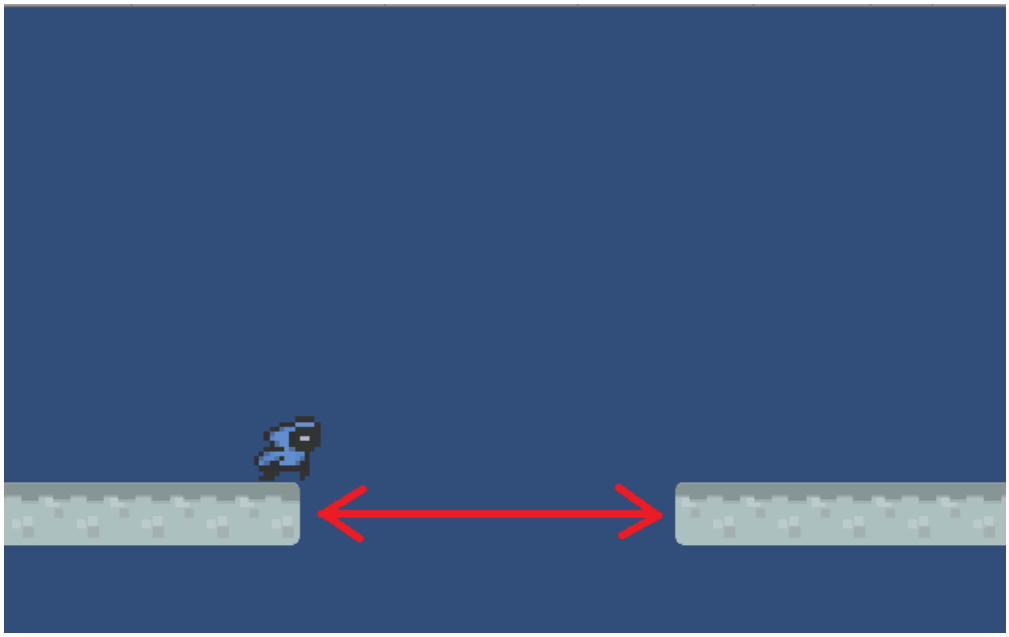
Instead of referencing one platform individually on each line, I decided to use a 'for' loop. This 'for' loop will get the x-value of the platform that has been randomly selected from the array. The x-value(length) retrieved from that particular element (e.g. element 1) is equal to the box collider of that element. This means that if a small platform is spawned, it will not have the player will not detect it as a larger platform. This prevents the player from walking on air. This statement is used to calculate the appropriate platform width:

```
widthOfPlatforms = new float[aPlatforms.Length]; //creates an array of elements from 0 to 3
for (int i =0;i<aPlatforms.Length;i++)
{
    widthOfPlatforms[i]=aPlatforms[i].GetComponent<BoxCollider2D>().size.x; //Gets the xValue of the platform selecte
}
```

I decided to use an 'if' I statement. This statement will get the x-value of the platform that has been randomly selected from the array. This value can be used to calculate the appropriate platform gap.

```
if (transform.position.x < generatePoint.position.x)
{
    distanceBetween = Random.Range(minDistanceBetween, maxDistanceBetween); //will pick random value for the space between

    thePlatformSelector = Random.Range(0, aPlatforms.Length); //Selects a platform between 0 and the length of the array

    transform.position = new Vector3(transform.position.x + widthOfPlatforms[thePlatformSelector] + distanceBetween, transform.position.y, transform.position.z); //Calculates width of platform

    Instantiate( aPlatforms[thePlatformSelector], transform.position, transform.rotation);//Creates a copy of an existing
```

When running the game, the gap problem was resolved, now there are no gaps that are extremely large where players cannot jump across. This makes gameplay feel more consistent as there are very little flaws that the player can see here, at this current point. This is shown below:



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Appropriate spaces between Platforms of varying length, that are being spawned in the game.	No input from the player as it is a method.	Without appropriate spaces for the player to jump across, the player loses instantly. This would ruin gameplay.	The game produces gaps of varying size between new platforms being generated, that the player can jump across.	The game produces gaps of varying size between new platforms being generated, that the player can jump across.

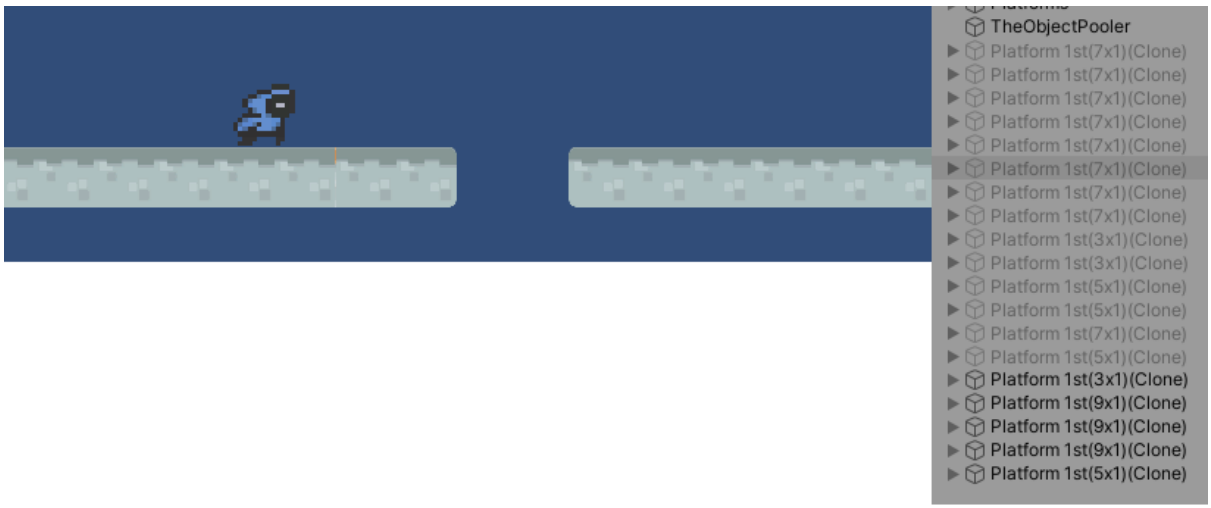
Test Data: Random platform sizes.

Test Data	Expected	Outcome
Different sized platforms on screen.	Valid.	Valid.
Different sized platforms together.	Invalid.	Invalid.
Platforms obey space randomization	Valid.	Valid.

Integrating object pooling again

Previously I deactivated object pooling, so that I can practice randomising platforms. Now that I have got platforms of random length implemented, I'll need to add object pooling in again so that the game reduced the required processing power of the CPU. Object pooling makes my game more efficient.

Due to deactivating the object pooling system, my game currently doesn't reuse inactive objects. Instead, it instantiates new objects, which is inefficient. This is shown below:



Setting up platform randomiser to work with the object pooling system:

Since I am using various unique platforms form an object pool my instantiate statement won't work in this case of spawning a new object, due to it being only assigned to one length of platform(7x1). I commented-out the instantiate statement. This is shown below:

```
//Instantiate( aPlatforms[thePlatformSelector], transform.position, transform.rotation);
```

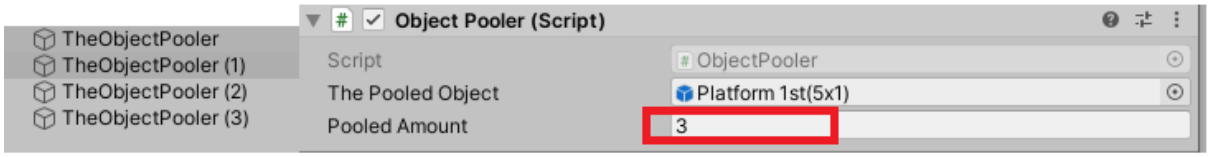
Previously I had my object pooling system work with one specific type of platform(7x1) but since I am using an array of new unique platforms, I had to create an array for my object pools.

```
public ObjectPooler[] myObjectPools;
```

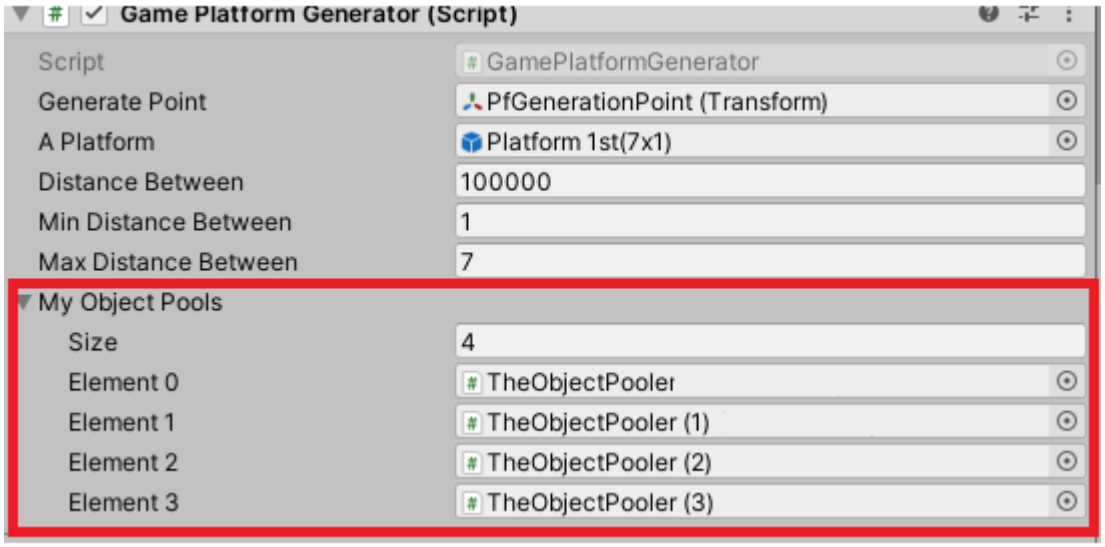
After I had created that, I replaced my mentions of my old array of platforms, that consisted of the (7x1) length platform, with the new array. When a new object is instantiated in this case, it looks for an available object in my platform object pool. This means that the game will not run into an error looking for a platform. The code is shown below:

```
if (transform.position.x < generatePoint.position.x)
{
    distanceBetween = Random.Range(minDistanceBetween, maxDistanceBetween); //will pick random value for the space between

    thePlatformSelector = Random.Range(0, myObjectPools.Length); //Selects a platform between 0 and the length of the array

    transform.position = new Vector3(transform.position.x + widthOfPlatforms[thePlatformSelector] + distanceBetween, transform.position.y, transform.position.z); //Calcul

    //Instantiate( aPlatforms[thePlatformSelector], transform.position, transform.rotation);//Creates a copy of an existing


    GameObject newPlatform = myObjectPools[thePlatformSelector].GetPooledObject(); //Knows where to look in the object pool for a pooled object
                                                                                   // Get an object is one of the available objects that are inisde my object pool
    newPlatform.transform.position = transform.position;

    newPlatform.transform.rotation = transform.rotation;
    newPlatform.SetActive(true);
```

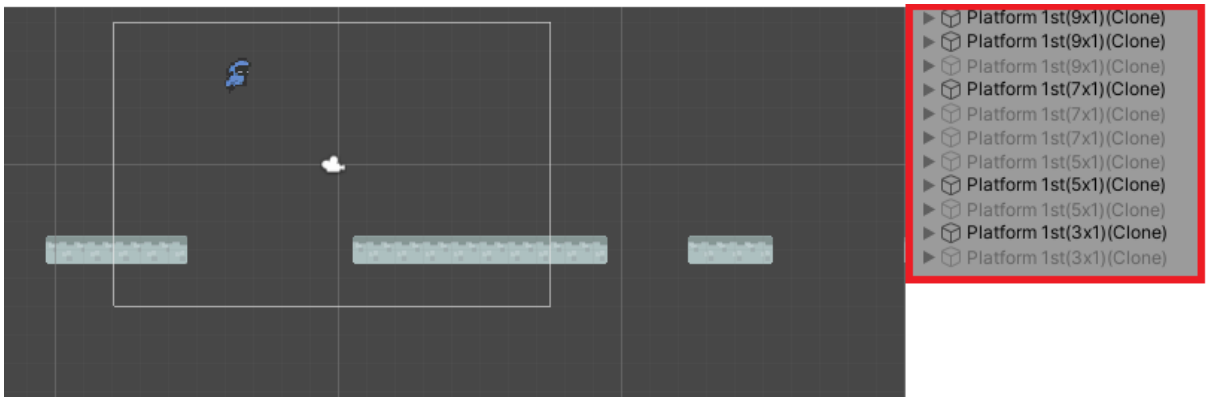
I created 4 new game objects and assigned each one to a specific platform. The pooled amount is how many of that specific platform will be pre-instantiated when the game begins. I kept it as 3 because the player won't be travelling so fast that the game won't be able to spawn new platforms. This is shown below:



After that I added each object to my platform generator script.This is shown below:



When I tested the game, I noticed the game was using the object pool for the platforms again, as shown below. Various Platforms were spawning at a consistent rate which I am sure will enhance the gameplay for my stakeholders.



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Object Pooling on generating platforms of varying lengths.	No input from the player as it is a method.	Using object pooling in my game reduces processing power required and reduces the amount of RAM needed.	The game reuses game objects that are inactive instead of creating new objects.	The game reuses various types of platforms that are not currently active instead of creating a new platform.

05/01/2021

Randomising Platform height

To make better use of the jump feature and to make the game less boring, I'll be including a feature where platform heights are randomised to be placed higher and lower from the default position. To do this efficiently, I'll be implementing object pooling into this.

To start with, I do not want my platform being spawned extremely high or extremely low during the game, so I set some float variables in the platform generation script. Much like my platform generation point and platform destruction point, my 'point of Zenith' is the max height which will be set using a game object. I'm planning to set my lowest spawn height as the same default height of the regular platforms that I've been using so far.

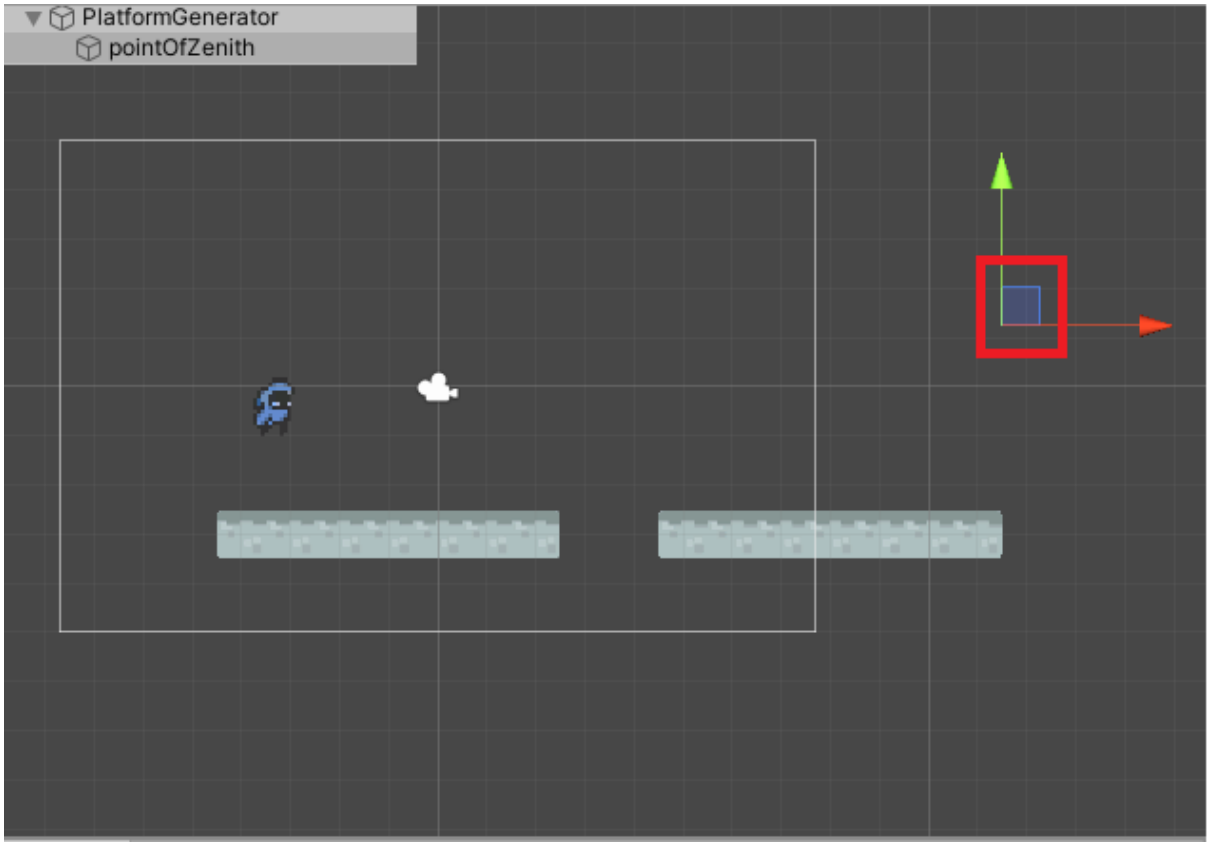
```
private float minimumHeight; //default height of platform
private float maximumHeight;

public Transform pointOfZenith; //Moves the platforms above the default position
private float heightDifference; //Max range for platform movement
public float maxHeightDifference;
```

I changed my default y-position for my platforms from being static to being random from using the 'Random. Range' method when they are being positioned after spawn. This is shown below:

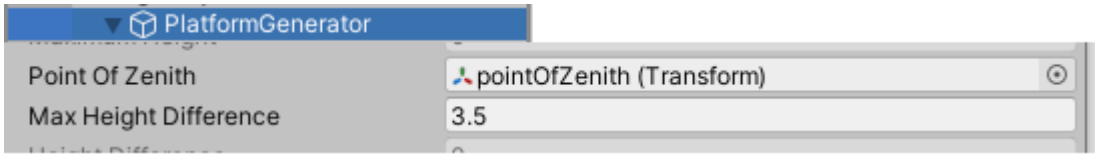
```
heightDifference = transform.position.y + Random.Range(maxHeightDifference,-maxHeightDifference);

transform.position = new Vector3(transform.position.x + (widthOfPlatforms[thePlatformSelector] / 2) + distanceBetween, heightDifference, transform.position.z);
```

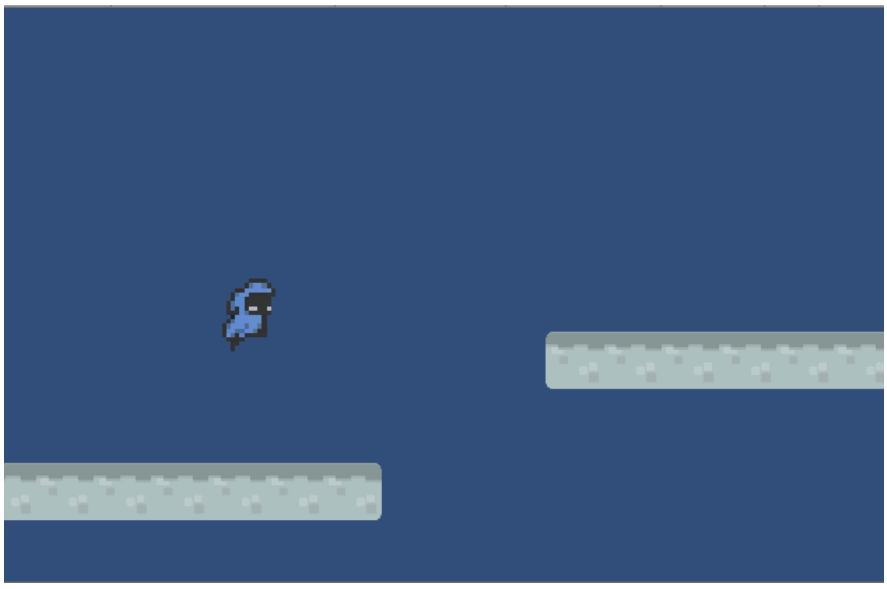
After I referred to the transform object in the generator script, I created a game object in unity which would represent the 'pointOfZenith'. Since this is being used by the platform generator, I set it as a child(underneath) of the generator and positioned it in the game. The red box highlights the position of the 'pointOfZenith' which represents the maximum height

of which a platform can spawn.



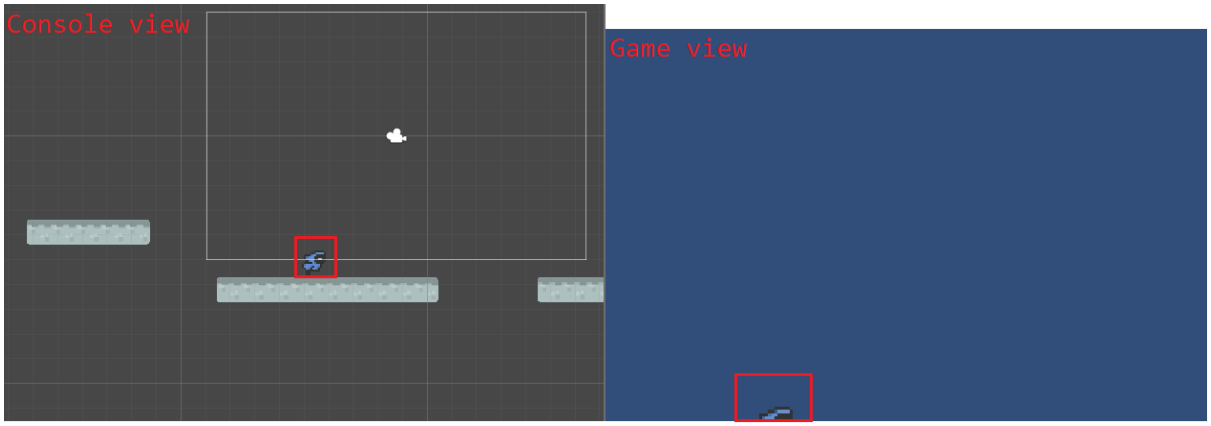
Here I attached the game object (from above) to my platform generator and set a height of 3.5 on the y-axis from the max spawn height.





When I tested the game, the height of each platform was spawned randomly which was a good start. However, I ran into a problem where platform was spawning below the default height (i.e., the minimum height) which can cause issues in the game as the

player cannot see where the platforms are. This is shown below:



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Platforms being generated at varying heights.	No input from the player as it is a method.	Adds a new interesting dynamic to the game and makes gameplay more enjoyable.	Platforms are generated at varying heights where the player can see and reach them.	Platforms are generated at varying heights but may spawn where the player cannot see them.

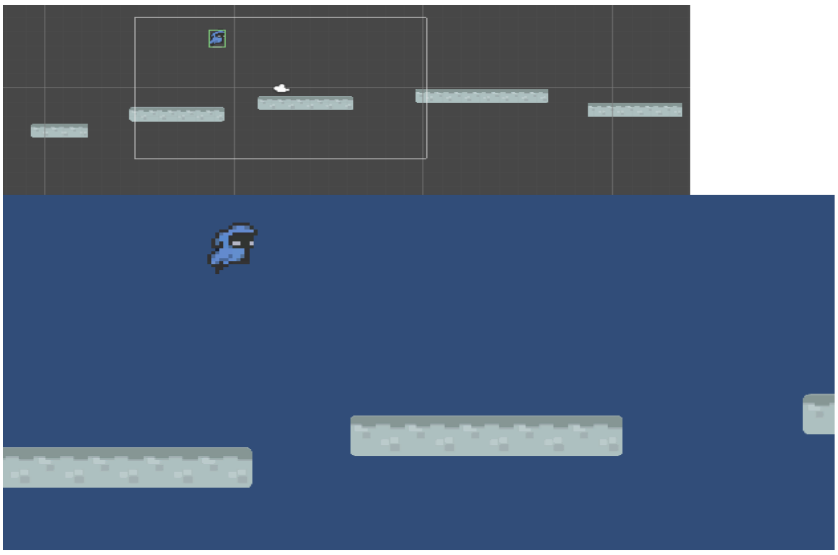
Test Data: Random platform height.

Test Data	Expected	Outcome
Platforms are loaded at different heights.	Valid.	Valid.
Platforms displayed at different heights on-screen.	Valid.	Valid.
Platforms displayed off-screen.	Invalid.	Valid.

To fix this, I set 2 if statements so that if a platform's height is randomly selected to be beyond the limits of the maximum or minimum height, they are automatically set to have

the highest or lowest position possible. This is shown below:

```
if (heightDifference > maximumHeight)//If above max height,it will be set as max height
{
    heightDifference = maximumHeight;
}
else if (heightDifference < minimumHeight)
{
    heightDifference = minimumHeight;
```



When running the game, no platforms were spawned outside the boundaries of the camera. This means that the player will not run in to any problems of not being able to see where they are going.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Platforms being generated at varying heights.	No input from the player as it is a method.	Adds a new interesting dynamic to the game and makes gameplay more enjoyable.	Platforms are generated at varying heights where the player can see and reach them.	Platforms are generated at varying heights where the player can see and reach them.

Test Data: Random platform height.

Test Data	Expected	Outcome
Platforms are loaded at different heights.	Valid.	Valid.
Platforms displayed at different heights on-screen.	Valid.	Valid.
Platforms displayed off-screen.	Invalid.	Invalid.

When running the game, I noticed that sometimes I overtook platforms by accident with a normal jump. This could make the controls seem too sensitive to the stakeholders, so I am planning on varying the height of the jump based on how long the key is pressed.

Player movement continuation.

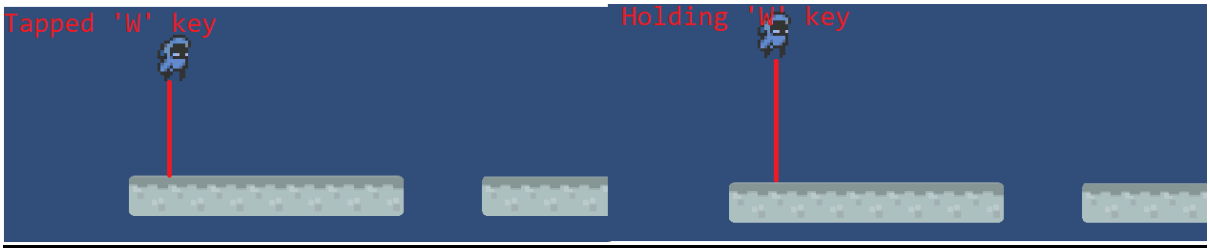
If the player holds a key down for a specific duration, they will continue jumping further. Therefore, I set 2 float variables. One times how long the key was pressed, the 'jumpDurationChecker' resets the 'jumpDuration' variable. I did this so that the player will be able to perform more than one prolonged jump.

```
public float jumpDuration;
private float jumpDurationChecker; //Resets duration of a jump after each jump
```

This 'if' statement makes the player go continually higher until the keystroke has reached the time limit of the jump duration.

```
if (Input.GetKey(KeyCode.W))
{
    if (jumpDurationChecker > 0)
    {
        playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, forceOfJump);
        jumpDurationChecker -= Time.deltaTime; //As the player holds down the button up to a
    }
}
```

After setting a value to the jump duration, I tested the game. When holding the 'W' key down, my player jumps higher than lightly tapping it. This is shown below:



To eliminate the possibility of double jump I made these 'if' statements. When the time to hold a key down is up, the player won't be able to jump again until they come into contact with the ground. This is shown below:

```
if (Input.GetKeyUp(KeyCode.W))
{
    jumpDurationChecker = 0;
}

if (onGround)
{
    jumpDurationChecker = jumpDuration;
}
```

I implemented this feature as to prevent players from breaking the game by jumping too high.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Jump height varies on how long the button is pressed.	The 'W' key for jumping.	Making the jump height dependant on how long the key is pressed, allows the player to have more control over the player sprite.	After a certain time has elapsed when the player has held the key, the player goes higher during their jump.	After a certain time has elapsed when the player has held the key, the player goes higher during their jump.
Absence of double jump.	The 'W' key for jumping.	I do not want to give the player the ability to spam jump after performing an extended jump.	The player does not have the ability to spam jump after performing an extended jump.	The player does not have the ability to spam jump after performing an extended jump.

06/01/2021

Difficult curve

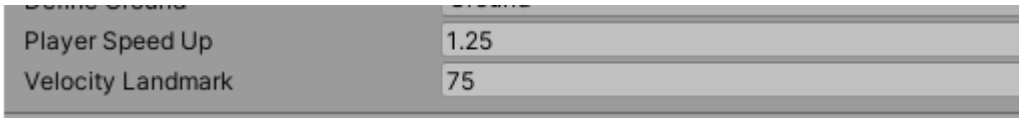
While playing the game, I want my stakeholders to be more engaged and not get bored overtime, therefore I decided, during my analysis stage, to add a difficulty curve into the game. This difficulty curve will cover one of my success criteria and will depend on how good the player is at surviving the endless platformer. The longer the player stays on, the faster the player will run which in turn will cause the difficulty to increase over time.

To start, I set 3 float variables in my player control script which will be used to increase the player's speed by a set amount. In the update void (What C# uses to update the game frame-by frame for animations, movement, platform generation etc.) I set an 'if statement so that if the player reaches a distance I have set, the player's speed will increase by a certain amount('playerSpeedUp). The set distance will also increase if the player has passed it once, this is to make sure that the player is speeding up over time and not just once.

```
public float playerSpeedUp;
public float velocityLandmark;
private float velocityLandmarkCounter;
```

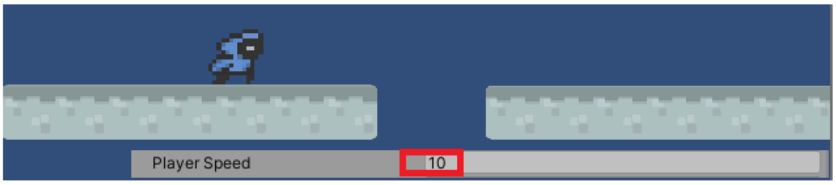
```
if (transform.position.x > velocityLandmarkCounter)
{
    velocityLandmarkCounter += velocityLandmark; //Continously adds a set distance for where the player will speed up

    playerSpeed = playerSpeed * playerSpeedUp;
}
```

After saving the script, I set values in the game for the landmarks the player must pass and the factor of which the speed will increase by. The player's speed will increase by 25% for this test and the landmark will be 75 on the x-axis.

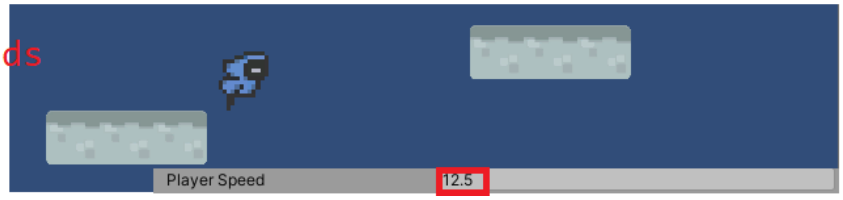


I started the game and made the player move for 5-10 seconds, I eventually passed the landmark (75 on the x-axis) and my player sped up. The player won't speed up by 25% after every landmark because it causes a sudden spike in difficult and may seem jarring to my stakeholders, I'm using this value to easily notice if the speed-up mechanic is working. In which this case, it is:

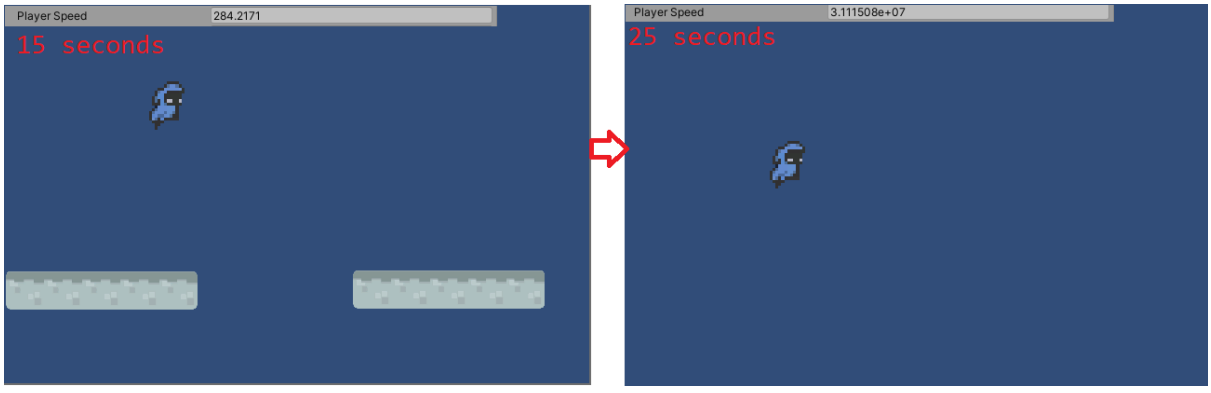
Start of game:



After a few seconds of playing:



After playing with the game for a bit, I quickly noticed a problem which was that my player sped up exponentially after the first speed-up. This is because that after each time the player's speed increased, they reached the landmark quicker each time. This caused the player to move way too fast to the point where the game can't spawn enough platforms for the player. This is shown below:



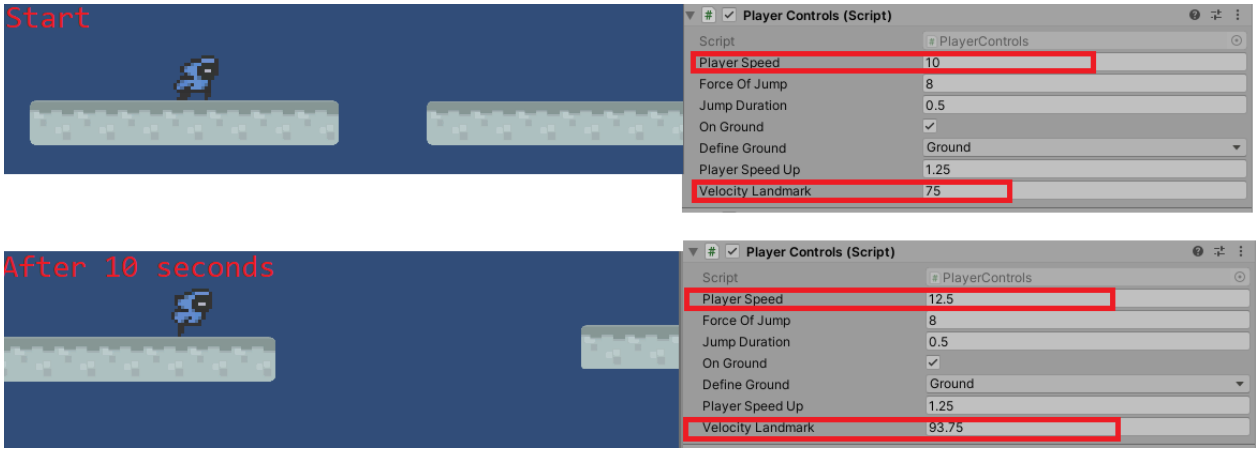
What is being tested?	Input	Justification	Expected outcome	Actual outcome
The player speeding up	The 'D' key as speeding up	Adds an element of	The player's sprite subtly	The player's sprite goes

over time.	depends on the player moving forward.	difficulty in the game which keeps the player engaged.	goes faster over time if the player is alive.	quite fast over a short period of time, which makes the game too difficult after 15 seconds of playing.

To make sure that the player's speed does not increase dramatically, the landmark will increase by the same factor that the player's speed will increase by. This will occur at the same time which makes gameplay more consistent and less speed issues. The line of code below shows that the landmark will increase by the same factor of the player's speed:

```
velocityLandmark = velocityLandmark * playerSpeedUp;
```

When running the game, the player's speed, and the landmark in which the player has to pass increased at the same time. This fixed the sudden increase in difficulty and makes the game more enjoyable and less tedious.

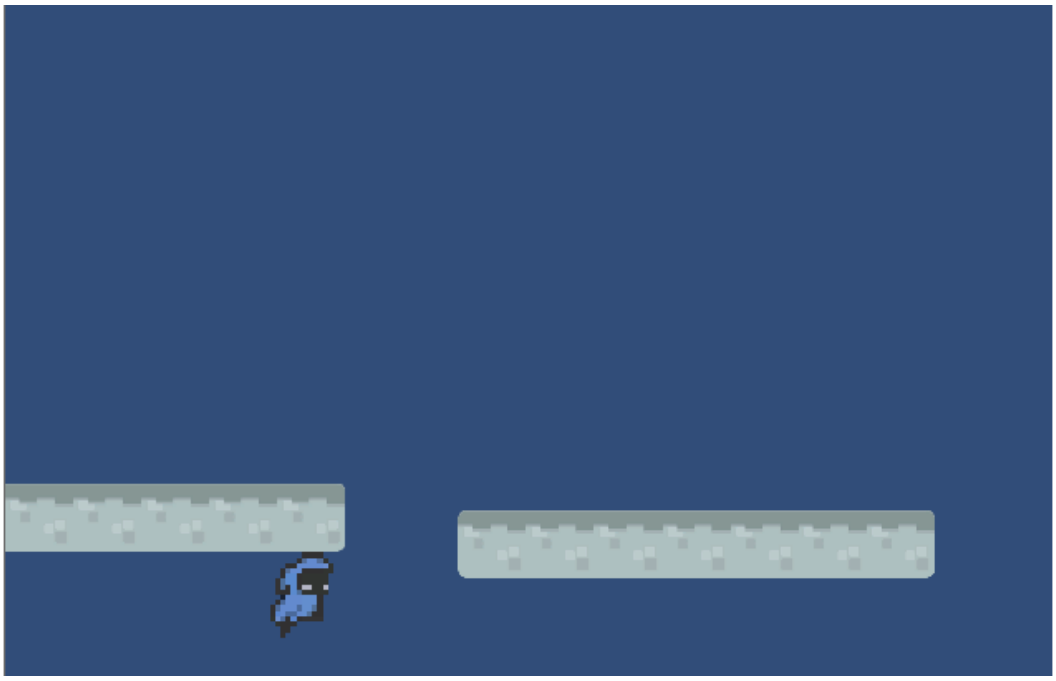


What is being tested?	Input	Justification	Expected outcome	Actual outcome
The player speeding up over time.	The 'D' key as speeding up depends on the player moving forward.	Adds an element of difficulty in the game which keeps the player engaged.	The player's sprite subtly goes faster over time if the player is alive.	The player's sprite subtly goes faster over time if the player is alive.

Test Data: Player speeding up.

Test Data	Expected	Outcome
Game speeds up gradually with time.	Valid.	Valid.
Game speeds quickly over a short time frame.	Invalid.	Invalid.

While I was playing the game, I noticed that my player was slid under a platform once so I tried going underneath the platforms and discovered that players can travel under the platforms. This could prevent the player from losing which then means that the difficulty is lost. This is shown below:



To fix this, I set 2 new float variables in the player controls script which will be used to determine if the bottom of the player sprite is touching the ground.

```
public Transform groundCollisionChecker;
public float groundCollisionCheckerRadius;
```

My game will recognise the player if the bottom centre of the player sprite is in contact with the platform. Physics between the platform and the player will only act if the bottom centre of the sprite is in contact with the platform. This means that the player can't slide under the platform anymore as the sprite's top is touching the platform instead of the bottom.

```
onGround = Physics2D.OverlapCircle(groundCollisionChecker.position,groundCollisionCheckerRadius, defineGround);
```

When running the game, this has resulted in the player not being able to use the previous exploit. I fixed this as there will be ways for the player to lose, such as falling of the platform

or encountering an enemy.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Whether the player can continue travelling by sliding under the platforms.	The 'W' key(jump) or 'D' key (running forward)	If the player can continue travelling by sliding under the platforms, they will be able to cheat the game and never fall to death.	The player will not be able to slide under the platforms if they fall off.	The player is not able to slide under the platforms if they fall off.

07/01/2021

Re-evaluating gameplay

In my analysis and design, I mentioned a 2-minute countdown timer that would be present in the game. Once this timer reached 0, the game would end with the player winning. I introduced this as a way for computers to avoid memory issues. However, with the use of the object destroyer and object pooling, memory will be used efficiently which nullifies the use of the timer. The speeding up mechanism in the game adds an increase in difficulty, having a timer suddenly stop the player during gameplay would feel jarring so I made the decision to remove the timer aspect as there is no use for it. This also means that there is not really a 'win' or 'lose' outcome here as the player may inevitably survive to the point where they are going too fast to react to anything. The scoring system will be used to show the player a measure of their skill.

11/01/2021

The player losing and restarting.

When the player has fell or has come across an obstacle such as an enemy cube, mentioned in my analysis and design, the player will "die". This is a requirement that I have set in my success criteria during my analysis. While playing the game I realised that having the player

see the game over menu and then go back to the title screen every time they lose, will quickly become annoying to them. For now, I will implement a system where the player restarts after death and adding a game-over menu later in development.

In my scripts folder, I created a new script which I will use to program the game restarting when the player dies. The new script I created is named 'GameManager'. GameManager is a feature in Unity which keeps track of what state the game is in and manages the main menu as well as pause menus. This explains why the script has a different Icon compared to the other scripts.

<u>Resetting player and platform generator back to the start after death</u>

The platform and the player (as well as the camera) must return to their default positions, like how they were at the start of the game, after the player has died. This is needed otherwise the platform destroyer will be well ahead of the player instead of behind the player which may result in the player having no platforms to stand on. That can cause an infinite death loop.

```
public Transform platformGenerator;
private Vector3 pgSpawnPoint;
public PlayerControls myPlayer;
private Vector3 playerSpawnPoint;
```

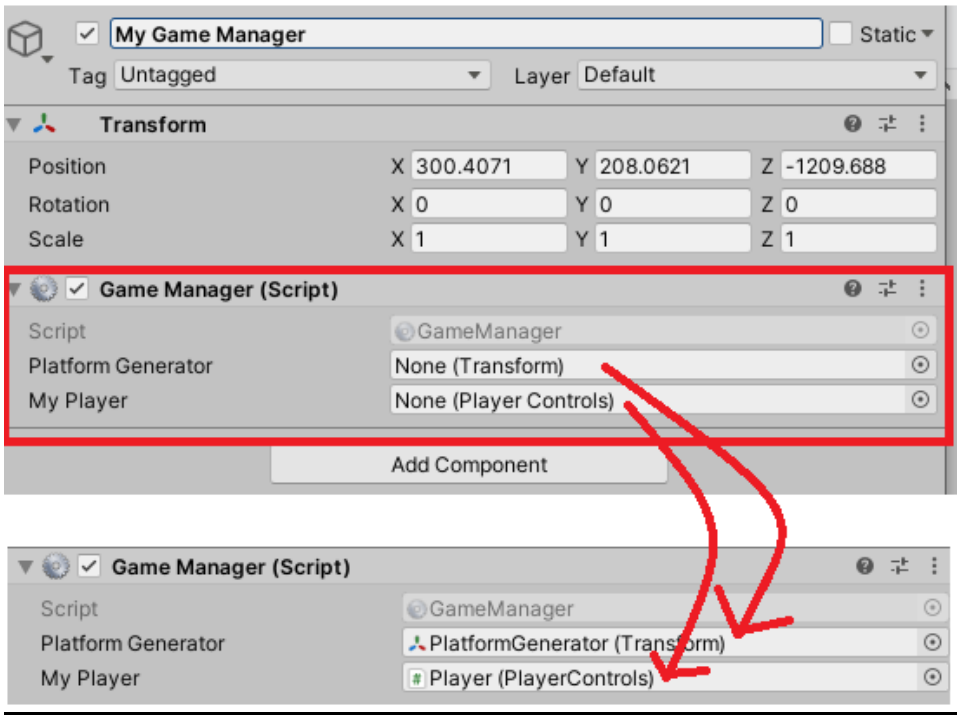
In my game manager script, I created 2 vectors which will have the coordinates of the original place where the player and the platform will spawn. I created 2 places to store my coordinates of the spawn points, one named 'platformGenerator' and the other one named 'my Player'.

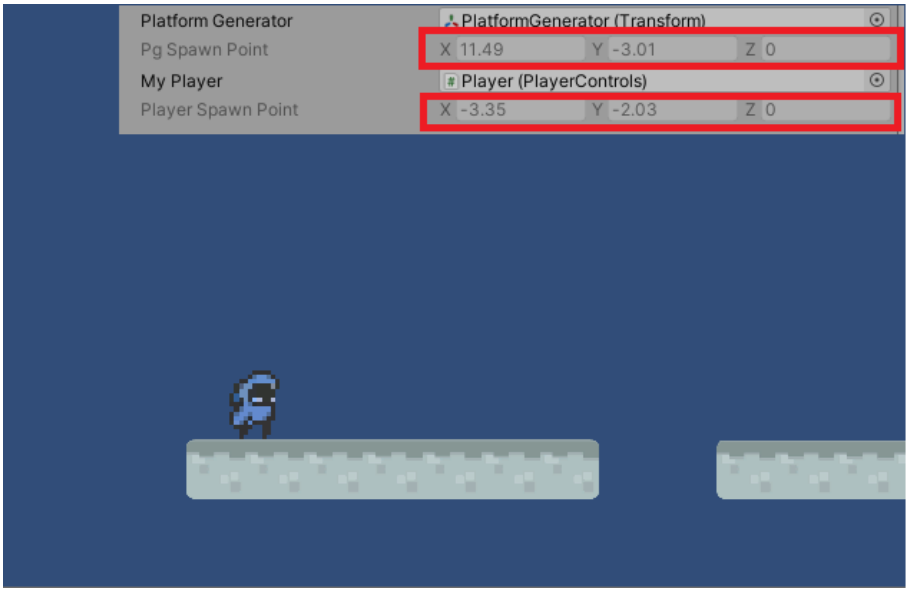
```
pgSpawnPoint = platformGenerator.position;
playerSpawnPoint = myPlayer.transform.position;
```

This statement is used to create the spawn points that will be saved.

I created a new game object to attach my new script to.

In my game manager object, I attached the new script and in the red box, I placed my player game object into 'My player' and dragged the Platformgenerator into 'Platform generator'. Dragging those game objects into the boxes means that my two variables will be able to read the coordinates of the player and the platform generator and store them. This is shown below:





When I was testing the game, I checked my 2 variables, and they displayed the coordinates where the platform and player starts from and does not change when the player is moving. These 2 sets of coordinates in the red box will be referred to when the game restarts.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The player's	No input from	This is to act as	The 2 variables	The 2 variables

coordinates and the platform generator's coordinates being read and saved.	the player.	the restart point for whenever the player dies. Otherwise, the player will just infinitely die with no platforms to land on.	show the coordinates of the player and the platform.	show the coordinates of the player and the platform and do not change.

Shown below, is a new function that I have created and is used to restart the game. The function is made public so I can reference it in the future. I have referenced a coroutine that will be used to restart the game. I decided to use a coroutine as It's a special of function used in Unity to stop the execution until a certain condition, that I have set, is met, and continues from where the player's spawn coordinates are.

```
public void RestartGame() //can be called form another script
{
    StartCoroutine ("RestartGameCo");
}
```

For my coroutine I have to use 'IEnumerator' as it is a requirement to use in C#'s .NET language for coroutines. This is shown below:

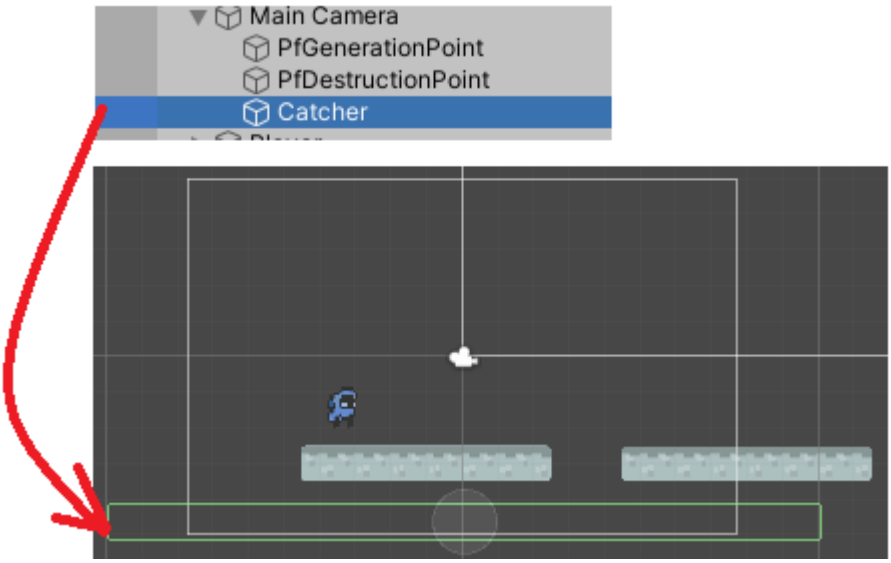
```
public IEnumerator RestartGameCo()
```

In the image below, the 'WaitForSeconds' method is a command that I will use which will wait a certain amount of time before activating a coroutine. In this case it's restarting the game. In the red box, the player's and platform generator's last position will be set back to the spawn coordinate values. This means that the game will set the player back to the start after death.

```
yield return new WaitForSeconds(1f);
myPlayer.transform.position = playerSpawnPoint;
platformGenerator.position = pgSpawnPoint; //sets the position back to the start
```

In the highlighted red boxes of the image below, I set where the player's sprite becomes active or inactive. When the coroutine starts (just after the player dies) the player game object will deactivate (the sprite disappears) and at the end of the coroutine the player game object will activate(Sprite appears).The end of the coroutine is when the player is spawned back at the beginning.

```
myPlayer.gameObject.SetActive(false);
yield return new WaitForSeconds(1f);
myPlayer.transform.position = playerSpawnPoint;
platformGenerator.position = pgSpawnPoint; //sets the position back to the start
myPlayer.gameObject.SetActive(true);
```



Under my main camera I created an invisible platform called ‘catcher’ which follows the camera and catches the player if they fall off the platform. This platform can be used as a zone where if the player game object comes into contact with it (i.e. they fall off), they ‘die’. This makes straightforward to implement the restarting mechanism.

```
public GameManager theGameManager;
```

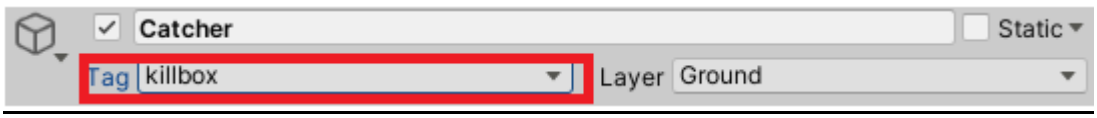
I referenced my GameManager, under a variable, script in my player control script as the player game object is involved in this process of restarting.

```
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "killbox")
        {
            theGameManager.RestartGame();
        }
}
```

In my player control script I made a new function as restarting the game, temporarily changes the game state. I’m going to add a tag called ‘killbox’ to my ‘catcher’ game object so that when my player comes into contact with an object with that specific tag, the game will call this function.

In my function, I referenced the game manager script in my if statement. This game manager will call the’.RestartGame’ routine which will restart the game if the ‘catcher’ game object and the ‘player’ game object collide.

In my ‘Catcher’ game object, I created a new tag called ‘killbox’ and added that tag to the platform. I’m using a tag as it allows the game object, that contains the ‘killbox’ tag, to be identified by the script during the game.



When testing the game, I tested the restart function by falling off a platform. The game brought me at the spawn point, however no platforms were being spawned. This meant

that the player kept falling off and dying, leading to an infinite death cycle. On top of that, more platforms are spawned on top of each other after every death. This is shown below:

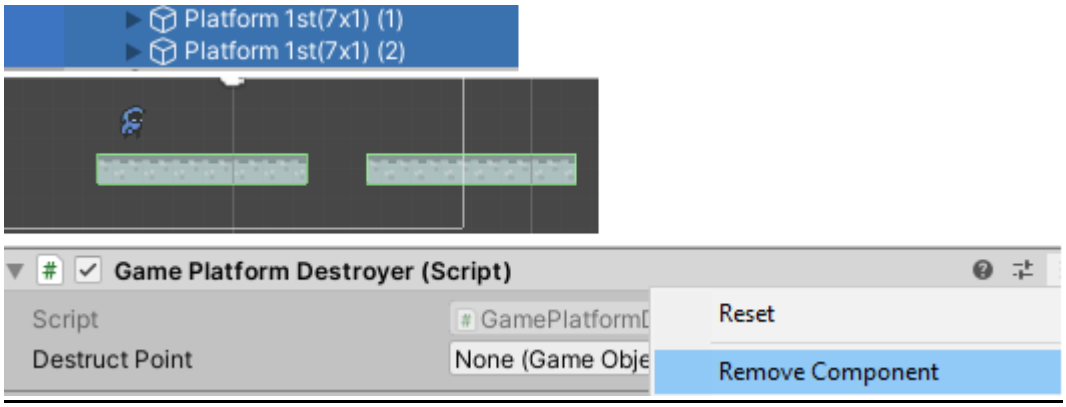
What is being tested?	Input	Justification	Expected outcome	Actual outcome
The game restarts after the player dies.	No input from the player.	The player does not need to navigate the game over menu and then the title screens every time they lose. Reduces annoyance.	The game restarts after the player have fallen off a platform.	The game does not restart properly. Instead has an infinite death loop and platforms spawning on top of each other.

Test Data: Player restarts after death.

Test Data	Expected	Outcome
Player is set at beginning after reset.	Valid.	Valid.

Player is spawned on platforms.	Valid.	Invalid.

My first 2 default platforms are already stored in my game, so they do not need for to be affected by the platform destroyer. This should mean that when I test the game, the platforms should not be destroyed, and that the player will respawn onto those platforms. Image is shown below:



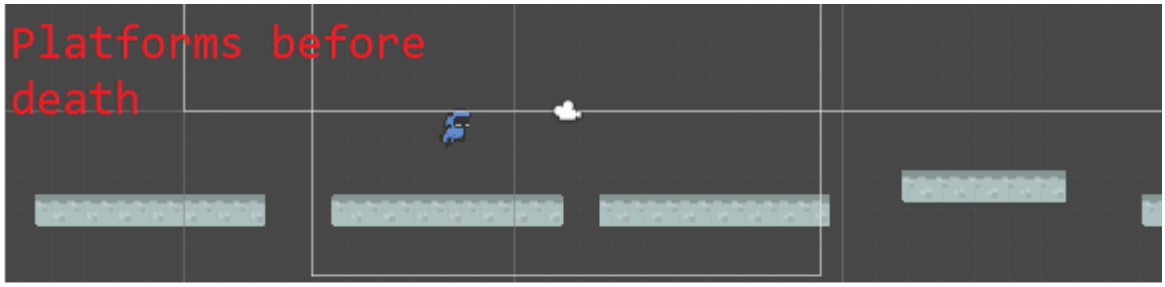

```
private GamePlatformDestroyer[] myPlatformList;
```

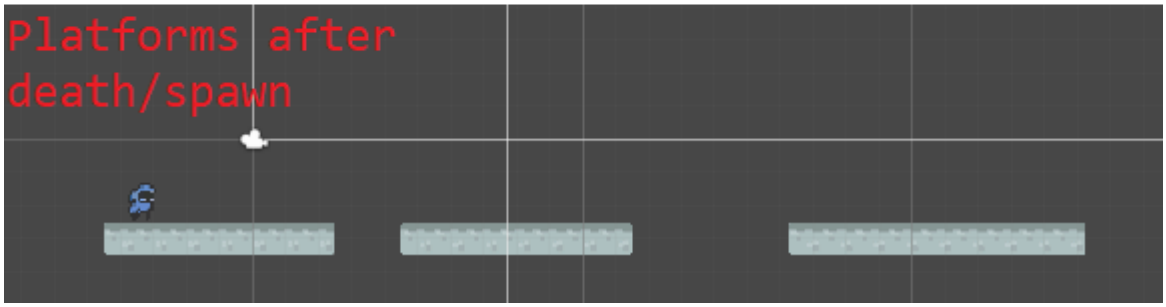
To prevent the platforms from stacking on top of one another, I created an array (called 'myPlatformList')which will highlight all the current active platforms in the game.

I made my active platform array, 'myPlatformList', go through all the active platforms in the game. In the 'for' loop, the game goes through every active platform and deactivates it (set to false). This means that when the player respawns, the previous platforms will not be present and new platforms will spawn. This should prevent platforms spawning on top of each other. This is shown in the image below:

```
myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
for (int i = 0; i < myPlatformList.Length; i++)
{
    myPlatformList[i].gameObject.SetActive(false);
}
```

When testing the game, I fell off the platform and the player spawned on a platform (shown in the image below). The active platforms at the time when the player died, also deactivated which improved gameplay as platforms will not stack on top of one another. I have noticed that the increased player speed does not reset after death, I will tackle this problem later. The initial problem seems to be resolved. Image of testing is shown below:





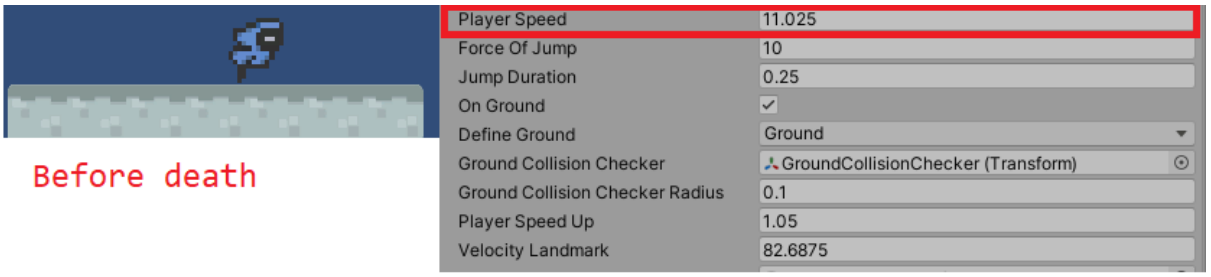
What is being tested?	Input	Justification	Expected outcome	Actual outcome
The game restarts after the player dies.	No input from the player.	The player does not need to navigate the game over menu and then the title screens every time they lose. Reduces annoyance.	The game restarts after the player have fallen off a platform.	The game restarts after the player have fallen off a platform.

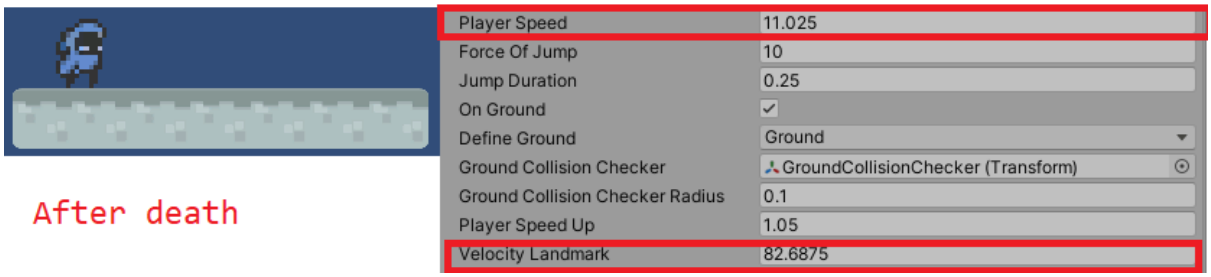
Test Data: Player restarts after death.

Test Data	Expected	Outcome
Player is set at beginning after reset.	Valid.	Valid.
Player is spawned on platforms.	Valid.	Valid.

I mentioned that the player still has their increased speed when they respawn. I see this as a problem as the landmark would be increased which as well could mean that the player will have to play for a while before they notice an increase in speed. Also, if the player died

while moving really fast, the player will still be able to move fast after they spawn which may be difficult for the play to enjoy and get used to. The image below shows the player having the same increased move speed and increased landmark:






```
public float playerSpeed;
private float playerSpeedStore;
public float velocityLandmark;
private float velocityLandmarkStore;

private float velocityLandmarkCounter;
private float velocityLandmarkCounterStore;
```

To make my player's speed and the speed landmark reset, I created variables which will hold their initial values at the start of the game. The have the same name as the previous variables but with 'store' attached to the end of them.

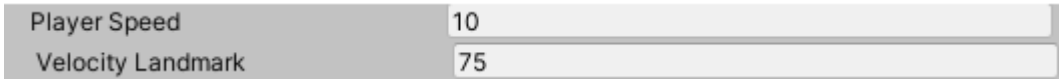
In the image below, I stored the initial values of my player's speed and speed-landmark into the 'store' variables.

```
playerSpeedStore = playerSpeed;
velocityLandmarkCounterStore = velocityLandmarkCounter;
velocityLandmarkStore=velocityLandmark;
```

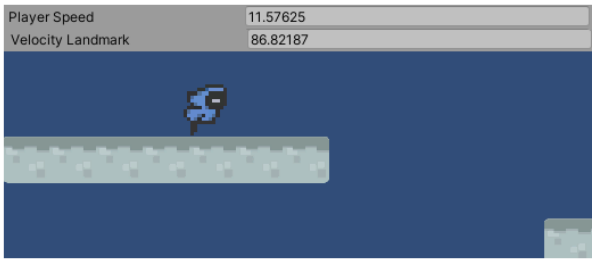
```
0 references
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "killbox")
        {
            theGameManager.RestartGame();
            playerSpeed = playerSpeedStore;
            velocityLandmarkCounter = velocityLandmarkCounterStore;
            velocityLandmark = velocityLandmarkStore;
    }
}
```

When the player dies, their speed and the landmark (for their speed to increase) resets to the initial value. This should mean that when I run the game, the player's speed and speedlanmark should be equal to the initial values after spawning.

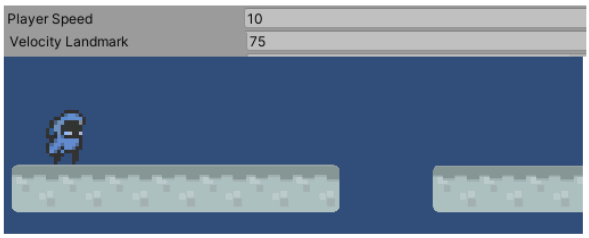
These are the initial values:



15 seconds into playing the game:



Spawning after death:



When I tested the game, I let my player speed increase for a bit and recorded it (shown in image) and then fell off the platform. When the player spawned back to the start, the velocity and landmark reset to their initial values. The increased-speed problem seems to have been resolved now.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
If the player's speed and the speed landmark is reset back to their initial values after the player dies.	No input from the player.	If the player respawns at the same speed, they died with they might move too fast at the beginning which could cause difficulty problems.	The player's speed and the speed landmark reset after the player dies and respawns.	The player's speed and the speed landmark reset to their initial values after the player dies and respawns.

12/01/2021

Interview with key-stakeholders

I am having an interview with my key-stakeholders to get their feedback on what they think of the game so far and the recent change I made with the timer feature.

What do you think of the player movements?

Azim: The player movements are easy to get a hang of. The standard WASD keys are usually a good choice, they work well here.

Hamid: They are fine, I have not noticed any problems with it so far.

Are the player animations fine for you?

Azim: Yeah, the player's animation is running smoothly considering how early you are into development, although it does not look like a cube sprite.

Have you noticed the variety in spawning of the platforms?

Azim: Yeah, while I was lasting a long run, I noticed that the game got progressively more difficult.

Hamid: I noticed it for a short time, but I kept dying, the difficulty curve is a nice feature.

I removed the timer feature due to finding a way to save memory, what are your thoughts?

Azim:

I think that the removal of the timer is a good thing because most of the tie I do not even survive for 2 minutes and even if I did, I would be annoyed with the game randomly stopping.

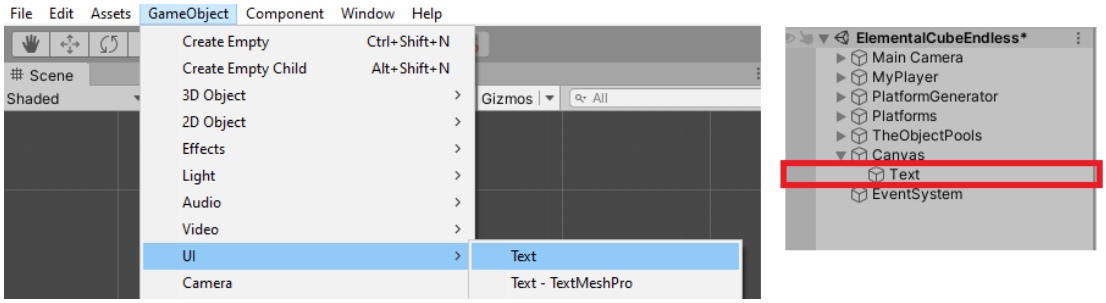
Hamid: I do not mind the timer being removed; I usually do not even survive for a minute anyway.

13/01/2021

Scoring system

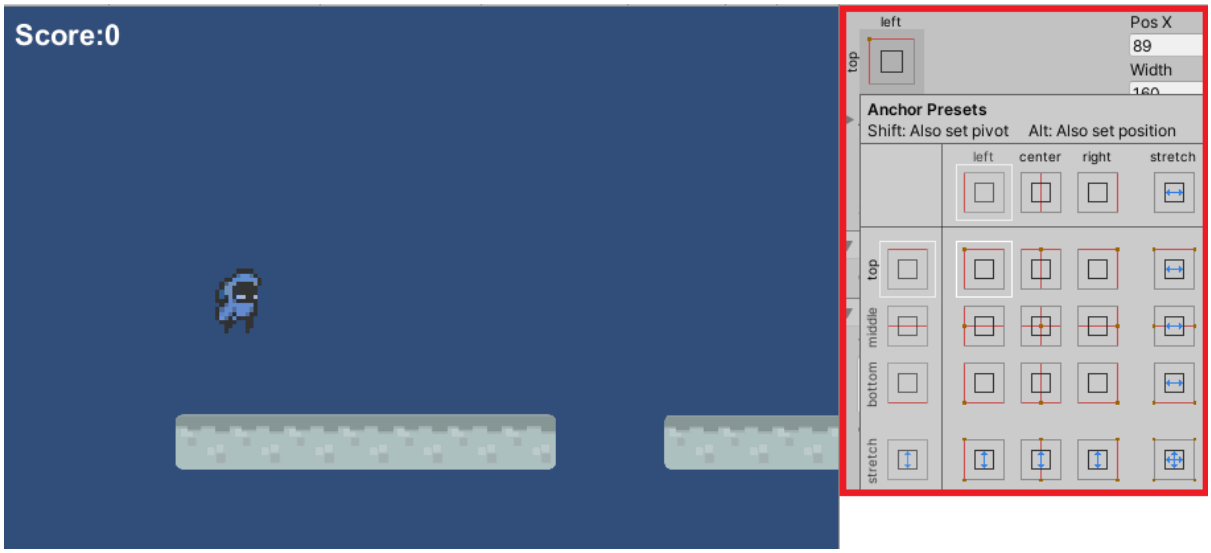
For the Player to see their measure of skill, there will be a scoring system implemented. This will give the player the incentive to carry on playing this game. A scoring system was stated in my analysis' success criteria and in my design for the leader board and I am planning to implement the system now.

First, I need to create the UI for the scoring system. In unity I can create a new text canvas through scrolling under the 'GameObject' section and by the 'UI' option, I can add some text into my game. First, I'll be implementing the score. This is shown below.

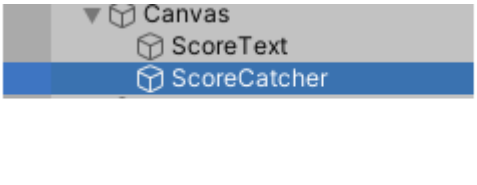


I initially set the score to 0 on the UI, as the player has not moved yet. To make sure that the player can see the score and that it will not show the score in the middle of the screen, I set

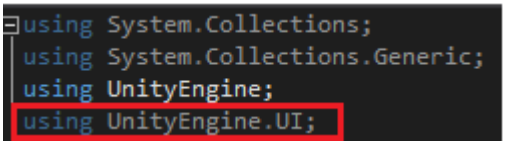
its position to the top left. This makes it easy for the player to read their score while they are playing. This is shown below:



Note: in my design I created an in-game UI where the score is shown on the right side. I decided to move the score to the left side because if the player's score goes too high, the score might appear off the right side of the screen.



On the left, I created an object that my script will be attached to. The script is named 'ScoreCatcher' and will be used to program the scoring system.



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

On the left, I added the statement using 'UnityEngine.UI' under the default settings in Unity. I needed to do this as it allows me to alter text objects in the game.

In the script, I referenced my text object where the score will be shown, otherwise the 'Score:' text will remain the same during the game and will not increase. The float variable,'score Count' will be used to calculate to the player's score. This is shown below:

```
public Text scoreText; //referencing my text object for score
public float scoreCount; //Used to keep track of score
```

The float variable, 'timePoints', adds on to the player's score by a certain amount every second. This means that the player is unlikely to die with a score of 0.

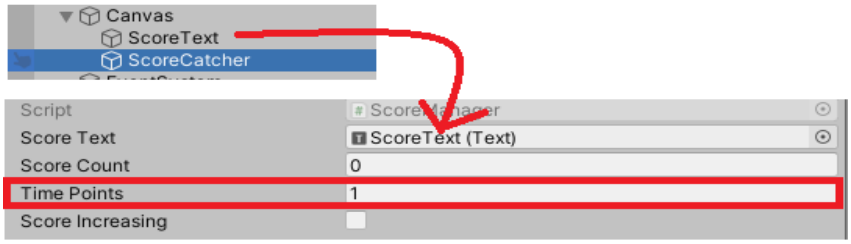
```
public float timePoints; //Player gets a certain point added to thier score every second
```

Here in the update section, I set the script to have access to the text object that I created in my game. The '.text' method allows me to change what it says on the score and the addition

of the other variable 'scoreCount'. I set 'scoreCount' to increase based on the value given in 'timePoints' which means that if I set 'timePoints' =2, the player's score will increase by 2 each second. This is shown in the code below:

```
void Update()
{
    scoreCount += timePoints * Time.deltaTime; //Time passed in each frame,adds a cetrain point each secodn to the score
    scoreText.text = "Score: " + scoreCount; //Accessing text within the text object

}
```

The ScoreCatcher is the game object that will manage the score in the game and is what the score script will be attached to. I added my text object, 'Score: 0' in the script and set the 'timePoints' =1. The player's score will increase by 1 point every second.



When running the game, my score was increasing but it showed decimals as well which doesn't look appealing to the player as round numbers are much easier to read.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Scoring system.	No input from the player.	A scoring system can be used to gauge a player's skill and make	The player's score increases by a certain amount every	The player's score increases but unexpectedly

		playing much more fun.	second and is shown on screen.	shows decimals in the score.

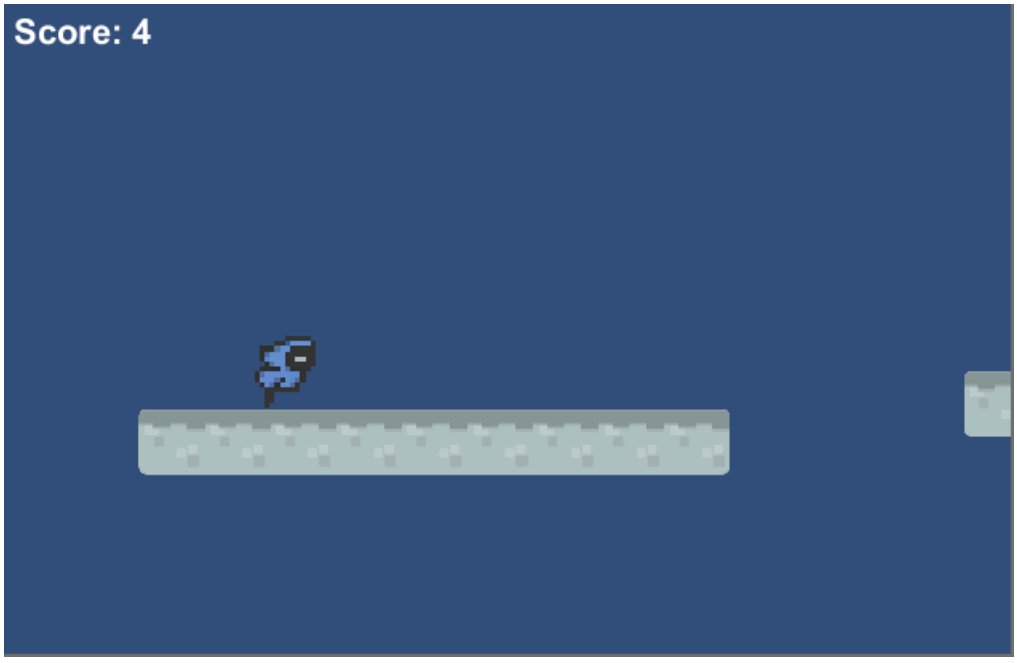
After going on one of Unity's forums, I discovered that using the '.Round' method from the 'Mathf' class, I am able to round floats to whole numbers. This means that the game should output whole numbers by the score.

```
void Update()
{
    scoreCount += timePoints * Time.deltaTime; //Time passed in each frame,adds a cetrain point each secodn to the score
    scoreText.text = "Score: " + Mathf.Round(scoreCount); //Accessing text within the text object
}
```

To make sure that the game does not increase the score if the player has died, I created a new Boolean variable. If the player is not moving across the platforms or has died from fall, the score will stop increasing in this 'if' loop containing the Boolean variable. This is shown below:

```
public bool scoreIncrease;

void Update()
{
    if (scoreIncrease)
    {
        scoreCount += timePoints * Time.deltaTime; //Time passed in each frame,adds a cetrain point each secodn to the score
    }
```

After creating that 'if' statement, I saved the script and tested the game. When I was running, the score increased by 1 point every second with whole numbers which fixed my previous problem.

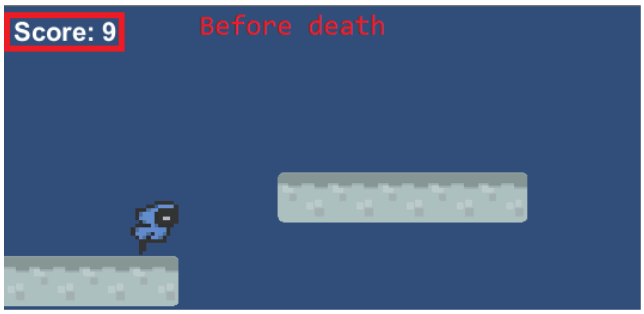


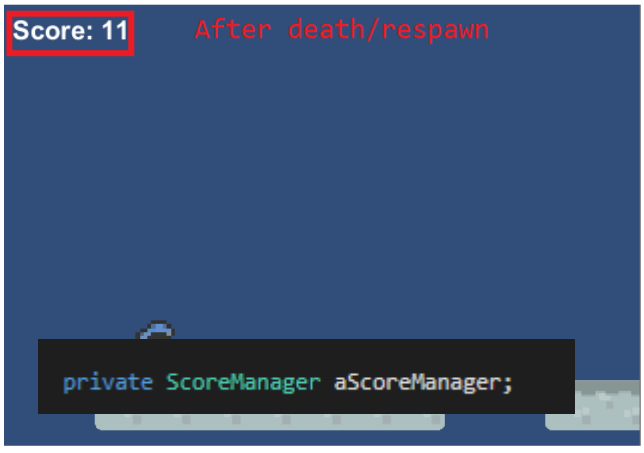
What is being	Input	Justification	Expected	Actual outcome

tested?			outcome	
Scoring system.	No input from the player.	A scoring system can be used to gauge a player's skill and make playing much more fun.	The player's score increases by a certain amount every second and is shown on screen.	The player's score increases in whole numbers.

Test Data: Scoring system.

Test Data	Expected	Outcome
Score accumulates over time.	Valid.	Valid.
Score rapidly increases.	Invalid.	Invalid.
Score is displayed.	Valid.	Valid.
Score is saved on Leader board.	Valid.	Invalid.
Score is 0 at restart/reset.	Valid	Invalid.





While I was running the game, I noticed another problem. Which is that when the player dies and respawns, the score does not reset and continue to increase. This problem has a huge effect on scoring and the leader board due to the player constantly being able to increase their score without penalty which could lead to the game storing infinitely high scores and crashing. This could pose a problem for my stakeholders as they may not have an incentive to carry on playing if they cannot measure their score accurately.

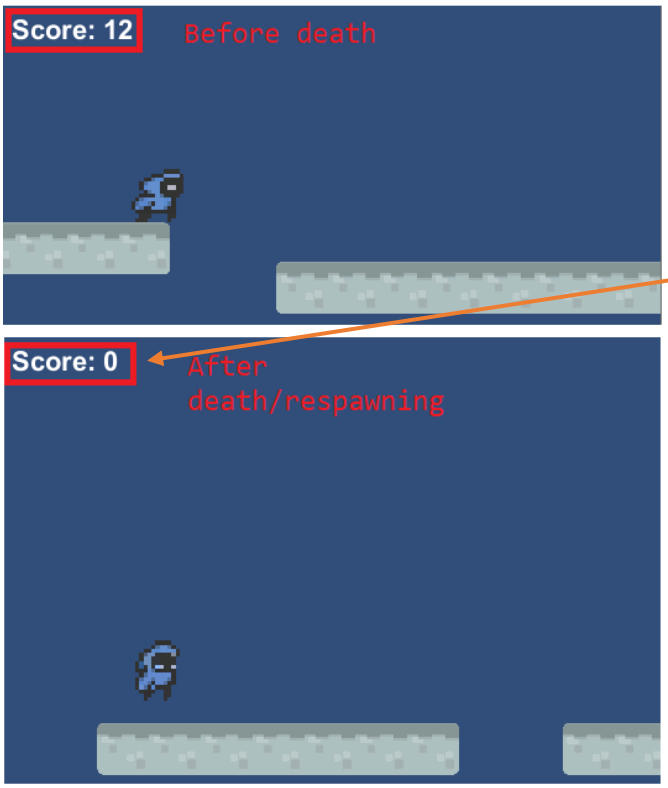
To fix this, I created a new class, 'aScoreManager', in my Game manager script which references the score manager script. I did this since the game manager is responsible for restarting the game; resetting the score relies on the game restarting.

As you can see below, I have set 'aScoreManager' to find the score manager script in my game. This will allow the class 'aScoreManager' to alter the value of the score.

```
aScoreManager = FindObjectOfType<ScoreManager>(); //Finding score manager in my game
```

In my coroutine for restarting my game, I set the 'scoreIncrease' to false as soon as the player dies. This means that when the player dies, the score will stop increasing., this prevents the player from getting an infinitely high score. This is shown in the first red box. Just before the coroutine finishes, I let 'aScoreManager' reset the current score back to 0. This signifies that the score has been reset. Just under that command, I let the Boolean, 'scoreIncrease', become true. This means that when the game restarts, the score will then start increasing again. This is shown below:

```
public IEnumerator RestartGameCo()
{
    aScoreManager.scoreIncrease = false; //Score stops increasing
    myPlayer.gameObject.SetActive(false);
    yield return new WaitForSeconds(1f);
    myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
    for (int i = 0; i < myPlatformList.Length; i++) //Goes through the array of active platforms
    {
        myPlatformList[i].gameObject.SetActive(false);
    }
    myPlayer.transform.position = playerSpawnPoint;
    platformGenerator.position = pgSpawnPoint; //sets the position back to the start
    myPlayer.gameObject.SetActive(true);

    aScoreManager.scoreCount = 0; //Resets score to 0
    aScoreManager.scoreIncrease = true; //Score starts increasing again
}
```



When I tested the game, I let the score increase for a bit and then fell off a platform. When I looked at the score, it stopped increasing at 12 and then went back to 0. This means that the game has successfully cleared the score. This is shown on the left.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Scoring system resetting.	None form the player.	The player is constantly able to increase their score without penalty which may lead to the game storing infinitely high-scores and crashing.	The score resets after the player have died.	The score resets after the player have died.

Test Data: Scoring system.

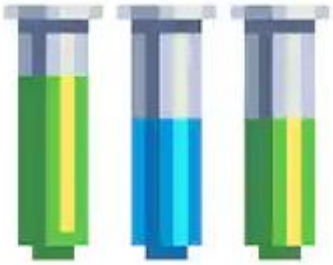
Test Data	Expected	Outcome
Score accumulates over time.	Valid.	Valid.
Score rapidly increases.	Invalid.	Invalid.
Score is displayed.	Valid.	Valid.
Score is saved on Leader board.	Valid.	Invalid.
Score is 0 at restart/reset.	Valid	Valid.

This is one of the steps I'll be using for my scoring system, as in my design I created a rough idea on how my scoring system will work. So far having the player see their score is a success.

15/01/2021

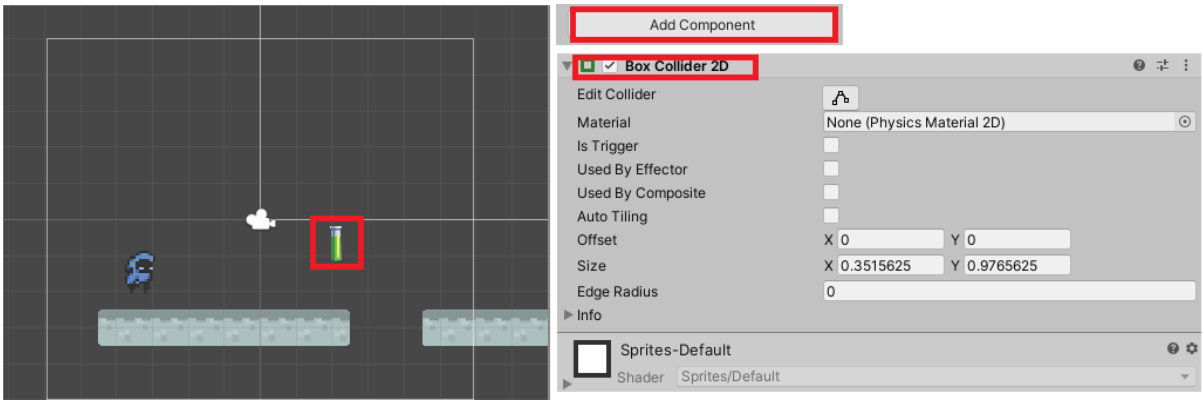
In my game analysis' success criteria and design, I mentioned having tubes that the player can pick up for extra points. This keeps the player engaged just like how I was when analysing existing solutions (such as 'New Super Mario Bros'). The tubes will be scattered across the platforms.

Right now, I will be working on letting the game recognise the tubes having a certain number of points attached to it and allowing the player to pick them up.

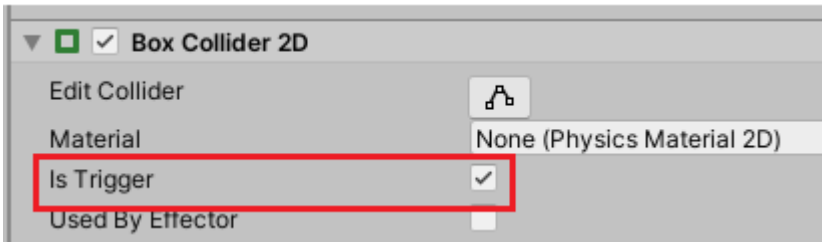


For the tube sprites, I found some royalty-free sprites that I can use for my game. These will be worth a certain number of points to the player. I will be using the green tube as the default tube that will be in the game.

After dragging the image into my game, I selected the green tube and added the box collider component to it. This means that the game will know when the player sprite's box collider and the tube sprite's box collider meet one another, just like how the player recognizes the platforms. This is shown below:



On the box collider, I set the 'is Trigger' option as active. This means that when the player comes into contact with it an event will trigger. In this case it is giving points and disappearing.



I created a new script which will be used to program my tubes so that they can be collected as points for the player's score when they encounter them. Once the player encounters a tube, it should disappear.

I created an integer variable named 'tubePoints' which will be the number of points that the player will get when they encounter the tube. Also, I created a new class, 'tubeScoreManager' that is referencing my score manager script. This is because the tubes will affect the player's score. This is shown below:

```
public int tubePoints; //The value that the tube will give to the player.

private ScoreManager tubeScoreManager;
```

I have set 'tubeScoreManager' to find the score manager script in my game. This will allow the class 'tubeScoreManager' to alter the value of the score when the player runs into a tube. This is shown in the statement below:

```
tubeScoreManager = FindObjectOfType<ScoreManager>();
```

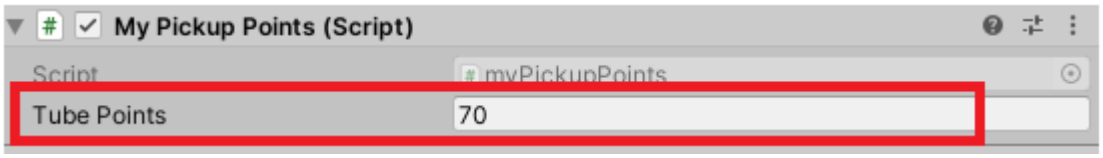
I added a new method in my script which is the 'OnTriggerEnter2D' method. When a game object collides with another game object that has the ColliderBox2D component, an event will trigger. In my if statement, I made the rigger the player. So, if a tube comes into contact with a game object called 'Player', the commands in the 'if' statement will be carried out. In this case, the value assigned to a tube will be added to the score.

```
void OnTriggerEnter2D(Collider2D other) //When something collides with the trigger
{
    if (other.gameObject.name=="Player")
    {
        tubeScoreManager.AddScore(tubePoints); //Adds value of tube to the player's score
    }
}
```

```
public void AddScore(int pointsDue)
{
    scoreCount += pointsDue;
}
```

In my score manager script, I mentioned the 'pointsDue' variable so that it can add that value on to the score.

I added my script to my tube game object and set a large value on the tube, so it'll be obvious if the tube's points are being added on to the player's score.



When testing the game, I ran into the tube and my score increased, significantly, as shown in the image below. This means that my game can recognise the value of the tube and is able to successfully add it onto my score when I pass through it. However, I noticed that the tube did not de-spawn, in-game, when the player went through it. This is something I will need to fix later because the player could just go back and repeatedly pass through it and increase their score very quickly with little effort.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Tubes adding on to the score.	None from the player. Points activate when the player is moving, in which case the input is then 'W','A','D' keys.	The tubes that the player can pick up for extra points is an easy way to keep the player engaged.	The player's score increases when the player passes through the tube.	The player's score increases when the player passes through the tube.

Test Data: Tube interaction.

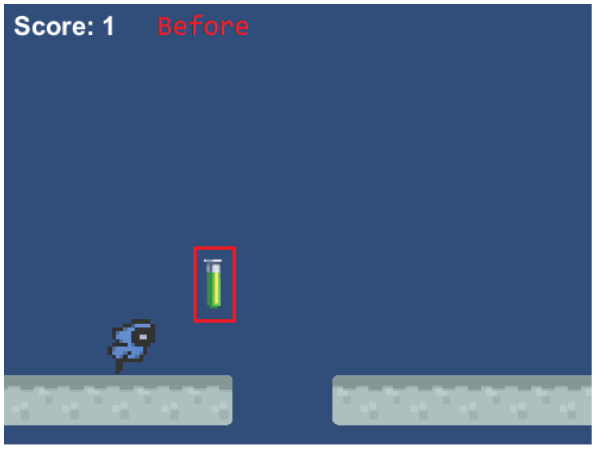
Test Data	Expected	Outcome
Tubes increase points.	Valid.	Valid.
Tubes de-spawn on contact.	Valid.	Valid.
Tubes remain after contact.	Valid.	Valid.
Player can click on tubes (Left mice click).	Invalid.	Invalid.

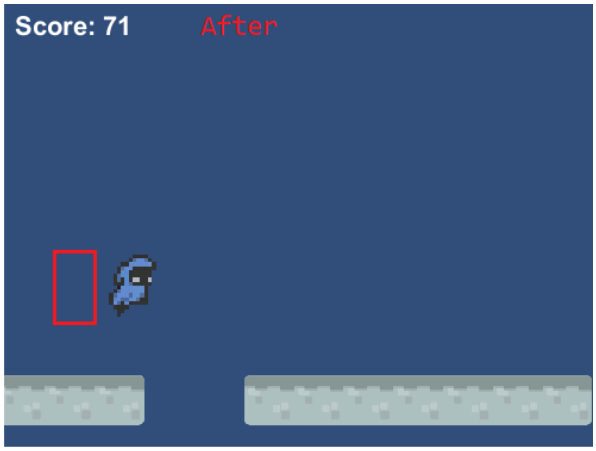
Deactivating tubes after collision

In the test above, I mentioned that the tube did not de-spawn in-game when the player went through it and it posed a problem for the scoring system. This is an easy fix. Setting the gameObject to false after the tube's value is added onto the score, means that the tube will de-spawn after contact.

```
void OnTriggerEnter2D(Collider2D other) //When something collides with the trigger
{
    if (other.gameObject.name=="Player")
    {
        tubeScoreManager.AddScore(tubePoints); //Adds value of tube to the player's score
        gameObject.SetActive(false);
    }
```

When I ran the game, the tube successfully disappeared after I encountered it. This prevents players from repeatedly going past it and infinitely increasing their score.





What is being tested?	Input	Justification	Expected outcome	Actual outcome
The Tubes deactivating after the player passes through them.	None from the player.	This prevents players from repeatedly going past the tube and infinitely increasing their score.	The Tube deactivates when the player passes through it.	The Tube deactivates when the player passes through it.

Test Data: Tube destruction.

Test Data	Expected	Outcome
Tubes de-spawn on contact.	Valid.	Valid.
Tubes remain after contact.	Invalid.	Invalid.

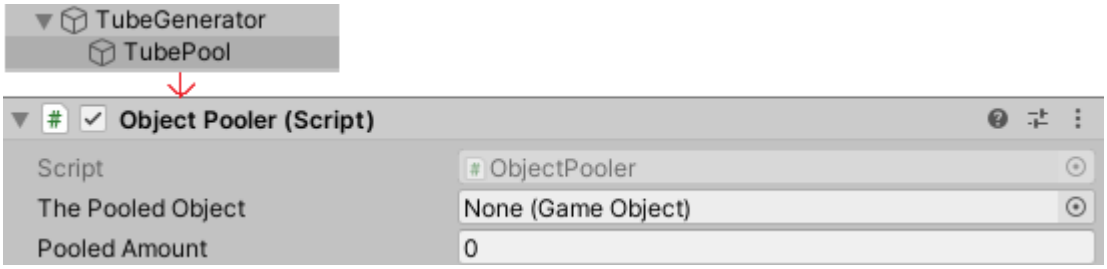
16/01/2021

Randomly placing tubes on my platforms

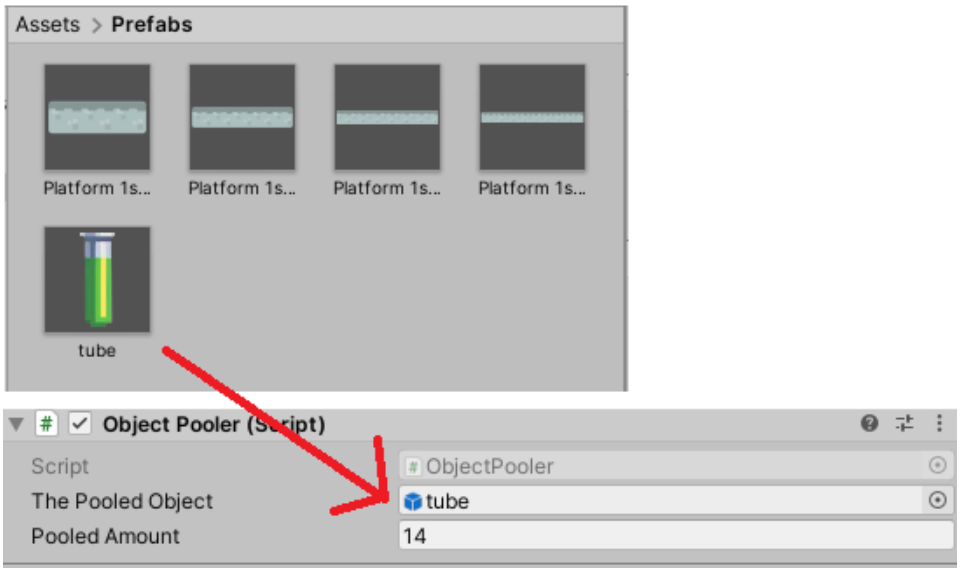
In my design I had images of what the game would look like, including having tubes scattered across platforms. I will be implementing this feature into my game today, but the tubes will be placed on random platforms, like how they were shown in my design stage, so that the gaining points seems less uniform. This is a way to keep the player engaged because tubes will not spawn on every platform.

To have tubes randomly generated, I will need an object-pooling system, like that of my platforms, for my tubes.

I created a game-object named 'TubeGenerator' which will randomly generate tubes onto the platforms. I created a child-game-object under it named, 'TubePool' which will be using object-pooling to activate and deactivate tubes. I am using this method as object pooling is an efficient method of saving CPU usage. After, I created those 2 game objects, I attached my Object Pooler script onto the 'Tube Pool is shown below:



I opened my prefab folder, where I keep all the platforms, of varied size, in and dragged my tube game-object into that. This makes my tube a reusable asset, just like the platforms in the game. After making it a prefab, I dragged the tube into the script so that the script knows what to reuse (shown in the image below). Where it says, 'Pooled amount', I set the value to a large number. In this case,14 as it means that the game will have 14 tubes in-game regardless of if it is in use or not in use. This also means that I will not have less tubes available then platforms at any given moment in the game. This is shown below:




```
public ObjectPooler tubePool; //Used

public float distanceBetweenTubes;
```

I created a new script named, 'TubeGenerator', where I will be programming the feature of spawning tubes on random platforms. (Shown on the left)

Just like my object pool for my platforms, I created a class for my tubes, which is named 'tube Pool'. The float variable, 'distanceBetweenTubes', is going to be the distance set between each tube that is spawned on a platform.

I created a new void function for spawning tubes, 'SpawnTubes', as this function will only be called at certain times during gameplay. The vector position, 'startPoint', will be the default position of where the tubes will spawn on each platform. The statement is shown below:

```
public void SpawnTubes(Vector3 startPoint)
{
```

In one of my design images, I had around 3 tubes per platform, so I will be using 3 tubes. I create 3 classes for my tubes, named 'tube1', 'tube2' and 'tube3' and I will use the '. GetPooledObject' method to them as game objects in the game. 'tube1' will be spawned at the middle of the platform. 'tube2' will be spawned on the left of tube 1 due to it taking away the 'distanceBetweenTubes' variable. and 'tube3' will be on the right due to it adding on the 'distanceBetweenTubes' variable. This means that the tubes are spread out and are not all in the same position, which replicates the images I created in my design stage. The statements are shown below.

```
GameObject tube1 = tubePool.GetPooledObject();
tube1.transform.position = startPoint;
tube1.SetActive(true);

GameObject tube2 = tubePool.GetPooledObject();
tube2.transform.position = new Vector3(startPoint.x - distanceBetweenTubes, startPoint.y, startPoint.z);
tube2.SetActive(true);

GameObject tube3 = tubePool.GetPooledObject();
tube3.transform.position = new Vector3(startPoint.x + distanceBetweenTubes, startPoint.y, startPoint.z);
tube3.SetActive(true);
```

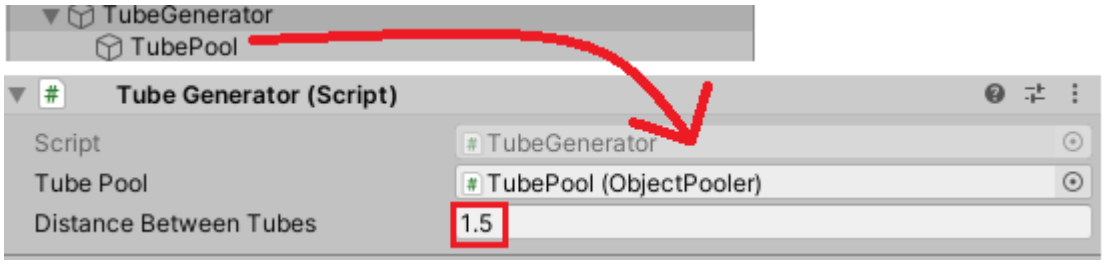
Since the tubes are spawning on my platforms, I must reference my tube generator script in my platform generator script. I named the class 'myTubeGenerator'. I went on to the Start void and I used the 'FindObjectOfType' method on 'myTubeGenerator' so that at the start of the game, any object that has the tube generator script attached to it will be used to spawn on the platforms. This is shown in the statement below:

```
private TubeGenerator myTubeGenerator;
void Start()
myTubeGenerator = FindObjectOfType<TubeGenerator>();
```

In the update void, I used the 'SpawnTubes' function to spawn the tubes onto the platforms. In the coordinate brackets, the y axis (underlined in red) is moved up by +1. I have done this so that the tubes don't spawn in the platform but on top of the platform, where the p[layer can reach them. This is shown below:

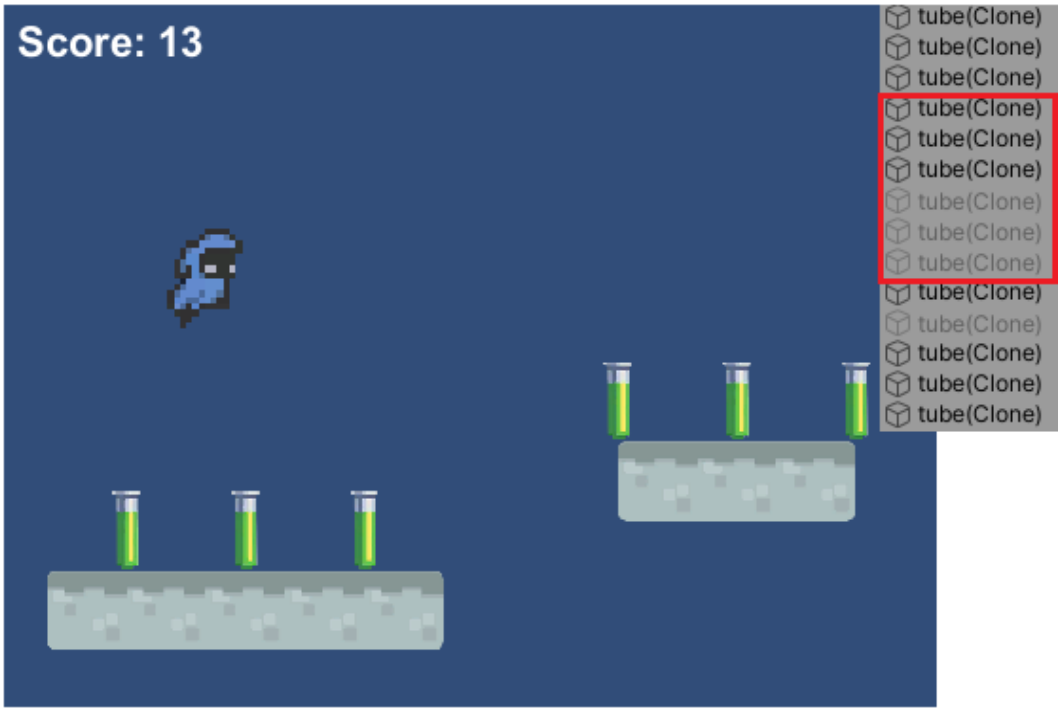
```
void Update()
myTubeGenerator.SpawnTubes(new Vector3(transform.position.x, transform.position.y+1f,transform.position.z) );
```

On my 'TubeGenerator' game-object, I attached the tube generator script. I dragged the child-game-object, 'TubePool' in to where the object pooling will occur and set the distance between each tube to be 1.5 apart on the x-axis. This is shown below:



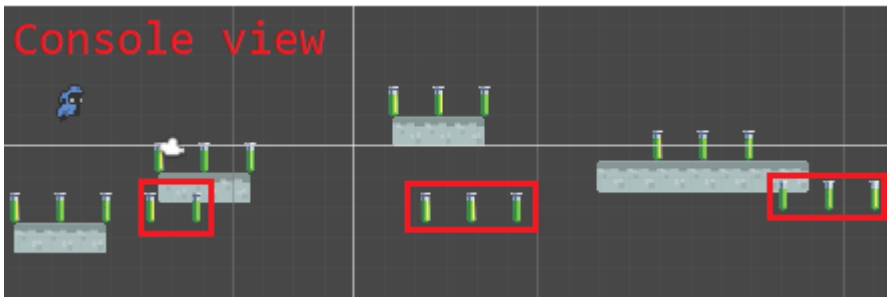
When I tested the game, the tubes began spawning on each platform at a set distance apart and when I checked on the console, the game was using object pooling (shown below in the

red box). This means I have achieved one of the ideas that I visualised in my design stage.

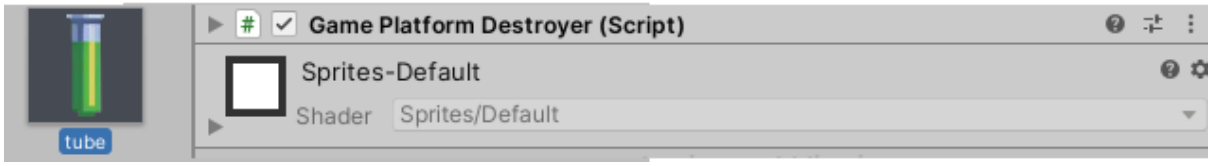


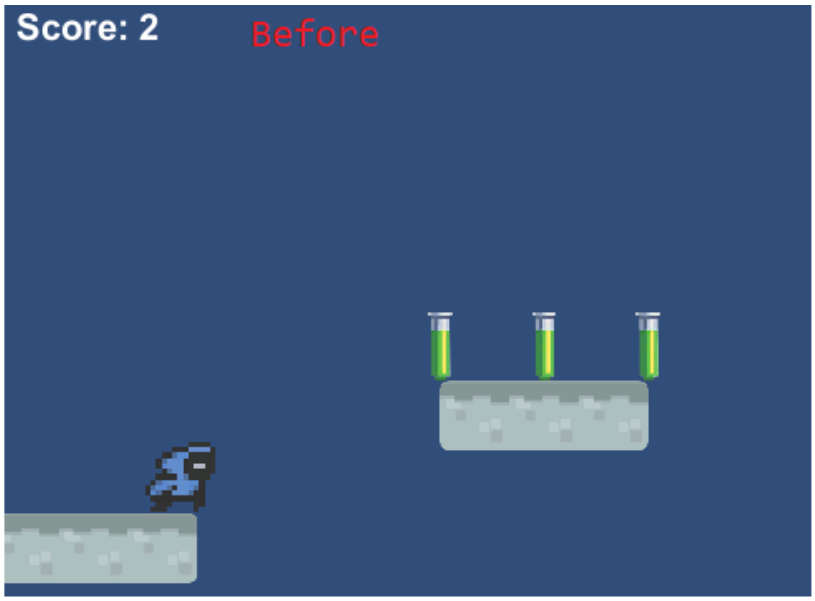
What is being tested?	Input	Justification	Expected outcome	Actual outcome
Tubes spawning on every platform.	None from the player.	To check that tubes are spawning on my platforms before I implement randomization	The tubes spawn on every platform at a set-distance away from each other.	The tubes spawn on every platform at a set-distance away from each other when the game is running.
The game using object-pooling to spawn tubes.	None from the player.	Using object pooling for the tubes my game	The game reuses tubes from the object pool while the game is running.	The game reuses tubes from the object pool while the game is running.

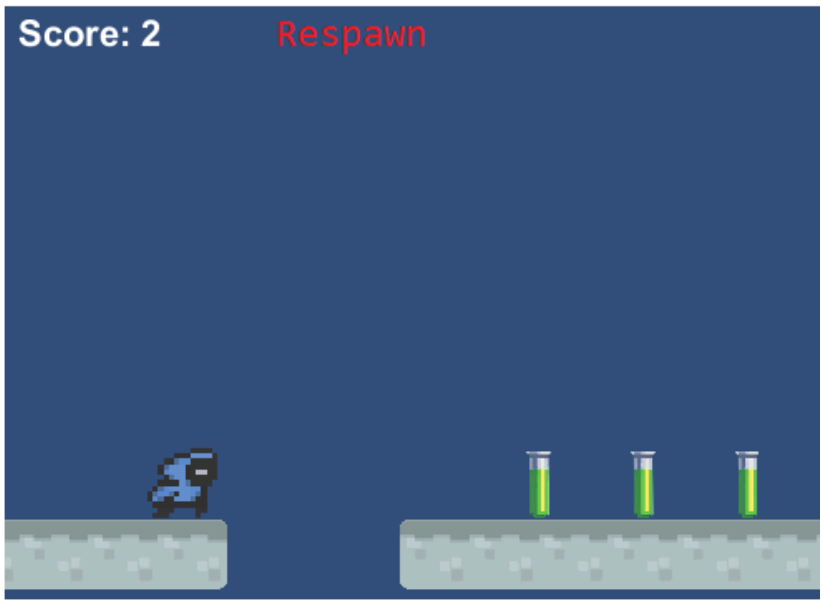
However, I ran into a problem while I was testing the game. Which was that if I carried on playing after I died, the tubes from the previous platform were still present. This is shown in the image below, in the red boxes. This may look messy to my stakeholders; it could give players extra points and may be misleading the player into dying very early in the game.



On my tube prefab, I gave it the platform destroyer script. This means when a platform is destroyed, any tubes left on it will also be destroyed.







When I tested the game, I fell off the platform before the platform with tubes on it. After I respawned, I headed forward and the tubes from the platform before death were gone. This means that the tubes were also destroyed along with the platform.

This prevents players from gaining extra points in places they should not gain them and prevents the player from being led to an early death.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
The tubes being destroyed after death.	None from the player.	Prevents players from gaining extra points not being led to an early death.	The tubes are destroyed after the player dies.	The tubes are destroyed after the player dies.

Tubes spawning on random platforms.

Right now, tubes are spawning on every platform that is spawned. To make the game less uniform and more engaging, the tubes will be spawned on random platforms.

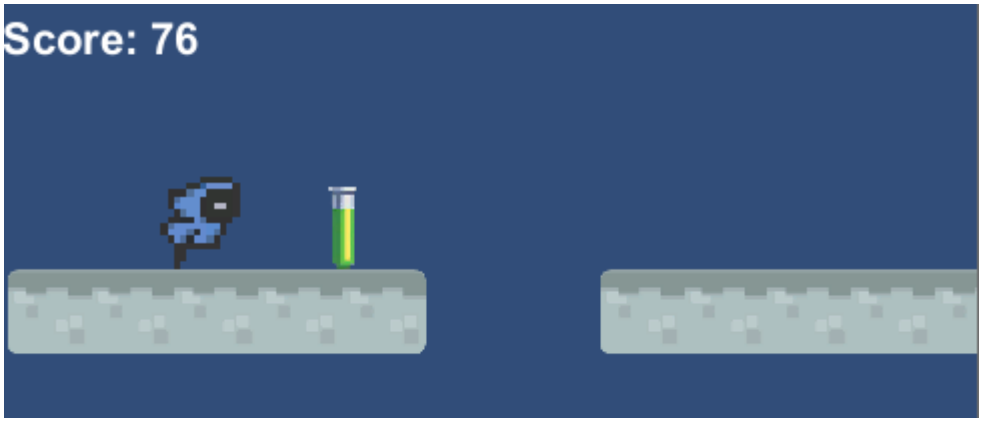
In my platform generator script, I created a float variable called 'tubeChanceLimit'. I am going to use a "Chance system" to determine if a tube will be spawned onto a new platform at any given time. In the 'if' statement I used the 'Random.Range' function to select a number between 0 and 100.If the selected number is less than the value assigned to 'tubeChanceLimit', then tubes will spawn onto a platform (e.g., if tubeChanceLimit is 50 and the randomly selected value is 36, tubes will be spawned onto a new platform.). The 'if' statement is shown below:

```
private TubeGenerator myTubeGenerator;
public float tubeChanceLimit;
```

```
if (Random.Range(0f, 100f) < tubeChanceLimit)
{
    myTubeGenerator.SpawnTubes(new Vector3(transform.position.x, transform.position.y + 1f, transform.position.z));
}
```

The value set is 61, which means there is a 60% chance of tubes spawning onto a new platform.

When I ran the game, some platforms did not have tubes. This means that my chance system is taking effect in the game. This is shown below:



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Tubes being spawned on random platforms.	None from the player.	Gaining points will seem less uniform. This keeps the player engaged because tubes will not spawn all the time, which increases rarity.	Tubes are not spawned onto every platform the player comes by.	Tubes are not spawned onto every platform the player comes by.

Test Data: Random Tube spawning.

Test Data	Expected	Outcome
Tubes do not appear on every platform.	Valid.	Valid.
Tubes spawn anywhere.	Invalid.	Invalid.

21/01/2021

Re-evaluating Gameplay

In my analysis and design, I mentioned a leader board that would be present in the game. The game would show the players with the highest scores achieved in the game, which can be accessed through the menu. The player can enter their name and it would be stored in the leader board. I introduced this as a way for the player to see how well they did compare to others. However, with the only one stakeholder present per game and the multiplayer limitation I mentioned, there is no use of having a leader board if there is no one to compete with. However, I still want my stakeholders to have an incentive to play, so considering time-limitations, I think the best option here is for me to go with a high-score counter that will be present during the game.

This High-score counter will be displayed at the top of the screen along with the score and will update when the player beats the highest score recorded in the game.

22/01/2021

Emailing my Key stakeholders

I want to know how my key-stakeholders, Hamid Chowdhury and Azim Khanazada, feel about the change in direction in terms of the game's scoring system. I sent an email to my

key stakeholders and asked for feedback as it allows me to get an idea of how I should approach me new idea so that my key-stakeholders are pleased with the end-result of my solution.

My email:

Hello Azim and Hamid,

I have been developing and creating changes to the game and I have decided that it is important to notify you two as the most recent change is big. I decided that it would be better and more logical if I removed the leader board.

My solution is to replace the leader board with a high-score counter. The high score counter will be displayed on the corner of your screen so you can see your high score. This seems to make more sense as the game is not designed to be neither online nor local multiplayer.

I would like to know both of your stances on this change, as it will give me a better sense of direction towards a game more suited to your liking.

Best Regards,

Ehsan Ahmed.

24/01/2021

Feedback from my Key stakeholders

Azim:

“Hello Ehsan,

I just read your email and I do not mind the removal of the leader board. It is not a feature that I was intending to use with other students as the game is not multiplayer. The new replacement idea seems fine”.

Best wishes,

Azim Khanazada.”

Hamid:

“Hi Ehsan,

I was going through the email you just sent to me and Azim about the leader board being scrapped. I was curious about using the feature against my peers, but since the game is not going to support multiplayer features, I am not really concerned too much about the leader

board. The replacement you proposed should work well for me anyway because I do not want extra RAM being used for the leader board.

Kind regards,

Hamid Choudhury".

What I can take from their responses are that the leader board feature was not an interest for everyone since Azim was not intending to use it and Hamid would not want to be using up extra RAM on his desktop pc. I will be trying to implement the high-score counter into the game soon to incentivise my key-stakeholders to keep on playing the game.

25/01/2021

Adding a High-score counter into my game

In my recent re-evaluation of my solution, I decided that it would be easier to add a high-score counter to my game. I will be implementing the high-score counter today.

Under my canvas, where I have the test for my score, I added a new text box and named it 'High ScoreText'. This will be where the high score will be displayed for the player. This is shown below:

I created a text variable to reference the 'High ScoreText' and a float variable which will display the high score. This is shown below:

```
public Text highScoreText;
public float highScoreCount; //Used to keep track of high-score
```

I opened my ScoreManager script and went to the update function (where the scripts update the game every frame). Underneath that functions I added the highscoreText variable with the words "High Score" assigned to it. I used the 'Mathf.Round' method to round the score that will appear on the screen to the nearest whole number, I also used this method earlier when implementing the scoring system. The statement is shown below:

```
// Update is called once per frame
0 references
void Update()
highScoreText.text = "High Score: " + Mathf.Round(highScoreCount);
```

In the same update function, I created an 'if' statement for when the score is greater than the current high score. The game will update the high score with the new score and will continue updating the player has died, hence why they are made to be 'equal' to each other in the 'if' statement. Underneath that, I used the 'PlayerPrefs' class as it allows me to store and access my preferences between each game sessions. Using this with the 'SetFloat'

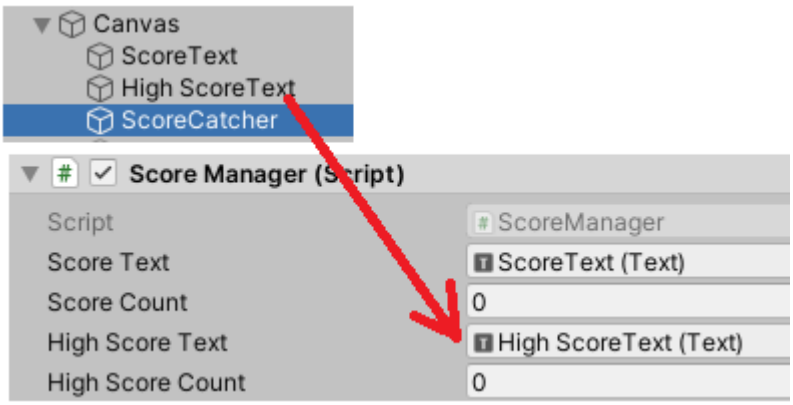
method allows me to store the value of the 'highScoreCount' variable. This value will remain even after the player leaves the game. This 'if' statement is shown below:

```
if (scoreCount > highScoreCount) //Updates High Score
{
    highScoreCount = scoreCount;
    PlayerPrefs.SetFloat("HighScore",highScoreCount);//Saves highscore
}
```

In the start function, where its contents are called before the first frame update, I created an 'if' statement. In the 'if' statement I mentioned that if the high score is not equal to 0, then the game will get the high score value that's stored in the 'PlayerPrefs' class by using the 'GetFloat' method. This 'if' statement is shown below:

```
if (PlayerPrefs.GetFloat("HighScore") !=null)
{
    highScoreCount = PlayerPrefs.GetFloat("HighScore");
}
```

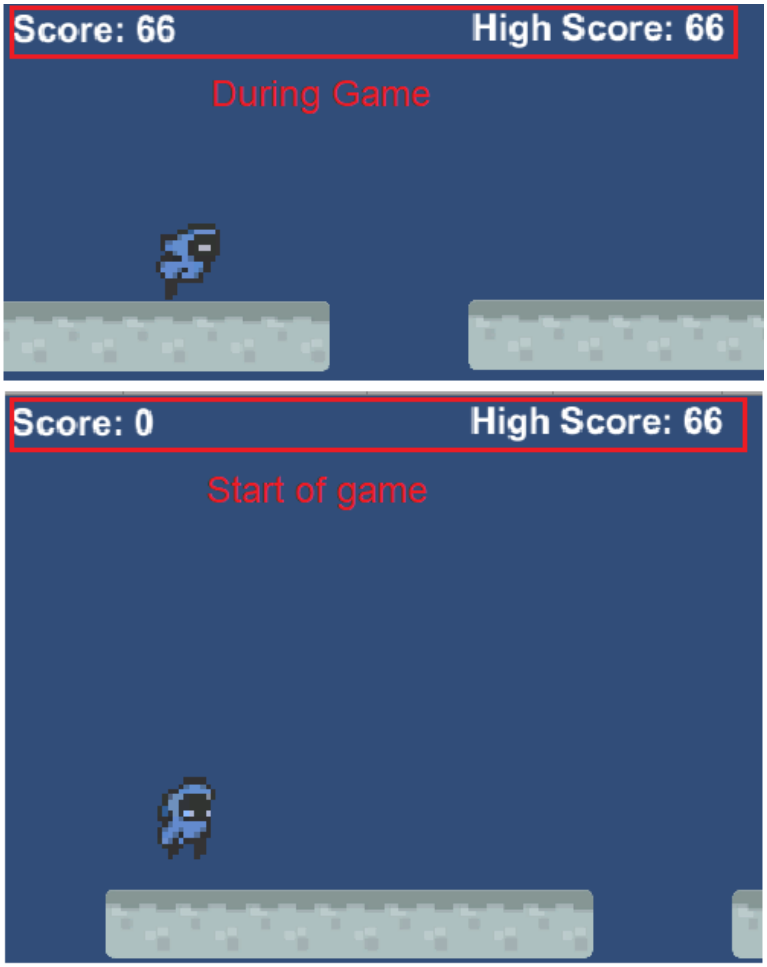
In my score catcher gameobject, I went under the ScoreManager script and dragged in my HighScore Text so that the ScoreManger script can access the text object. This is shown below:



I have set the High Score Count to 0(Shown on the left) so that I can record a highscore when I test the game. Hopefully after setting a highscore, it should remain

there when restarting the game.

When testing the game (shown in the image below), I played for a bit so that I can build up a High Score ('During Game' image), I then died and restarted the game. In the 'Start of game' image, my high score was the same as the one in the first session (top image) of the game. This means that I have successfully created a way for the player to see their high score and have it automatically stored for them in the game. This could give my stakeholders the incentive to carry on playing my game as they will always have their own score to beat.



What is being tested?	Input	Justification	Expected outcome	Actual outcome
High Score counter updating.	None from the player.	Will let the player know if their current score is greater than the last high score	The high score is displayed and updates if the player has beaten the last high score	The high score is displayed and successfully updates when the player has beaten the last

		achieved.	stored.	high score stored.
High Score being stored and displayed.	None from the player.	Allows the player to see the highest score that they have achieved and gives them an incentive to carry on using my solution.	The last high score stored in the game is displayed once the game is loaded or restarted	The last high score stored in the game is successfully displayed once the game is loaded or restarted.

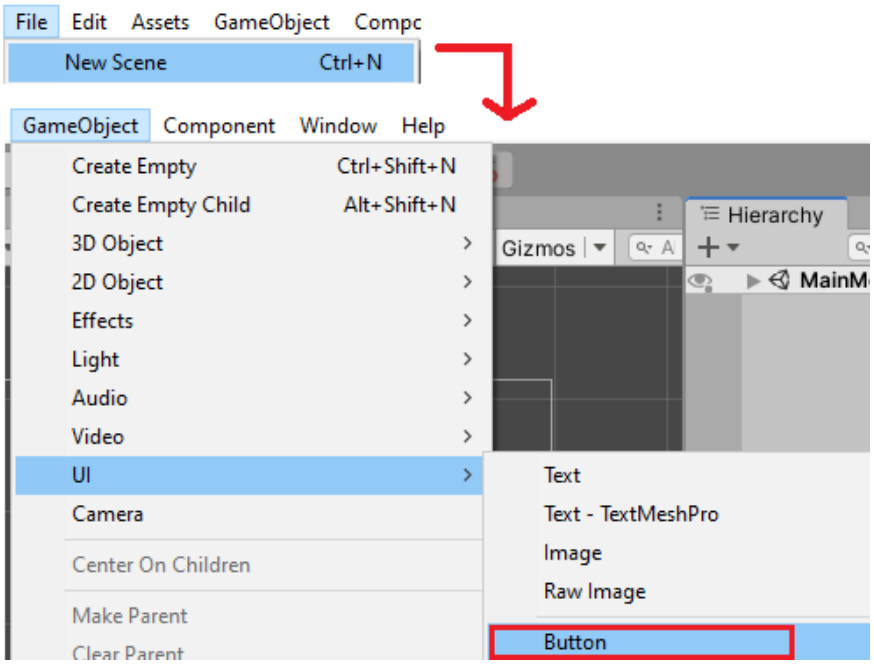
01/02/2021

Making a main menu

The main menu will be used as a base where the player can branch into the different options available in the game. Previously in my design, I created a mock-up image of a home screen which was a good starting point to refer to. It had the 4 options of 'Play', 'Leader board', 'Options' and 'quit'. Today I will be creating the main menu for my game. Also, this main menu will only consist of playing the game and leaving the game as the leader board idea has been removed and 'options' will be implemented later. The 'Options' option is to only turn music on or off, which is of less priority in development.

Creating a basic menu

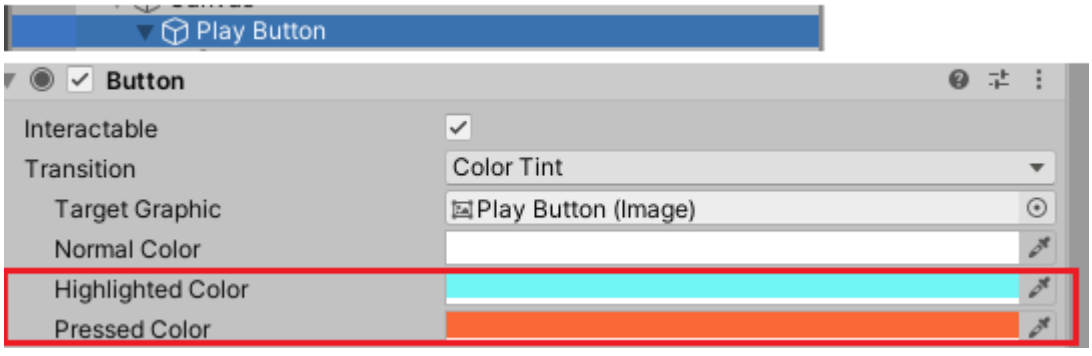
I will be creating a basic menu for the time being so that I can get the functionality right before heading towards aesthetics.



In Unity, I created a new scene for my main menu. A scene contains the environment and the menus of a game. After that, I navigated to the 'UI' option of 'GameObject' and selected 'Button'. Buttons are what player's use to interact with the menu, they click on buttons and they get a response.

After creating the button, I clicked down on the text part of the game object and renamed the button to 'Play'. This button will be used to allow the player to play the game. This is shown on the left.

I selected my 'Play' Button and went through its settings. I changed the highlighted colour and the pressed colour to bright colours so if the player hovers or clicks on a button, they'll have a good visual indicator that they did. This is shown below:



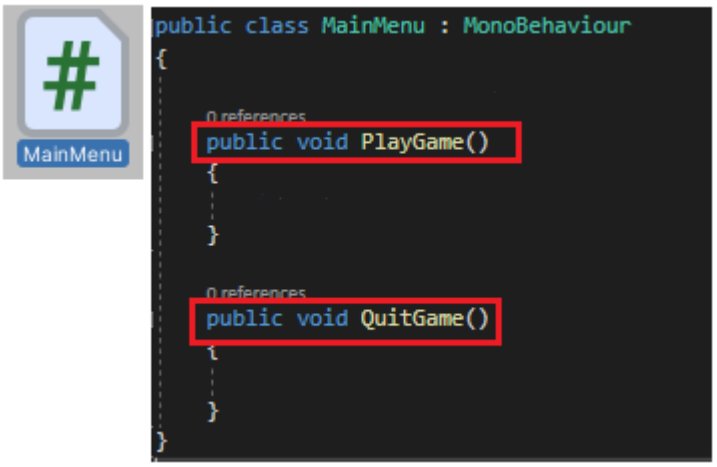
I duplicated the same button and renamed it to 'quit'. This button will be used to allow the player to leave the game. The preferences such as the colours selected for my 'play' button have been carried over to this button as well.

Highlighted:

Clicked:

When running the menu, I hovered my mouse over the 'Play' button and it turned blue. When clicking a button, the button turns from blue to orange which indicates that the player has pressed something. These colours are used as visual feedback for the player.

I added the title of my game onto the main menu. I am using basic fonts for now but will use different fonts and colours later in development.



My buttons don't actually do anything yet because I haven' created a script for the buttons to follow. I created a new script called 'Main Menu' and created 2 functions. 'PlayGame' and 'QuitGame' are the functions that the 'Play' and 'Quit' will us in the game.

```
public void QuitGame()
{
    Application.Quit();
}
```

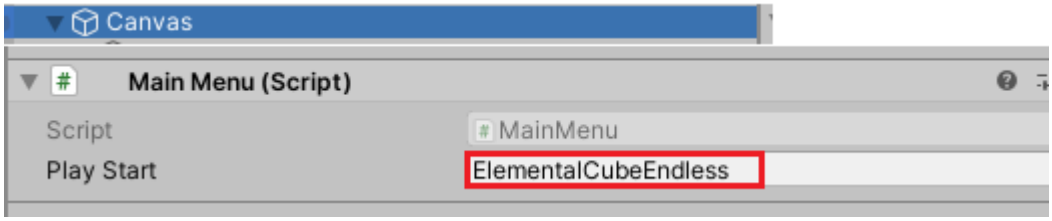
For the 'QuitGame' function I can use a built-in method in Unity which allows me to close the application. This

statement 'Application.Quit();' will close the application for the player if the 'Quit' button is pressed.

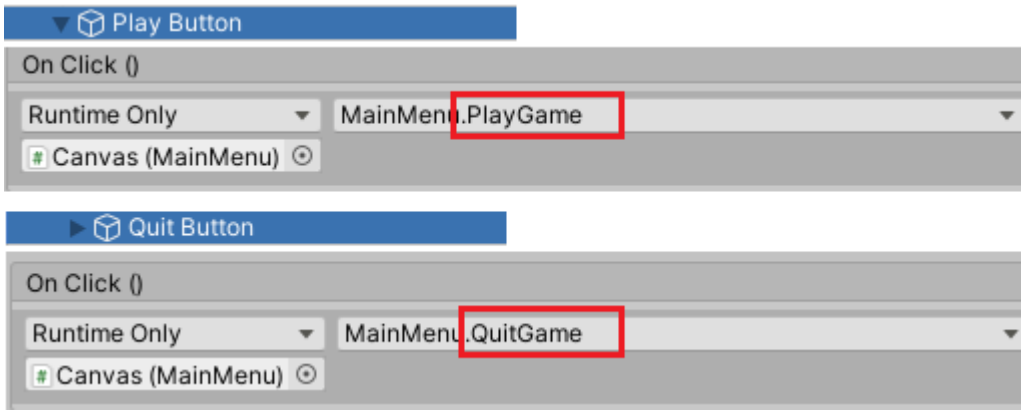
For the 'PlayGame' function, it loads the actual game based on the name of the string that is set. I will use this string to store my Elemental Cube Runner.

```
public string playStart;
0 references
public void PlayGame()
{
    Application.LoadLevel(playStart);
}
```

After saving the script, I selected the canvas and entered the name of the scene where the actual gameplay occurs, in this case it's called 'ElementalCubeEndless'. This means that when the 'PlayGame' function occurs, the game should be loaded and will allow the player to play. This is shown below:

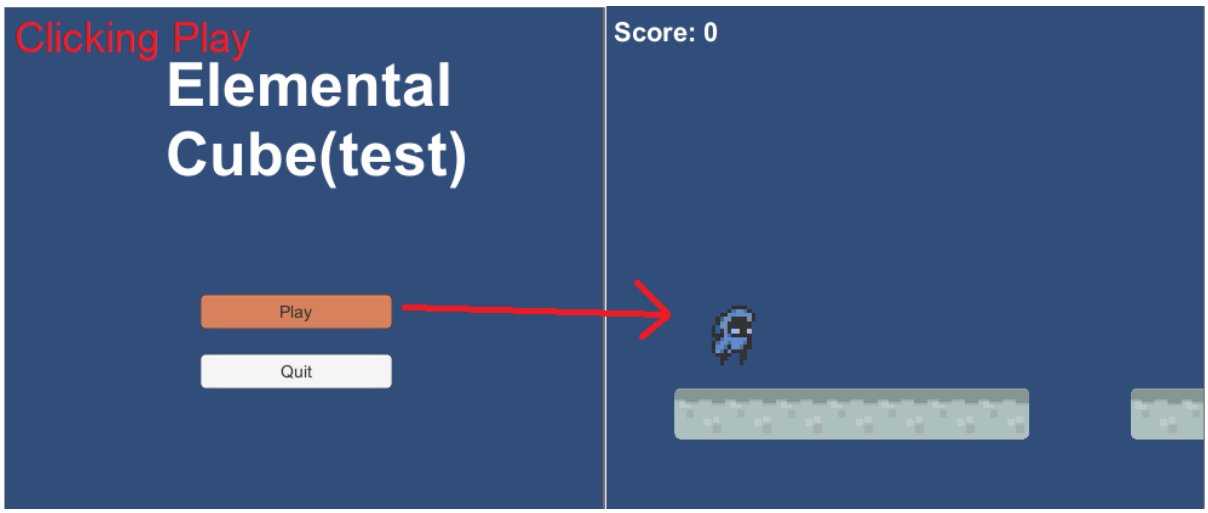


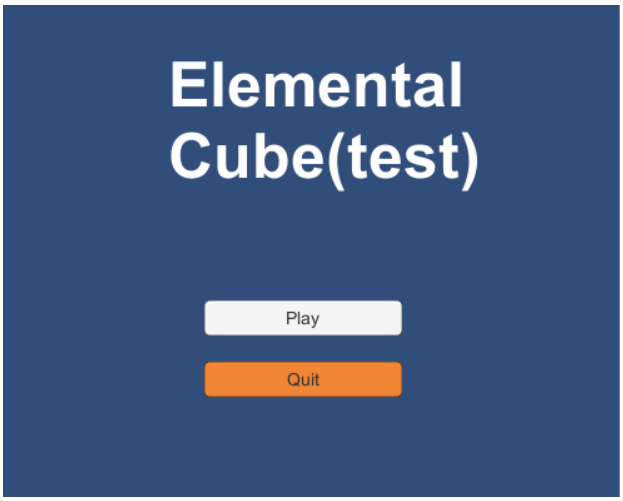
On each button, I scrolled down to the 'On Click ()' section and added the respective function to each button. This means that the functions are assigned to the actual buttons which means that my buttons should be able to change the state of the game.



I went on the build settings and added the main menu in front of the ElementalCube game which means that when the player runs the game, the menu is shown first. This is shown below:

When I tested the game, I clicked the 'Play' button which changed to orange for a second before successfully transitioning into the play state of the game.





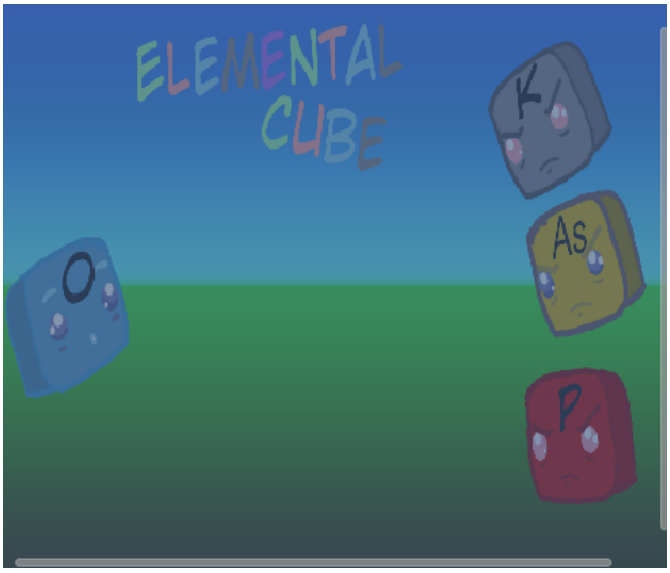
Clicking quit does what it is supposed to do and exits out the game.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Play Button.	Left mouse click.	Allows the player to transition to the gameplay from the menu.	The game transitions to the play state of the game.	The game transitions to the play state of the game.
Quit Button.	Left mouse click.	Allows the player to leave the game from the menu.	The application stops running and closes.	The application stops running and closes.

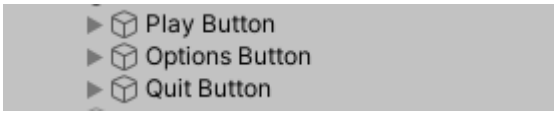
03/02/2021

Updating main menu design

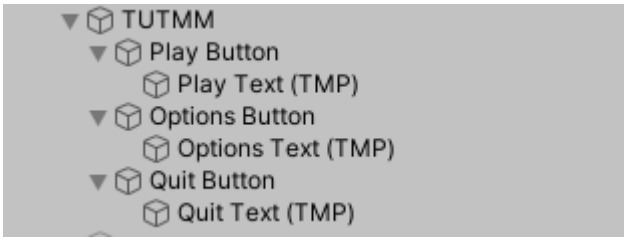
Now that I am able to create a basic main menu in Unity that can transition into the game, I'll proceed to make the menu look more similar to the one I created in my design. This is so that the games may look more interesting to my stakeholders.



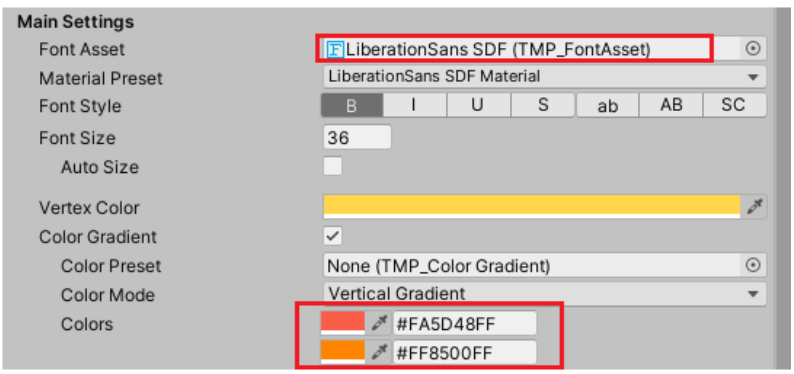
I imported the same background that I used in my design. This will be the background for the main menu as well as the option menu. I will be adding buttons just like I did in my basic menu, but they will not be completely visible as I want the player to select the text instead. This is to make the game look neater and less aesthetically-jarring.



I created 3 buttons for my main menu. One for playing the game, one for accessing the options menu and one for leaving the game.

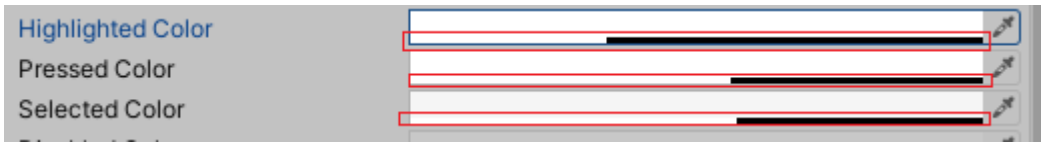


For my text design, I'll be using an add-on which is available on the unity store and is royalty-free. The add-on is called TextMeshPro (TMP) and allows me to stylize my texts, such as using different fonts and adding gradients to my text.



When I click on one of the texts, I can change the fonts and colours. I have chosen to make the gradient a yellow-orange and chose a nice font.

Since I do not want buttons surrounding the text, so I deactivated the image and have altered one of the colour channels (Alpha) on the highlighted, pressed and selected colours. This is shown below. This should mean that when I hover over and press on the texts, it should be in a darker colour.

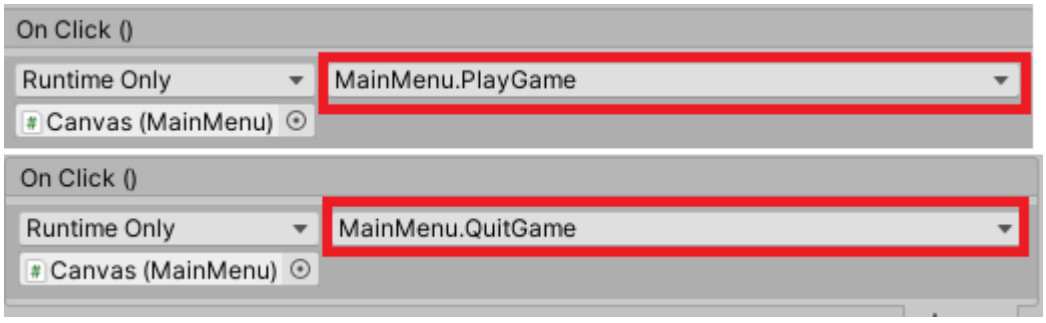


Hovering on a button

Clicking a button

When running the main menu, I hovered my mouse over the 'Options' button and the button surrounding the text went dark. When clicking a button, the button shadow goes even darker which indicates that the player has pressed something. These colours are used as visual feedback for the player.

On each button, I scrolled down to the 'On Click ()' section and added the respective function to each button. However, since I have not made the options menu yet, nothing will happen if 'Options' is clicked. Like how I made my basic menu, there is functionality assigned to the actual buttons which means that my buttons should be able to change the state of the game. The function-assigning is shown in the image below:



I went on the build settings and got rid of the previous basic main menu and replaced it with the new menu in the build settings. The menu should be shown first when the game is run by the player. The build settings are shown below:

When I tested the game, I clicked the 'Play' button which changed the shadow of the button to a darker shade for a second before successfully transitioning into the play state of the game.

Clicking quit does what it is supposed to do and exits out the game. I think this a success as of reaching my proposed solution due to my menu being a near-replica of the menu I created in my design.

What is being tested?	Input	Justification	Expected outcome	Actual outcome

Play Button on new menu.	Left mouse click.	Allows the player to transition to the gameplay from the menu.	The game transitions to the play state of the game.	The game transitions to the play state of the game.
Quit Button on new menu.	Left mouse click.	Allows the player to leave the game from the menu.	The application stops running and closes.	The application stops running and closes.

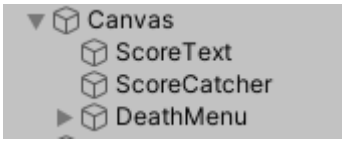
Test Data: Main menu.

Test Data	Expected	Outcome
Left Mice Click (LMC).	Valid.	Valid.
1	Invalid.	Invalid
w	Invalid.	Invalid.
#	Invalid.	Invalid.

05/02/2021

Making a game-over menu

As of right now, I got a system where the player restarts when they have fallen off the platforms. I mentioned at the start of introducing the restart-system that I will be adding a game-over menu in my game. Today I will be trying to introduce that into my game.

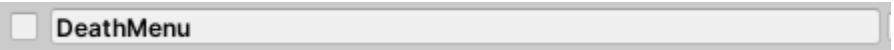


Under my canvas, where I previously added my test for the score, I created a new game-object for the Game-Over menu for when the player dies which is named ‘DeathMenu’.

Under the 'DeathMenu' game-object, I added a blank image which has been reduced in the alpha channel. This means that the background will appear darker than usual. This is so that the player's attention can be averted from the game to the game-over menu when the player has died. I added a simple title saying "You died. Game Over.". It is put in a large font so that the player knows immediately that they have lost.

I added a restart button where the player can play again if they wish to. The button beneath it allows the player to quit to the main menu in case they want to do something else in the game or leave the game. This previously was not intended in my game-over menu during my design stage of the project, but I decided to add it in as it would be more convenient for my stakeholders when navigating the game.

I set deactivated the DeathMenu as it will not be active throughout the entirety of the game. The DeathMenu will be active when the player dies. The box that is left unticked deactivates the game, object, this is shown below:



I made a new script for my Death Menu, named 'GameOverMenu'. This is where I will be programming my game to transition from the play-state to the game-over menu.

```
public class GameOverMenu : MonoBehaviour
{
    public string mainMenuLoad;

    0 references
    public void RestartGame()
    {

    }

    0 references
    public void QuitToMainMenu()
    {

    }
}
```

I created a new string variable called 'mainMenuLoad', where the main-menu scene will be stored in (Shown in the red box).In the script, I also added 2 functions: 'RestartGame' and 'QuitToMainMenu' which will be assigned to their respective button on the game-over menu. This is shown on the left.

In the 'QuitToMainMenu' function, I used one of Unity's present methods which allows me to load a scene, in this case the main menu. Since the main menu is stored in a spring (mentioned earlier),I can use this string in the statement. This is shown below

```
public void QuitToMainMenu()
{
    Application.LoadLevel(mainMenuLoad); //Loads up the main menu form game-over menu
}
```

Since the game is transitioning from the play-state to the game-over menu, I need to reference my Game-over script in my Game Manager script.

```
public GameOverMenu gameOverDeathScreen; //referencing GameOverMenu script.
```

In my function where the game manager restarts the game, I commented out the usual statement that activates restart-coroutine. When I comment something, it means that it will not be read by the script. This is due it being replaced with a game-over menu. It may have a use in the future, so I have not removed it. This is shown below:

```
public void RestartGame() //can be called from another script
{

    //StartCoroutine ("RestartGameCo"); //(Not being used at the moment)
}
```

I added the 2 statements from my restart-coroutine function and in the red box underneath, I have set my game-over screen to true. This means that it activates the death-menu and displayed it to the player. This statement is shown below:

```
aScoreManager.scoreIncrease = false; //Score stops increasing
myPlayer.gameObject.SetActive(false);
gameOverDeathScreen.gameObject.SetActive(true); //turn game-over menu on
```

I created a new function 'Reset' which performs the actions that restart the game. I copied the statements from my restart-coroutine function and added them here as I am not immediately restarting the game after the player dies. The Reset function is shown in the image below:

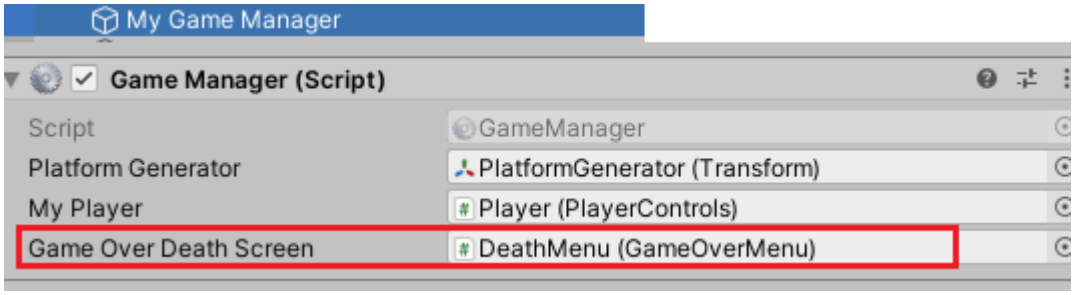
```
public void Reset()
{
    myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
    for (int i = 0; i < myPlatformList.Length; i++) //Goes through the array of active platforms
    {
        myPlatformList[i].gameObject.SetActive(false);
    }
    myPlayer.transform.position = playerSpawnPoint;
    platformGenerator.position = pgSpawnPoint; //sets the position back to the start
    myPlayer.gameObject.SetActive(true);

    aScoreManager.scoreCount = 0; //Resets score to 0
    aScoreManager.scoreIncrease = true; //Score starts increasing again
}
```

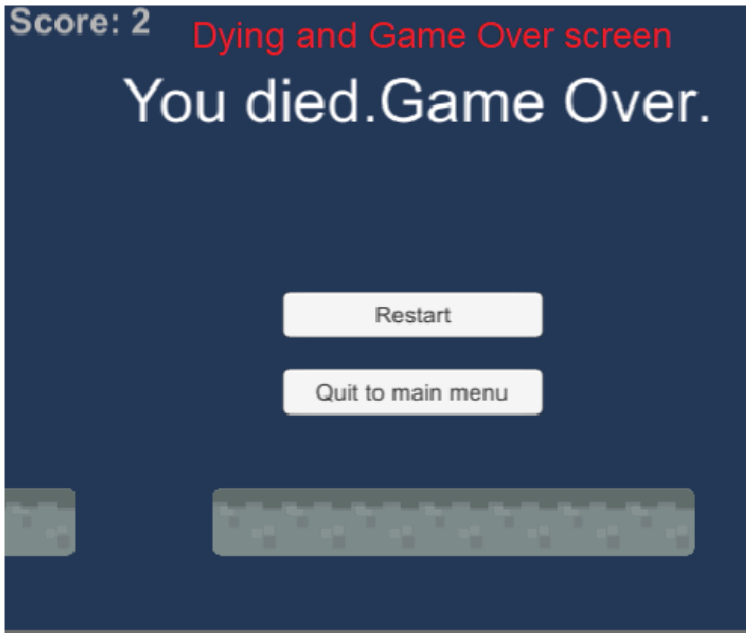
Since I am not using the restart-coroutine that I programmed earlier in development for restarting, I have commented it out. Commenting a statement out means that's not read by the script. The statements in this coroutine have been copied to the 2 functions that I mentioned previously: 'RestartGame' and 'Reset'.

```
/*public IEnumerator RestartGameCo()
{
    aScoreManager.scoreIncrease = false; //Score stops increasing
    myPlayer.gameObject.SetActive(false);
    yield return new WaitForSeconds(1f);
    myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
    for (int i = 0; i < myPlatformList.Length; i++) //Goes through the array of active platforms
    {
        myPlatformList[i].gameObject.SetActive(false);
    }
    myPlayer.transform.position = playerSpawnPoint;
    platformGenerator.position = pgSpawnPoint; //sets the position back to the start
    myPlayer.gameObject.SetActive(true);

    aScoreManager.scoreCount = 0; //Resets score to 0
    aScoreManager.scoreIncrease = true; //Score starts increasing again
}*/
```

I saved the scripts and selected my DeathMenu game-object. I copied and pasted the name of Main menu into the 'Main Menu Load' box which means that the game knows where to transition to if the player decides to quit the main menu.

In my Game Manager game-object, I dragged my DeathMenu game-object into my game manager (shown in the image below).This should mean that when I run the game, the game manager will transition from the play-state to the game-over menu.





When I tested the game, I let the player run for a little bit and then let it die. The screen went darker and the game over menu popped up with the 2 buttons that allows the player to restart the game or quit to the main menu. The game has successfully transitioned from the play-state to the menu which means i can now go on to add functionality to the buttons.

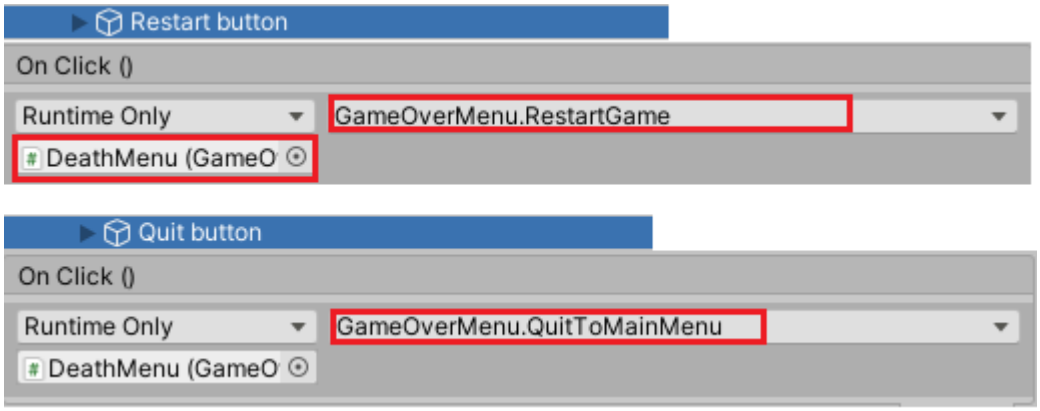
What is being tested?	Input	Justification	Expected outcome	Actual outcome
Game Over Screen activating.	None from the player.	The player should be given the choice if they wish to play again or go back to the main menu when they want to stop playing.	The game shows the game-over menu when the player dies.	The game shows the game-over menu when the player dies.

07/02/2021

Adding functionality to the game-over menu's buttons

Today, I'll be assigning a function to each button on the game-over menu which allows the player to navigate through the game after they die.

On each button, I Scrolled down to the 'On Click ()' feature in Unity and dragged the DeathMenu game-object into each button. This is so that I can assign the functions that I created in my 'GameOver' script. I assigned the 'RestartGame' function to the restart button and the 'QuitToMainMenu' to the quit button.



However, I have not referenced my functions from my GameManager script into my GameOver script, so I'll have to do that in order for my buttons to work.

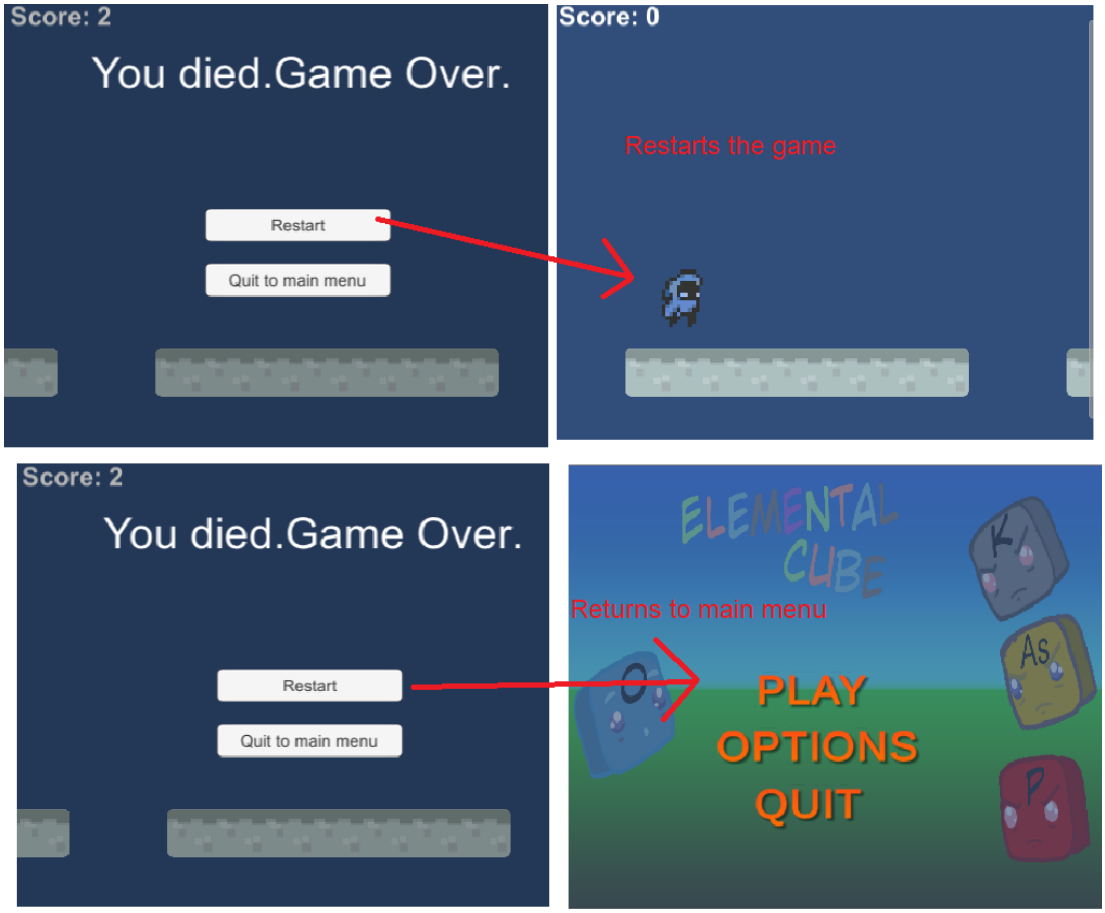
In the 'RestartGame' function, I added the 'Reset' function from the game manager into the GameOver script by the 'FindObjectOfType' method. This will allow the script to perform the function, that resets the game, from the GameManager script. This is shown below:

```
public void RestartGame()
{
    FindObjectOfType<GameManager>().Reset(); //Calls the reset function on game manager
}
```

In my GameManager script, I created a new line where I set the DeathMenu to false, which means that it deactivates again when the player chooses to restart the game.

```
public void Reset()
{
    gameOverDeathScreen.gameObject.SetActive(false); //turn game-over menu off
```

When I tested the game, I let the score accumulate for a little bit and then let the player fall off the platform. The game-over screen showed up and allows me to either restart the game or quit to the menu if I wanted to (shown in the image below). I was able to successfully create a game-over menu which has functionality to it. I'll improving the look of the game-over menu to how it was in the design-stage, later on in development. I am on step closer to my proposed solution. Testing images shown below:



What is being tested?	Input	Justification	Expected outcome	Actual outcome
Restart Button.	Left mouse click.	Allows the player to restart the game if they wish to and play again.	The game is restarted, and the player's score is 0.	The game is restarted, and the player's score is 0.
Quit to the main menu Button.	Left mouse click.	Allows the player to return to the main menu if they do not wish to restart.	The game goes from the play-state to the main menu scene.	The game goes from the play-state to the main menu scene.

Test Data: Game over menu.

Test Data	Expected	Outcome
Left Mice Click (LMC) on Restart button.	Valid.	Valid.
Left Mice Click (LMC) on Quit	Valid.	Valid.

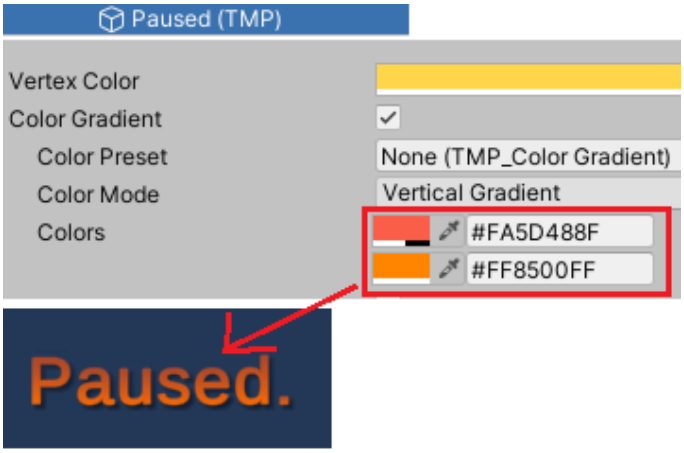
button.		
w	Invalid.	Invalid.
#	Invalid.	Invalid.

10/02/2021

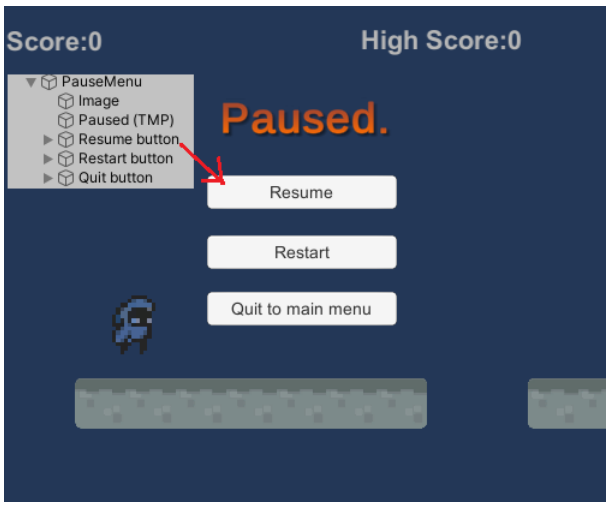
Adding a pause menu into my game

Adding a pause menu is an idea that I proposed in my design. If my stakeholders wish to take a small break from the game or must do something outside of the game, they can activate the pause menu to do so.

My pause-menu will have similar features from my death (game over) menu; therefore, it would be easier to duplicate the death menu and rename it to 'PauseMenu'.



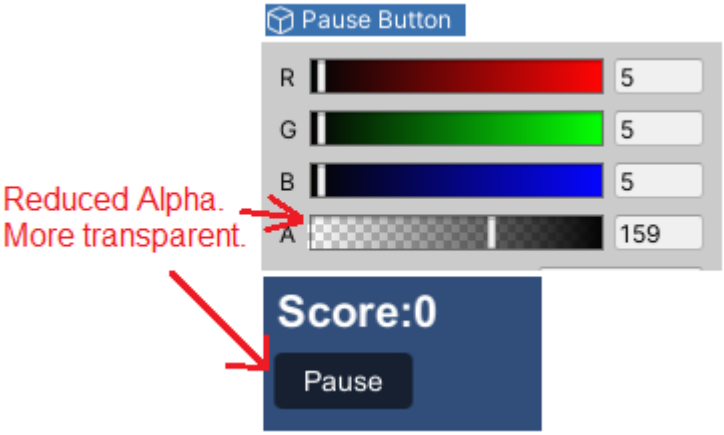
I created a new text object in my pause menu and named it 'paused'. I added a colour gradient and used the same colours that I used in my main menu to keep consistency. This will make it apparent to the stakeholder that they have paused the game.



I created a new button which the player will press to resume the game. This will be the first option that will be available to the player in case if they accidentally pause the game.

Note: After giving it some thought, I decided that I will replace the idea of pressing the ESC key to pause the game to pressing a pause button that is on-screen. I think it's more time

effective as I am more aware of linking buttons to menus than linking keyboard keys to menus.



In the image shown to the left, I have created a new button named 'Pause Button'. This is what the player can press if they want to pause the game and will be placed underneath the score text. I have gone into its colour settings and have reduced the alpha channel so that it's more transparent and won't stand out to distract the player.

I created a new script and named it 'Pause Menu is where I will program how the pause menu will work such as adding pause and resume functions.

```
public class PauseMenu : MonoBehaviour
{
    // Start is called before the first frame upda
    public string mainMenuLoad;

    public GameObject pauseMenu;

    0 references
    public void PauseGame()
    {

    }

    0 references
    public void ResumeGame()
    {

    }

    0 references
    public void RestartGame()
    {
        Time.timeScale = 1f;//Time is set back to
        pauseMenu.SetActive(false);//Deactivates p
        FindObjectOfType<GameManager>().Reset(); /
    }

    0 references
    public void QuitToMainMenu()
    {
        Time.timeScale = 1f;//Time is set back to
        Application.LoadLevel(mainMenuLoad); //Loa
    }
```

Since the pause menu will be sharing some of the features that the game-over menu has, I copied-and-pasted the same functions that the game-over menu has (shown in the red boxes on the left).

Above that I created 2 new functions that will be exclusive to the pause menu which is 'PauseGame', which pauses the game and 'ResumeGame' which will allow the player to resume the game after pausing it. The two mentioned functions are shown in the red underline in the image to the left.

```
public GameObject pauseMenu;
```

The Game-Object 'PauseMenu' is a class that will carry out the functions assigned to the 'PauseMenu' script.

In my 'PauseGame' function, I referenced the time class and used the 'Timescale' method to set the time to '0f'.This means that the game will slow down form '1f' which is the normal

speed of the game to being frozen(0f).This is something I need to use to make the pause menu effective as it'll essentially freeze the game so the player won't carry on moving after the player has clicked the pause button. This is shown below:

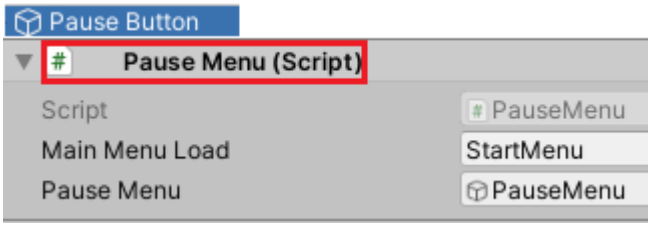
I also used the Set Active method and set it to true which means that the pause menu will activate when the Pause Game function occurs.

```
public void PauseGame()
{
    Time.timeScale = 0f;// Game pauses,player won't move
    pauseMenu.SetActive(true); //Activates pause menu

}
```

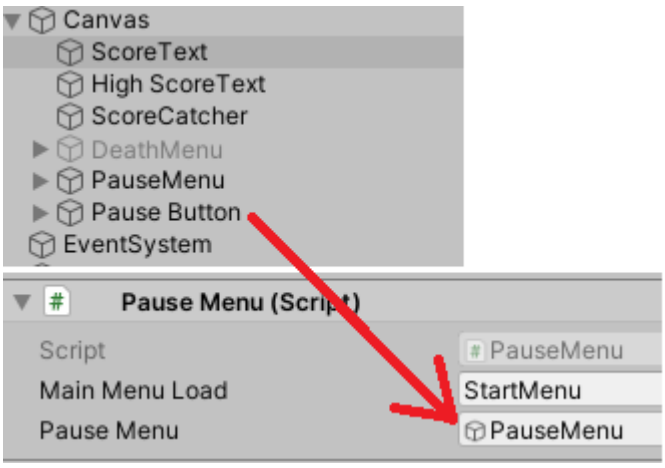
13/02/2021

In the 'Resume Game' function (shown below), I used reverted the speed of the game back to normal by changing the speed from '0f' to '1f' in the 'Time' class, so that the game will not be frozen and will run normally. The first statement is also present in 'Restart' function and 'Quit' function in this script. Underneath that statement, I used the 'Set Active' method on the 'pause Menu' class again and set it too false. This should mean that when the player chooses to resume the game, the 'Resume Game' function will occur, and the pause menu will deactivate. The 'Resume Game' function is shown in the image below:

```
public void ResumeGame()
{
    Time.timeScale = 1f;//Time is set back to normal
    pauseMenu.SetActive(false);//Deactivates pause menu
}
```

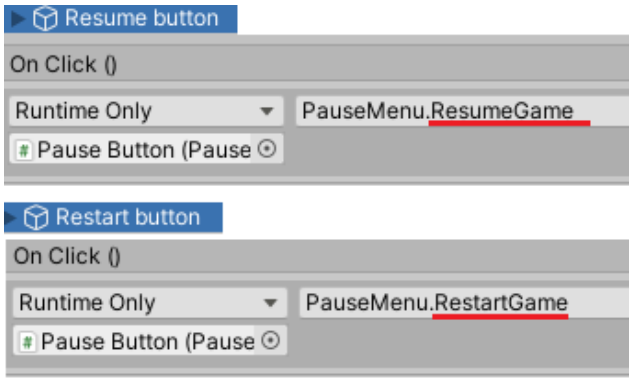


I saved the script then went back into Unity and added the Pause Menu script onto my Pause Button (Shown on the left). This is due to the Pause Menu only being read by the game when the Pause Button is pressed.



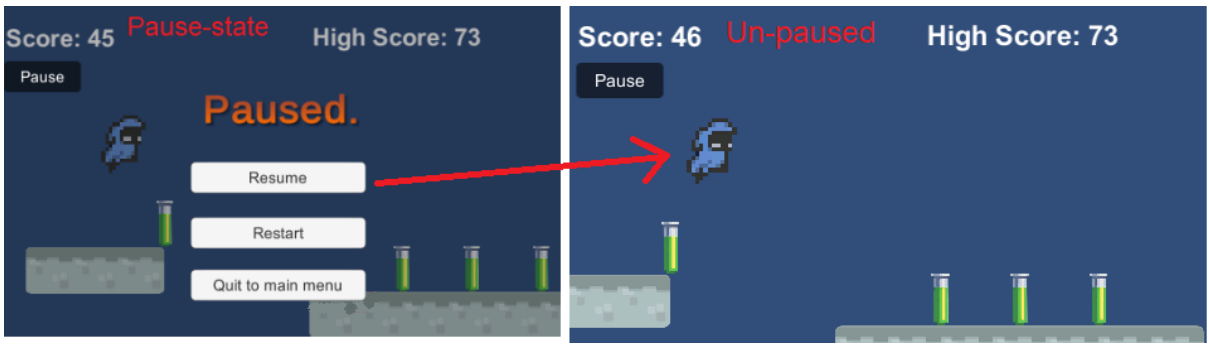
Also, on my Pause Button, I added the 'PauseMenu' game-object which contains all the buttons that will show in the pause menu. This means that the script will be able to activate and deactivate the pause menu.

On each button I scrolled down to the 'On Click' feature and added the respective method to each button (shown below in red underline), like what I did for the buttons in the game-over menu.

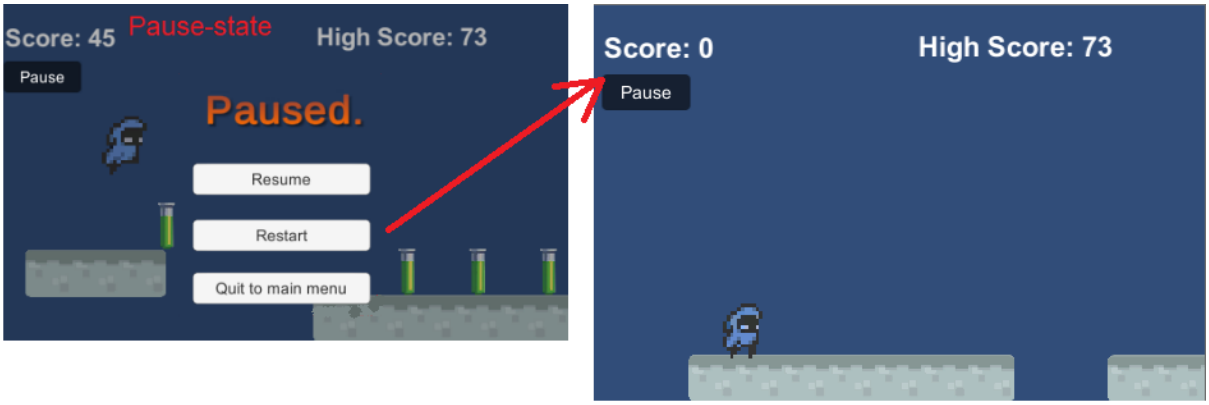


When testing the pause menu, I let the player stack up some points and I clicked on the pause button. It successfully brought up the pause menu with all the buttons need. It also froze the game, so that the player does not die.

I decided to test the 'Resume' button out first. When I clicked on it, the pause menu successfully deactivated, and the game was running again after being frozen. This is shown in the image below:



I tested the restart button again to make sure it still worked like it did in the game-over menu and it did. When selected it successfully placed the player back at the start and reset the score to 0. This is shown image below:



I tested the Quit-to-main-menu button and it left the game and successfully changed the game-state to the main-menu. With the success of this, I now know that I created a pause-menu that has functioning buttons.

I think the end-result of the pause screen satisfies most of the needs that have been requested by my stakeholders in the analysis stage. The aesthetics of the pause menu and the game-over menu may be improved later in development.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Pause button.	Left mice click from player.	Allows the player to pause the game if they wish to take a break, restart, or go to the main menu.	The pause menu is activated and is displayed for the player and the game freezes.	The pause menu is successfully activated and is displayed for the player when it is clicked and the game freezes.
Resume button	Left mice click from player.	Allows the player to return to the	The pause menu is	The pause menu is

		game from the pause menu.	deactivated, and the game runs again from being frozen.	successfully deactivated, and the game runs again from being frozen.
Restart button	Left mice click from player.	Allows the player to restart the game if they wish to and play again.	The game is restarted, and the player's score is reset to 0 and the game unfreezes.	The game successfully restarts, and the player's score is reset to 0 and the game unfreezes.
Quit to main menu button	Left mice click from player.	Allows the player to return to the main menu if they do not wish to restart.	The game goes from the play-state to the main menu scene.	The game successfully goes from the play-state to the main menu scene and the game unfreezes.

Test Data: Pause menu.

Pause

Test Data	Expected	Outcome
Left Mice Click (LMC) on Pause button.	Valid.	Valid.
'ESC' key	Valid.	Invalid
w	Invalid.	Invalid.
#	Invalid.	Invalid.

Un-pause.

Test Data	Expected	Outcome
Left Mice Click (LMC) on Resume button.	Valid.	Valid.
'ESC' key	Valid.	Invalid

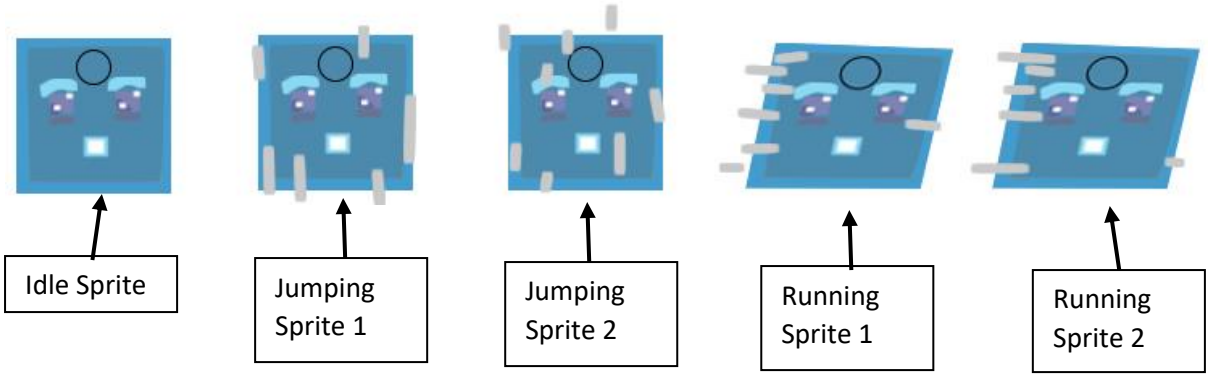
w	Invalid.	Invalid.
#	Invalid.	Invalid.

20/02/2021

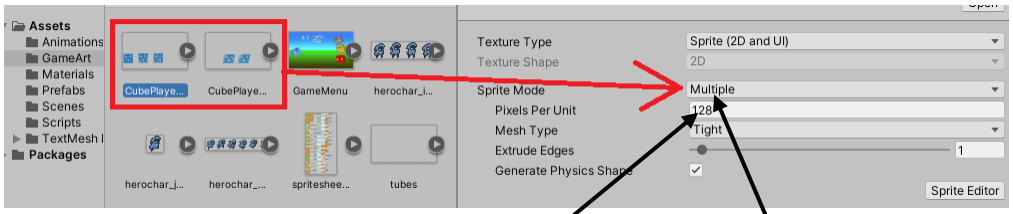
Adding my own playing sprite into the game

At the start of my development, I decided to use royalty-free sprites so that I can focus on the main mechanics of the game before heading into the looks of the game. Since I have completed most of the mechanics needed my game, I am going to use my own sprites now.

I created sprites like the ones I created during the design-stage of the project. Each sprite is for a different animation and they have been put together in a transparent sprite sheet. The sprite sheet is shown below:



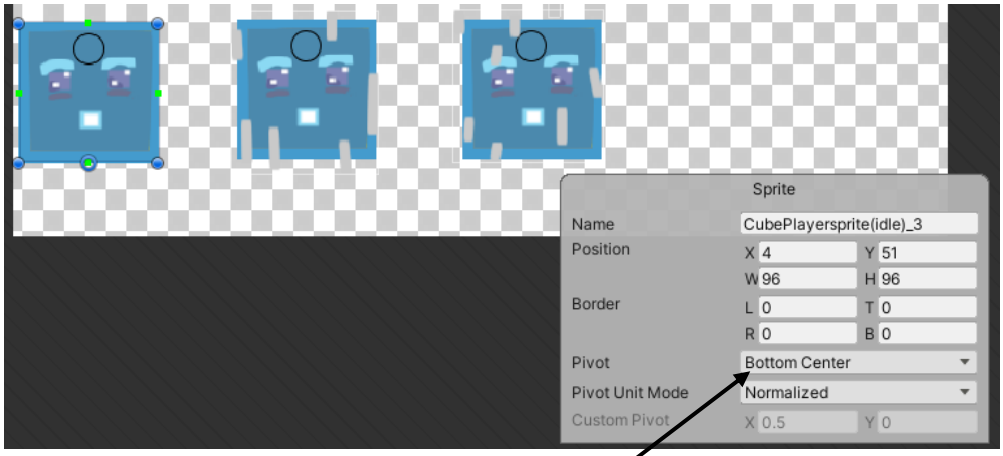
I have dragged my player-sprite from my game into the 'GameArt' folder of Unity, where I keep all my other sprites, and have set the sprite mode to multiple. This is shown in the image below:



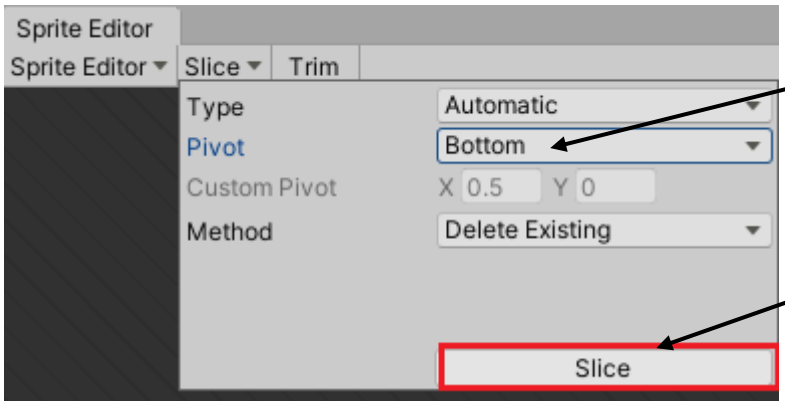
I set the Pixels Per Unit to 128 as it is the It is expressing how many pixels from my Sprite will fit into Unity's game object scaling. Since my other sprites use 128 pixels per unit, I am setting the same value for consistency.

Setting the Sprite mode to multiple allows me to separate the sprites from the sprite sheet. This will allow me to use each sprite individually when it comes to animating the player.

I opened the sprite editor to select my sprites. I made a box around each sprite that will be used in the game. I set the pivot of each sprite at the bottom, as that is the point of the sprite that will be meeting the platforms.



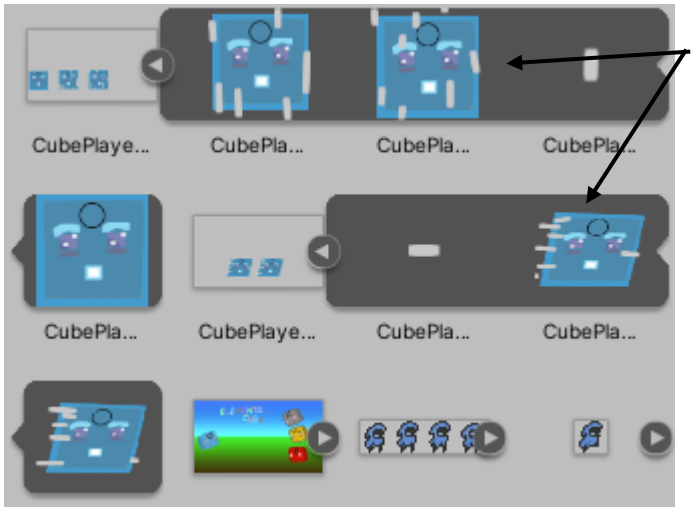
The green box at the bottom centre of the sprite is the pivot of the sprite.



The pivot is set at the bottom in the slicing option as well, to make sure that the pivot is set at the bottom.

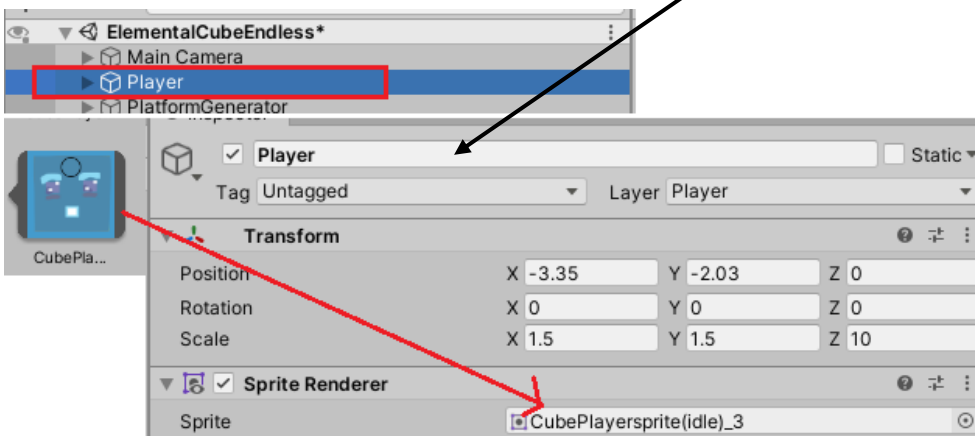
The Slice option allows me to separate the sprites from the sprite sheet for future use.

After slicing each sprite, so that they are separated, I click on 'Apply' so that the preferences I selected are saved and applied onto the sprites. The preferences here, is the pivot's location on the sprite.

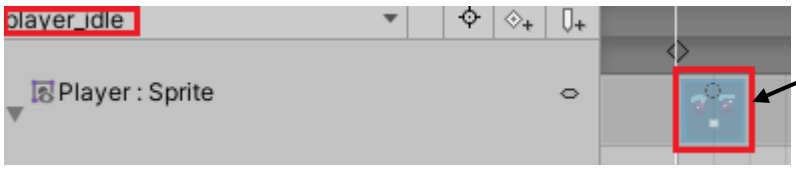


My sprites have been separated after applying my preferences and slicing each sprite.

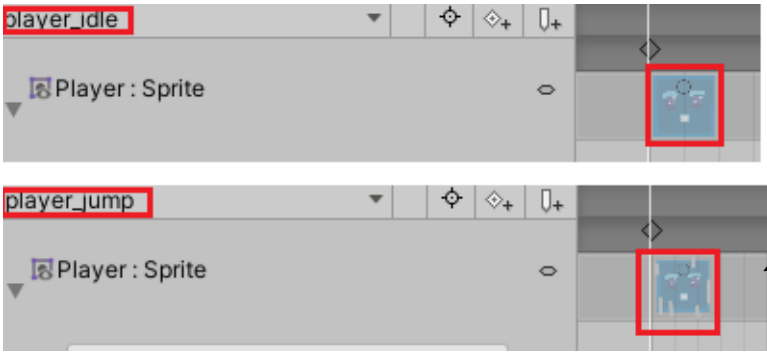
I selected the player game object which was using the royalty-free sprite and selected its sprite renderer.



In Unity's animator, I selected on the old animation for the player being idle and changed it to the new idle sprite. This is to prevent the game form using the old sprite in the animation.

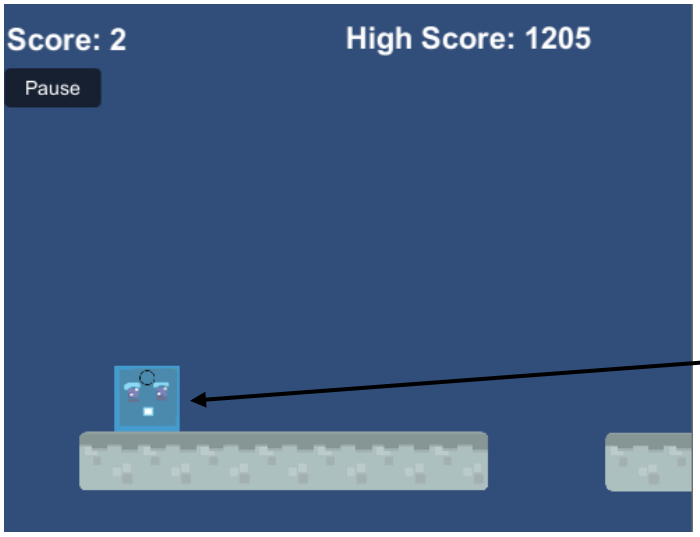


I dragged the idle player sprite from the sprite sheet into the sprite renderer so that it is the



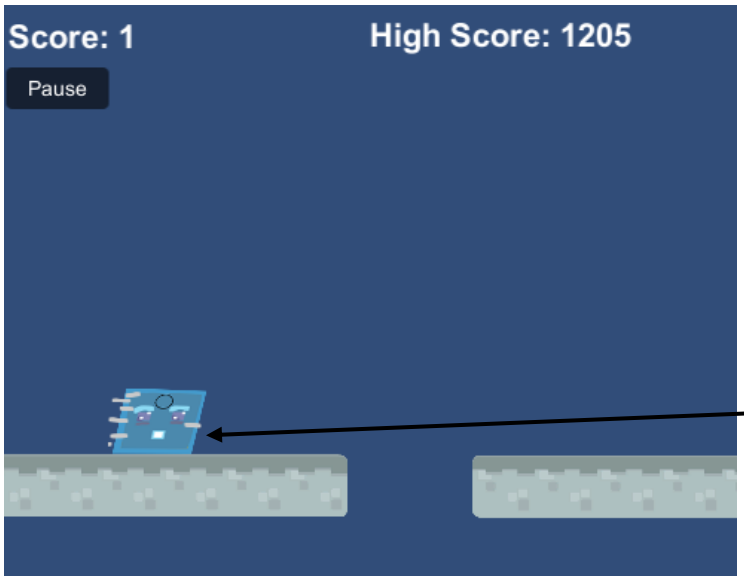
For the jumping animation, I selected on the old animation for the player jumping and changed by using the new jumping frames. This should mean that this new animation should appear when the player is jumping.

For the running animation, I selected on the old animation for the player running and changed by using the new running frames. This should mean that this new animation should appear when the player is running.



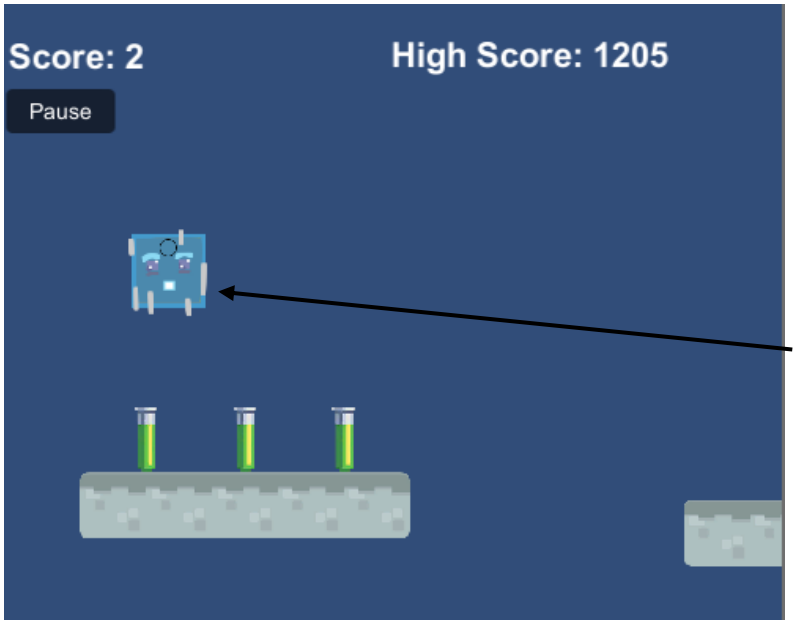
When testing the game, I did not touch any of the player controls so that I can see if the player's sprite remains as the one, I placed in the animator.

The player uses the idle sprite animation when the player is not moving. This is shown to the left.



When testing the running, I pressed on the 'D' key to see if the player game object uses the animation frames that I placed in the animator for the running animation.

The player uses the running animation sprite when I click the 'A' or 'D' key. This is shown on the left.



When testing the jumping, I pressed on the 'W' key to see if the player game object uses the animation frames that I placed in the animator for the jumping animation.

The player uses the jumping animation sprite when I click the 'W' key. This is shown on the left.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
When the Player moves left or right, the player's sprite will transition into the running animation.	'A' or 'D' key.	Animation brings life into a game and makes it look less static and boring. It acts as a clear visual indicator that the player is moving.	The player sprite will transition into a running or jumping animation depending what key, for the controls, is being pressed.	The player sprite transitions into a running or jumping animation depending what key, for the controls, is being pressed.
When the Player moves up and down, the player's sprite will transition into the jumping animation.	'W' key.	Animation brings life into a game and makes it look less static and boring. It acts as a clear visual indicator that the player is moving.	The player sprite will transition into a Jumping animation if the 'W' key is being pressed.	The player sprite transitions into the jumping animation if the 'W' key is being pressed.

Testing the player's movement animations:

Test data	Expected	Actual
'A' key transitions player to running animation.	Valid.	Valid.
'D' key transitions player to running animation.	Valid.	Valid.
'W' key transitions player to jumping animation.	Valid	Valid.
'1' key	Invalid.	Invalid.
'Space bar'	Invalid.	Invalid.

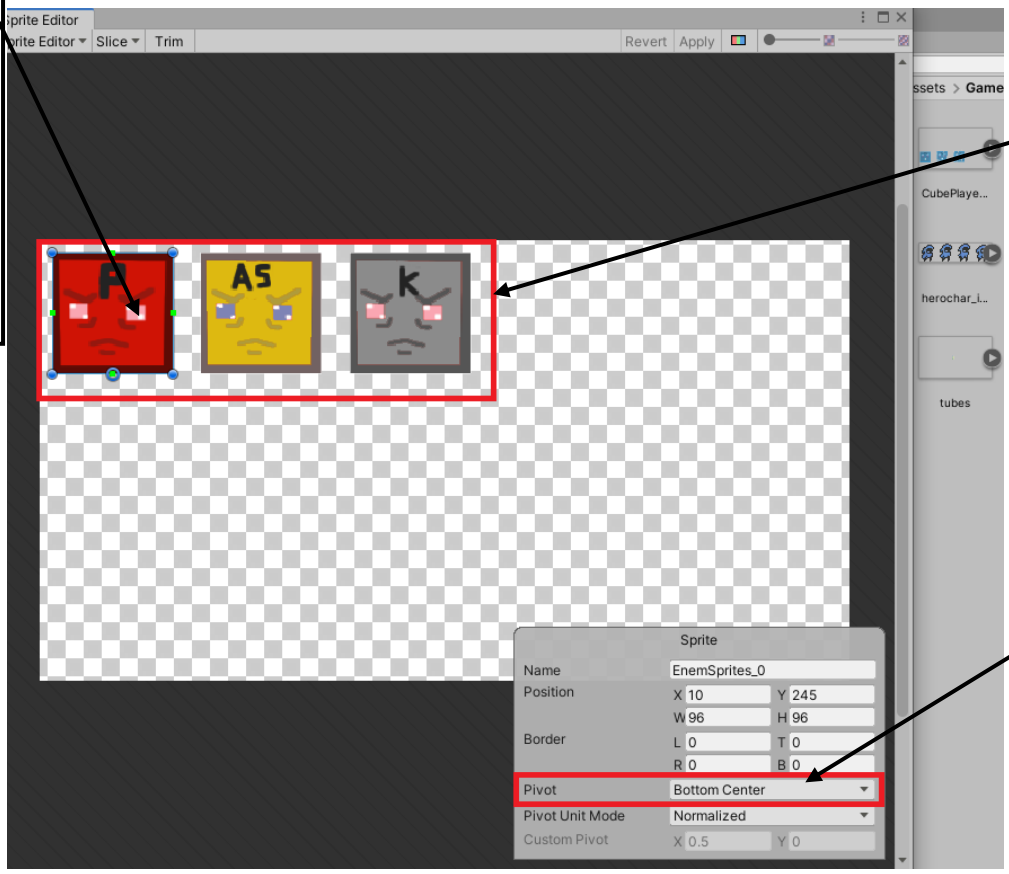
1/03/2021

Implementing enemies into my game

I wanted enemies to be part of my game since the analysis stage and have them be an extra challenge to the player. I created a mock-up of how they would look during gameplay. Today I will be trying to implement enemies into my game with the aim that they will cause the player to die.

I created 3 enemy sprites in a drawing software and changed the colour scheme and symbol so that they show different elements (e.g., 'k' is potassium). These are the sprites that will be used in the game.

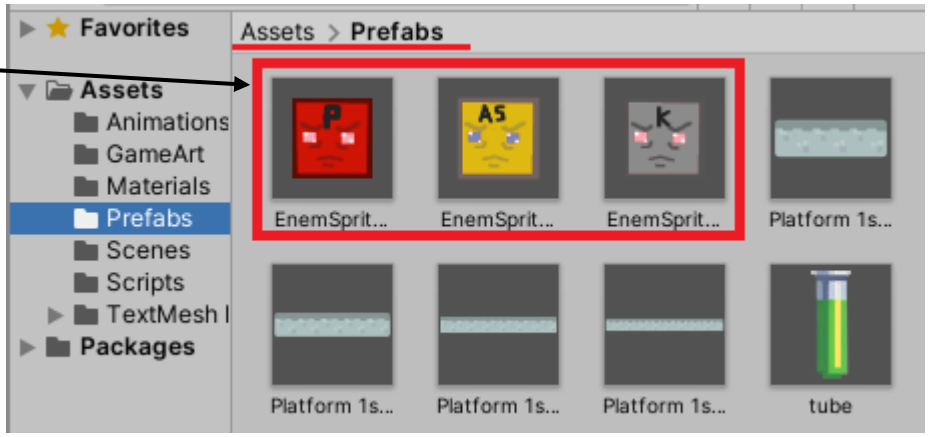
I drew squares around the sprites so that the sprite editor will not cut out anything outside the sprites.



I uploaded the png file of the enemy sprites into Unity's editor and opened the sprite editor where I can separate the sprites.

After I selected a sprite, I create a pivot at the bottom of the sprite. This will be the point at which the sprite will meet the platforms.

After separating the sprites, I uploaded them into my 'prefabs' folder where I keep all my other re-usable game-objects.



I am re-using a void function, from my tube generator script, for spawning enemies, 'SpawnEnemy', as this function will only be called at certain times during gameplay. The vector position, 'startPoint', will be the default position of where an enemy will spawn on each platform.

```
public class EnemyGenerator : MonoBehaviour
{
    // Start is called before the first frame update
    public ObjectPooler enemyPool; //Used to access object pool for tubes


    1 reference
    public void SpawnEnemy(Vector3 startPoint)
    {
        GameObject enemy1 = enemyPool.GetPooledObject();
        enemy1.transform.position = startPoint;
        enemy1.SetActive(true);

    }
}
```

I created a new class called 'enemypool' which will be using my object pooling script.

I create a class for my enemies, named 'enemy1' and I will use the '. GetPooledObject' method to create it as a game object in the game.

Since enemies are spawning on my platforms, I must reference my enemy generator script in my platform generator scrip for this to work. I named the class 'myEnemyGenerator'. I went on to the Start void and I used the 'FindObjectOfType' method on 'myEnemyGenerator' so that at the start of the game, any object that has the enemy generator script attached to it will be used to spawn on the platforms.

```
private EnemyGenerator myEnemyGenerator;

void Start()
{
    //widthOfPlatform = aPlatform.GetComponent<BoxCollider2D>().

    widthOfPlatforms = new float[myObjectPools.Length]; //create
    for (int i =0;i< myObjectPools.Length;i++)
    {
        widthOfPlatforms[i]= myObjectPools[i].thePooledObject.Ge
    }

    minimumHeight = transform.position.y; //Getting the default
    maximumHeight = pointOfZenith.position.y; //Max height of wh

    myTubeGenerator = FindObjectOfType<TubeGenerator>(); //Finds
    myEnemyGenerator = FindObjectOfType<EnemyGenerator>();
}
```

To make this easier, enemy spawning will follow the same idea that the random tube generation follows. I added a float variable into the platform generator script to make the chance of the enemies spawning more random.

```
private EnemyGenerator myEnemyGenerator;
public float enemyChanceLimit;
```

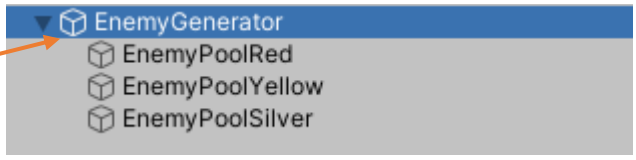
This float variable, 'enemyChanceLimit', will be the chance that an enemy will spawn on a platform.

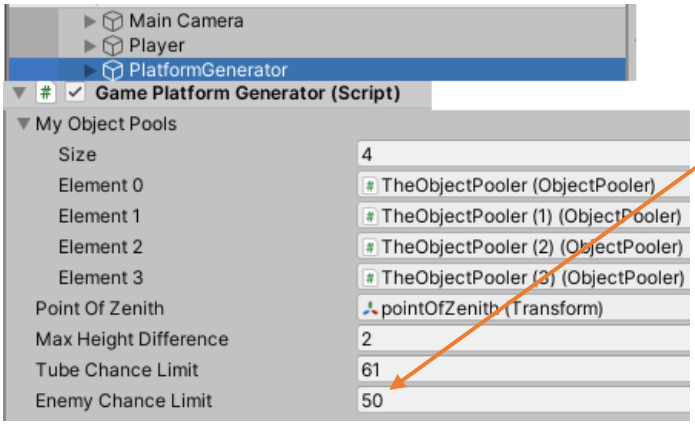
I am going to use the "Chance system" again to determine if an enemy will be spawned onto a new platform at any given time.

```
if (Random.Range(0f, 100f) < enemyChanceLimit)
{
    myEnemyGenerator.SpawnEnemy(new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z));
}
transform.position = new Vector3(transform.position.x + (widthOfPlatforms[thePlatformSelector] / 2) , transform.position.y, transform.position.z);
```

In the 'if' statement I used the 'Random.Range' function to select a number between 0 and 100.If the selected number is less than the value assigned to 'enemyChanceLimit', then tubes will spawn onto a platform.

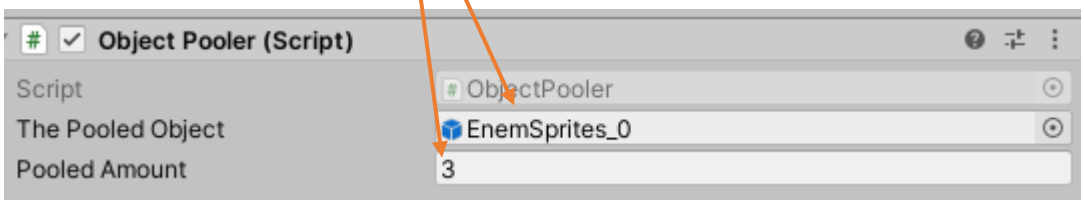
I created a game object in unity and named it to 'EnemyGenerator'. This is what the enemy script will be attached to and is what the script will use to spawn the enemies.





I selected the platform generator game object and set the Enemy Chance Limit to 50. This means that there is a 49% chance of an enemy spawning.

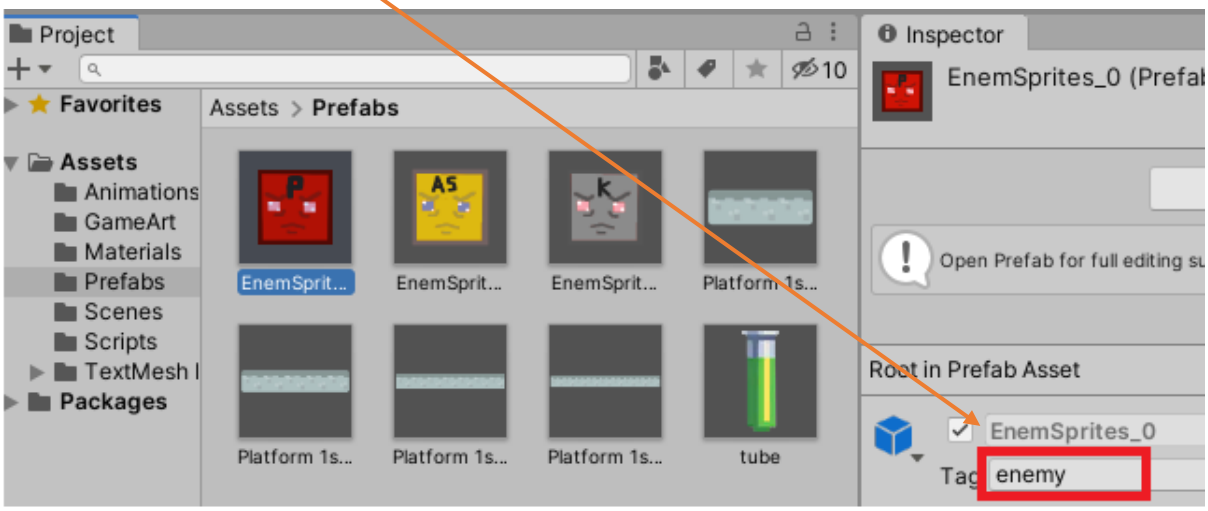
I am going to use the red sprite for now so that I can see if enemies are able to spawn. I attached the object pooler script to my red sprite and set the pooled amount to 3. I am only having 3 red sprites available in the game due to the chance being of spawning is 50%.



To make sure that the enemies influence the player during gameplay, I went into the player controller script and added a statement in the section where the player dies on contact with an object. The statement in the red box means that when the player comes collides with a game object called “enemy”, the game will transition to the game over menu to indicate the player has died.

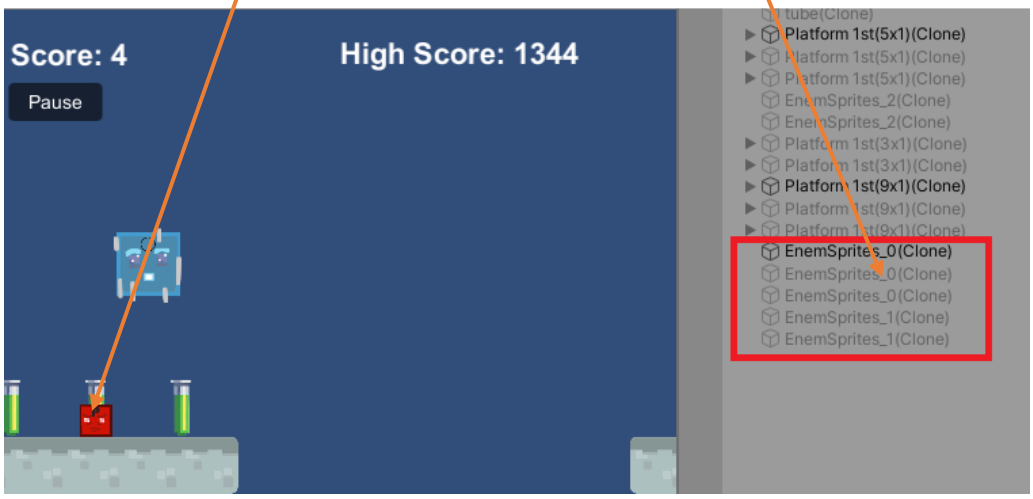
```
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "killbox" || other.gameObject.tag == "enemy")
        {
            theGameManager.RestartGame();
            playerSpeed = playerSpeedStore;
            velocityLandmarkCounter = velocityLandmarkCounterStore;
            velocityLandmark = velocityLandmarkStore;
    }
```

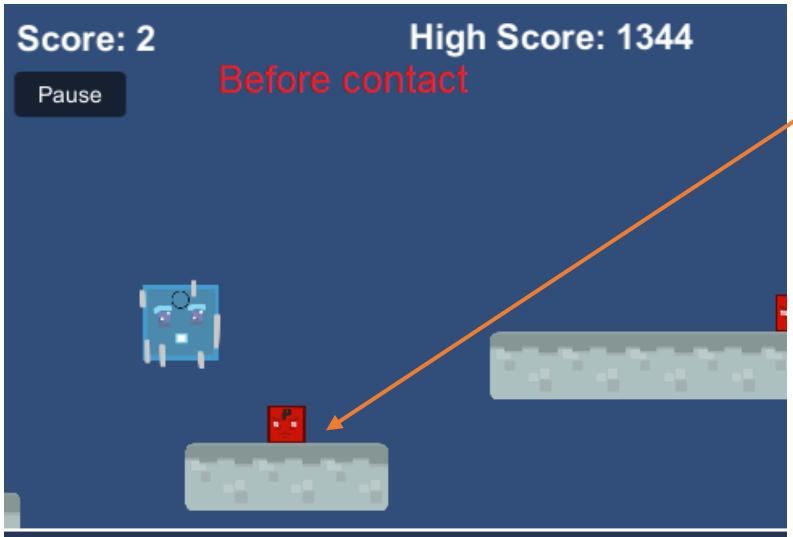
I created a tag called “enemy” and attached it to the enemy sprites. When the player collides with those sprites, the game should end, and the player is redirected to the game. over menu.



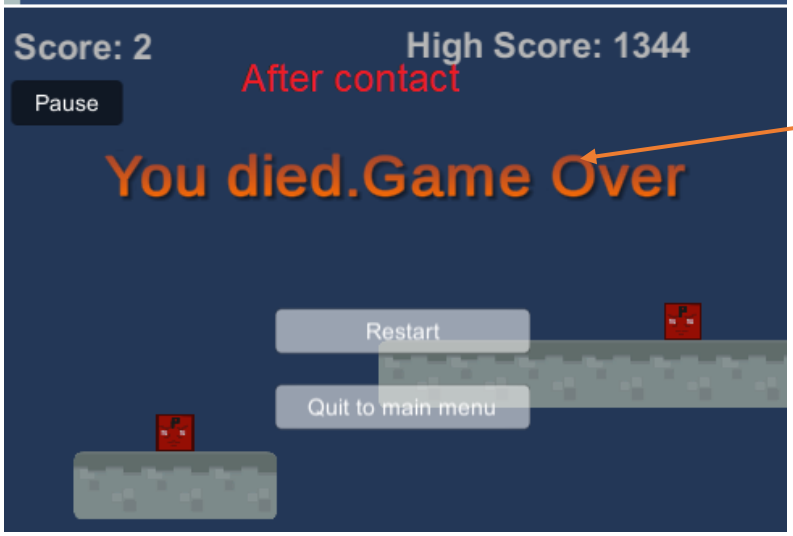
Testing enemy spawn

When testing the game, I moved the player forward and looked at the Unity editor and saw the game using object pooling for the red enemies. The enemy spawns in the middle of the platform





When testing if the enemies cause the player to die, I waited for an enemy to spawn on one of the platforms in front of the player.



After encountering the enemy and colliding with it, the player dies and the game over menu is loaded. This was one of the features that I wanted in my analysis, followed up by a product screen.

Test Data: Random enemies spawning.

Test Data	Expected	Actual
Enemies do not appear on every platform.	Valid.	Valid.
Enemies spawn anywhere.	Invalid.	Invalid.

What is being tested?	Input	Justification	Expected outcome	Actual outcome
Enemies spawn randomly.	None from the player.	An extra challenge for the player and introduces elements into the game.	Enemies randomly spawn on all platforms.	Enemies randomly successfully spawn on all platforms.
Player dies when comes into contact.	None from the player.	Poses a threat to the player and keeps them engaged in the game.	Player dies when they collide with an enemy.	Player dies when they collide with an enemy.

I think the enemy spawning turned out well but there are improvements to be made to it such as spawning random enemies and programming the game to transition from a player colliding with an enemy to the product menu that I stated I wanted to implement since my Analysis and Design stages. However, due to time limitations, I will not be able to implement some of the features that I was intending to such as power-ups, the product menu, options menu, music/sfx.

I will be conducting an interview with my key stakeholders and ask about their opinion of the game.

03/03/2021

Interview with key-stakeholders

Me:

Most features that I stated in my analysis and design, were coded, and implemented into my game; with the main exceptions being the power-ups, the product menu, options menu, and music. I have implemented the high-score counter, main menus, pause menus and game over menus as well as enemies have been implemented into the game since I last

contacted the both of you. I feel like I have created the fundamental features of an endless side scroller.

Azim Khanazada:

Yeah, I can agree with you on that the game is resembling an endless runner. The high-score counter what just as what I expected from the last email you sent me, and the menus are easy to navigate through as well. I believe you should stop coding the game and move on with your project as you do not have much time.

Although I was looking forward to seeing the product screen feature being implemented, I think that you progressed well with the game.

Hamid Choudhury:
The game is fun to play with the difficulty curve and the enemies spawning. I am impressed with how you were able to have the menus transition the game and the high-score counter that you mentioned was alright. I wanted to see the power-ups, but I think it is better to stop programming and focus on the rest of your project.

Evaluation

Testing for the evaluation

Menus

This section will test the navigation of buttons in menus through the game. Due to time constraints, I did not implement the options menu. The leader board feature was replaced with a high score counter.

No.	Tests	Input	Expected outcome	Actual outcome
1.	Menu with the title and the 3 options being displayed which are Play, Options and Quit.	N/A.	The menu will have three options available where the player can go into the game-state, the options menu, and the quit option.	As expected.
2.	Left mice click on the play button to access the play state.	Valid: When in the menu state, the left mice button is clicked.	Valid: The game state will change from the menu state to play state.	As expected.
3.	Pressing the pause button to enter the pause state.	Valid: When in the play state, the left mice	The game state will change from	As expected.

		button is clicked on the pause button.	the play state to the pause state.	
4.	Pressing the resume button with the Left mice click to enter the play state.	Valid: When in the pause state, the left mice button is clicked on the resume button.	The game state will change from the pause state to the play state.	As expected.
5.	Pressing the restart button with the Left mice click to enter the play state.	Valid: When in the pause state or game, the left mice button is clicked on the restart button.	The game state will change from the pause state back to the play state.	As expected.
6.	Pressing the quit to main menu button with the Left mice click to enter the menu state.	Valid: When in the pause state or game, the left mice button is clicked on the restart button.	The game state will change from the pause state back to the play state.	As expected.
7.	Left mice click on the options button to access the options state.	Valid: When in the menu state, the left mice button is clicked.	Valid: The game state will change from the menu state to options state.	This feature was not implemented.
8.	Game over Menu with the title and the 2 options being displayed which are Restart and Quit to main menu.	N/A.	The game over menu will have two options available where the player can restart or quit to the main menu.	As expected.
9.	Left mice click on the Leader board button to access the Leader board state.	Valid: When in the menu state, the left mice button is clicked.	Valid: The game state will change from the menu state to leader board state.	This feature was not implemented.

Screen elements

This section will test the screen elements of the game.

	Tests	Input	Expected outcome	Actual outcome
10.	Score counter is displayed on the top left.	N/A	When the game is initialised, the	As expected.

			score is displayed on the top left.	
11.	Pause button is displayed under the score counter	N/A	When the game is initialised, the pause button is displayed on the top left under the score counter.	As expected.
12.	Tubes are displayed on random platforms.	N/A	Tubes are spawned on to random platform for the player to collect. Chance of spawn is 49%	As expected.
13.	Tubes disappear on contact with the player.	N/A	The tubes disappear when the player collides with it	As expected.
14.	Platforms appear during gameplay.	N/A	The platforms are displayed in play state.	As expected.
15.	Enemies are displayed on random platforms.	N/A	Enemies are spawned on to random platform to challenge the player.	As expected.
16.	High-score counter is displayed on the top right.	N/A	The high score counter is displayed on the top right corner during gameplay.	As expected.

Scoring

This section will test the scoring system that I have implemented into the game. The rank/grade feature was not implemented due to time constraints. The user entering their name into the leader board wasn't implemented due to the leader board idea being replaced with another feature.

	Tests	Input	Expected outcome	Actual outcome
17.	The user's score increases by +1 every second.	N/A	The user's score is increased by 1	As expected.

			ever second and is displayed on the score counter.	
18.	Each tube increases the score's value by +3, on contact.	Valid: Player control keys cause the player to collide with the tubes.	The user's score increases by +3 whenever the player collides with a tube.	As expected.
19.	The player is given a grade depending on the score that they achieved.	Valid: Player score causes the game to respond with a rank.	The game displays a rank depending on what score bracket the player's score falls in to.	This feature was not implemented.
20.	The user entering their name for the leader board.	Valid: When in the end play state, the player uses the keyboard.	Valid: The player uses the keyboard to enter their name between 2 and 10 characters.	This feature was not implemented.
21.		Valid: When in the end play state, the player uses the keyboard.	Invalid: The player uses the keyboard to enter their name between 2 and 10 characters.	This feature was not implemented.
22.	High score counter is updated.	N/A	The high score counter is displayed and is updated if the player's current score is greater than the last recorded high score.	As expected.

Gameplay

This section will test the main gameplay elements in the game.

No.	Tests	Input	Expected outcome	Actual outcome

23.	The player's sprite is displayed in the play state.	N/A	When the game is initialised, the player's sprite is displayed on a platform.	As expected.
24.	The player's running animation is displayed when the player is running.	Valid: The 'D' or 'A' key.	Valid: When either of those keys are pressed, the player's running animation is displayed.	As expected.
25.		Invalid: Any other keystrokes.	Invalid: If any key apart from the 'A' or 'D' key is pressed, the player's running animation will not be displayed.	As expected.
26.	The player's jumping animation is displayed when the player is jumping.	The 'W' key.	When the 'W' key is pressed, the player's jumping animation is displayed.	As expected.
27.	The player dies when they fall off.	N/A	If a player falls off a platform they will die and be directed to the game over screen.	As expected
28.		N/A	The player's rank is displayed, and they are shown the leader board.	This feature was not implemented.
29.	Product screen displays after a reaction with the player's score and rank.	N/A	The game will display the score that the player achieved and the rank the gained.	This feature was not implemented.
30.	The player's speed increases over time.	Valid: Occurs when the 'D' key is pressed.	Valid: The player's speed increases over	As expected.

			time during gameplay while the 'D' (forward) key is pressed.	
31.	Reaction animation is displayed when the player collides with an enemy.	N/A	When the player encounters and enemy there is a reaction animation followed up by the game over screen.	As expected.

Mechanisms

No.	Tests	Input	Expected outcome	Actual outcome
32.	Platforms of random sizes spawning.	N/A	Platforms of varying widths are chosen and spawned into the game in a random order.	As expected.
33.	Platforms spawned at random heights.	N/A	When a platform is chosen, they are spawned into the game at a random height.	As expected.
34.	Random gaps between platforms.	N/A	Platforms are spawned into the game at a random distance between every platform.	As expected.
35.	Platforms are destroyed after a certain point beyond the player.	N/A	After a certain point, platforms are destroyed.	As expected.

Music/SFX

This section will test the music and sound effects features of the game. Due to time restrictions, I was unable to implement any sound.

No.	Tests	Input	Expected outcome	Actual outcome

36.	Music starts playing when the game is loaded.	N/A	When the stakeholder runs the game, music will start playing.	This feature was not implemented.
37.	Music can be toggled off in the options menu.	N/A	The stakeholder can turn off the music when accessing the options menu.	This feature was not implemented.
38.	Sound effects (SFX) can be toggled off in the options menu.	N/A	The stakeholder can turn off the sound effects when accessing the options menu.	This feature was not implemented.
39.	Sound effect for when the player is running.	Valid: The player is running with either the 'A' key or the 'D' key.	Valid; There is a footsteps sound effect for when the player is running.	This feature was not implemented.
40.	Sound effect for when the player is jumping.	Valid: The player is running with the 'W' key.	Valid; There is a wind blowing sound effect for when the player is jumping.	This feature was not implemented.
41.	Sound effect for when the player has collected a tube.	Valid: A 'ping' sound effect for when the player has collected a tube.	Valid; There is a ping sound effect that is played for when the player is running into a tube.	This feature was not implemented.
42.	Sound effect for when the player has reacted with an enemy	Valid: A 'steam/hissing' sound effect for when the player collides with an enemy.	Valid: A 'steam/hissing' sound effect is played for when the player reacts with an enemy.	This feature was not implemented.

Important!

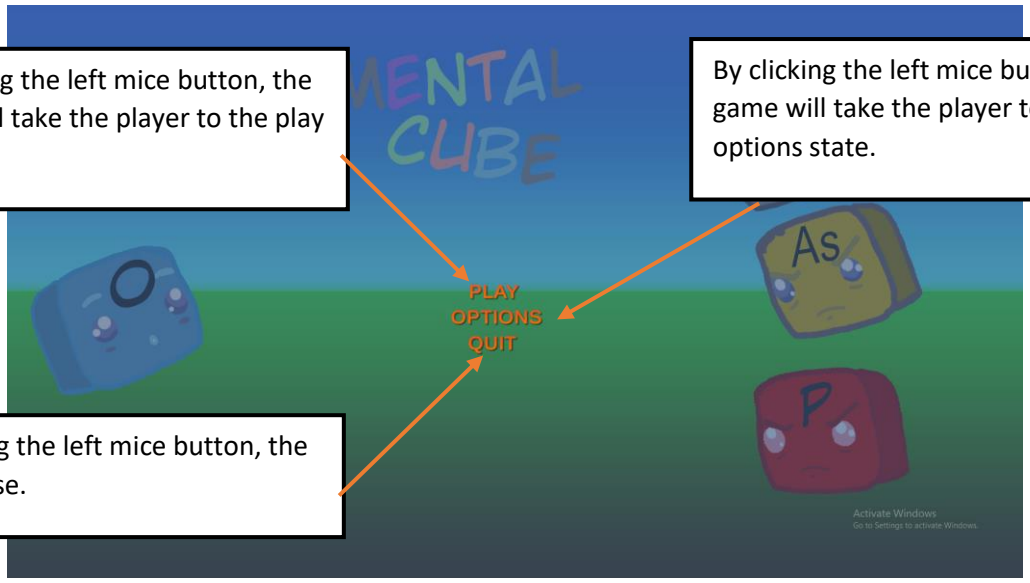
I have created a video showcasing my project where I show how the game works and showing how the game responds to the test data. it is visible by copying and pasting this link into your browser:

https://youtu.be/p3iZXdZ6xv4

Usability features

In this section, I will annotate an image of the game and highlight its usability features. I will explain each section and then get feedback from either of my key-stakeholders: Azim Khanazada or Hamid Choudhury.

Menu Navigation



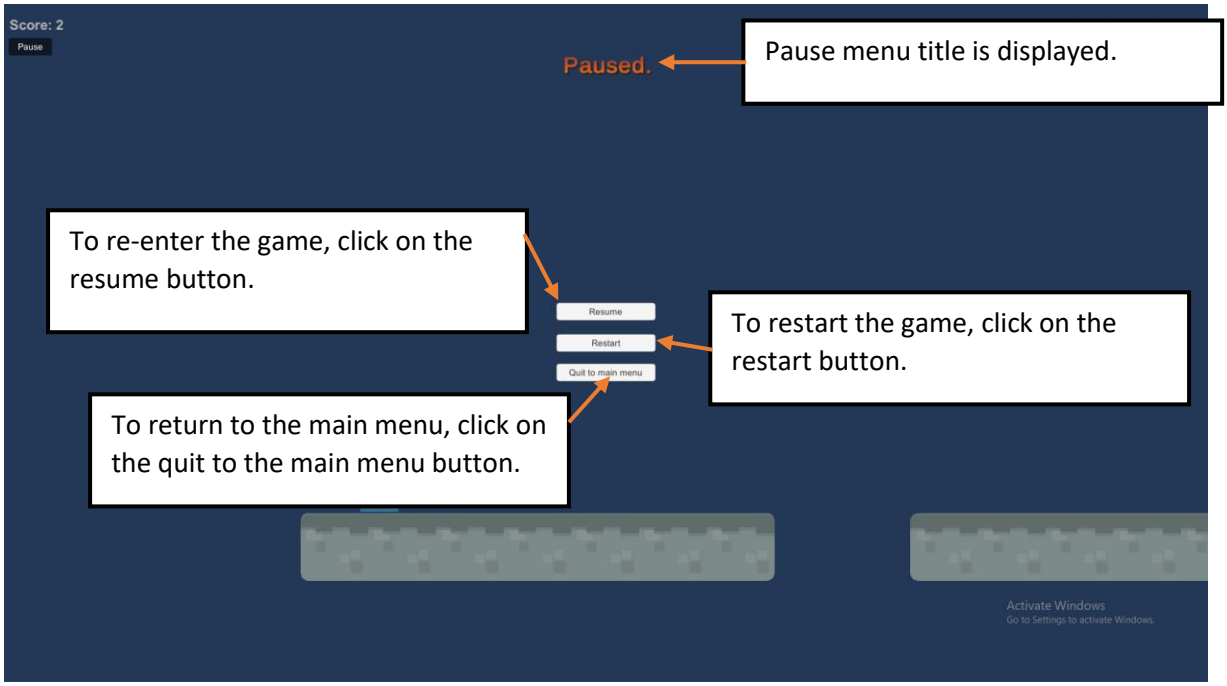
Pause menu

Me:

Here in the menu, the three options are displayed, and they can be clicked on by pressing the left-mice button. Due to time limitations, I was not able to implement the options menu. Given more time, this problem can be easily fixed by implementing music/SFX and being able to toggle it on or off in the options menu.

Azim:

Navigating through the menu was easy because using the mouse was a standard across most games. I understand time limitations but an options menu not being included would not be acceptable in the final game. It would have been nice to also use the WASD keys as well for navigating but the mouse is fine.

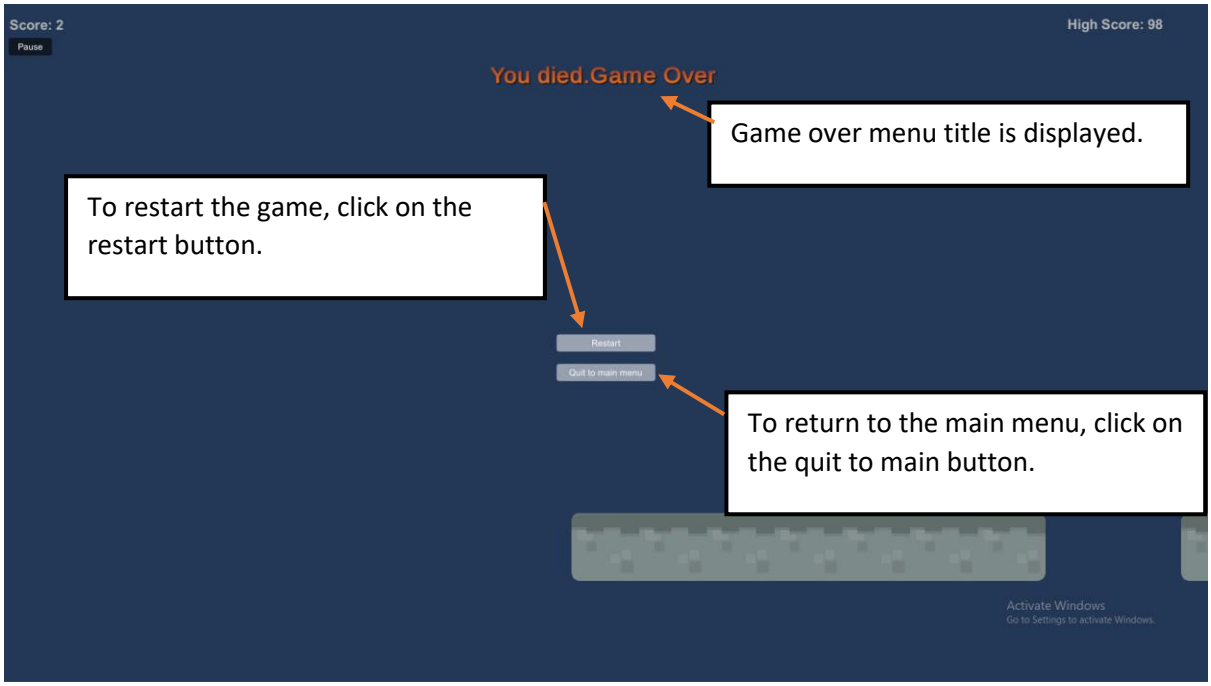


Me:
The pause menu was implemented so that the player can take a break from it. They can click on the first button to resume the game, the second button to restart the game and the last button to return to the main menu. I was not familiar with assigning keys to menus, which meant that I was not able to assign the pause menu to the 'P' key. I was unsure on why the buttons turned out to be so small. Given more time, I may have found a solution to it online.

Hamid Choudhury:

I like how the pause menu is easy to understand and shows the buttons in the centre, which is convenient, but I would not accept the small sizes of the buttons in the final game; It is off-putting. The gameplay in the background being darkened was a nice touch though.

Game over menu

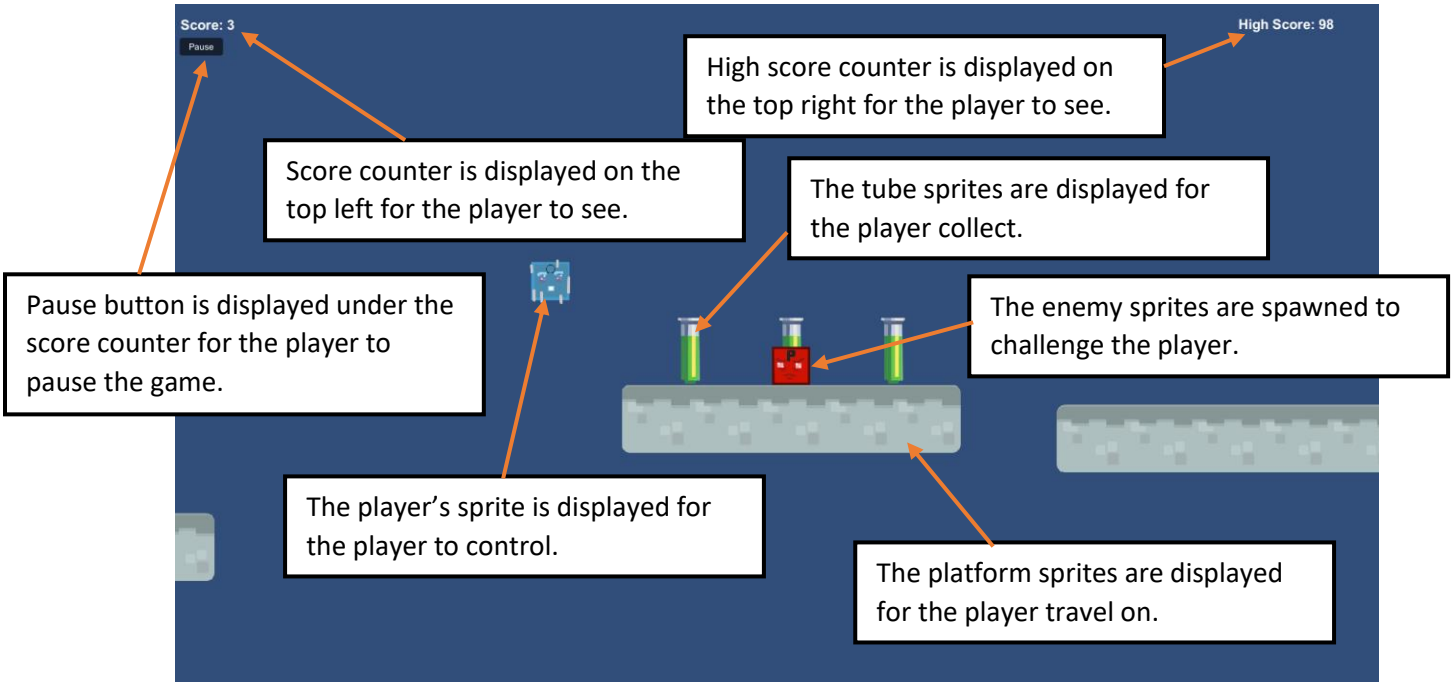


Me:
The pause menu was implemented so that the player can take a break from it. They can click on the first button to restart the game and the last button to return to the main menu. I mentioned this to before, but I was unsure on why the buttons turned out to be so small. Given more time, I may have found a solution to it online.

Azim Khanazada:

I like the clean user interface for the game over menu with the buttons in the centre, but the size of the buttons is a bit inconvenient when trying to click on them. Overall, it is alright.

<u>Screen elements</u>



Me:
The score counter on the top left is for the player to view their score and the pause button is under it so that it is easy to notice. The top right shows the high-score counter so that the player knows what the high-score is. The player' can view their own sprite and is able to control it. There are tubes spawned on random platforms for the player to increase their score. Enemies are also spawned on random platforms to pose as a challenge.

Azim Khanazada:

The pause button, score and high score counters being displayed on the top of the screen is nice to see as it is not covering the screen. The colour and the font could look better. I remember seeing the animations on the old player sprite (in the first interview during development) and thought they were good, the animations for your own player sprite are just as good as well. I like the tubes and enemies randomly spawning but I wish that they did not spawn in the middle of the platforms all the time; it creates unnecessary difficulty.

Hamid Choudhury:

The animations for your own player sprite are pretty good. Collecting tubes are fun but I was there was sound effects for when I am collecting them. The enemies are always spawning in the middle of the platforms, like the tubes, all the time which makes collecting tubes difficult. The pause button is easy to notice but I would prefer if there was a keyboard shortcut to activate the pause menu. The score counter is convenient and the high score counters updating is cool.

Controls

Me:

The controls for the player are the standard WASD keys and the user just needs a mouse for navigating through the menus. The mouse does not require an explanation for what it is used for but the player may not have any game messages for the WASD keys due to time limitations.

Azim: The controls for the player would be guessed anyway by anyone who was played a pc game before; the mouse controls are also fine.

How well does the solution match the success criteria success criteria?

In this section, I will be going over how well my solution matches the success criteria that I made after the research stage in my analysis. I will also talk about any changes that were made to the design of the solution during development and how I completed any unmet requirements.

Menus/Navigation.

REQUIREMENT	JUSTIFICATION
Main Menu.	The menu will need to function properly as it acts as a point where the game can branch out to its other features such as “play”, “options”, “quit” etc. The player will be able to select what they want to do here.
Pause menu.	Pausing allows the player to take a quick break from the game or leave the game entirely.
Restarting.	Allows the player to restart gameplay if they wish.
Game-over menu.	This screen should display when the player has fallen. It should give the players the option to restart or leave.
Leader board.	The leader board will display 5 players who have the highest scores. The Scores will be stored in a file which is used in the game to collect data for the leader board. This makes the game seem arcade-esque.
Options.	If the user does not wish to hear the music or the sound effects, they can toggle it off in settings.
Product screen displays the product of a reaction.	The educational aspect of the game is within the product screen. The game giving feedback on what the kind of element the user has reacted to. This lets the player’s knowledge expand.

I was able to implement most these requirements except for the ‘Leader board’ and ‘Options’ requirement.

The ‘Main Menu’ requirement was partially met as I learned how to link the menus to the play state of my game at the date :01/02/2021 during development. However, the options menu could not be implemented due to time limitations and the idea of the leader board was replaced.

However, if I were able to implement music and sound effects into my game (which is what the options menu is being used for), I would be able to create the options menu to be like that of the game over menu and have two button/icons which would either activate or de-activate the music and sound effects game objects.

The ‘Pause menu’ was met easily as the play-state is frozen and the player is given three options to choose from in the pause menu: ‘Resume’, ‘Restart’ and ‘Quit to main menu’. The first button allows the player to resume the play-state. The second button allows the player to restart the play-state which also resets their score. The last button lets the player to go from the pause state to the menu state. These buttons were tested with the left mice click at :10/02/2021 during development.

The ‘Game-over menu’ requirement was easily met as well as the game would go from the play state to the endplay state when the player fell off or collided with an enemy. The player is given two options to choose from which are ‘Restart’ and ‘Quit to main menu’ and they can use the left mice click to select a button. This was tested at 05/02/2021 during development.

The 'leader board' requirement was not met. A design change I made during development was for the leader board state/menu requirement. I considered a design change from 21/01/2021 to 24/01/2021 during development. I realised that if there is no multiplayer, there is no point of having a leader board as only one person will be playing it. Therefore, I replaced it with a high-score counter that is visible on the top left corner. It was implemented at 25/01/2021 during development.

The 'Product screen displays the product of a reaction' requirement was not met due to time limitations. Given more time, I would have implemented a script attached to a game object where depending on the name of the enemy, the product screen would output a chemical reaction and a fact about the product of the reaction.

Screen elements

REQUIREMENT	JUSTIFICATION
Tubes (collectible points).	Gives extra points to the player when they are picked up.
Tubes on random platforms.	Keeps the payer engaged as tubes will not be spawning on every platform.
Score Counter.	Allows the player to see their score in-game.
Random Platform sizes.	Makes the gameplay feel less uniform and more engaging for the player.

The 'tubes (collectible points)' requirement was fully met with the player being able to increase their score by +3 after colliding with a tube. This requirement was implemented at 15/01/2021 during development.

The 'Tubes on random platforms' requirement was met which made the chances of tubes spawning, random. This feature was implemented at 16/01/2021 during development.

The score counter was fully met where the player can see their score on the top left of the screen. This feature was implemented at the 13/01/2021 during development.

The 'Random Platform sizes' requirement was fully met as there are 4 different kind of sprites that can be spawned into the game during gameplay. This feature was implemented at 04/01/2021 during development.

Gameplay

REQUIREMENT	JUSTIFICATION
Speed-up.	The game speeds up the longer the player is running. This adds a difficulty curve to the game.
Restarting.	Allows the player to restart gameplay if they wish.
Power-ups.	This will help my solution have a game-like atmosphere and let the player have more fun.
Visual effects/Animations.	Visual effects like animation keeps the game engaging to my stakeholders but should not take up a lot of screen space so that it's not distracting.
Enemies.	The elementary cubes that the player will face acts as enemies as they provide the stakeholder's a challenge as well as a threat of losing. The elements of the cubes act as the gateway to the educational aspects of the game.

The 'Speed-up' requirement was met successfully as the player's velocity gradually increases over time. This idea was included in my analysis stage as I wanted to have a challenge for the player. This feature was implemented at 06/01/2021 during development.

Power-ups was not implemented into my game due to time limitations. Given more time, I would have created an if statement in the player controls script where if the player collides with an object with the tag "power up", colliding with enemies will not affect the player for a certain amount of time. I used this tag idea when I was implementing a restart system at 11/01/2021 and when I implemented enemies at 01/03/2021 during development.

The 'Visual effects/Animations' requirement was partially met as I was only able to implement animations for the player. Given more time, I would be able to create sprites for an animation when the player collects a tube or when the player reacts with an enemy. I implemented animation features at 01/01/2021 and 20/02/2021.

The 'restarting' requirement was fully met as the player can restart the game from either the pause menu or the game over menu. This feature was originally implemented at 11/01/2021 and then introduced into menus from the 10/02/2021 to the 13/02/2021 during development.

The 'enemies' requirement was partially met with only one type of enemy spawning and not causing the game to go into the product screen when a reaction has occurred. When the player collides with an enemy, they are sent to the game-over screen instead (This was tested at 01/03/2021). I was not able to implement all enemies into a random spawning system due to time limitations. Given more time, I would have used a script like the platform generator script and using the object pooling method that I used when generating random platforms.

<u>Scoring</u>

REQUIREMENT	JUSTIFICATION
Score-system.	Implementing a functional score system will give the player an incentive to carry on and play the game. Collecting tubes (test tubes in chemistry) will give the player and number of points which can be added to their score. The game also displays the high score.

The 'Score-system' requirement was partially met as there were changes to how the player's score was saved. A new high score counter was implemented at 25/01/2021 during development to save the player's highest score and have it displayed on the top right of the screen. The tubes add to the player's score and this was tested at 15/01/2021 during development.

<u>Mechanisms</u>

REQUIREMENT	JUSTIFICATION

Platforms being re-used by a function.	Having platforms being activated and de-activated in the game, reduces CPU usage. This makes the game more stable when running on non-gaming PCs.
Random Platform Spaces.	Makes the gameplay feel less uniform and more engaging for the player.
Platforms of varied height.	Makes the gameplay feel less uniform and more engaging for the player and makes more use of the jump mechanic.
Tubes on random platforms.	Keeps the payer engaged as tubes will not be spawning on every platform.
Game messages.	Messages such as “Ready and “go” may be added at the start so that it lets the user prepare to play the game.

The ‘Platforms being re-used by a function’ requirement was fully met as I used the object pooling function to activate and de-activate platforms instead of creating new ones. This feature was proposed during my design stage, which was then implemented and tested at 03/01/2021 during development.

The ‘Random Platform Spaces’ requirement was fully met with the spaces between platforms ranging between 2 variables. This was mentioned in my analysis stage and was implemented and tested 04/01/2021.

The ‘Platforms of varied height’ requirement was fully met as the platforms can be spawned at heights above the player, in which they can reach. There was a location and values in which the game uses to determine where a platform would spawn. This was implemented and tested at 05/01/2021 during development.

The ‘Tubes on random platforms’ requirement was also fully met, which means that tubes aren’t spawned on all platforms. The game relies on a value set by the developer(me) in which that is the chance of the tubes spawning (like that of the enemies). This was shown through the implementation and testing of this feature at 16/01/2021.

The ‘Game messages’ requirement was not met as this was a low priority and I didn’t end up implementing it due to time limitations. Given more time, I would have had a text game object that would display for the player for a certain amount of time and then be deactivated so that the player can concentrate on the game.

<u>Sound (Music/SFX)</u>

REQUIREMENT	JUSTIFICATION
Sound.	This will bring life to the game. Sound effects should be present when the player jumps or lands on a platform, reacts with a cube or clicks an option in a menu. Background music will be added into the background of the menu.

The ‘Sound’ requirement was not met due to time requirements. This would have given the player auditory feedback and could be turned on or off in the options menu. Given more time, I would have downloaded royalty-free sound files that will be called by a script for

whenever something happens in the play-state. Another music file would be implemented to function as background music.

Controls

REQUIREMENT	JUSTIFICATION
Controls.	Controls will allow the player to be able to move across platforms using 'W','A','S','D' or the directional arrow keys.

The 'Controls' requirement was fully met was the Player can use the WASD keys to move the player. Clicking the 'A' key goes backwards, 'D' key makes the player go forward and the 'W' key makes the player jump. This was tested and implemented at 01/01/2021.

Maintenance

Current and future maintenance

Right now, I have not implemented any maintenance features that the player could access or use. However, if this were professionally maintained and distributed, my priority would be to add all the features that I was not able to add due to time limitations such as the sound effects, the product screen etc.

Since my solution is not complete, I would like to have an update feature integrated into my game where the player can update the game easily. This is due to high internet speeds being common and would also mean that the player would not have to download copies of the game after every update is available. I would also like to have an IT team that could help with fixing bugs and help with integrating the missing features. Depending on the reception of the game by my target demographic, more features could be implemented such as multiplayer or 3D models that can be used in the game.

Limitations and how I would approach them.

REQUIREMENT	JUSTIFICATION
Moving enemies	Enemies that move on platforms adds a dynamic to the game, which is the enemies' behaviour

To complete this requirement, I would make a new script where there would have a time variable whose value is dependent on the size of the platform. Enemies will spawn at the end of the platform and will go in the opposite direction for a specific amount of time. At the end of that time, the enemy should be at the other end of the platform. Once that occurs, the enemy will travel back again. This will mean that the enemies will move back and forth on a platform. There would be an animation created for the enemy's movement and would be implemented in a similar fashion in which I implemented the player's animation.

REQUIREMENT	JUSTIFICATION
2-player mode	Having 2 players playing on the game locally will make the game enjoyable and last longer.

To complete this requirement, I would create a new scene (in Unity) in which the player will be split into 2 screens horizontally. This would mean that there would be a top screen and a bottom screen for the players. I would duplicate the player game object and slightly change the name; I would add the same scripts that the original player sprite has. On the player's control script, I would change the control keys for the 2nd player to '8','4','5' and '6' keys so that the 2 players have enough space to play.

REQUIREMENT	JUSTIFICATION
Online play	A sizable limitation is that my proposed solution will not have online multiplayer available. Setting up a server to take in many inputs, and outputting scores from many people may be overwhelming and it may not send data back to each client within an acceptable time.

To complete this requirement, I would create a new scene (in Unity) in which the player will be split into 2 screens horizontally. This scene would occur if the player clicked an 'Online play' option. There would be a screen layout which is the same as the local multiplayer. This is so that the 2 players can see each other's score. If a player has died before the other player, the loser can choose 3 options from a menu between watching the winner gain a score, play again against someone else or leave the game.

REQUIREMENT	JUSTIFICATION
3D models and visuals	A 3D variant of the game can be downloaded by the player. Nice visuals make the player more likely to replay the game.

To implement this requirement, I would use a 3D modelling software and create sprites for everything such as the player, tubes, enemies' platforms etc. I would also have to create animation models for each animation such as a 3D running model for the player. The visuals aspect would be lighting and shading which would affect each sprite so that it's more aesthetically pleasing for the player. The game would be in a 2.5D view which means that the player will have a 2D view, but the game will be using 3D visuals.

Code Listings

```
public class CameraChaser : MonoBehaviour
{
    public PlayerControls myPlayer; // Finds where the player is#
    private Vector3 previousPlayerPosition;
    private float distanceChase;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        myPlayer = FindObjectOfType<PlayerControls>(); //Finding the player's controls for the camera
        previousPlayerPosition = myPlayer.transform.position; //Initial position of player
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        distanceChase = myPlayer.transform.position.x - previousPlayerPosition.x;
        /*camera*/
        transform.position = new Vector3(transform.position.x + distanceChase, transform.position.y, transform.position.z); //Camera will only move left to right
        previousPlayerPosition = myPlayer.transform.position;
    }
}
```

```
public class EnemyGenerator : MonoBehaviour
{
    // Start is called before the first frame update
    public ObjectPooler enemyPool; //Used to access object pool for tubes


    1 reference
    public void SpawnEnemy(Vector3 startPoint)
    {
        GameObject enemy1 = enemyPool.GetPooledObject();
        enemy1.transform.position = startPoint;
        enemy1.SetActive(true);

    }
}
```

```
public class GameManager : MonoBehaviour
{
    public Transform platformGenerator;
    private Vector3 pgSpawnPoint;
    public PlayerControls myPlayer;
    private Vector3 playerSpawnPoint;

    private GamePlatformDestroyer[] myPlatformList;

    private ScoreManager aScoreManager;

    public GameOverMenu gameOverDeathScreen; //referencing GameOverMenu script.

    // Start is called before the first frame update
    0 references
    void Start()
    {
        pgSpawnPoint = platformGenerator.position;
        playerSpawnPoint = myPlayer.transform.position;

        aScoreManager = FindObjectOfType<ScoreManager>(); //Finding score manager in my game
    }

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

    }

    1 reference
    public void RestartGame() //can be called from another script
    {
        aScoreManager.scoreIncrease = false; //Score stops increasing
        myPlayer.gameObject.SetActive(false);
        gameOverDeathScreen.gameObject.SetActive(true); //turn game-over menu on


        //StartCoroutine ("RestartGameCo"); //(Not being used at the moment)
    }

    2 references
    public void Reset()
    {
        gameOverDeathScreen.gameObject.SetActive(false); //turn game-over menu off
        myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
        for (int i = 0; i < myPlatformList.Length; i++) //Goes through the array of active platforms
        {
            myPlatformList[i].gameObject.SetActive(false);
        }
        myPlayer.transform.position = playerSpawnPoint;
        platformGenerator.position = pgSpawnPoint; //sets the position back to the start
        myPlayer.gameObject.SetActive(true);

        aScoreManager.scoreCount = 0; //Resets score to 0
        aScoreManager.scoreIncrease = true; //Score starts increasing again
    }

    /*public IEnumerator RestartGameCo()
    {
        aScoreManager.scoreIncrease = false; //Score stops increasing
        myPlayer.gameObject.SetActive(false);
        yield return new WaitForSeconds(1f);
        myPlatformList = FindObjectsOfType<GamePlatformDestroyer>();
        for (int i = 0; i < myPlatformList.Length; i++) //Goes through the array of active platforms
        {
            myPlatformList[i].gameObject.SetActive(false);
        }
        myPlayer.transform.position = playerSpawnPoint;
        platformGenerator.position = pgSpawnPoint; //sets the position back to the start
        myPlayer.gameObject.SetActive(true);

        aScoreManager.scoreCount = 0; //Resets score to 0
        aScoreManager.scoreIncrease = true; //Score starts increasing again
    }*/
}
```

```
1 reference
public class GameOverMenu : MonoBehaviour
{
    public string mainMenuLoad;

    0 references
    public void RestartGame()
    {
        FindObjectOfType<GameManager>().Reset(); //Calls the reset function on game manager
    }

    0 references
    public void QuitToMainMenu()
    {
        Application.LoadLevel(mainMenuLoad); //Loads up the main menu form game-over menu
    }
}
```

```
public class GamePlatformDestroyer : MonoBehaviour
{
    public GameObject destructPoint;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        destructPoint = GameObject.Find ("PfDestructionPoint"); //Finds the object called platform destruction point
    }

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

        if (transform.position.x < destructPoint.transform.position.x)
        {
            //Destroy (gameObject); //Destroy whatever the script is attatched to

            gameObject.SetActive(false); //Deactivates object instead of deleting it
        }

    }
}
```

```
public class GamePlatformGenerator : MonoBehaviour
{

    public Transform generatePoint;  //Point where platforms need to be generated
    public GameObject aPlatform;  //The platform that'll generate ahead of the player.One individual platform.
    public float distanceBetween;
    private float widthOfPlatform;

    public float minDistanceBetween;
    public float maxDistanceBetween;

    //public GameObject[] aPlatforms; //Array of platforms
    private int thePlatformSelector;
    private float[] widthOfPlatforms; //Using an array due to multiple widths

    public ObjectPooler[] myObjectPools; //Made an array of object pools

    private float minimumHeight; //default height of platform
    private float maximumHeight;

    public Transform pointOfZenith; //Moves the platforms above the default position
    private float heightDifference; //Max range for platform movement
    public float maxHeightDifference;

    private TubeGenerator myTubeGenerator;
    public float tubeChanceLimit; //Any number below this float will result in tubes being spawned on a platform

    private EnemyGenerator myEnemyGenerator;
    public float enemyChanceLimit;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        //widthOfPlatform = aPlatform.GetComponent<BoxCollider2D>().size.x;  //Get's the length from the box collider's x-value

        widthOfPlatforms = new float[myObjectPools.Length]; //creates an array of elements from 0 to 3  /*aPlatform*/
        for (int i =0;i< myObjectPools.Length;i++)
        {
            widthOfPlatforms[i]= myObjectPools[i].thePooledObject.GetComponent<BoxCollider2D>().size.x; //Gets the size of an object from an object pool
        }

        minimumHeight = transform.position.y; //Getting the default height of a spawn platform
        maximumHeight = pointOfZenith.position.y; //Max height of which platforms can spawn

        myTubeGenerator = FindObjectOfType<TubeGenerator>(); //Finds object in the game with the tube generator script
        myEnemyGenerator = FindObjectOfType<EnemyGenerator>();
    }

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

        if (transform.position.x < generatePoint.position.x)
        {
            distanceBetween = Random.Range(minDistanceBetween, maxDistanceBetween); //will pick random value for the space between

            thePlatformSelector = Random.Range(0, myObjectPools.Length); //Selects a platform between 0 and the length of the array

            heightDifference = transform.position.y + Random.Range(maxHeightDifference,-maxHeightDifference);


            if (heightDifference > maximumHeight)//If above max height,it will be set as max height
            {
                heightDifference = maximumHeight;
            }
            else if (heightDifference < minimumHeight)
            {
                heightDifference = minimumHeight;
            }

            transform.position = new Vector3(transform.position.x + (widthOfPlatforms[thePlatformSelector] / 2) + distanceBetween, heightDifference, transform.position.z);

            //Instantiate( aPlatforms[thePlatformSelector], transform.position, transform.rotation);//Creates a copy of an existing


            GameObject newPlatform = myObjectPools[thePlatformSelector].GetPooledObject(); //Knows where to look in the object pool for a pooled object
                                                                                           // Get an object is one of the available objects that are inisde my object pool
            newPlatform.transform.position = transform.position;

            newPlatform.transform.rotation = transform.rotation;
            newPlatform.SetActive(true);

            if (Random.Range(0f, 100f) < tubeChanceLimit)
            {
                myTubeGenerator.SpawnTubes(new Vector3(transform.position.x, transform.position.y + 1f, transform.position.z));
            }

            if (Random.Range(0f, 100f) < enemyChanceLimit)
            {
                myEnemyGenerator.SpawnEnemy(new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z));
            }
            transform.position = new Vector3(transform.position.x + (widthOfPlatforms[thePlatformSelector] / 2) , transform.position.y, transform.position.z); //Calculates
```

```
public class MainMenu : MonoBehaviour
{
    public string playStart;
    0 references
    public void PlayGame()
    {
        Application.LoadLevel(playStart);
    }

    0 references
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

```
public class myPickupPoints : MonoBehaviour
{
    public int tubePoints; //The value that the tube will give to the player.

    private ScoreManager tubeScoreManager;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        tubeScoreManager = FindObjectOfType<ScoreManager>();

    }

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

    }

    0 references
    void OnTriggerEnter2D(Collider2D other) //When something collides with the trigger
    {
        if (other.gameObject.name=="Player")
        {
            tubeScoreManager.AddScore(tubePoints); //Adds value of tube to the player's score
            gameObject.SetActive(false);
        }
    }
}
```

```
public class ObjectPooler : MonoBehaviour
{
    public GameObject thePooledObject; //Object that's going to be used

    public int pooledAmount; //How many objects that are going to be used

    List<GameObject> thePooledObjects;

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

        thePooledObjects = new List<GameObject>();

        for (int i = 0; i < pooledAmount; i++)
        {
            GameObject myObject = (GameObject)Instantiate(thePooledObject); //used gameobject in brackets to cast whateber is behind it,inot a game object
            myObject.SetActive(false); //created new object but don't want  it set active in the scene,deactivated
            thePooledObjects.Add(myObject); //adds one to i
        }

    }

    5 references
    public GameObject GetPooledObject() //Looking for a game object to get
    {
        for (int i = 0; i <thePooledObjects.Count; i++)
        {
            if (!thePooledObjects[i].activeInHierarchy) //Goes to position 0 in the list. activeInHierarchy:is it inactive in the scene at the moment
            {
                return thePooledObjects[i];  //returns an object that's not currently active ,the script won't loop afterwards
                                             //if an object is active, 'id' statement is false
            }
        }
        GameObject myObject = (GameObject)Instantiate(thePooledObject); //Adds a new inactive object to the list
        myObject.SetActive(false);
        thePooledObjects.Add(myObject);
        return myObject; //returns the object that has just been instantiated

    }
}
```

```
public class PauseMenu : MonoBehaviour
{
    // Start is called before the first frame update
    public string mainMenuLoad;

    public GameObject pauseMenu;

    0 references
    public void PauseGame()
    {
        Time.timeScale = 0f;// Game pauses,player won't move
        pauseMenu.SetActive(true); //Activates pause menu

    }

    0 references
    public void ResumeGame()
    {
        Time.timeScale = 1f;//Time is set back to normal
        pauseMenu.SetActive(false);//Deactivates pause menu
    }

    0 references
    public void RestartGame()
    {
        Time.timeScale = 1f;//Time is set back to normal
        pauseMenu.SetActive(false);//Deactivates pause menu
        FindObjectOfType<GameManager>().Reset(); //Calls the reset function on game manager
    }

    0 references
    public void QuitToMainMenu()
    {
        Time.timeScale = 1f;//Time is set back to normal
        Application.LoadLevel(mainMenuLoad); //Loads up the main menu form game-over menu
    }
}
```

```
public float velocityLandmark;
private float velocityLandmarkStore;

private float velocityLandmarkCounter; //When a specific distance is reached, the landmark for when the speed increases is moved further along.
private float velocityLandmarkCounterStore;

public GameManager theGameManager;

// Start is called before the first frame update
0 references
void Start()
{
    playerRigidBody = GetComponent<Rigidbody2D>(); //GetComponent searches the player object for the Rigid Body2D
    //playerCollider = GetComponent<Collider2D>(); //search on player object for collider
    playerAnimator = GetComponent<Animator>(); //Finds animator that's attatched to the player
    jumpDurationChecker = jumpDuration;
    velocityLandmarkCounter = velocityLandmark; //Milestone is not at 0 everytime
    playerSpeedStore = playerSpeed;
    velocityLandmarkCounterStore = velocityLandmarkCounter;
    velocityLandmarkStore=velocityLandmark;


// Update is called once per frame
0 references
void Update()
{
    //onGround = Physics2D.IsTouchingLayers(playerCollider, defineGround); //onGround is true if the two colliders from the Player and Ground are touching.

    onGround = Physics2D.OverlapCircle(groundCollisionChecker.position, groundCollisionCheckerRadius, defineGround);

    if (transform.position.x > velocityLandmarkCounter)
    {
        velocityLandmarkCounter += velocityLandmark; //Continously adds a set distance for where the player will speed up
        velocityLandmark = velocityLandmark * playerSpeedUp;

        playerSpeed = playerSpeed * playerSpeedUp;
    }

    if (Input.GetKey(KeyCode.D))
    {
        playerRigidBody.velocity = new Vector2(playerSpeed, playerRigidBody.velocity.y);
    }

    if (Input.GetKey(KeyCode.A))
    {
        playerRigidBody.velocity = new Vector2(-playerSpeed, playerRigidBody.velocity.y);
    }

    if (Input.GetKeyDown(KeyCode.W))
    {
        if (onGround)
        {
            playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, forceOfJump);
        }
    }

    if (Input.GetKey(KeyCode.W))
    {
        if (jumpDurationChecker > 0)
        {
            playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, forceOfJump);
            jumpDurationChecker -= Time.deltaTime; //As the player holds down the button up to a certain time limit.he'll continue going higher into the air
        }
    }

    if (Input.GetKeyUp(KeyCode.W))
    {
        jumpDurationChecker = 0;
    }

    if (onGround)
    {
        jumpDurationChecker = jumpDuration;
    }

    playerAnimator.SetFloat("Speed", playerRigidBody.velocity.x); //Uses the speed value of the RigidBody for the animator
    playerAnimator.SetBool("OnGround", onGround);

}

0 references
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "killbox" || other.gameObject.tag == "enemy")
        {
            theGameManager.RestartGame();
            playerSpeed = playerSpeedStore;
            velocityLandmarkCounter = velocityLandmarkCounterStore;
            velocityLandmark = velocityLandmarkStore;
    }
}
```

```
public class ScoreManager : MonoBehaviour
{
    public Text scoreText; //referencing my text object for score
    public float scoreCount; //Used to keep track of score

    public Text highScoreText;
    public float highScoreCount; //Used to keep track of high-score

    public float timePoints; //Player gets a certain point added to thier score every second

    public bool scoreIncrease;


    // Start is called before the first frame update
    0 references
    void Start()
    {
        if (PlayerPrefs.GetFloat("HighScore") !=null)
        {
            highScoreCount = PlayerPrefs.GetFloat("HighScore");
        }
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        if (scoreIncrease)
        {
            scoreCount += timePoints * Time.deltaTime; //Time passed in each frame,adds a cetrain point each second to the score
        }

        if (scoreCount > highScoreCount) //Updates High Score
        {
            highScoreCount = scoreCount;
            PlayerPrefs.SetFloat("HighScore",highScoreCount);//Saves highscore
        }

        scoreText.text = "Score: " + Mathf.Round(scoreCount); //Accessing text within the text object and rounding
        highScoreText.text = "High Score: " + Mathf.Round(highScoreCount);

    }

    1 reference
    public void AddScore(int pointsDue)
    {
        scoreCount += pointsDue;
    }

}
```

```
public class TubeGenerator : MonoBehaviour
{
    public ObjectPooler tubePool; //Used to access object pool for tubes

    public float distanceBetweenTubes;

    1 reference
    public void SpawnTubes(Vector3 startPoint)
    {
        GameObject tube1 = tubePool.GetPooledObject();
        tube1.transform.position = startPoint;
        tube1.SetActive(true);

        GameObject tube2 = tubePool.GetPooledObject();
        tube2.transform.position = new Vector3(startPoint.x - distanceBetweenTubes, startPoint.y, startPoint.z);
        tube2.SetActive(true);

        GameObject tube3 = tubePool.GetPooledObject();
        tube3.transform.position = new Vector3(startPoint.x + distanceBetweenTubes, startPoint.y, startPoint.z);
        tube3.SetActive(true);

    }
}
```