COMPUTER SCIENCE PROJECT

Bartlomiej Wlodarski

Candidate Number: 1359
Centre Number: 56120

Table of Contents

Project: Platformer, Side-Scrolling Game

My project is a platformer, side-scrolling game. The objective of the game is to complete each level, while gaining as many points as possible through getting rid of enemies, collecting money and completing the game as quickly as possible. I got the idea for this project from the game called Super Mario Run which I have covered under 'existing solutions'.

I plan on making this game, because currently, on Android, there aren't many games like this apart from Super Mario Run, and the existing ones are all pay-to-win (you must use microtransactions to gain features that help you throughout the game) which isn't fair in my opinion. This makes it less enjoyable for the people who can't play. Therefore, I decided to create a free-to-win (everyone has access to the same features and additional ones can usually be unlocked by completing certain activities) side-scrolling platformer game.

Features:

- Score counter (increases through collecting items in the level). This is computationally approachable, as the score counter involves calculations and statistics; it displays your score and increases or decreases it at appropriate times.
- Leaderboard, to encourage people to play again (to beat high scores). This can be computationally approachable, as this involves calculations and it involves logical reasoning, such as comparing places on the leaderboard itself, seeing if a score is higher or lower than a certain score, then calculating the place where the new score should be placed
- Timer to add an extra level of challenge (can be disabled if the user finds the level too hard with it). This is computationally approachable as this involves calculations and statistics; the new time is calculated every second and it is displayed on the screen for the player to see.

Computational approaches:

The idea of a side-scrolling game is computationally approachable, because:

- It will have an on-screen score counter which is calculated in real-time (this involves processing the score through calculations).
- The game will have inputs and outputs, the touchscreen of a phone being both of those.
- The game will contain features that are processed concurrently, for example, the player can move while the time remaining, and the score is calculated.
- The game can be easily decomposed into smaller modules; for example, each button can use the same press detection script instead of having every button have its own individual script.
- The game will have physics simulation.

Stakeholders:

The stakeholders for my game will be teenagers and adults who are interested in playing video games on their mobile devices. These people usually spend much of their time playing games and this makes them suitable for this project, as obviously, people interested in video games will be interested in playing a mobile game, if they like the type of game it will be. They're also suitable stakeholders for my project, as most gamers hate microtransactions; they'd rather unlock features through playing the game and completing activities, rather than with their money. Since I plan on making my game free-to-win, they will probably enjoy it. Anyone who isn't a teenager, or an adult wouldn't be suitable for this project, as young children don't have the necessary knowledge to play a complicated or hard platformer game. They also don't have the attention span to play games like this and need to be a bit older to appreciate the game. Because of that, I will not focus on child-friendly features.

My stakeholders will be able to make use of this game, as gamers like to occasionally play games on their phones but a lot of the time, they don't find them very enjoyable, due to features being blocked by microtransactions (you must pay a small amount of money to unlock features that allow you to be better at a game). I will create the game free of these paywalls to make a game that is enjoyable for my stakeholders. Also, in general, they will make use of this solution by having fun while playing the game; that is my primary objective.

The stakeholders will also be able to test the game during the development of it, to allow them to give me their opinions on it, which will allow me to make it as enjoyable for them as possible.

Existing Solutions:

Super Mario Run

Super Mario Run is a game available for Android and iOS and its primary objective of this game is to complete levels with a score that is as high as possible. To complete a level, you must not touch any enemies (unless you jump on them) and complete the level within the time limit.

Figure 1: An advertisement image for Super Mario Run, showing some levels of the game.

This is the game that originally inspired me to create a side-scrolling game in a similar style. What I like about this game is the simple gameplay; you only collect coins and try to beat your high scores on each level. It also seems to be well optimised as it seems to run well on the devices I tested it on. I will try to make my game run well on most devices, as not everyone has powerful devices and I want the game to be available for as many people as possible.

However, there were more things wrong with it than there were good things, in my opinion. I don't like the fact that you don't control the movement of the character fully. You can only tap the screen to jump or glide when you're falling, which makes the game quite repetitive. I will try to avoid this in my game by allowing the player to control the movement completely, e.g. by allowing the user to control when the character moves. I also think that the game has too many features; there are several game modes and menus which can be rather hard to use at first, which is surprising as this game is targeted towards children. Although my game isn't targeted towards children, I will try to avoid complicated menus, to make the app as easy to use as possible. There are also in-app purchases, which are good for-profit, but make the game less enjoyable, as some features are locked for you unless you pay. I will make all of the game's content available for everyone in my game, as I want it to be equally as fun for everyone.

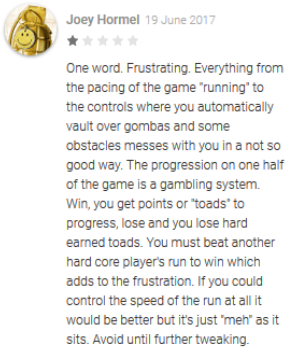


Figure 2: One of the top-rated reviews on the Play Store, which is negative.

The reviews for Super Mario Run on the Android Play Store were quite mixed; The mean score is currently 3.6/5 stars and approximately 44% of the reviews were 1/5 stars. From the reviews that I looked at, people have very similar issues with the game to me, for example, the person in Figure 2. He also complained about the lack of controls which he states can make the game less enjoyable. I will definitely avoid this lack of control, as many other reviewers complained about this feature too.

Geometry Dash

In this game, you play as a square and need to complete each level without touching any obstacles. While completing a level you will also come across some collectables such as coins. There are three coins per level.

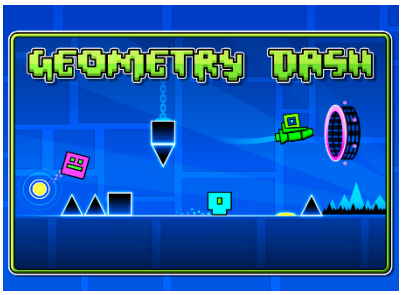


Figure 3: An advertisement image for Geometry Dash, showing some of the features of the game (spikes, flying etc.)

I really like this game because of how varied it is; each level feels truly unique, unlike the levels in Super Mario Run. For example, in some levels, you will come across new features, such as jump pads that will make the square fly into the sky. The game is also challenging in an enjoyable way; for example, the controls are very much like those in Super Mario run (if not even simpler), but in this case, it's a positive thing, as the lack of horizontal control makes the game more challenging, unlike in Super Mario Run, where they don't make the game any harder, just more annoying to play. These controls are good in this game, as since you can't choose when you go horizontally or vertically, you need to time your jumps correctly, otherwise, you will fall onto spikes or other obstacles and you'll fail the level.

In comparison to Super Mario Run, this game has a lot more features. These features in this game are also much better than the ones in Super Mario Run, however what makes up for that is the fact that they are very simple features and even with such a large amount of them, you can still understand how to use and play the game without trouble, unlike the features in Super Mario Run, where it's hard to navigate the menus and to understand the features in general.

The menus are simplistic, meaning that you can't get lost very easily (there are only a few buttons on the main screen, one being the level selection button and the other notable one being the settings button.

From this game, I will take the idea of having almost every level add something new to the game. I believe that this would be quite pleasing to my stakeholders, as gamers like to have a variety of features; to us, it would be boring to have a game that never introduces anything new.

Investigation and analysis:

To implement features that the stakeholders would like to be in the game, I have created a quick survey and sent it out directly to them, through gaming groups on social networks. These groups were made entirely of people who enjoy playing video games, meaning I would get accurate results, as these people are my stakeholders. This has helped me make several decisions, such as how long the levels (and the game itself) should be, how hard the game should be, what features I should focus on the most, and so on.

This survey has also allowed me to find out if people who play games are mostly teenagers and adults, which also helped me with deciding how mature the game should be.

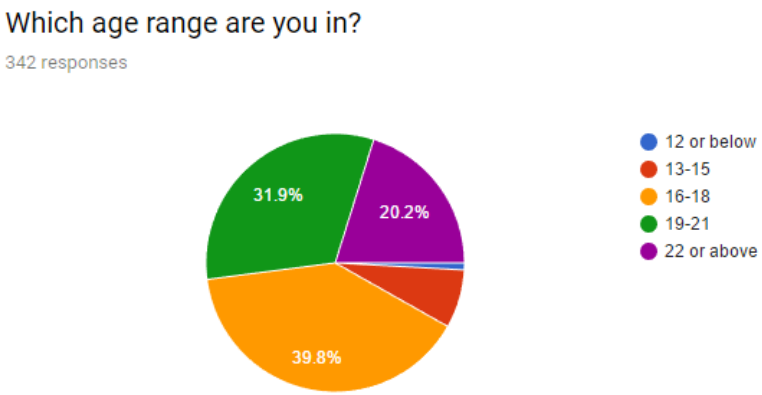


Figure 4: A pie chart of results for the first question of the survey.

The first question I asked in the survey was the one above: “Which age range are you in?” As we can see from the chart, a clear majority of people said that they are 16-18, followed by 19-21 and so on.

From this, we can confirm that most gamers are in fact teenagers or adults. As the results say that most people who play games are at least 16 years old, I can make the contents of it somewhat mature and I can make the game quite complex, as older people have more knowledge with video games, so they’d be able to use the more complex mechanics easily. For example, small children would probably only be able to play a game well if the controls are basic, for example, if the game only allowed you to jump, while a teenager would be able to use more complex controls easily.

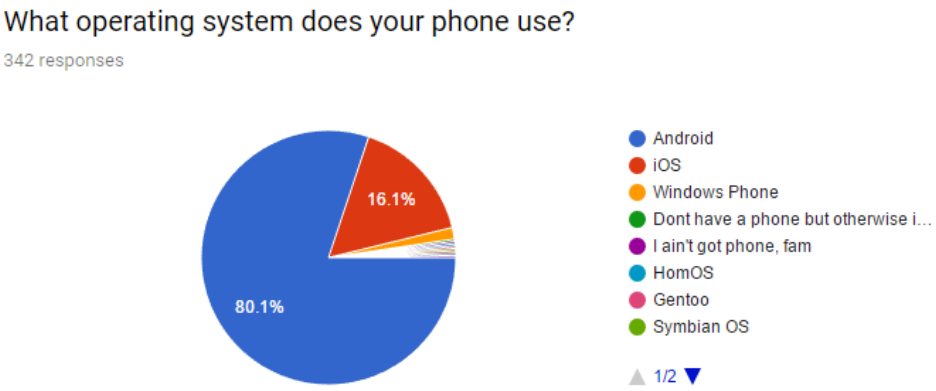


Figure 5: A pie chart of results for the second question of the survey.

The second question was “What operating system does your phone use?” Without a doubt, we can see that most people use the Android operating system; a staggering 80% of people said that they use Android, which is good, as that is the operating system I initially planned on making the game for. This means that if I were to release this to the public, a clear majority of people would be able to play it. Unfortunately, I would not be able to create the game for iOS even if I wanted to, as there is a rather large amount of money you need to pay before you can develop games for iOS, plus you can’t develop iOS devices on Windows, so I would need to own a desktop or laptop Apple device.

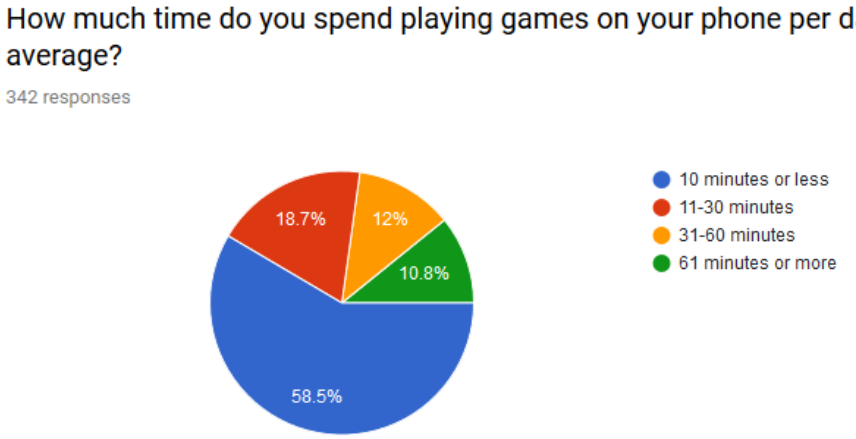


Figure 6: A pie chart of results for the third question of the survey.

After that, I asked the question “How much time do you spend playing games on your phone per day, on average?” The results for this question were unsurprising; most people don’t spend a lot of time playing games on their phones. For me, this means that for my game to be enjoyable, I need to make the gameplay very short, by making short levels, for example. Of course, I will add longer levels too, as the rest of the people said they play at least 11 minutes or more. However, I will not add too many longer levels, as it would be a waste since a wide majority of people who did my survey said they play for only 10 minutes or less.

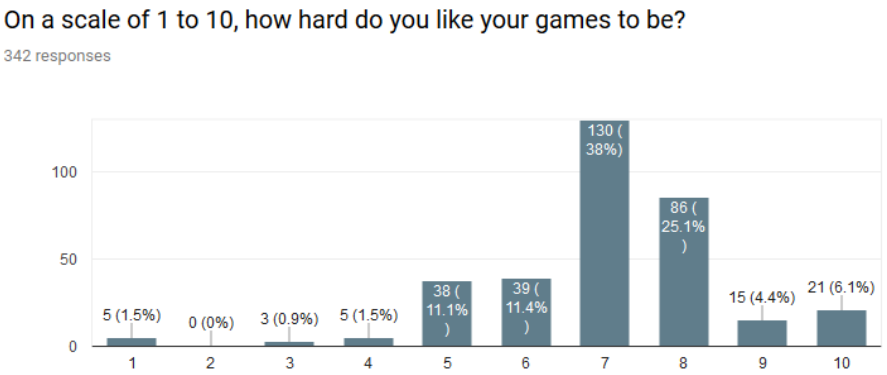


Figure 7: A column chart showing the results for the fourth question of the survey.

The next question was “On a scale of 1 to 10 (1 being extremely easy, 10 being extremely hard), how hard do you like your games to be?” Most people said that they like their difficulty to be 7/10, followed by 8/10. This is quite surprising since most people said they play for 10 minutes or less per day. This means that for my game to be successful with adults and teenagers, I will have to make the game somewhat hard. Of course, since the results were a bit varied for the rest of the results, I will add difficulty modifiers so that those people could still enjoy the game too. The default difficulty will be based on 7, so by default, it will be quite hard.

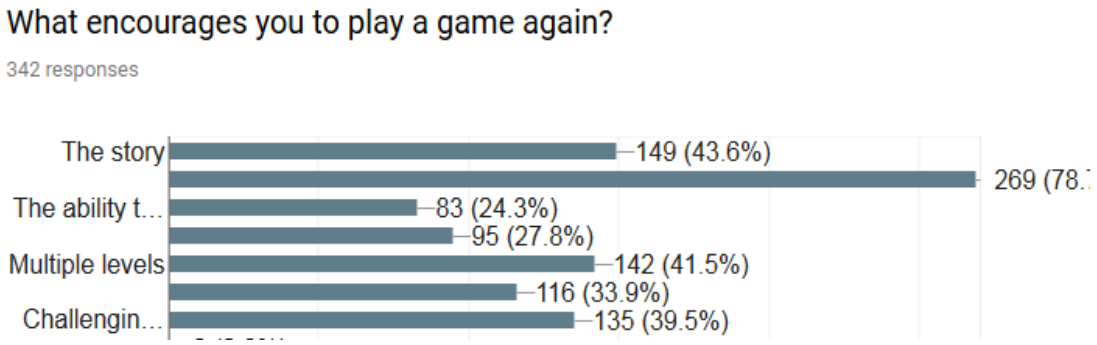


Figure 8: A bar graph showing the results for the fifth question.

Then, I asked the question “What encourages you to play a game again?” This was the last question I asked unless earlier you answered that your phone used an Android operating system. I asked this question as I want to know what feature of a mobile game is the most important aspect of it so that I know what I should focus on the most. Again, these results were somewhat unsurprising, it’s what I expected. Over 78% people said that the gameplay is important to them, so I’m going to focus on the gameplay the most. Of course, I’m still going to work on all the other features mentioned above, but I won’t focus on them as much, as people said that the gameplay is the most important part of the game.

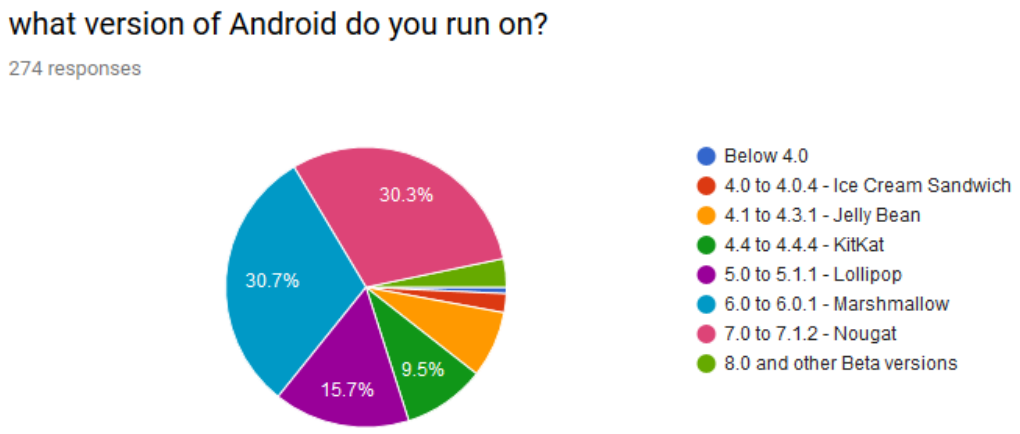


Figure 9: The answers to the last question of the survey. It is only available if the person answering selected Android as their phone's operating system earlier.

The final question was asked when the person doing the survey said their phone runs on Android. I asked this question so that I know what version of the Android operating system most people use, which allows me to see what features I could confidently use (later Android versions have more features). So, in this case, since most people use android ~6.0 and ~7.0, it will be fine for me to use features from these operating systems; in total, 61% of people use these two versions, so I think it will be fine to use features from them, but I will try to use them as little as possible, so that the other 39% can still play the game (this way I'll be able to reach a greater audience).

Essential features:

The essential features I will implement in the game are as follows:

- Main menu screen that lets you select different screens (game mode selection screen, settings etc.). This is necessary as without this main menu, you wouldn't be able to access all the other necessary menus and the app would go straight to the game. The menu will contain:
  - A button to select levels
  - A button to go into options
- Allow the user to have good control over the character (can move up, down left and right when possible), as most people stated that the gameplay is one of the essential features, and the ability to control the movement as much as possible would improve the gameplay.
- Have multiple levels that are quite different from each other, to make the gameplay better. The levels will include:
  - Multiple platforms for the player to jump onto.
  - Obstacles, both stationary and moving, for example, spikes or enemies respectively.
  - Collectibles, such as coins.
  - A finish point, so that the player can beat the level.
- Have some sort of a story in the game, as that was one of the more popular things in my survey that supposedly make a mobile game good.
  - The story will be somewhat simple, as most people said they only play mobile games for 10 minutes or less, so putting in a complex story would be pointless.
- Have difficulty modifiers, as that will let more people enjoy the game, as some people like games to be easier or harder (this allows me to reach out to a greater amount of people).

    - The modifiers will affect the score, so if you lower the difficulty, during the score calculation at the end of a level, the score would be lower than usual, and the opposite would happen if you increase the difficulty.
- Allow the player to find collectable items, as this adds to the replayability (as some players like to play games to collect all the items and/or achievements).
    - I plan on adding several types of collectables that are worth different amounts, for example, a regular coin could be worth just 5 points while a diamond could be worth 100 points.
- There will be a leaderboard, as quite a lot of people said in my survey that that's a feature that makes them want to play a game again (they play the game again to be above everyone)
    - The leaderboard would update after every game, so if you fail a level, the final score will not be put on the leaderboard.
- The player must be controlled by the touchscreen of a phone. This is because, I have decided to make this game for mobile devices, and there are no other ways that can easily allow you to control the player.
    - This will be done by pressing on-screen buttons that control the motion of the player. So, pressing the left button will move the character left, the right button will move them right and so on.
- Each level must also be completable because if they are not completable, the stakeholders will not be able to make high scores on the leaderboard. Since quite a few people said in my survey that they like to beat high scores, it would be bad if they couldn't set them.

Limitations:

Since I don't have a lot of experience with using Unity to create games or C# (the language that I plan on using in Unity), there will be quite a few limitations for me. For example, I don't think I'd be able to do things such as randomizable levels, as this would require a lot of knowledge, which I don't have. I imagine this would be very hard to code, as you must consider things such as paths: there always needs to be a path for the player to go through and I don't think I'd have the skill to produce code that will do that for me. Therefore, I will have to stick with levels generated by me.

I will also not be able to add features such as multiplayer (some people in my survey said that's one of the things that makes a game enjoyable for them). This is because I am not familiar enough with Unity to add features that are this complicated. I am very new to Unity, so I doubt I'd be able to add that feature.

I doubt I'll also be able to add custom physics, such as bounce effects (if I wanted to make objects that let the player bounce when collided with) since Unity has its own physics components built-in and I don't have the skills required to create my own physics. I will have to stick with what the game engine gives me by default.

Requirements:

Since I plan on making this game on Android, I had to research the minimum device requirements for the user to run the game. Unity provides this information on their website: "Android: OS 4.1 or later; ARMv7 (Cortex) CPU with NEON support or Atom CPU; OpenGL ES 2.0 or later". These will be the absolute minimum requirements of the game, but they may change slightly, depending on how demanding the game turns out to be.

I will be developing the game in the Unity engine since it is the easiest to use game creation engine. That is good for me, as I don't have much experience with making games and this will allow me to create a good game with little struggle.

Of course, the end-user will also have to own an Android device, otherwise, they will not be able to play the game unless they use some emulation software on their device of choice that can execute APK files (these files are what you use to install apps on Android).

Measurable success criteria:

The measurable success criteria are as follows:

- It must have multiple levels and be able to switch between them. This should be done through the main menu. If there weren't multiple levels, the players would not find the game enjoyable, they'd find it repetitive instead. People like variation, therefore I will have to add several levels.
- There needs to be a leaderboard that contains all the top scores of the game. This is going to be accessed through the main menu. The highest score will be displayed at the top. The high-score leaderboard will also show what level the person got the high score on, as not all the levels will be the same, so that will allow people to see what levels they'd need to play to gain the most points.
- There will have to be the main menu that lets you modify the game's settings and access the leaderboards, as well as the levels. Without this feature, you wouldn't be able to access various levels and the app will go right into the game. I chose this as the stakeholders will want to be able to modify the game's settings and leaderboard and such.
- Have obstacles that can prevent you from completing the game if a collision occurs. These obstacles will have to be in the game as in my survey earlier in the analysis, people said that they like their games to be quite hard. Obstacles and enemies will make the game harder that way.

There must be difficulty modifiers for the stakeholders, as not everyone likes hard games and some people like their games to be even harder than usual. The players will be rewarded depending on the difficulty modifier. For example, if the player set the difficulty modifier to less than usual, their score will be slightly lower than usual too, and the opposite would happen if they set their game to be harder. This will be accessible within the settings of the game and will affect the movement speed of the player, as well as the jump height and enemy speed.

- There must be a quit button within the game, otherwise, the stakeholders will have to fiddle around with their phone's settings or other things to close the game. This is necessary to reduce the memory usage on the player's phone. Most phones don't have too much memory and if a game is left running in the background, it will slow down the device. I don't want that to happen as my stakeholders will probably be upset about that.

Design model:

This is where the stages of the spiral model can also be used, but they may not be completed in a linear sequence; at some point of the process, you may wish to go back to the quick design stage as you may still be researching some other part of the requirements.

Throughout this, feedback will be obtained from testers and because of that, this is an iterative process during which changes may be made to one part of the program as another is being built.

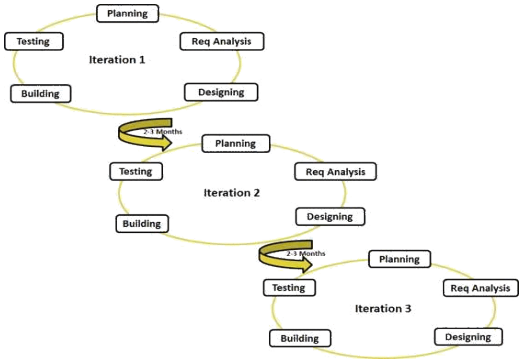


Figure 9: An example of the agile model

A prototype is made during every stage with the users so that we can see if the prototype follows what the users want. For this type of software development to be successful the following needs to be done:

- The model must be kept simple.
- You need constant feedback from the users.
- Need to understand that the requirements could change in development.
- Be prepared to make drastic changes to the software.

I have decided to use the agile model during the development of my game. This is because, this model is very simple, and it requires you to constantly get feedback from the stakeholders, which makes sure that the while I develop the game, it is still what the stakeholders want. This way, the game will always be up to date with what the users want.

Project Design and Development Stage

Development Plan:

Stage 1: User Interface

In this stage, I will create a dummy version of the game, where there's going to be only a user interface and no game itself. This is going to be the first step of the project as then I'd be able to see if my stakeholders like the visuals. It means that I won't have to go back and re-design the interfaces after I've started making the other parts.

The only algorithms that will be added at this point are going to be ones that allow you to switch between the different screens and a script to close the game when the exit button is pressed.

The features of this will be:

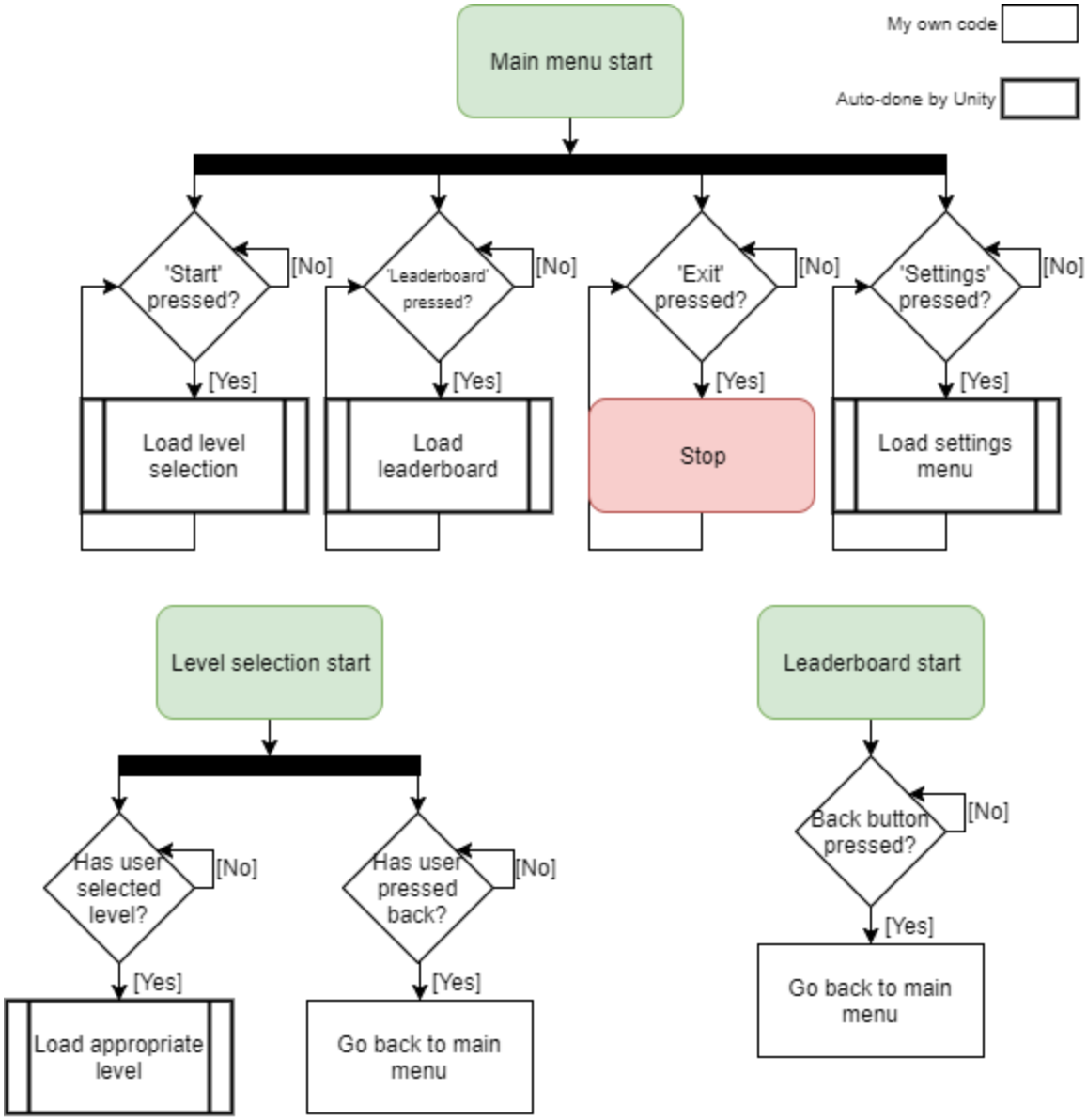
- Menus that you can move between (e.g. pressing a button will take you from one screen to another)
- Have scalable buttons so that they can work on a larger range of devices than static size buttons
- Have an exit button so that the game doesn't stay open in the background

To test this part of the app, I will use black box testing, using the table below.

Test #	Description	Data to Test	Result Expected	Result Observed
1	Valid Does the app open when the icon is pressed?	Icon	The app opens when the icon is clicked.	
2	Valid Do the buttons in-game do appropriate things?	Buttons	The buttons take you to correct screens.	
3	Valid Does the exit button close the game?	Exit button	The exit button closes the game.	
4	Valid Do the buttons scale with the screen resolution?	Buttons	The buttons increase in size as the resolution goes up and vice versa.	

After I have completed the interface, I will ask some of my stakeholders in person and through social media for feedback. If they like the interface, then I will not apply any changes to it, and if they don't, then I will change it, keeping their comments in mind.

User interface activity diagram



Stage 2: Creating Levels

After the first stage, I will design the levels for the game. For this, I will have to create 2D sprites for objects such as collectables, the floor and obstacles. I will then have to place them appropriately within Unity.

I plan on doing this early on, as I believe it will make it easier for me to fine-tune the player controls later (such as the jump height). It will make my job a lot easier.

I will show my stakeholders the design of each level and will modify them, depending on how the stakeholders respond to the levels. For example, if a level is too short, then I will make it longer.

Each level must include:

- Collectable items, so that you can increase your score
- Different obstacles to make the levels more varied
- A finish line, so that the level can be completed

Test #	Description	Data to Test	Result Expected	Result Observed
5	Valid Does each level contain collectable items?	Levels	The level contains collectable items such as coins.	
6	Valid Are there obstacles in each level?	Levels	There are obstacles such as boxes you'd have to jump over.	
7	Valid Is there a finish line in each level?	Levels	There is a finish line at the end of each level.	
8	Valid Is there a clear path to complete each level?	Levels	You can easily see a path you must take to carry on in each level.	
9	Valid Do menu buttons take you to the right level?	Buttons	The buttons take you to the correct level.	

After I have completed the interface, I will then ask the stakeholders what they think of them: so, what they like and what could be improved. At a later stage, I will then get back to this and address any of their issues.

Stage 3: The Player

In this stage, I will focus on creating controls for the player and I will also be testing to see if the player with the controls can complete a level. I'm doing this after the level creation stage, as this will allow me to adapt the player controls to the levels, which like I said earlier, will make this task a lot easier.

In this stage, I will add scripts that will allow the player to move and shoot projectiles.

The player must:

- Collide with obstacles so that the player doesn't fall out of the map
- Collide with coins to collect them to increase the score
- Fail the level if they fall out of the map, to prevent endless falling
- Collide with the finish line to end the level

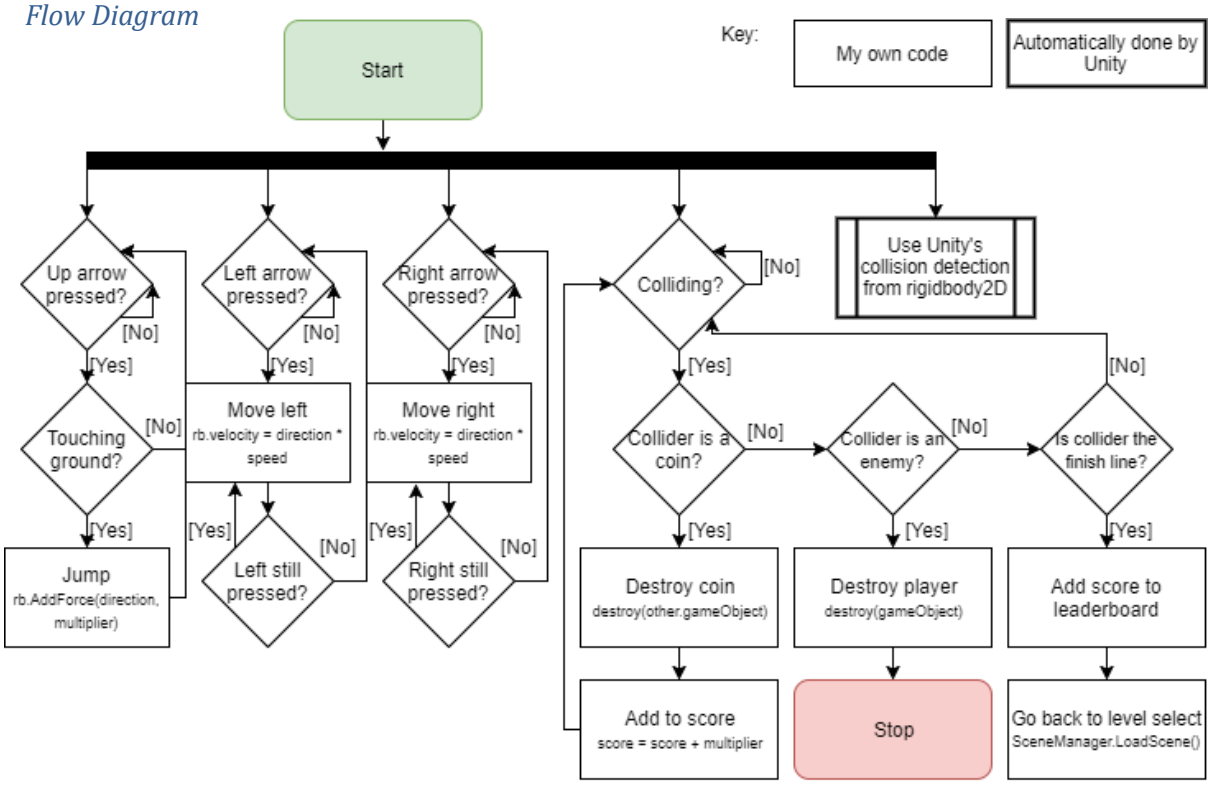
Test #	Description	Data to Test	Result Expected	Result Observed
10	Valid Does the player score go up as the player moves closer to the finish line?	Score counter	The score goes up at the player moves to the right.	
11	Valid Does the player collide with walls?	Player	The player collides with walls and stops moving.	
12	Valid Does the score increase as the player collides with coins?	Score, Coins	The coins disappear as they collide with the player and the score goes up.	
13	Valid Does the player fail if they fall out of the world?	Player	The player fails, and they must restart the level.	
14	Valid Does the player complete the level when they cross the finish line?	Player, Finish Line	The player finishes the level.	
15	Valid Is the player able to shoot projectiles?	Player	Yes, the player shoots projectiles.	
16	Valid Do the projectiles stop when they hit an obstacle?	Player, obstacle	The projectile stops when it touches an obstacle.	

Once I have completed the player controls, I will reach out to my stakeholders and ask them about these controls and gather their opinions on them. I will then take their comments into consideration and adjust the code, until the stakeholders like it.

Data Table

Name	Data Type	Description
rb	Rigidbody2D	A class built into the Unity engine. This allows you to control the player's position with physics simulation. Public functions such as 'AddForce' (`rb.AddForce(direction, magnitude)`) can be used to apply a force on the object to accelerate it or you could use variables such as 'Velocity' (`rb.velocity = direction * magnitude`) to apply a constant velocity when a key is pressed, for example. If this class is present in the player's code, they will be affected by gravity and other objects acting upon them.
Speed	Float	This will be the magnitude used for the left and right movement when paired up with the velocity variable from Rigidbody2D.
jumpHeight	Float	I will use this variable to control the height the player will jump when the jump button is pressed. This will be used with the velocity variable from Rigidbody2D.
up, left, right	Boolean	These variables are needed because of the way unity works with buttons. They will be needed to detect in which direction the player should move.
other	Collider2D	This variable will be used with a trigger detection function. It will be used to manipulate the other object or check information about it.
coins	Integer	The coin count will be stored here.

Flow Diagram



Pseudocode

```
//This is run once at the beginning of the gameplay
Void Start () {
	// GetComponent allows you to use the variables and functions of
	Rigidbody2D.
	rb = GetComponent<Rigidbody2D> ();
}
//This only runs when the players collider enters another collider
Void OnTriggerEnter2D (Collider2D other) {
	if (other.gameObject.tag == "Coin") {
		Destroy (other.gameObject);
		coins = coins + 1;
	}

	if (other.gameObject.tag == "Enemy") {
		Destroy (gameObject); //Destroys the player upon contact
	}
}
//Runs every frame, made specifically for physics calculations
Void FxiedUpdate () {
	rb.velocity = new Vector2 (rb.velocity.x * damping,  rb.velocity.y);
	//makes sure that the object doesn't stop moving after the directional
	buttons aren't pressed, instead the player slowly glides until friction
	stops them.
}
Void Update () {
	if (up) { //See 'public void pointerLeftDown/Up' for how this
		works
		rb.velocity = new Vector2 (rb.velocity.x, jumpHeight);
		up = false;
	}

	if (left) { //works in the same way as above
		rb.velocity = new Vector2 (-speed, rb.velocity.y);
	}

	if (right) {
		rb.velocity = new Vector2 (speed, rb.velocity.y);
	}
}

//These functions are assigned to buttons within the Unity editor. For example,
when a button is pressed down, a certain function can be assigned to that trigger,
same with when the buttons is released.

Public void pointerLeftDown() {
	left = true;
}
Public void pointerLeftUp() {
	left = false;
}

Public void pointerRightDown() {
	right = true;
}
Public void pointerRightUp() {
	right = false;
}
```

Stage 4: The Enemies

This stage will focus on creating the enemies for this game, which are the next most important thing, as without the features before this, such as the player and the level, this stage wouldn't be tested properly until later stages.

This stage will have a few scripts, the first one being the script that will allow them to move and jump over the obstacles, and the second one being one that makes the player fail the level if they hit the enemy. The third script will check if a projectile from the player has hit the enemy.

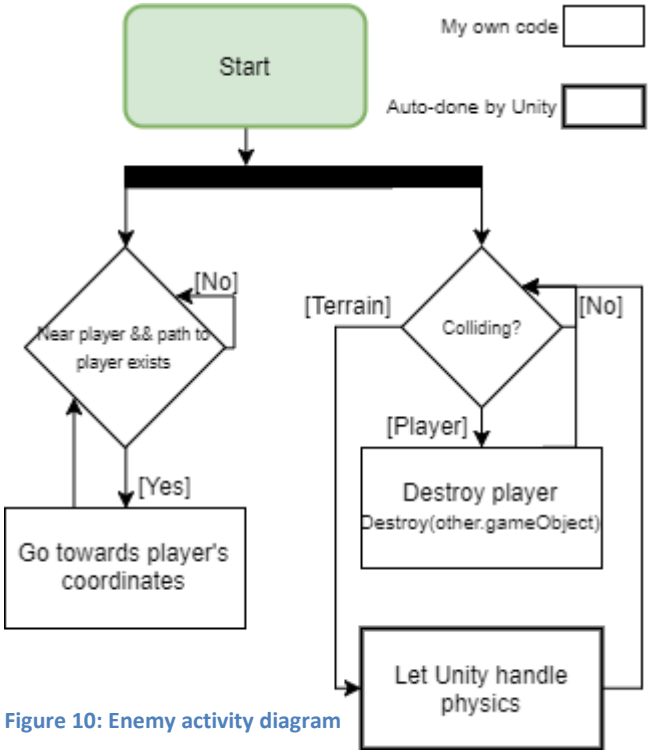


Figure 10: Enemy activity diagram

The enemy must:

- Collide with walls and the floor like the player
- Be immune to any obstacles what would hurt the player
- Make the player lose when they collide with them
- Jump over any stationary obstacles and holes
- Not pick up any collectables for the player

Test #	Description	Data to Test	Result Expected	Result Observed
17	Valid Does the enemy collide with walls and the floor?	Enemy, floor, obstacles	The enemy collides with all floors and obstacles.	
18	Invalid Does the enemy get destroyed when they collide with obstacles harmful to the player?	Harmful obstacles, enemy	The enemy does not get destroyed by the obstacles, nothing happens to the enemy.	
19	Valid Does the player lose when they collide with the enemy?	Enemy, player	The player fails the level and must start again.	
20	Valid Is the enemy able to go move and jump over obstacles?	Enemy, obstacles	The enemy moves towards the player and avoids obstacles.	
21	Invalid Does the enemy pick up collectables that only the player should?	Enemy, collectables	The enemy just goes through the collectables.	
22	Valid Is the enemy destroyed when a projectile from the player is fired at them?	Player, enemy	The enemy is destroyed upon contact with the projectile.	

Once the code for this is complete, I will then ask my stakeholders to see if they like how the enemy moves and if it should be easier or harder to avoid them. If the majority says that it's too hard or easy to avoid them, then at a later stage I will fix that. If there are also any other suggestions, which could make the game better, then I will implement those too.

Stage 5: The Leaderboard

In this stage, I will create a leaderboard that is accessible through the main menu, which shows all the top scores in the game. I believe this should be one of the last modules of the program that should be created, as this is probably the least important part of the game, as it doesn't really affect the gameplay at all.

At this stage, I will add a script that will display the high scores in descending order.

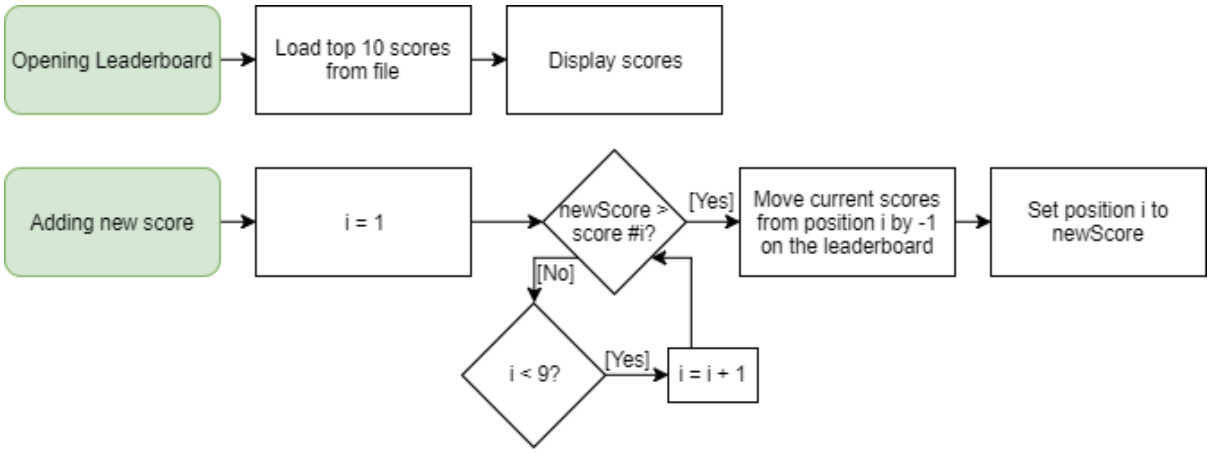
The leaderboard must:

- Show the high scores
- Display the high scores in descending order
- Say what level the score was achieved in and what difficulty it was achieved on

Test #	Description	Data to Test	Result Expected	Result Observed
23	Valid Does the leaderboard display the high scores?	Leaderboard	The leaderboard displays the highest scores	
24	Valid Does the leaderboard display the scores in descending order?	Leaderboard	The scores go down as you go down in the leaderboard	
25	Valid Is the correct level displayed along with the score?	Leaderboard	The leaderboard groups data correctly	

As with the other parts of this project, I will reach out to my stakeholders to see if they like this leaderboard. If they believe that I should change anything, then I will take their comments into consideration and alter it to their liking.

Flow Diagram



Stage 6: Options and Settings

This will be the final part of the development plan. In this part, I will be creating settings which will be accessible from the main menu screen via a settings button. This will allow the player to change the difficulty of the game, which will have its drawbacks and advantages, depending on the difficulty setting they choose. There will also be volume controls for sound effects.

In this part, I will only add a script that will change the volume of sound effect and one that changes the difficulty of the enemies and the score multiplier.

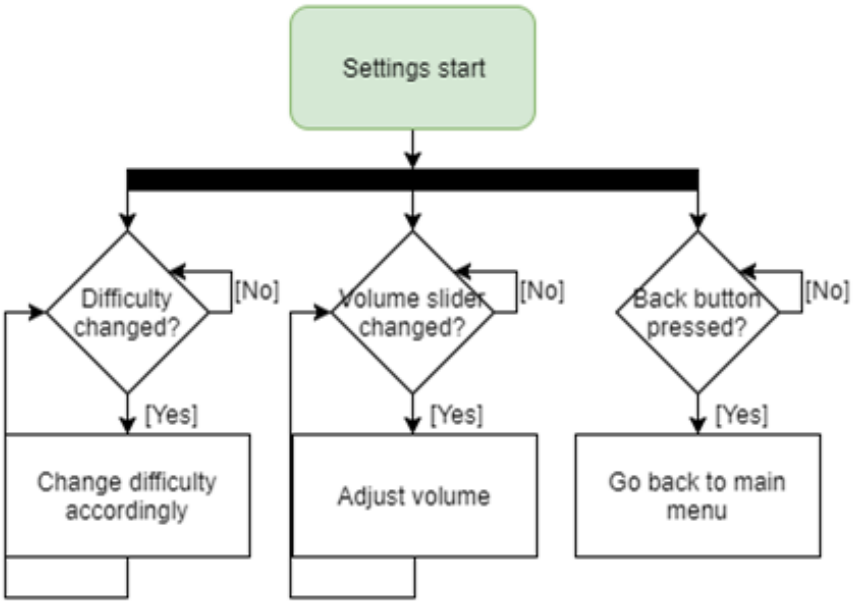
The settings must:

- Change the volume of sound effects
- Change the difficulty of the game

Test #	Description	Data to Test	Result Expected	Result Observed
26	Valid Does the volume change as the slider is moved?	Volume slider	The sound effect volume changes accordingly.	
27	Valid Does the difficulty of the game change when the difficulty setting is changed?	Difficulty buttons, enemies	The enemies are a lot more aggressive when the Hard button is pressed and a lot less aggressive when the Easy button is pressed.	
28	Valid Does the coin multiplier change, depending on the difficulty?	Difficulty buttons, score	On the default difficulty, you get a regular amount of coins (multiplier of 1x), on the Easy difficulty you get fewer coins (0.75x) and on the Hard difficulty, you get more coins (1.5x).	

After this stage is completed, I will ask my stakeholders if they think the coin multipliers are appropriate for each difficulty. If the multipliers are not appropriate, then I will change them accordingly.

Flow Diagram



Design Plan:

In this stage, I will be creating plans for my user interfaces, for each screen of the game.

Mock-Up

Figure 11: Main Menu

Figure 11: Leaderboard

Figure 12: Settings

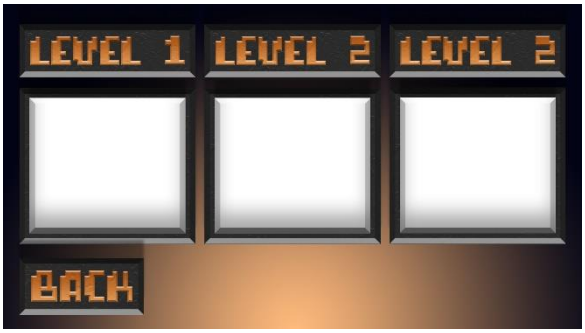


Figure 13: Level Select

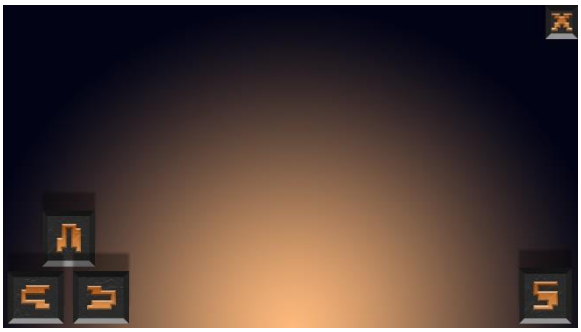


Figure 13: Level 1 / 2 / 3

Positions:

Everything is positioned to fill the screen as much as possible while staying at the centre of the screen (apart from the 'back' buttons) to avoid scaling issues with smaller size and resolution screens. The back button is located at the bottom left of each screen that has it to prevent the users from hitting the buttons accidentally. In the game screen, everything is in the corners of the screen, to allow for the best visibility of the game itself, plus having everything off to the side allows the player to easily reach all the buttons when they need to.

Font Style and Size:

The font used is called Pixelife, I used it as it fits the theme of the game (it will have a retro style to it, so it must be sharp and chunky). This ensures that the text is UI-friendly, as the text fills every button it's on almost completely to ensure the text is as readable as possible.

Colours Used:

The colours I will use primarily will be orange and black. This makes the UI user-friendly, as they contrast with each other, ensuring that all text is easily readable.

Menu Choices:

I have decided to make the menus simple, to not confuse the users. There are also not many of them, as if there were any more menus than needed, that would over complicate things, making the UI less user-friendly.

Stakeholders' Feedback

After creating these plans for the user interfaces, I reached out to some people online and some of my friends to ask them for opinions on the interfaces and if anything should be changed. Their responses were all almost identical: everyone seems to like how the UI looks as it is, but a lot of them recommended me to change the background as it seems too dark on some of the corners. They suggested replacing the background with a video of the gameplay. When I develop the game itself, I will try my best to implement this into each screen, apart from the game screen of course, as there will be actual gameplay there instead of the background.

Development Stage

Stage 1: User Interface

As I stated in the plan, I started the project with the user interface. I began this stage by creating all the individual scenes within the Unity engine: I created the main menu screen, level select, leaderboard and settings. After that, I added all the individual buttons as well as other elements that make the UI look like on the design ideas I have created.

```
public Button button;

    void Start () {
        source = GetComponent<AudioSource> ();
        Button btn = button.GetComponent<Button> ();
        btn.onClick.AddListener (TaskOnClick);
        //The above line sets a listener on the button attatched and
        //directs it to the TaskOnClick subroutine.
    }
```

The above code is what I first created in the script menuButtonPress.cs which is used on all the buttons that used for changing scenes, except for the 'Yes' and 'No' buttons that have been added later. Originally, I intended on having sound effects when the buttons are pressed, so I added a variable of type AudioSource called 'source'. Within Unity, I then assigned an audio file for a button press, using the 4th line above which allows me to do so. I also added the 5th line to tell Unity what button the script should refer to (This was necessary as later I use a switch statement and this script is used on all the buttons that take you between different screens.). The next line is explained using the comment below it.

On the next page is the subroutine I have created to check what button is being pressed. I did this by creating a switch statement which takes in the object's name as the case and executes appropriate code based on the name. After I created this, I noticed that I got runtime errors. Here's one of them:

Figure 13: The first error I received

I fixed this by adding all the scenes under 'Build Settings'. Doing so allowed me to switch between scenes within the editor and a compiled version of the game. Earlier on I decided that I wanted a sound effect to play when buttons are pressed, so the first thing I got the program to do was to play the audio clip. I did so by adding a component to all buttons called Audio Source which allowed me to play audio clips attached to the scripts. What I did below did work, however, I noticed that the audio cut off when switching between scenes. Due to this, I removed the code that played the audio clips and I removed the components I added to all the buttons until I found a better way of playing the sound effects.

```
    private AudioSource source;
    public AudioClip buttonPress;
//When the listener detects a button press, this subroutine is called.
    public void TaskOnClick () {
        source.PlayOneShot (buttonPress, 1);
        //This subroutine checks what button is pressed based on the name
        //and does the appropriate things like loading the right scene.
        switch (gameObject.name) {
        case "Start":
            SceneManager.LoadScene ("Level Select");
            break;
        case "Leaderboard":
            SceneManager.LoadScene ("Leader Board");
            break;
        case "Settings":
            SceneManager.LoadScene ("Settings");
            break;
        case "Exit":
            Quit ();
            break;
        case "Yes":
            Application.Quit ();
            break;
        case "No":
            CancelQuit ();
            break;
        case "Select_Back":
            SceneManager.LoadScene ("Main Screen");
            break;
        case "Leaderboard_Back":
            SceneManager.LoadScene ("Main Screen");
            break;
        case "Settings_Back":
            SceneManager.LoadScene ("Main Screen");
            break;
        case "Button 1":
            SceneManager.LoadScene ("Level 1");
            break;
        case "Button 2":
            SceneManager.LoadScene ("Level 2");
            break;
        case "Button 3":
            SceneManager.LoadScene ("Level 3");
            break;
        }
    }
```

Then, I needed to create the Quit and CancelQuit subroutines. I wanted these to create a small box that asks you if you want to quit the game or not: if the Exit button is pressed, the box pops up and if you press Yes, the game closes and if you press No, the box disappears.

```
    public GameObject question; //Any unassigned variables are
    public GameObject yes;      //assigned in the editor.
    public GameObject no;

    //This is executed when the 'Quit' button is pressed.
    private void Quit () {
        Debug.Log ("Activating question box");
        question.SetActive (true);
        yes.SetActive (true);
        no.SetActive (true);
        Debug.Log ("Done");

    }

    //This is exectued when the 'No' button is pressed.
    private void CancelQuit () {
        Debug.Log ("Closing question box");
        question.SetActive (false);
        yes.SetActive (false);
        no.SetActive (false);
        Debug.Log ("Done");
    }
}
```

I had to create new variables of type GameOject that I assigned the text box, yes and no buttons to be able to activate or deactivate them, depending on what button is pressed. Within Unity, I assigned the appropriate objects to these variables. Then I created the above scripts that link in with the previous part of the script; When the Quit button is pressed, the text box, yes and no buttons are activated (initially they're de-activated) and so they pop up on the screen in front of everything else. The opposite happens when you press the No button; the objects are de-activated and so disappear from the screen. I used the debug log command to do some troubleshooting, as I noticed that the text box would not appear when I pressed the Quit button. I swiftly realised that was due to me not assigning the game objects within the Unity engine, meaning that nothing was being activated.

After that was done, I went back to working on the button sound effects. Through experimenting, I found out that the easiest way to get the sounds to play was to do the following:

```
    private AudioSource source;
    public AudioClip buttonPress;

    // Use this for initialization
    void Start () {
        DontDestroyOnLoad (this.gameObject);
        source = GetComponent<AudioSource>(); //Gets AudioSource component
    }

    public void PlaySound () {
        source.PlayOneShot (buttonPress, 1); //Plays the sound
    }
```

This code was added to the SoundController.cs file which was attached to a game object of the name Sound Controller. The idea was to have the previous script I made call the procedure PlaySound. I made this all work by adding the following lines of code:

```
    public SoundController soundController;
```

The above line was added to the first script I made so that I can access the procedure PlaySound from the Sound Controller object. Then, to execute the procedure, I added this line of code within the TaskOnClick procedure:

```
        soundController.PlaySound ();
```

Then within Unity, I assigned the soundController variable to the Sound Controller object. I then realised that to have all buttons in all scenes play sounds, I needed to duplicate the Sound Controller game object to all scenes so that I can assign it to the first script, used in all the menu buttons.

After this was made, I tested this feature and noticed that every time a scene is loaded, a duplicate of the Sound Controller object was created, therefore I needed a way to remove the unnecessary ones. I did the following changes in the sound controller script:

```
    public void PlaySound (bool checkIfDestroy) {
        source.PlayOneShot (buttonPress, 1); //Plays the sound
        StartCoroutine(Wait (checkIfDestroy));
    }

    IEnumerator Wait (bool checkIfDestroy) {
        yield return new WaitForSecondsRealtime (0.288f);
        if (checkIfDestroy == true) {
            Destroy (this.gameObject);
        }
    }
}
```

I have created the co-routine Wait to remove the Sound Controller object after it has finished playing the sound effect if the parameter is true. The parameter is true if the button you press takes you to a different screen; it needs to stay if it's a button press that doesn't change the screen, as if I removed the object, no sounds would play until I changed screens. I needed to create a co-routine as I needed to wait a certain amount of time before I destroyed the game object, as there's no other way of pausing a script without pausing the whole game. The PlaySound procedure gets the parameter from the button press script I initially created:

```
        case "Leaderboard":
            soundController.PlaySound (true);
            SceneManager.LoadScene ("Leader Board");
            break;
```

Each name case in the switch statement within the previous script now has a PlaySound procedure in it with a parameter which is either true or false, depending on whether the scene is changed. If the scene will be changed, the parameter is true and if it's not then it's set to false.

With this last bug fixed, everything was working correctly, and the first stage of my development has been completed.

Test #	Description	Data to Test	Result Expected	Result Observed
1	Valid Does the app open when the icon is pressed?	Icon	The app opens when the icon is clicked.	Yes, the app opens properly.
2	Valid Do the buttons in-game do appropriate things?	Buttons	The buttons take you to correct screens.	Yes, all the buttons do what they should.
3	Valid Does the exit button close the game?	Exit button	The exit button closes the game.	Yes.
4	Valid Do the buttons scale with the screen resolution?	Buttons	The buttons increase in size as the resolution goes up and vice versa.	Yes, the entire UI scales with the screen resolution.

Test #	Evidence
1	https://drive.google.com/open?id=1e_WplJhJ2gVSymMo551OqW5TRat18vxS
2	https://drive.google.com/open?id=1jmgU-FuPh16uHCJspHzThUQ8Vk3G7G0u
3	Can be seen in evidence 2.
4	https://drive.google.com/open?id=1aj1x4f0C7LWid07zNOo7qxulTvf_u3Q3

After testing all the elements in the table above in my current state of the product, I have observed the above results. At the end of this stage, every element has been working correctly. I believe this stage has been very successful as I have met all the success criteria that I created in the development plan: so, you can move between the different menus and screens the way I planned in the success criteria, the buttons scale properly with the screen and the exit button closes the game fully.

I asked the stakeholders for some feedback on this and nobody had any sort of complaints. There was only one person who said the font was a bit weird but they could read it fine so I interpreted that as a personal preference, therefore I see no reason to change the font.

Stage 2: Creating Levels

Before starting designing the levels for the game, I decided that it would take too long to place all objects manually, therefore I have decided to write a script that will generate the level for me, based on colour data of an image. In short, you assign the level image within unity to the script and the script will use for loops to go through the entire image, placing appropriate objects based on the colour data of each pixel. For example, if the pixel being checked is back, it will place a generic platform block and if it's completely transparent with no colour, it won't place anything.

I began this script by creating all the necessary variables:

```
    public Texture2D level;

    public Color lightPlatform;
    public Color darkPlatform;
    public Color enemy;
    public Color player;
    public Color spikes;
    public Color leftBush;
    public Color midBush;
    public Color rightBush;

    private Color pixel;

    public GameObject lightPlatformPrefab;
    public GameObject darkPlatformPrefab;
    public GameObject enemyPrefab;
    public GameObject playerPrefab;
    public GameObject spikesPrefab;
    public GameObject leftBushPrefab;
    public GameObject midBushPrefab;
    public GameObject rightBrushPrefab;

    private Vector3 position;
```

The Texture2D level variable is the variable that will hold the image the script will generate the level from. The Color variables will be used for comparisons to individual pixels on the image, as stated above. Depending on which comparison gives a True outcome, a game object will be instantiated. The GameObject variables will be used to hold the prefabs of all the platforms, enemies, players etc.

I then created some code in the void Start() procedure (so that it only creates the level once when the level is loaded) that will loop through all the pixels of the image and will compare the values of individual pixels to the colours assigned to each individual game object above. Here is the code:

```
        for (int i = 0; i < level.width; i++) {
            for (int j = 0; j < level.height; j++) {

                pixel = level.GetPixel (i, j);
                position.Set (i, j, 0.0f);

                if (lightPlatform.Equals (pixel)) {
                    Instantiate (lightPlatformPrefab, position, Quaternion.identity);
                } else if (darkPlatform.Equals (pixel)) {
                    Instantiate (darkPlatformPrefab, position, Quaternion.identity);
                } else if (spikes.Equals (pixel)) {
                    Instantiate (spikesPrefab, position, Quaternion.identity);
                } else if (leftBush.Equals (pixel)) {
                    Instantiate (leftBushPrefab, position, Quaternion.identity);
                } else if (midBush.Equals (pixel)) {
                    Instantiate (midBushPrefab, position, Quaternion.identity);
                } else if (rightBush.Equals (pixel)) {
                    Instantiate (rightBrushPrefab, position, Quaternion.identity);
                }
```

After compiling the code, I assigned all the variables such as the colours and prefabs accordingly and ran the game. I instantly noticed that the player and enemies weren't spawning. To solve this, I looked back at the code and noticed that I forgot to add code to add the player and the enemies. I fixed it by adding the following code at the end of the long if statement in the code above:

```
                } else if (player.Equals (pixel)) {
                    Instantiate (playerPrefab, position, Quaternion.identity);
                } else if (enemy.Equals (pixel)) {
                    Instantiate (enemyPrefab, position, Quaternion.identity);
                }
```

After this, all the prefabs were instantiated correctly, and the second stage of the program was technically done. However, I have decided that I want to allow players to make custom levels as I know that I'm not very creative with my levels. Therefore, I have decided to add an extra button in the level select menu that will allow the player to select their own image and play the level generated from the image.

```
        if (EditorSceneManager.GetActiveScene().name == "Custom Level") {
            filePath = EditorUtility.OpenFilePanel ("Select level", "", "");
            Debug.Log (filePath);
            WWW www = new WWW ("file:///" + filePath);

            yield return www;

            level = www.texture;
        }
```

I have created the following code to check if the level that the script is used on is called "custom level". If it isn't, the script will just use the level image provided within unity. If the name is Custom Level, then it will give the player a file select box. This file's path will then be accessed, and the selected file will be set as the level image. However, it turns out that this code doesn't work outside of the Unity Editor, therefore I am forced to leave this code out for now until I find out more about it.

Then I created levels using Adobe Photoshop. Here's the images of the three levels:



In this image, the black represents the walkable surface, orange represents spikes, red represents enemies, yellow represents coins, cyan represents keys, white represents the finish line (note: this image is meant to be transparent where there is nothing, so all empty space appears to be white), green represents the spawn position and the three shades of purple represent the bushes.

I tested each aspect of the above levels: I checked if it's possible to collect all coins, complete the level, get to all platforms and so on. After several attempts of adjusting the above levels, I believe that they are now as good as they can be. I asked a few stakeholders to give me comments on the levels. Most of them said they like the look of the layout and thought that nothing should be changed, while others said some things should be changed. There was no pattern to the suggestions of change, therefore I didn't change anything in the above levels apart from the things I thought were appropriate. All levels have coins, spikes, enemies, keys and a finish line.

Test #	Description	Data to Test	Result Expected	Result Observed
5	Valid Does each level contain collectable items?	Levels	The level contains collectable items such as coins.	Yes.
6	Valid Are there obstacles in each level?	Levels	There are obstacles such as boxes you'd have to jump over.	Yes, there are walls and drops for the player to overcome.
7	Valid Is there a finish line in each level?	Levels	There is a finish line at the end of each level.	Yes, although it can't be seen on the images due to the colour I used.
8	Valid Is there a clear path to complete each level?	Levels	You can easily see a path you must take to carry on in each level.	Yes, all levels are completable.
9	Valid Do menu buttons take you to the right level?	Buttons	The buttons take you to the correct level.	Yes.

Test #	Evidence
5	Can be seen in the level images I have created. Yellow pixels stand for coins, light blue stand for keys (items worth more than coins).
6	Can be seen in the level images I've created. Black pixels are walls and orange pixels are spikes (also obstacles, but they harm the player).
7	Finish lines are represented as white pixels on time level images, however since I can't display transparency in a document, it looks like the background.
8	You can see that there is a clear path to the finish line on the level images.
9	Level 1: https://drive.google.com/open?id=1yAEbPrGU8cg0b0eEHtgvGLAni_jNrD3_ Level 2: https://drive.google.com/open?id=1bviMAlAcNfsRlypMcyzKX4XGH-8Ks0tb Level 3: https://drive.google.com/open?id=1hLBeaQoK5904QMpmDcJXU0StCa6haBs8

After all the testing has been completed, everything was working correctly, apart from the finish line, which I have not implemented yet, as I couldn't test it due to the lack of a player script to control the player.

Other than that, I believe this stage has been very successful. This is because, again, I met all the success criteria: there are collectables that will increase your score (code for this will be implemented when I create player scripts to avoid having to re-write code), There are varied obstacles throughout all the levels and there is a finish line, but it doesn't function yet for the same reason as the collectables.

The stakeholders for this game said they rather liked the layouts of the levels and how large they are, so I believe this is further evidence that this stage has in fact been successful.

Stage 3: The Player

After the code to create levels had been completed, it was time to implement the player. I began this by making a simple sprite with the rigidbody2D component. At first, I decided to focus on making the physics work. I began this by creating simple controls for PC, as that is easier than setting up mobile controls and I can easily convert them to mobile controls when I feel that I'm happy with how they play.

```
    void FixedUpdate () {
        #if UNITY_STANDALONE
        if (Input.GetKeyDown (KeyCode.Space){
            rb.AddForce (transform.up * jumpMultiplier);
        }
        if (Input.GetKey (KeyCode.A)) {
            rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
            if (direction == "right") {
                direction = "left";
                transform.localRotation = Quaternion.Euler (0f, 180f, 0f);
                //changes rotation of player
            }
        }
        if (Input.GetKey (KeyCode.D)) {
            rb.velocity = new Vector2(forceMultiplier, rb.velocity.y);
            if (direction == "left") {
                direction = "right";
                transform.localRotation = Quaternion.Euler (0f, 0f, 0f);
            }
        }
        #endif
    }
```

I created these physics controls inside FixedUpdate, as according to Unity, this is specialised for physics calculations and similar things. The first line and last line of code inside this are there so that I can test the game on both mobile and PC once I implement the mobile controls, without filling up the memory on mobile devices (this code doesn't get compiled when exporting to Android), which is crucial as mobile devices don't have much memory.

After that, I added the three individual controls for jumping, moving left and moving right. I used rb.AddForce for jumping as from previous experience, using force to simulate jumping looks the most realistic. This used a vertical vector for direction (transform.up) and a jumpMultiplier variable that I can adjust within unity.

The controls for left and right work identically so I'll only describe one. For moving left, I simply changed the velocity of the player as I didn't want to use a force because a force would make the player accelerate which I didn't want. The velocity of the player in the horizontal plane was set to a variable that can be changed within the editor, and I set the vertical velocity to what it was before pressing a movement key, because this is the only way to implement realistic movement without using forces; the player would not move vertically when moving horizontally. Then, a simple string is used to check which way the player is facing. By default, this string is set to "Right", however, if you press the left movement key, the player needs to be flipped, so this piece of code does that.

After testing this, I have realised that the player can jump as many times in the air as they want. That is not something I wanted; I only wanted the player to be able to jump from the ground once and in the air once like many platformer games have. Because of this, I needed to create a function that checks if the player is touching the ground.

```
    private bool CheckIfGrounded () {
        if (rb.velocity.y <= 0) {
            foreach (Transform checker in groundCheckers) { // BELOW: creates coll
iders at each ground checker
                Collider2D[] colliders = Physics2D.OverlapCircleAll (checker.posit
ion, radius, ground);
                for (int i = 0; i < colliders.Length; i++) {
                    if (colliders [i].gameObject != gameObject) { //if collider lo
oked at isn't player
                        maxJumpCount = 2;
                        return true; //return true if colliding with object other
than player
                    }
                }
            }
        }
        return false;
    }
```

Because of this, I created the above code after creating three empty game objects below the player's feet and attached them to the player, so that they move with the player and stay in the same position relative to the player. To reduce checking time, I added an if statement that checks if the player's velocity is smaller than or equal to zero. This is because, if your vertical velocity is greater than zero, you can't physically be touching the ground as you're travelling upwards, so when the vertical velocity is greater than zero it just skips the big chunk of code and returns false, meaning that jumping is more accurate. However, if the vertical velocity is not greater than zero, we need to

check if the player is touching the ground using colliders. Earlier in the code, I added a transform (tells you the position, scale and rotation of an object) array to store all the ground detectors. Then, for each of the ground detectors, the script creates 2D colliders where the ground detectors are. The script then goes through all the colliders to see if they're touching anything, and if any of them are touching something, the script returns true. If none of them touch the ground, the script returns false.

I then altered the jumping script to the following:

```
        if (Input.GetKeyDown (KeyCode.Space) && (isGrounded || maxJumpCount != 0))
{
            rb.AddForce (transform.up * jumpMultiplier);
            maxJumpCount--;
            Debug.Log(maxJumpCount);
        }
```

The script now only lets you jump if the player is grounded or if the player hasn't run out of jumps. The maxJumpCount variable is set to 2 as the player can jump once from the ground and then once in the air. Once the player hits the ground, the maxJumpCount is reset to 2, in the ground check function, if the script confirms that the player is touching the ground.

The next most appropriate feature to include was the player animations. Since I downloaded an asset pack (details about this are in the bibliography), All the animation frames for the player and enemies walking have already been created, however, as separate frames. Therefore, I needed to find a way of making these animations work within Unity. After several hours of experimentation, I have figured out that this can be achieved by adding an animator component to the player within Unity and by creating a playerAnimator variable that has the animator component attached to it. I did this within the script like this:

```
    void Start () {
        rb = GetComponent<Rigidbody2D> ();
        playerAnimator = GetComponent<Animator> ();
    }
```

As you must do with the rigidbody component, we need to use GetComponent to get the animator component script to allow us to use animator-related commands within our script. Then, I went back to the Unity editor and created a new animation. I sliced up the image with all the player movement frames and placed them one after another appropriately in the animator timeline to create a working animation for the player.

After that, I attached that animation to the player's animator script, so that it could be accessed within the code. I did the same with the jumping and falling animations. Then, to use these animations on the player, I created the following code:

```
    void Update () {
        if (!(Input.GetKey(KeyCode.A)) && !(Input.GetKey(KeyCode.D)) && isGrounded) {
            playerAnimator.Play ("Player Idle"); //Plays the animations
        }

        if (!isGrounded && rb.velocity.y > 1f && (!Input.GetKey (KeyCode.A) || !Input
.GetKey (KeyCode.D))) {
            playerAnimator.Play ("Player Jump");
        }

        if (!isGrounded && rb.velocity.y < -
1f && (!Input.GetKey (KeyCode.A) || !Input.GetKey (KeyCode.D))) {
            playerAnimator.Play ("Player Fall");
        }

        if (isGrounded && (Input.GetKey (KeyCode.A) || Input.GetKey (KeyCode.D))) {
            playerAnimator.Play ("Player Walk");
        }
    }
```

This part of the script checks whether certain animations should be played. The first if statement checks to see if the player isn't checking any keys. If they're also grounded, then the player idle animation will play (there will just be a static image of the player). Then, if the player isn't grounded and their vertical velocity is greater than 1 (not greater than zero, to avoid visual bugs) and if the player isn't grounded and if none directional keys are pressed, the jump animation is played and so on.

After testing these animations, I have decided that they work as I intended them to. However, while testing the animations while walking and jumping, I realised that the player has inconsistent jump heights. During the debugging of this issue, I have concluded that the player jumps too high whenever they touch multiple hitboxes. Unfortunately, the only solution to this that fixed the issue for me was to manually set the player's vertical velocity, instead of applying a vertical force:

```
        if (Input.GetKeyDown (KeyCode.Space) && (isGrounded || maxJumpCount != 0)) {
            //rb.AddForce (transform.up * jumpMultiplier);
            rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
            isGrounded = false;
            maxJumpCount--;
            Debug.Log(maxJumpCount);
        }
```

This means that the jumping isn't as smooth, however, I believe that it's more important that the player jumps the same height every time they jump, instead of the jump height being different every jump. Fixing this issue means that the player won't be able to jump up entire walls they should be able to get over, which is a good thing, as it limits the number of ways they can get to certain locations, meaning they must be more creative with how they get there, instead of simply jumping to the place they want to get to.

Then, I decided that I wanted to change how the player works; I decided that I want the player to eliminate enemies by stomping on them, just like in Super Mario Run, as I could not find any sprites for projectiles that could fit the theme of the game, also it would make it more challenging to take out enemies; being able to just shoot them would be too easy. Because of this, I needed a way of finding out whether the player is stomping on the enemies or if they're just touching them.

To do this, I decided to re-use the code from ground detection, but I altered it to only detect enemies:

```
    public bool OnEnemy () {
        if (rb.velocity.y <= 0) {
            foreach (Transform checker in groundCheckers) {
                Collider2D[] colliders = Physics2D.OverlapCircleAll (checker.posit
ion, radius, enemy);
                for (int i = 0; i < colliders.Length; i++) {
                    if (colliders [i].gameObject != gameObject) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

The only parts of this that I altered was the layer mask that the 2D colliders were detecting, and I got rid of the line that reset the jump count of the player. To test this out, I created the following code:

```
    void OnCollisionEnter2D (Collision2D other) {
        Debug.Log ("Colliding...");
        if (other.gameObject.tag == "Enemy") {
            Debug.Log ("Touching enemy...");
            if(OnEnemy ()) {
                Debug.Log ("On top of enemy");
            }
            else {
                Debug.Log ("Not on enemy?");
            }
        }
    }
```

At first, this code didn't work, as I used Collider2D instead of Collision2D, which didn't give me information about the collision, but about the collider of the other object the player collided with. After that was fixed, the stomping detection worked correctly.

After more debugging, I have noticed that when the player sprite rotates, the camera rotates too; the whole screen is flipped along with the sprite. I realised that this is because, the player camera is attached to the player, so it has all the same coordinates and rotations applied to it. Because of this, I needed to separate the player camera from the player, so that it is no longer a child of the player in the object hierarchy. However, this meant that the player camera would no longer follow the player around, so I needed to create a script that would make the player camera follow the player, without it rotating with the player's sprite. I created the following code:

```
public class goToPlayer : MonoBehaviour {

    public Transform playerTransform;
    public float distance;

    // Update is called once per frame
    void Update () {
        playerTransform = GameObject.FindWithTag ("Player").transform;
        transform.position = new Vector3 (playerTransform.position.x, playerTransf
orm.position.y, -distance);
    }
}
```

I called the script goToPlayer, and it has two variables: the transform of the player, which is used to get the coordinates of the player, and a float distance so that I can adjust the area that the player can see. Every frame, this script looks for a game object with the “Player” tag, which is assigned through unity. It then modifies the position of the transform of the camera by using a three-dimensional vector with the x-coordinate of the player, the y-coordinate of the player, and negative the distance float variable. It’s negative, as making it positive inverts everything.

Upon testing this script, I realised that there was a performance drop, so I went back to the code and realised that the player transform variable only needs to be assigned to the player game object once at the beginning of the game. So, I moved that line into the Update procedure of the script that Unity sees as one of its procedure, in this case, it’s a procedure that only runs once when the level is loaded.

Then, I needed to convert the controls so that they’re for mobile. So, I re-enabled all the UI elements I disabled at the beginning of the player creation and made the move left, move right and jump scripts as separate, public procedure that can be called when buttons are pressed:

```
    public void Jump () {
        if (isGrounded || maxJumpCount != 0) {
            //rb.AddForce (transform.up * jumpMultiplier);
            rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
            isGrounded = false;
            maxJumpCount--;
            //Debug.Log(maxJumpCount);
        }
    }

    public void MoveLeft () {
        rb.velocity = new Vector2 (-forceMultiplier, rb.velocity.y);
        if (direction == "right") {
            direction = "left";
            transform.localRotation = Quaternion.Euler (0f, 180f, 0f); //changes r
otation of player
        }
    }

    public void MoveRight () {
        rb.velocity = new Vector2 (forceMultiplier, rb.velocity.y);
        if (direction == "left") {
            direction = "right";
            transform.localRotation = Quaternion.Euler (0f, 0f, 0f);
        }
    }
```

So instead of each if statement responsible for movement executing the code in the above procedures, I put the code in separate procedures so that I can call them from a script I'll make for the movement buttons later.

However, I have reached the point where I am unable to continue making this game for Android, as I simply cannot find a way to make the player controls work. There is no option in Unity to easily detect when a button is being pressed down and released (UI button, you can use procedures to see if keyboard/mouse buttons are being pressed). Therefore, I need to adapt the project so that it can run on other platforms. I have decided to simply port it to Windows, as I do not own any other operating systems that I could test the game on. So, to adapt the game to PC, I have removed all the on-screen buttons in the level scenes and left in the code I used for debugging within unity (the code that lets you control the player with keyboard keys) and removed the tags that don't export the code during compilation, so now it has been ported to Windows.

Because of this complication, I have decided to move the movement code back to the if statements they were originally in, as this way I am not creating unnecessary functions or procedures.

The next stage for me was to create the code that controls the score that is displayed for the player to see. To do this, I have created a text UI element and modified it within the playerController script, to keep the number of scripts to a minimum. Within OnCollisionEnter2D, I have created the following code:

```
        switch (other.gameObject.tag) {
            case "Key":
            score += 50;
            UpdateScore();
            break;

            case "Burger":
            score += 5;
            UpdateScore();
            break;

            case "Enemy":
            if (isOnEnemy) {
                score += 20;
                UpdateScore();
            }
            break;
        }
```

Here, I use a switch statement to go through the tag of the object the player is colliding with and updating the score accordingly. The procedure UpdateScore does the following:

```
    void UpdateScore () {
        if (score < 10) {
            scoreText.text = "00" + score.ToString();
        } else if (score < 100) {
            scoreText.text = "0" + score.ToString();
        }
    }
```

If the score is under 10 the code adds two zeros before the score for a nice effect, same if the score is under 100, except it only adds one zero.

Upon testing this, I realised that this will not work as the objects I have created that the player can collect are triggers so they won't be detected in collisions, so I moved the switch statement into the OnTriggerEnter2D procedure.

However, this still didn't work so I needed to look for another solution. The solution for this was to remove the "Enemy" case in the switch statement and move the code that handles score calculation when some enemy dies over to the enemy controller script. I did the following:

```
public IEnumerator EnemyDeath () {
    if (!isDying) {
        GameObject.Find("Player(Clone)").GetComponent<playerController>().score += 20;
        GameObject.Find("Player(Clone)").GetComponent<playerController>().UpdateScore();
    }
    isDying = true;
    yield return true;
}
```

(Note: I switched to Visual Studio from MonoDevelop during development as I felt more comfortable with it, so the code here will have different syntax highlighting and background colours.)

The above code is called within the OnEnemy procedure within the player controller's script. It is an IEnumerator, as later, I want to further develop this procedure when I get to the enemy script properly. There is also a Boolean variable isDying. This is, as the name suggests, to check if the enemy is dying. If the enemy is dying, just before the isDying variable is set to true, the score in the playerController script is incremented by 20 and the score is updating using a procedure within the same script I just mentioned. Then, isDying is set to true, so if the player jumps on the enemy while they're dying, they will not get extra points for doing so. This feature was now working.

I then noticed that the player could jump three times instead of two times. Because of that, I had to drop the value maxJumpCount is assigned to within the check if grounded function from 2 to 1. This fixed the issue and you could no longer jump more than two times.

Then, I needed to create code for sound effects, since in the settings menu I have a volume slider, so I decided to give the player walking and falling sound effects. Later, I will also give the enemy sound effects too, but only when I get to the enemy stage.

At first, I created the following code:

```
IEnumerator PlayWalkSound () {
    if (isGrounded && (rb.velocity.x != 0f) && readyToPlay) {
        audioSource.PlayOneShot(walk, 1f);
        readyToPlay = false;
    }
    yield return new WaitForSeconds(walk.length + 1f);
    readyToPlay = true;
    yield return true;
}
```

This coroutine is called in the Update procedure, however, it only starts when readyToPlay is true (this is a variable to check if the sound can be played again yet, to ensure that you don't hear the sound every frame). However, I had to alter this code as the sound still played every frame after you start walking.

I altered the following:

```
IEnumerator PlayWalkSound () {
    if (isGrounded && (rb.velocity.x != 0f)) {
        audioSource.PlayOneShot(walk, 1f);
        yield return new WaitForSeconds(walk.length + 1f);
    }
    readyToPlay = true;
    yield return true;
}
```

I removed the Boolean readyToPlay from the if statement and instead moved it to the if statement in the Update procedure:

```
void Update () {
 if (readyToPlay) {
     readyToPlay = false;
     StartCoroutine(PlayWalkSound());
 }
```

This ensures that another instance of the coroutine isn't started until the sound effect has finished playing. I have also moved the delay WaitForSeconds inside the if statement in PlayWalkSound to make sure that there's no delay in execution of the sounds when the player doesn't move.

Then, I realised that the sound doesn't play often enough, so I removed the extra second from the WaitForSeconds and replaced it with the time it takes for the player to take one step in their animation. I have also later changed the conditions in the if statement within PlayWalkSound; I removed the check for a velocity greater than zero on the x-axis and replaced with a check for key presses responsible for moving the player, which in the end was much more reliable than checking the velocity.

Unfortunately, I was unable to get the falling sounds to work, as I could not figure out a way to make the game realise that the player has fallen a certain height and hit the ground. Because of this, I decided to skip this stage and move on to creating code that detects if the player has touched the finish line.

For now, since I haven't started the leaderboard stage, I only created temporary code for the finish line:

```
if (other.gameObject.name == "Finish Block") {
    Debug.Log("Player Won");
}
```

This if statement is within the OnCollisionEnter2D procedure, so this check is done with every object the player collides with. When I get to the leaderboard stage, I will replace this with code that will save the score to the leaderboard.

Then, the final part of the player stage was to create the health bar. Firstly, I created a slider within the Unity editor and disabled its slider handle, to make sure that the player cannot give themselves health. Then, I proceeded to create the following code:

```
        if (other.gameObject.tag == "Enemy") {
                //Debug.Log ("Touching enemy...");
                if(OnEnemy ()) {
                        Debug.Log ("On top of enemy");
                        rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
                        other.gameObject.SendMessage("Score");
                        other.gameObject.SendMessage("EnemyDeath");
                }
                else {
                  healthBar.value -= damage;
                  if (healthBar.value <= 0) {
                      Debug.Log("Player is dead");
                }
         }
```

The part of the code I just wrote is the is the code inside the else statement (this is all within the OnCollisionEnter2D procedure). If the player's health falls to or below zero after touching the enemy and they didn't stomp the enemy, they will die. However, I'll only add this code once I get to the leaderboard stage as I want the scores to save even after a player dies, so that good scores aren't lost forever. The healthBar variable is of type Slider. The initial value of the slider is 10 and the value of 'damage' is 2, however, this will differ with difficulty later, once I get to stage 6.

Now that I've done the health bar I needed something that will discourage the player from touching enemies, as right now, you can touch an enemy and only lose health which wouldn't be very important until you are very low on health. Therefore, I have decided to punish the player when they touch an enemy; I will do this by making the player jump and lose control over the character until they land. I believe that this will discourage players from touching the enemies.

I decided to use the following code:

```
            else {
                playerAnimator.Play("Player Bonk");
                rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
                enableControls = false;
                healthBar.value -= damage;
                if (healthBar.value <= 0) {
                    notDead = false;
                    Debug.Log("Player is dead");
                }
            }
```

This is the 'else' part of the if statement that checks if the player is on top of an enemy. So if the player isn't on top of an enemy, a "bonk" animation plays, which looks like the player has tripped. The player also jumps up and loses controls over the character. I made use of the enableControls variable wherever an if statement for movement controls is used, for example:

```
            if (Input.GetKey (KeyCode.A) && notDead && enableControls) {
                    //MoveLeft();
                    rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);

                    if (direction == "right") {
                            direction = "left";
                            transform.localRotation = Quaternion.Euler (0f, 180f, 0f);
                    }
            }
```

Instead of just checking for if the player is pressing one of the buttons that control movements and that the player isn't dead, we're also now checking for if the controls are enabled. As I showed in the other code, enableControls is set to false when colliding with an enemy. It's enabled back on when the player touches the ground:

```
    private bool CheckIfGrounded () {
              if (true/*rb.velocity.y <= 0*/) {
                     foreach (Transform checker in groundCheckers) {
                            Collider2D[] colliders = Physics2D.OverlapCircleAll
(checker.position, radius, ground);
                            for (int i = 0; i < colliders.Length; i++) {
                                   if (colliders [i].gameObject != gameObject) {
                                          maxJumpCount = 1;
                                          Debug.Log ("Is grounded!");
                                          enableControls = true;
```

As you can see on the last line here when the player hits the ground, the enableControls is set back to false so when they land, the player controls go back to normal. I have also added a check for this variable within the animation if statements:

```
              if (isGrounded && !(left || right || Input.GetKey(KeyCode.A) ||
Input.GetKey(KeyCode.D)) && notDead && enableControls) {
                     playerAnimator.Play ("Player Idle"); //Plays the animations
              }
```

This check for if the player is not dead and if the player controls are enabled is to potentially remove any bugs, e.g. the "Player Bonk" animation stopping before the player lands.

During testing everything was successful, however, I have decided to also add a horizontal force that pushes you away from the enemy so that you don't stomp them as the punishment jump occurs:

```
                //player.sprite = bonk;
                playerAnimator.Play("Player Bonk");
                if (direction == "left") {
                    Debug.Log("Left");
                    rb.velocity = new Vector2(forceMultiplier, jumpMultiplier);
                } else if (direction == "right") {
                    Debug.Log("Right");
                    rb.velocity = new Vector2(-forceMultiplier, jumpMultiplier);
                }
```

This code above sends the player flying in the opposite direction they're facing to; I believe that this is sufficient enough to stop the player from carelessly colliding with enemies as it could make them collide with things such as spikes and they wouldn't be able to stop that from happening.

Upon testing, I found a bug where if the player stands still and collides with the enemy, they aren't sent flying in the horizontal direction but only up. To do this, I needed to change the order of the code in the if statement that disables the controls (it needs to be moved so that the controls are disabled before the direction check is done and the player is launched). This is because I added a check for this variable within the if statement that stops the player from sliding on the floor as that was blocking the player from being launched:

```
        if (isGrounded && !(Input.GetKeyDown(KeyCode.A) && Input.GetKeyDown(KeyCode.D) &&
Input.GetKeyDown(KeyCode.Space)) && enableControls) {
            //Debug.Log("Forcing player to stop");
            rb.velocity = new Vector2(0f, rb.velocity.y);
        }
```

Once those two things were done, the player was launched in the correct directions.

The final thing to do with the player was to create code that handles collisions with spikes:

```
            case "Spikes":
            if (notDead) {
                rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
                healthBar.value -= damage;
                if (healthBar.value <= 0) {
                    notDead = false;
                    Debug.Log("Player is dead");
                }
            }
            break;
```

The above code is in the OnTriggerEnter2D procedure and the “Spikes” case it the tag of the trigger the player enters. If the player’s not dead, a vertical velocity is applied to the player and the player’s health is reduced slightly. Upon testing this feature, everything worked fine.

Now, everything in the player script has been completed and it's time to see how successful this stage of the development was:

Test #	Description	Data to Test	Result Expected	Result Observed
10	Valid Does the player score go up as the player moves closer to the finish line?	Score counter	The score goes up at the player moves to the right.	No, see below.
11	Valid Does the player collide with walls?	Player	The player collides with walls and stops moving.	Yes, player collides with walls.
12	Valid Does the score increase as the player collides with coins?	Score, Coins	The coins disappear as they collide with the player and the score goes up.	Yes, score increases.
13	Valid Does the player fail if they fall out of the world?	Player	The player fails, and they must restart the level.	No, see below.
14	Valid Does the player complete the level when they cross the finish line?	Player, Finish Line	The player finishes the level.	Yes, however, the score will be saved at a later stage.
15	Valid Is the player able to shoot projectiles?	Player	Yes, the player shoots projectiles.	No, the player jumps on enemies instead.
16	Valid Do the projectiles stop when they hit an obstacle?	Player, obstacle	The projectile stops when it touches an obstacle.	No, see above.

Test #	Evidence
10	N/A (Feature removed)
11	https://drive.google.com/open?id=1uI5Z3krWck8XR7H6iWC0l6F5tHyi3CAU
12	https://drive.google.com/open?id=1tgKFLq10sdnOiffonqm0IjzgYmk-bTn4
13	N/A (Feature removed)
14	https://drive.google.com/open?id=1yH_cS9Vdarx1mzem-6pGF5RoBrUDSQfr
15	N/A, instead I'll provide evidence for being able to stomp enemies: https://drive.google.com/open?id=13aHHnWXjInRaF3bVteNc4xTAy76wtJm0
16	N/A (Feature removed)

Considering the above table, I would say that this stage has not been very successful. This is because I had to switch platforms from Android to PC due to the limitations of Unity; I could not figure out a way of making the UI buttons controls the player movements, it almost seemed as if it was a bug within the engine itself as all code but the rigidbody scripts were executed. I also changed several fundamentals of how I wanted the game to turn out; for example, I made it so that the score doesn't increase as the player moves closer to the finish line as the finish line could be below the player so they would get no extra score for going towards it, or I'd need to create very complex code which I probably wouldn't be capable of doing. I also decided to not allow the player to fall out of the world at all, I instead decided to have spikes in any holes to hurt the player. This is because, I believe it

would be too frustrating for players as they'd fail immediately. Instead, I'll just have more spikes, so that the player just loses some health instead of making them fail completely which I thought was a bit unreasonable. Lastly, I decided that the player will instead have to stomp enemies, which I believe is more challenging than simply shooting enemies, which I believe to be better as the survey I carried out earlier before the development said that players liked challenges.

I asked a few stakeholders for their opinions on how the player works. Everyone said that the player movements were fine, the only issue they seemed to have with the controls was how far the player moves with a single tap of either of the keys I used for moving from left to right and vice versa: they said the player moved a bit far, so I decreased the speed multiplier within the Unity editor and after that it was fine.

I also asked them about how they feel about this game now being made for Windows: they didn't seem to mind. This may be because this is such a simple game that people don't mind what they play it on, which would make sense, so the change of operating systems might not be such a big deal after all.

Stage 4: The Enemies

This stage will involve creating code for the enemies. I began this by creating simple animations for the enemies with assets I have downloaded from the internet for free. I added an animator component to the enemy and added animations such as walking and dying.

Then I went on to creating code that rotates the enemy based on the direction they're heading in:

```
void Update () {
	if (rb.velocity.x > 0) {
		transform.localRotation = Quaternion.Euler (0f, 0f, 0f);
	} else if (rb.velocity.x < 0) {
		transform.localRotation = Quaternion.Euler (0f, 180f, 0f);
	}
}
```

So, if the player is going to the right, their rotation is reset and if they're going to the left, they are rotated by 180 degrees.

Then to turn the enemy, I have decided to write code that turns the enemy around when on the edge of a block or when hitting a wall. I created the following code:

```
    public bool IsOnEdge () {
        Transform checker = edgeChecker.transform;
        Collider2D[] colliders = Physics2D.OverlapCircleAll(checker.position, radius,
ground);
        for (int i = 0; i < colliders.Length; i++) {
            if (colliders[i].gameObject != gameObject) {
                //Debug.Log("Not on edge");
                return false;
            }
            //Debug.Log("On edge");

        }
        return true;
    }

    public bool IsntTouchingWall () {
        Transform checker = wallChecker.transform;
        Collider2D[] colliders = Physics2D.OverlapCircleAll(checker.position, radius,
ground);
        for (int i = 0; i < colliders.Length; i++) {
            if (colliders[i].gameObject != this.gameObject) {
                //Debug.Log("Not touching wall");
                return false;
            }
            //Debug.Log("Touching wall");

        }
        return true;
    }
```

In IsOnEdge, a new variable checker of type Transform is assigned to the wallChecker gameObject's transform. This is done to extract the transform from the empty gameObject that I will use to detect whether the enemy is on the edge or not. Then at the position of the checker, I create an array of colliders and I loop through them all to see if the enemy is on the edge. This works, because the edge checker is placed right below and slightly in front of the enemy, so when the checker object is colliding with anything, it means that the enemy is not on the edge, so it returns false. However, when the collider doesn't collide with anything, it means that the enemy is right on the edge so true is returned. The same explanation can be used for IsntTouchingWall, except the checker is placed slightly in front of the enemy but not below them. So, when the collider detects a collision, it means that the enemy is touching a wall and so false is returned, and when there are no collisions detected it means that the enemy isn't touching a wall and so true is returned.

These two functions were used to detect if the enemy should be flipped or not, so then I created the following code to use them:

```
    void FixedUpdate() {
        //rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
        if (IsOnEdge() || !IsntTouchingWall()) {
            if (direction == "left") {
                direction = "right";
            }
            else if (direction == "right") {
                direction = "left";
            }
            //rb.velocity = new Vector2(-rb.velocity.x, rb.velocity.y);
        }
```

Within FixedUpdate, I check for if the enemy is on an edge or if they're touching a wall. Either one of these conditions is true, the string variable "direction" changes the direction to the opposite to the current direction, for example, if the current direction is left, it will check if it's left and change it to right, and vice versa. By default, I have set the direction string to "left". There is no reason for that, it just needed to be assigned before any checks are done as there needs to be a direction in the first place for the next code I write to work:

```
        if (direction == "left") {
            //rb.velocity.Set(-forceMultiplier, rb.velocity.y);
            rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
        } else if (direction == "right") {
            //rb.velocity.Set(-forceMultiplier, rb.velocity.y);
            rb.velocity = new Vector2(forceMultiplier, rb.velocity.y);
        }
```

This is still in the FixedUpdate procedure, right below the code before this. It checks the direction of the enemy and assigns a velocity accordingly.

Earlier in the player stage, I have created lines of code that send a message to the enemy they're colliding with, called EnemyDeath and Score. I decided that EnemyDeath should be within the enemy script so that I don't have to mess around with detecting which object should be removed and to increase stability in general. Same for the Score procedure, it needed to be within the enemy script otherwise the player would get too much score when stomping enemies.

I have already created the code for the EnemyDeath procedure in the player controller stage so all I needed to write was the Score procedure:

```
    public void Score () {
        if (!isDying) {
            GameObject.Find("Player(Clone)").GetComponent<playerController>().score +=
20;

GameObject.Find("Player(Clone)").GetComponent<playerController>().UpdateScore();
        }
        isDying = true;
    }
```

This code first checks to see if the enemy is dying or not. If they're not dying, the script will access the score variable from the player and increment it by 20 (this score will be greater once I create difficulties). This also calls the UpdateScore procedure. Then, isDying is set to true so that the player doesn't get to stomp the enemy multiple times while they're dying and get extra score.

I then wanted to add sound effects to the enemy, so I created the following code:

```
    public IEnumerator PlayWalkSound () {
        audioSource.PlayOneShot(walk);
        yield return new WaitForSeconds(walk.length + 1f);
        readyToPlay = true;
        yield return true;
    }
```

This is essentially the same coroutine I used with the player so there's nothing to explain.

When this was done, the enemy stage has been completed.

Test #	Description	Data to Test	Result Expected	Result Observed
17	Valid Does the enemy collide with walls and the floor?	Enemy, floor, obstacles	The enemy collides with all floors and obstacles.	The enemy does collide with walls.
18	Invalid Does the enemy get destroyed when they collide with obstacles harmful to the player?	Harmful obstacles, enemy	The enemy does not get destroyed by the obstacles, nothing happens to the enemy.	No, enemy stays alive.
19	Valid Does the player lose when they collide with the enemy?	Enemy, player	The player fails the level and must start again.	Instead, the player's score is decremented.
20	Valid Is the enemy able to go move and jump over obstacles?	Enemy, obstacles	The enemy moves towards the player and avoids obstacles.	See below.
21	Invalid Does the enemy pick up collectables that only the player should?	Enemy, collectables	The enemy just goes through the collectables.	The enemy doesn't collect anything.
22	Valid Is the enemy destroyed when a projectile from the player is fired at them?	Player, enemy	The enemy is destroyed upon contact with the projectile.	See below.

Test #	Evidence
17	https://drive.google.com/open?id=1L3f20e5bhuYo8Y5uAe2YmGMbrQgD0eOZ
18	https://drive.google.com/open?id=1-Cogn9dNslCCvE4GXtYBpDRyomDoKsKy
19	Video of player being hurt by enemy: https://drive.google.com/open?id=1QCaDUL63uAdpN-uJ_Nz61lPdSLzbM7vG Video of player dying because of enemy: https://drive.google.com/open?id=1FvJmHDb84-42DDwr1CjOBanObnUTukdS
20	Can be seen in evidence 17, enemy jumping feature wasn't implemented.
21	Can be seen in evidence 17.
22	Replaced with stomping: https://drive.google.com/open?id=10e7iVZueWvpeeQlAEg7zTRsKzKhtGyIa

I would say that this stage has been very successful if not for the fact that it was affected by the previous stage, as in the player stage I decided to remove the feature of shooting enemies. However, in this stage, I have also decided to not allow the enemy to go towards the player when they are nearby, as I thought that would make the game simply too difficult, so I skipped this feature.

Apart from this, I have met all the other success criteria: the enemy collides with walls properly, they're immune to any obstacles that would hurt the player, the player loses if they collide with the enemy too much and they do not pick up any items the player would; instead, they just walk straight through the items.

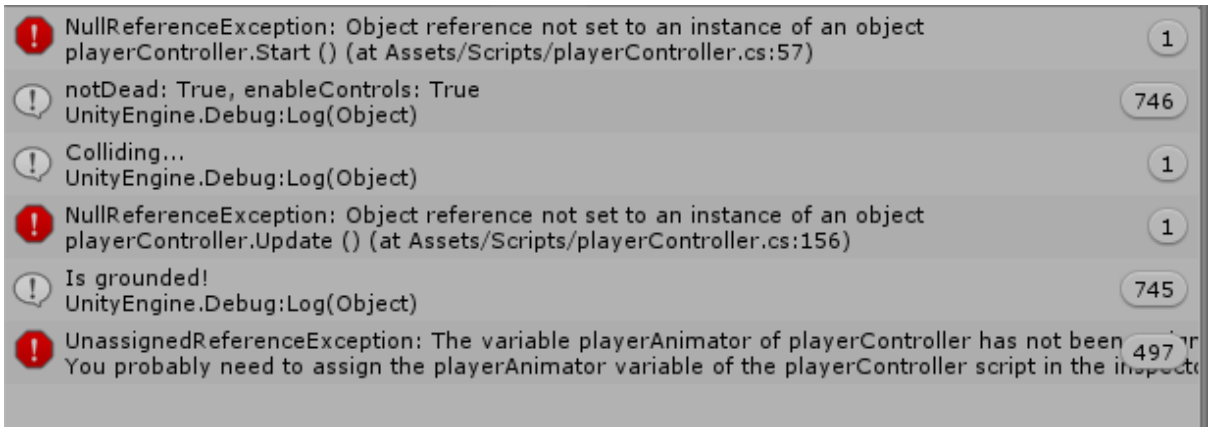
I asked a few stakeholders about how they feel about the enemy movements. There were no complains or features they'd like to see added apart from maybe more enemies with different behaviours. However, I don't think this will be possible due to time limitations so I will skip this for now. If I did have more time, I would add more enemies though.

Stage 5: Leaderboard

In this stage, I will create code that handles the leaderboard by storing, sorting and displaying highs-scores. Firstly, I have created two new text objects, one that displays a message saying, “Level Completed” and one saying, “Level Failed”. These will display when the player finishes the level or dies, respectively. I have set these objects to disabled mode and they will only be enabled at the right time. So, I went back to the player controller script:

```
            healthBar.value -= damage;
            if (healthBar.value <= 0) {
                failed.enabled = true;
                PlayerPrefs.SetInt("CurrentScore", score);
                PlayerPrefs.SetInt("CurrentLevel", 1);
                notDead = false;
                Debug.Log("Player is dead");
            }
        }
          }
    if (other.gameObject.name == "Finish Block") {
        completed.enabled = true;
        PlayerPrefs.SetInt("CurrentScore", score);
        PlayerPrefs.SetInt("CurrentLevel", 1);
        notDead = false;
        Debug.Log("Player Won");
    }
```

I have simply altered this code by adding variables that are stored when the game is closed. This will allow me to set high scores. I have decided to save the current level and current score as those are the two things that are visible on the leaderboard. I have also added code that enables the text objects accordingly. However, within unity I started getting several errors:



After debugging I found out that this is due to the two text objects being initially disabled. To avoid this issue, I have decided to keep the two objects constantly active but make them initially have no text attached to them. Then, once the player completes a level or fails a level, I will assign the text:

```
failed.text = "Level Failed!";
```

```
completed.text = "Level Completed!";
```

This fixed the issue. Then, I wanted to change what the exit button does when the player dies or finishes the level, so I altered the menuButtonPress script as follows:

```
            case "Exit_level":
            soundController.PlaySound(true);
            if
(GameObject.FindGameObjectWithTag("Player").GetComponent<playerController>().notDead == false)
{
                SceneManager.LoadScene("Leader Board");
            } else {
                SceneManager.LoadScene("Level Select");
            }
                    break;
               }
```

What this does it make the player go to the leaderboard if they were to fail or complete a level since I re-used notDead for when the player finishes the level and when they die, as notDead is used for the same things I would have used a levelFinished variable for (disabling controls etc).

```
    void Start() {
        playerScore = PlayerPrefs.GetInt("CurrentScore", 0);
        playerLevel = PlayerPrefs.GetInt("CurrentLevel", 0);

        scores[0] = PlayerPrefs.GetInt("s0", 0);
         …
        scores[8] = PlayerPrefs.GetInt("s8", 0);

        levels[0] = PlayerPrefs.GetInt("l0", 0);
         …
        levels[8] = PlayerPrefs.GetInt("l8", 0);

        if (playerScore > scores[8]) {
            scores[8] = playerScore;
            levels[8] = playerLevel;
        }

        for (int i = 8; i > 0; i--) {
            if (scores[i] > scores[i - 1]) {
                tempScore = scores[i - 1];
                tempLevel = levels[i - 1];
                scores[i - 1] = scores[i];
                levels[i - 1] = levels[i];
                scores[i] = tempScore;
                levels[i] = tempLevel;
            }
        }
        for (int i = 0; i < 9; i++) {
            Debug.Log("Setting score: " + i);
            scoreTexts[i].text = scores[i].ToString();
            levelTexts[i].text = levels[i].ToString();
        }

        PlayerPrefs.SetInt("s0", scores[0]);
         …
        PlayerPrefs.SetInt("s8", scores[8]);

        PlayerPrefs.SetInt("l0", levels[0]);
         …
        PlayerPrefs.SetInt("l8", levels[8]);

        PlayerPrefs.SetInt("CurrentScore", 0);
        PlayerPrefs.SetInt("CurrentLevel", 0);
    }
```

The first two lines of the code are used to retrieve the CurrentScore and CurrentLevel variables stored in the game's files and assigns them to a variable. Then, the top 8 scores are loaded along with the levels that go along with them (the … represents a skip, there are variables going from 0 to 8 there and I decided to cut them out because it would take up too much space). Then I wrote an if statement that checks if the new score is big enough to be placed on the leaderboard: I check this by seeing if the latest score is at least bigger than the lowest score. If it is, it replaces the lowest score

(since it won't appear on the leaderboard again, ever) and then a bubble sort puts the new score in the right location. Then I have a for loop that applies the text to the correct UI elements (scores[] and levels[] are arrays of type int and scoreTexts[] and levelTexs[] are arrays of type Text). Then, the new high scores are stored locally in the right variables (the first score is stored in s0, the last score is stored in s8 etc.) And the CurrentScore and CurrentLevel are reset to zero so that the scores aren't duplicated on the leaderboard every time a player presses the leaderboard button.

Upon testing, everything was working correctly so it was then time to evaluate the success of this stage:

Test #	Description	Data to Test	Result Expected	Result Observed
23	Valid Does the leaderboard display the high scores?	Leaderboard	The leaderboard displays the highest scores	As expected.
24	Valid Does the leaderboard display the scores in descending order?	Leaderboard	The scores go down as you go down in the leaderboard	As expected.
25	Valid Is the correct level displayed along with the score?	Leaderboard	The leaderboard groups data correctly	As expected.

23	
24	Can be seen in evidence 23.
25	Can be seen in evidence 23.

I believe that this stage has been rather successful. This is because, the leaderboard works exactly as I planned in my success criteria, so it does the following: It displays the high scores, in the correct order (they go in descending order, so from highest score to lowest), and along with the high scores the level the score was obtained in is also displayed to the right side of the score.

The few stakeholders I asked about this screen have said that it looks good and they had no complaints about it, so I went straight on to the next stage.

Stage 6: Options and Settings

This will be the final stage of development. In this stage, I will write code that handles setting the game's difficulty as well as volume.

I first created the following code:

```
public Button button;
public Slider volume;

   // Use this for initialization
   void Start () {
    Button btn = button.GetComponent<Button>();
    btn.onClick.AddListener(TaskOnClick);
}

   public void TaskOnClick () {
          switch (gameObject.name) {
        case "Easy":
        PlayerPrefs.SetInt("Damage", 1);
        PlayerPrefs.SetFloat("ScoreMultiplier", 0.8f);
        break;

        case "Normal":
        PlayerPrefs.SetInt("Damage", 2);
        PlayerPrefs.SetFloat("ScoreMultiplier", 1f);
        break;

        case "Hard":
        PlayerPrefs.SetInt("Damage", 3);
        PlayerPrefs.SetFloat("ScoreMultiplier", 1.2f);
        break;
    }
   }

void Update () {
    PlayerPrefs.SetFloat("Volume", volume.value);
}
```

This is essentially identical to the code I used to check what menu buttons are being pressed in menuButtonPress.cs. When a button with this script assigned is pressed, a check is carried out to see what button is pressed. The damage multiplier variable is stored locally according to the difficulty, same with the score multiplier. The higher the difficulty setting, the more damage the player will take and the more score they will get. Then I did the following in the player controller script:

```
public int damage = PlayerPrefs.GetInt("Damage", 1);
public float scoreMultiplier = PlayerPrefs.GetFloat("ScoreMultiplier", 1f);
public float volumeMultiplier = PlayerPrefs.GetFloat("Volume", 1f);
```

The damage, score multiplier and volume multiplier are stored as a variable at the start when the player is instantiated. The same is done within the enemy controller script.

After that, I edited the static scores given to the player when collecting items or killing enemies:

```
            case "Key":
            Debug.Log("Key");
            //previousScore = score;
            score += Mathf.RoundToInt(50 * scoreMultiplier);
            UpdateScore();
            //StartCoroutine(UpdateScore());
            Destroy(collision.gameObject);
            break;
```

Instead of simply giving the player a score of 50 for collecting a key, for example, they are given 50 multiplied by the score multiplier. So, on the easy difficulty, they'd instead get 40 points and on hard they'd get 60, and so on for the other items and for killing enemies. Then with all volume-related scripts, I did the following:

```
audioSource.PlayOneShot(walk, randomVol * volumeMultiplier);
```

I have given every PlayOneShot procedure used either a second parameter (volumeMultiplier) or I multiplied the already existing parameter by the volume multiplier. Doing so adjusts the volume according to the setting on the slider in the settings menu.

Fortunately, I only found one bug: whenever you go back into settings, the volume is reset. To fix this I simply added the following line in the settings controller's start procedure:

```
volume.value = PlayerPrefs.GetFloat("Volume", 1f);
```

This makes sure that whenever you go into settings, the volume is set to the stored value.

Test #	Description	Data to Test	Result Expected	Result Observed
26	Valid Does the volume change as the slider is moved?	Volume slider	The sound effect volume changes accordingly.	As expected.
27	Valid Does the difficulty of the game change when the difficulty setting is changed?	Difficulty buttons, enemies	The enemies are a lot more aggressive when the Hard button is pressed and a lot less aggressive when the Easy button is pressed.	As expected.
28	Valid Does the coin multiplier change, depending on the difficulty?	Difficulty buttons, score	On the default difficulty, you get a regular amount of coins (multiplier of 1x), on the Easy difficulty you get fewer coins (0.75x) and on the Hard difficulty, you get more coins (1.5x).	As expected.

Test #	Evidence
26	I cannot provide evidence for this as files with audio would be too large.
27	I cannot provide evidence for this as I am limited by the recording length of my software.
28	I cannot provide evidence for this as I am limited by the recording length of my software.

During this stage I came across very few bugs and everything is working as I originally planned it, I didn't have to change any features. I also met all of the success criteria: The volume changes accorting to the position of the slider and the difficulty changes when the buttons for the different difficulties are pressed.

I asked my stakeholders about how they feel about the multipliers for the score and enemy damage. All of the few people I asked said the score inscrease/decrease scales well with the difficulty increase from the enemies dealing more damage. This is further evidence for this stage being successful.

End-Stage Testing

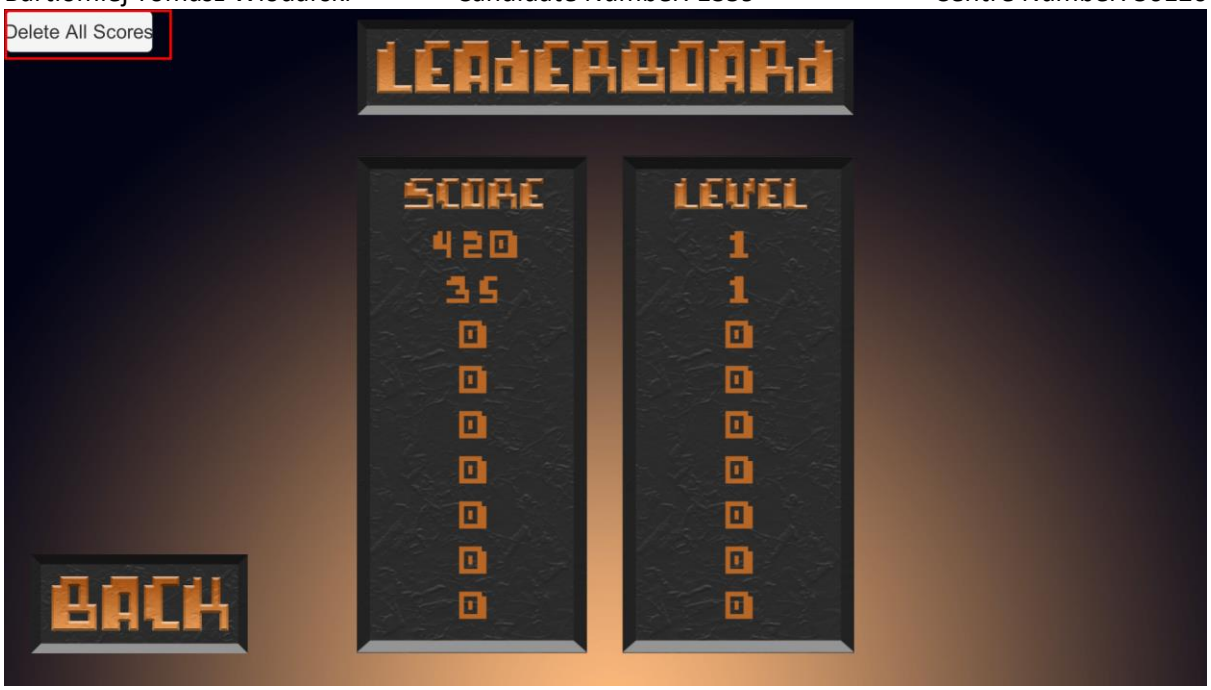
In this stage, I will be testing the game again to find any bugs I have not found yet. I will also be allowing some stakeholders to play the game for a short time and I will ask them for feedback.

Final Testing

Upon inspection, I immediately noticed that the game still had the “Development build” watermark in the bottom-right corner:

I fixed this by re-compiling the game without the “Development Build” setting. After I did that, the text in the image above has disappeared:

Then upon further investigation I noticed that I left the “Delete All Scores” button that I have left in the game during development for debugging purposes:



I do not want this to be in the final release so I decided to delete this button within the editor:

Then, I realised that I left the “Custom Level” button on the level select screen. Since I never figured out how to get this feature working, I will need to remove this button. Also, I have not added the preview images for the levels, the buttons are just empty:

I added the appropriate level images on top of the level buttons and I have also removed the custom level button:

Stakeholder testing

After all this was fixed, I needed to do stakeholder testing, as other people may be able to find any bugs in the game that I have not been able to find and I want a general opinion on the game from the target audience that I was creating the game for. I have come up with the following questions that I will ask the stakeholders after allowing them to play the game for a short period of time. Each bullet point under a question is a separate person answering the question:

- **User Interface:**
    - Does the user interface fill the screen well?
        - Yes
        - Yes
        - Yes
        - Yes

Obviously this is a personal opinion of the stakeholders, however since they all agree that the interface fits the screen well, then I think that most if not all of the stakeholders will like this scale. This means that I have met the success criteria for this question.

    - Does the interface scale well with different resolutions?
        - Yes
        - Yes
        - Yes
        - Yes

Again, this was liked by everyone, there were no complaints about the scale of the UI which means that this success criteria has been met.

    - Is the text easy to read?
        - Font is weird but it's okay to read – didn't inhibit ability to use UI.
        - Yes, it's clear
        - Yes I can read it fine
        - Yes it's good

There were some mixed opinions on this, even in the testing after the user interface has been completed. If I conducted the test on a larger scale and there were many complaints about the font, I would change it but since I asked such a small group, I will assume that's just their personal preferences being different from everyone else so I believe that this success criteria has been met.

    - Is it easy to understand what each button does?
        - It's clear except for keeping track of which difficulty you're on.
        - Yes
        - Yes
        - Yes

The first stakeholder in this stage pointed out something very important that I should have considered: you can't tell what difficulty you're playing on. Technically since the buttons do change the difficulty, I do meet the success criteria, however this would reduce the user experience as they can't tell which difficulty they will be playing on. This makes the game less usable.

- Does the colour scheme contrast well?
  - Yes
  - Yes
  - Yes
  - Yes

Since there were no complaints about this, I would say I did a good job choosing the colours for the game menus. Since the colours contrast well, it means that the game is very usable.

- Are the menus easy to navigate?
  - Yes
  - Yes
  - Yes
  - Yes

Since there were no complaints about this stage, this means that the menus are simple and therefore the game is very usable.

- **Levels:**
  - Do the levels have enough collectible items?
    - Yes they do
    - Yes
    - Yes
    - Yes

The stakeholders agree that they have enough collectible items, therefore I believe that the success criteria of there being collectible items has been met.

- Are the levels easy to navigate?
  - Yes
  - No – kept dying early on so it was difficult to keep track of where the player needs to go.
  - Partially, it would be useful to have a minimap though
  - Yes

The answers to this question pointed out to me that to some people the levels were either too difficult to ever find the finish line and some were completely lost, so this stage wasn't successful, as I did not meet the cirteria of the levels being easy to navigate.

- Are the obstacles varied throughout?
  - Yes
  - Yes
  - Yes
  - Yes

The stakeholders agreed with this question, therefore I met the success criteria of there being many obstacles in levels.

- Is it easy to identify the finish line?
  - No but it's easy to figure out that it is a finish line.
  - No because didn't reach it.
  - I didn't reach it
  - No I did not reach it

Just like with the question about navigation, I have realised that I may have made the levels too difficult to navigate, as nobody ever reached the finish line. This means that the success criteria for this stage have not been met in the end.

- Originally I was going to have endless holes in each level where you die if you fall in them. Do you feel that the lack of them impacts the game negatively or not?
  - It would be frustrating to immediately die however it would add more difficulty for advanced players so maybe have then on later levels.
  - It would impact it negatively as it's hard enough as it is.
  - I don't think it needs them as the game is hard enough
  - No it doesn't

The stakeholder did not see a problem with there being no holes in the level, which is a good thing as this will make the game more enjoyable for them.

- **Player:**
  - Does the player collide with all walls properly?
    - Yes
    - Yes
    - Yes
    - Yes

This means that I have met the success criteria of the player detecting collisions correctly.

- Originally, I was going to have the player shoot the enemies but I replaced it with stomping them. How do you think this impacts the gameplay? Was this a good idea or not?
  - Stomping them is better because it is more challening.
  - It's more difficult so ultimately makes the game more fun.
  - It's good because shooting them would look weird
  - It's good as it's more challenging and fun.
- **Enemies:**
  - Do the enemies hurt the player upon contact?
    - Yes
    - Yes but it feels like it hurts too much.
    - Yes
    - Yes

This question, like some others, made me realise that there's a possibility that I made the game too difficult, even on lower difficulties.

- Originally I planned on the enemies being able to jump over single blocks. Do you think it was a good idea to not let them jump or fall down blocks?
  - It's good having them confined to a closed space but maybe have them roam more freely on higher difficulties.
  - The harder the better so it's not a good idea to remove the feature
  - It was good because it is a good level of challenge. Any more would make it too challening.
  - Yes as otherwise the game would be too difficult.

The stakeholders answered this question with positive replies, therefore I believe that most stakeholders would agree that removing this feature was a good idea.

- **Leaderboard (No questions needed to be asked)**
- **Settings:**
  - Are the settings self-explanatory?
    - Yes
    - Yes
    - Yes
    - Yes

The responses to this brought further evidence that this game is rather usable, as everything made sense to the stakeholders.

- **General:**
  - How was your overall experience with the game?
    - It was good
    - It was very challening
    - It was fun
    - It was a lot of fun despite being challenging

This is probably the most important question of all, as it asks about the overall experience, and since the overall experience was good, this means that the whole point of creating this game has been accomplished.

  - Is there anything you'd change?
    - Allow the player to fall straight down if you don't press directional keys, more hidden areas (e.g. blocks you can walk through)
    - The game feels way too difficult at the normal and high difficulties. I'd lower the amout of damage you take.
    - No
    - No

There were very few things the testers wanted to see changed, which means that the product has been successful, as the game is fun as it is at the moment. However if I were to add more features to this game or ever go back to it, I would take these suggestions into consideration.

Evaluation

After completing the entire game, I can conclude that for the most part, this project has not been very successful. This is because, I have not met the requirements of many major features that were a part of my success criteria. However, there were also features that turned out very well and some features that I changed that the stakeholders still liked.

Missing feature: Timer

As I stated on page 2, I was originally going to have a timer to add an extra level of challenge. In the end, I didn't include this, as I believe that this feature would have made the game even more difficult than it already is. Also, the levels are massive, as seen on page 30, which means that it's harder for the players to find the exit. In fact, everyone that tested the game failed to find the finish line as seen on page 59. Therefore, I believe it was a good idea to leave out the timer; the players would have never been able to find the finish line within the time bounds.

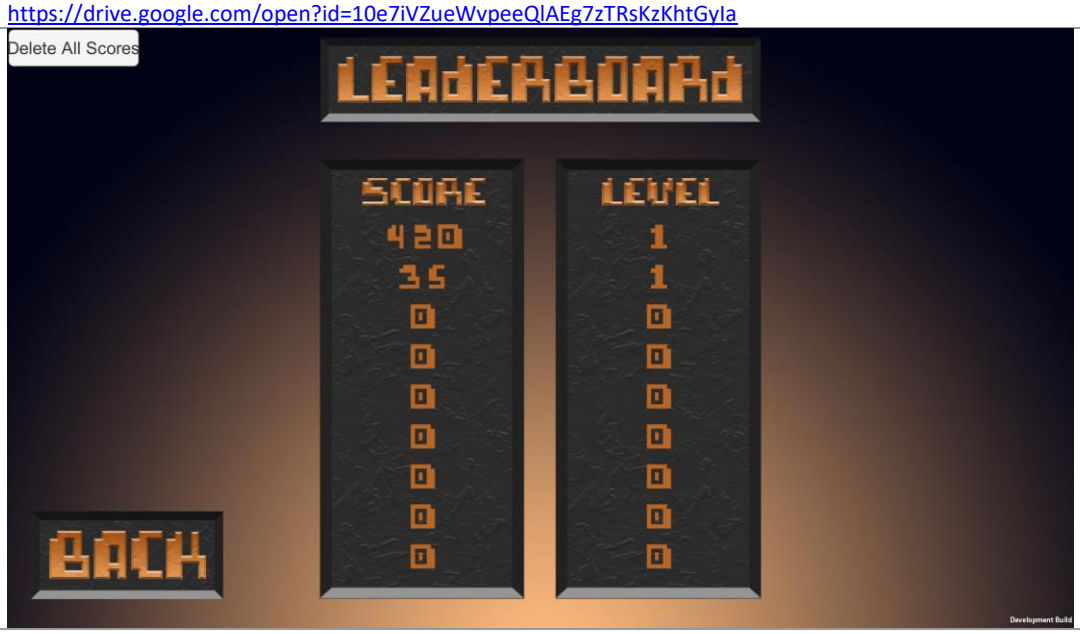
If I were to further develop this game, I could add a timer, if I reduce the level size and if I create the levels easier to navigate in any way, for example, by adding a mini-map, as one of the testers suggested on page 58.

Success Criteria

Test #	Description	Data to Test	Result Expected	Result Observed
1	Valid Does the app open when the icon is pressed?	Icon	The app opens when the icon is clicked.	Yes, the app opens properly.
2	Valid Do the buttons in-game do appropriate things?	Buttons	The buttons take you to correct screens.	Yes, all the buttons do what they should.
3	Valid Does the exit button close the game?	Exit button	The exit button closes the game.	Yes.
4	Valid Do the buttons scale with the screen resolution?	Buttons	The buttons increase in size as the resolution goes up and vice versa.	Yes, the entire UI scales with the screen resolution.
5	Valid Does each level contain collectable items?	Levels	The level contains collectable items such as coins.	Yes.
6	Valid Are there obstacles in each level?	Levels	There are obstacles such as boxes you'd have to jump over.	Yes, there are walls and drops for the player to overcome.
7	Valid Is there a finish line in each level?	Levels	There is a finish line at the end of each level.	Yes, although it can't be seen on the images due to the colour I used.
8	Valid Is there a clear path to complete each level?	Levels	You can easily see a path you must take to carry on in each level.	Yes, all levels are completable.
9	Valid Do menu buttons take you to the right level?	Buttons	The buttons take you to the correct level.	Yes.
10	Valid Does the player score go up as the player moves closer to the finish line?	Score counter	The score goes up at the player moves to the right.	No, feature was removed.
11	Valid Does the player collide with walls?	Player	The player collides with walls and stops moving.	Yes, player collides with walls.

12	Valid Does the score increase as the player collides with coins?	Score, Coins	The coins disappear as they collide with the player and the score goes up.	Yes, score increases.
13	Valid Does the player fail if they fall out of the world?	Player	The player fails, and they must restart the level.	No, feature was removed.
14	Valid Does the player complete the level when they cross the finish line?	Player, Finish Line	The player finishes the level.	Yes.
15	Valid Is the player able to shoot projectiles?	Player	Yes, the player shoots projectiles.	No, the player jumps on enemies instead.
16	Valid Do the projectiles stop when they hit an obstacle?	Player, obstacle	The projectile stops when it touches an obstacle.	No, projectiles were removed.
17	Valid Does the enemy collide with walls and the floor?	Enemy, floor, obstacles	The enemy collides with all floors and obstacles.	The enemy does collide with walls.
18	Invalid Does the enemy get destroyed when they collide with obstacles harmful to the player?	Harmful obstacles, enemy	The enemy does not get destroyed by the obstacles, nothing happens to the enemy.	No, enemy stays alive.
19	Valid Does the player lose when they collide with the enemy?	Enemy, player	The player fails the level and must start again.	Instead, the player's score is decremented.
20	Valid Is the enemy able to go move and jump over obstacles?	Enemy, obstacles	The enemy moves towards the player and avoids obstacles.	Feature removed to make game easier (would be too difficult).
21	Invalid Does the enemy pick up collectables that only the player should?	Enemy, collectables	The enemy just goes through the collectables.	The enemy doesn't collect anything.
22	Valid Is the enemy destroyed when a projectile from the player is fired at them?	Player, enemy	The enemy is destroyed upon contact with the projectile.	Projectiles were not added in the end.
23	Valid Does the leaderboard display the high scores?	Leaderboard	The leaderboard displays the highest scores	As expected.
24	Valid Does the leaderboard display the scores in descending order?	Leaderboard	The scores go down as you go down in the leaderboard	As expected.
25	Valid Is the correct level displayed along with the score?	Leaderboard	The leaderboard groups data correctly	As expected.
26	Valid Does the volume change as the slider is moved?	Volume slider	The sound effect volume changes accordingly.	As expected.
27	Valid Does the difficulty of the game change when the difficulty setting is changed?	Difficulty buttons, enemies	The enemies are a lot more aggressive when the Hard button is pressed and a lot less aggressive when the Easy button is pressed.	As expected.
28	Valid Does the coin multiplier change, depending on the difficulty?	Difficulty buttons, score	On the default difficulty, you get a regular amount of coins (multiplier of 1x), on the Easy difficulty you get fewer coins (0.75x) and on the Hard difficulty, you get more coins (1.5x).	As expected.

Evidence

Test #	Evidence
1	https://drive.google.com/open?id=1e_WplJhJ2gVSymMo551OqW5TRat18vxS
2	https://drive.google.com/open?id=1jmgU-FuPh16uHCJspHzThUQ8Vk3G7G0u
3	Can be seen in evidence 2.
4	https://drive.google.com/open?id=1aj1x4f0C7LWid07zNOo7qxulTvf_u3Q3
5	Can be seen in the level images I have created. Yellow pixels stand for coins, light blue stand for keys (items worth more than coins).
6	Can be seen in the level images I've created. Black pixels are walls and orange pixels are spikes (also obstacles, but they harm the player).
7	Finish lines are represented as white pixels on time level images, however since I can't display transparency in a document, it looks like the background.
8	You can see that there is a clear path to the finish line on the level images.
9	Level 1: https://drive.google.com/open?id=1yAEbPrGU8cg0b0eEHtgvGLAni_jNrD3_ Level 2: https://drive.google.com/open?id=1bviMAlAcNfsRlypMcyzKX4XGH-8Ks0tb Level 3: https://drive.google.com/open?id=1hLBeaQoK5904QMpmDcJXU0StCa6haBs8
10	N/A (Feature removed)
11	https://drive.google.com/open?id=1uI5Z3krWck8XR7H6iWC0l6F5tHyi3CAU
12	https://drive.google.com/open?id=1tgKFLq10sdnOiffonqm0IjzgYmk-bTn4
13	N/A (Feature removed)
14	https://drive.google.com/open?id=1yH_cS9Vdarx1mzem-6pGF5RoBrUDSQfr
15	N/A, instead I'll provide evidence for being able to stomp enemies: https://drive.google.com/open?id=13aHHnWXjInRaF3bVteNc4xTAy76wtJm0
16	N/A (Feature removed)
17	https://drive.google.com/open?id=1L3f20e5bhuYo8Y5uAe2YmGMbrQgD0eOZ
18	https://drive.google.com/open?id=1-Cogn9dNslCCvE4GXtYBpDRyomDoKsKy
19	Video of player being hurt by enemy: https://drive.google.com/open?id=1QCaDUL63uAdpN-uJ_Nz61lPdSLzbM7vG Video of player dying because of enemy: https://drive.google.com/open?id=1FvJmHDb84-42DDwr1CjOBanObnUTukdS
20	Can be seen in evidence 17, enemy jumping feature wasn't implemented.
21	Can be seen in evidence 17.
22	Replaced with stomping: https://drive.google.com/open?id=10e7iVZueWvpeeQlAEg7zTRsKzKhtGyIa
23	
24	Can be seen in evidence 23.
25	Can be seen in evidence 23.
26	I cannot provide evidence for this as files with audio would be too large.
27	I cannot provide evidence for this as I am limited by the recording length of my software.
28	I cannot provide evidence for this as I am limited by the recording length of my software.

Success Criteria Evaluation

Looking at the results of all the tests, most of the tests have been successful. The evidence for the successful tests shows that the criteria for most tests have been met (For example, the videos linked show exactly what happens and therefor reinforces the individual criteria being successful).

Unmet and Partially Met Success Criteria

There are many things that have gone wrong in this project. The main issue with this whole project was that I did not think much about the features that I wanted to have in the game. I simply thought of some ideas and decided that I will add them to the game, regardless of if they made sense or not. For example:

10	**Valid** **Does the player score go up as the player moves closer to the finish line?**	**Score counter**	**The score goes up at the player moves to the right.**	**No, feature was removed.**

During the development of the player scripts, I realised that there's absolutely no need to have a score that goes up as you move along, which meant that this success criteria could not be met. This is because, my levels have multiple layers so score based on position would make no sense. However, I could implement this feature if in future development I added an 'infinite mode' where the level is progressively generated and the game only ends when the player dies. Only in a situation like this would a position based score make sense, as the further you'd go, the higher score you'd have.

13	**Valid** **Does the player fail if they fall out of the world?**	**Player**	**The player fails, and they must restart the level.**	**No, feature was removed.**

The next success criteria that was unmet was the feature of falling out of the world, which was simply not implemented as I thought it would make the game too difficult for players. However, if I were to further develop the game, this feature could be implemented in later levels, if I ever created any. Like the stakeholders suggested, I would only add them in later levels to progressively increase the difficulty instead of throwing in new players into the most difficult levels possible.

15	**Valid** **Is the player able to shoot projectiles?**	**Player**	**Yes, the player shoots projectiles.**	**No, the player jumps on enemies instead.**

This success criteria has only been met partially, but only because shooting was replaced with stomping. I believe it would have been too easy to eliminate enemies through shooting and I believe that writing code for shooting would have been too difficult for me, and I may have run out of time if I did try to implement shooting mechanics. However, I believe that if I were to ever further develop this game, I would add the shooting mechanic as a pickup item that would reward the player for getting to a hard to reach area, allowing them to shoot enemies a few times before the item runs out of charge, for example. So, in short, it would only be implemented as a special reward for the most daring players.

19	**Valid** **Does the player lose when they collide with the enemy?**	**Enemy, player**	**The player fails the level and must start again.**	**Instead, the player's score is decremented.**

The above feature was partially met, because I have decided to not allow enemies to instantly kill the player upon impact. This is because, allowing the enemy to do so would make the game much harder, as it's very easy to accidentally come into contact with the enemies as there's so many of them walking around each level. Instead, I only have the enemies slightly damage the player so that the game isn't as hard. This feature could be implemented in a future version of the game in later levels, through enemies that are rarer and more harmful, like a rare boss for example.

20	**Valid** **Is the enemy able to go move and jump over obstacles?**	**Enemy, obstacles**	**The enemy moves towards the player and avoids obstacles.**	**Feature removed to make game easier (would be too difficult).**

This feature was also only partially met, because allowing all of the enemies to track the player's movements and follow them would have made the game very, very difficult since it's already really easy to accidentally collide with the enemies and lose health. Instead, in the future I could have rare, more aggressive enemies that do track the player, but only in small numbers, possibly one or two per level. Any more would make the game too hard, I believe.

All of the other success criteria were fully met. This is because, as the evidence shows, the game behaves as expected in the success criteria tests.

Limitations

There are a few limitations I can think of:

There are only three levels in the game. This is a limitation because, the players can only play the three levels I have created. This is an issue, because players would quickly get bored of the game and would stop playing it. To solve this, in a future version I could implement a level creator or custom level button where the player can upload their own level, like I tried doing during the development but failed to do so. I could also create an online database of user-created levels so that everyone can share their creations.

The game performs really badly on less powerful devices. This is a massive limitation as it essentially limits the number of users to only those with powerful computers, meaning less people will be able to play my game without upgrading their computers, which I doubt would happen as not many people upgrade their computers to play a single game. A solution to this would be to come up with a more efficient way of creating levels in the game, for example, I could merge all the wall/ground blocks that are touching into a single game object to cut down on the memory usage on the stakeholders' devices.

Unmet Usability Features

Fortunately, there is only one partially met usability feature: some stakeholders complained that the font is weird. Luckily that can be easily changed in the future by changing the font within Unity.

Scripts

enemyController.cs

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

public class enemyController : MonoBehaviour {


        private Animator enemyAnimator;
        private Rigidbody2D rb;

        public float forceMultiplier;


    public Transform edgeChecker;
    public Transform wallChecker;

    public float radius;
    public LayerMask ground;

    private string direction = "left";

    private bool isDying = false;

    public AudioSource audioSource;
    public AudioClip walk;
    public AudioClip hurt;
    bool readyToPlay = true;

    public float scoreMultiplier;
    public float volumeMultiplier;

    // Use this for initialization
    void Start () {
        scoreMultiplier = PlayerPrefs.GetFloat("ScoreMultiplier", 1f);
        volumeMultiplier = PlayerPrefs.GetFloat("Volume", 1f);

        rb = GetComponent<Rigidbody2D> ();
                enemyAnimator = GetComponent<Animator> ();
                enemyAnimator.Play ("Enemy Walk");
        rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
        audioSource = GetComponent<AudioSource>();
        //rb.velocity.Set(-forceMultiplier, rb.velocity.y);
        }

        void FixedUpdate() {
        if (readyToPlay) {
            readyToPlay = false;
            StartCoroutine(PlayWalkSound());
        }
        //rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
        if (IsOnEdge() || !IsntTouchingWall()) {
            if (direction == "left") {
                direction = "right";
            }
            else if (direction == "right") {
                direction = "left";
            }
            //rb.velocity = new Vector2(-rb.velocity.x, rb.velocity.y);
        }

        if (direction == "left") {
            //rb.velocity.Set(-forceMultiplier, rb.velocity.y);
            rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
        } else if (direction == "right") {
            //rb.velocity.Set(-forceMultiplier, rb.velocity.y);
            rb.velocity = new Vector2(forceMultiplier, rb.velocity.y);
        }
        }

        // Update is called once per frame
        void Update () {
                if (rb.velocity.x > 0) {
```

```
                        transform.localRotation = Quaternion.Euler (0f, 0f, 0f);
                } else if (rb.velocity.x < 0) {
                        transform.localRotation = Quaternion.Euler (0f, 180f, 0f);
                }
        }

    public bool IsOnEdge () {
        Transform checker = edgeChecker.transform;
        Collider2D[] colliders = Physics2D.OverlapCircleAll(checker.position, radius, ground);
        for (int i = 0; i < colliders.Length; i++) {
            if (colliders[i].gameObject != gameObject) {
                //Debug.Log("Not on edge");
                return false;
            }
            //Debug.Log("On edge");

        }
        return true;
    }

    public bool IsntTouchingWall () {
        Transform checker = wallChecker.transform;
        Collider2D[] colliders = Physics2D.OverlapCircleAll(checker.position, radius, ground);
        for (int i = 0; i < colliders.Length; i++) {
            if (colliders[i].gameObject != this.gameObject) {
                //Debug.Log("Not touching wall");
                return false;
            }
            //Debug.Log("Touching wall");

        }
        return true;
    }

    public IEnumerator EnemyDeath () {
        enemyAnimator.Play("Enemy Die");
        rb.velocity = new Vector2(0f, 0f);
        GetComponent<Collider2D>().enabled = false;
        rb.isKinematic = true;
        audioSource.PlayOneShot(hurt, volumeMultiplier);
        yield return new WaitForSecondsRealtime(0.5f);
        Destroy(gameObject);
        yield return true;
    }


    public void Score () {
        if (!isDying) {
            GameObject.Find("Player(Clone)").GetComponent<playerController>().score +=
Mathf.RoundToInt(20 * scoreMultiplier);
            GameObject.Find("Player(Clone)").GetComponent<playerController>().UpdateScore();
        }
        isDying = true;
    }

    public IEnumerator PlayWalkSound () {
        audioSource.PlayOneShot(walk, volumeMultiplier);
        yield return new WaitForSeconds(walk.length + 0.35f);
        readyToPlay = true;
        yield return true;
    }
}
```

generateLevel.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
/*#if UNITY_EDITOR
using UnityEditor;
using UnityEditor.SceneManagement;
#endif*/

public class generateLevel : MonoBehaviour {

    public Texture2D level;

    [SerializeField]
    public Color[] colors;
    [SerializeField]
    public GameObject[] objects;


    public Color lightPlatform;
        public Color darkPlatform;
        public Color enemy;
        public Color player;
        public Color spikes;
        public Color leftBush;
        public Color midBush;
        public Color rightBush;

        private Color pixel;

        public GameObject lightPlatformPrefab;
        public GameObject darkPlatformPrefab;
        public GameObject enemyPrefab;
        public GameObject playerPrefab;
        public GameObject spikesPrefab;
        public GameObject leftBushPrefab;
        public GameObject midBushPrefab;
        public GameObject rightBrushPrefab;

        private Vector3 position;

        private string filePath;
        private Texture2D download;

        // Use this for initialization
        IEnumerator Start () {

                /*#if UNITY_EDITOR
                if (EditorSceneManager.GetActiveScene().name == "Custom Level") {
                        filePath = EditorUtility.OpenFilePanel ("Select level", "", "");
                        Debug.Log (filePath);
                        WWW www = new WWW ("file:///" + filePath);

                        yield return www;

                        level = www.texture;
                }
                #endif*/

                for (int i = 0; i < level.width; i++) {
                        for (int j = 0; j < level.height; j++) {

                                pixel = level.GetPixel (i, j);
                                position.Set (i, j, 0.0f);

                                if (lightPlatform.Equals (pixel)) {
                                        Instantiate (lightPlatformPrefab, position,
Quaternion.identity);
                                } else if (darkPlatform.Equals (pixel)) {
                                        Instantiate (darkPlatformPrefab, position,
Quaternion.identity);
                                } else if (spikes.Equals (pixel)) {
                                        Instantiate (spikesPrefab, position, Quaternion.identity);
                                } else if (leftBush.Equals (pixel)) {
                                        Instantiate (leftBushPrefab, position, Quaternion.identity);
                                } else if (midBush.Equals (pixel)) {
```

```
                                        Instantiate (midBushPrefab, position, Quaternion.identity);
                                } else if (rightBush.Equals (pixel)) {
                                        Instantiate (rightBrushPrefab, position,
Quaternion.identity);
                                } else if (player.Equals (pixel)) {
                                        Instantiate (playerPrefab, position, Quaternion.identity);
                                } else if (enemy.Equals (pixel)) {
                                        Instantiate (enemyPrefab, position, Quaternion.identity);
                                }
                        }
                }
                yield return 0;
        }
}
```

goToPlayer.cs

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

public class goToPlayer : MonoBehaviour {

        public Transform playerTransform;
        public float distance;

        void Start () {
        }

        // Update is called once per frame
        void Update () {
                playerTransform = GameObject.FindWithTag ("Player").transform;
                transform.position = new Vector3 (playerTransform.position.x,
playerTransform.position.y, -distance);
        }
}
```

menuButtonPress.cs

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

public class menuButtonPress : MonoBehaviour {

	public SoundController soundController;

	public Button button;
	public GameObject question; //Any unassigned variables are
	public GameObject yes;      //assigned in the editor.
	public GameObject no;

	//private bool checkIfDestroy = false;

	// Use this for initialization
	void Start () {
		Button btn = button.GetComponent<Button> ();
		btn.onClick.AddListener (TaskOnClick);
		//The above line sets a listener on the button attatched and
		//directs it to the TaskOnClick subroutine.
	}

	//When the listener detects a button press, this subroutine is called.
	public void TaskOnClick () {
		//This subroutine checks what button is pressed based on the name
		//and does the appropriate things like loading the right scene.
		switch (gameObject.name) {
		case "Start":
			soundController.PlaySound (true);
			//If the parameter is true, the Sound Controller object is destroyed
			//after the sound is played. This is because, Unity creates a new Sound
			//Controller object every time you load a new scene. Therefore, this
			//makes sure the memory is not clogged up by the objects over time.
			//If it's set to false, then it's not destroyed.
			SceneManager.LoadScene ("Level Select");
			break;
		case "Leaderboard":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Leader Board");
			break;
		case "Settings":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Settings");
			break;
		case "Exit":
			soundController.PlaySound (false);
			Quit ();
			break;
		case "Yes":
			soundController.PlaySound (false);
			Application.Quit ();
			break;
		case "No":
			soundController.PlaySound (false);
			CancelQuit ();
			break;
		case "Select_Back":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Main Screen");
			break;
		case "Leaderboard_Back":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Main Screen");
			break;
		case "Settings_Back":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Main Screen");
			break;
		case "Button 1":
			soundController.PlaySound (true);
			SceneManager.LoadScene ("Level 1");
			break;
```

```
                case "Button 2":
                        soundController.PlaySound (true);
                        SceneManager.LoadScene ("Level 2");
                        break;
                case "Button 3":
                        soundController.PlaySound (true);
                        SceneManager.LoadScene ("Level 3");
                        break;
                case "Easy":
                        soundController.PlaySound (false);
                        break;
                case "Normal":
                        soundController.PlaySound (false);
                        break;
                case "Hard":
                        soundController.PlaySound (false);
                        break;
                case "Exit_level":
            soundController.PlaySound(true);
            if (GameObject.FindGameObjectWithTag("Player").GetComponent<playerController>().notDead ==
false) {
                SceneManager.LoadScene("Leader Board");
            } else {
                SceneManager.LoadScene("Level Select");
            }
                        break;
                }
        }

        //This is executed when the 'Quit' button is pressed.
        private void Quit () {
                Debug.Log ("Activating question box");
                question.SetActive (true);
                yes.SetActive (true);
                no.SetActive (true);
                Debug.Log ("Done");

        }

        //This is exectued when the 'No' button is pressed.
        private void CancelQuit () {
                Debug.Log ("Closing question box");
                question.SetActive (false);
                yes.SetActive (false);
                no.SetActive (false);
                Debug.Log ("Done");
        }
}
```

playerController.cs

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

public class playerController : MonoBehaviour {

        private Rigidbody2D rb;
        public float forceMultiplier;
        //public float maxVelocity;
        public float jumpMultiplier;
        Vector2 leftVelocity;
        Vector2 rightVelocity;
        //bool isGrounded;
        public string direction = "right";
        public Animator playerAnimator;

        [SerializeField]
        private Transform[] groundCheckers;

        public float radius;
        public LayerMask ground;
        public LayerMask enemy;
        public bool isGrounded;
    public bool isOnEnemy;
        private float maxJumpCount = 2;

        Collider2D playerCollider;

    //int previousScore;
    public int score = 0;
    public Text scoreText;

    private AudioSource audioSource;
    public AudioClip walk;
    public AudioClip hurt;
    public bool readyToPlay = true;
    private float randomVol;
    private float minVol = 0.5f;
    private float maxVol = 1.0f;

    public Slider healthBar;
    //public int damage = 2;
    public bool notDead = true;
    //private SpriteRenderer player;
    //public Sprite bonk;
    bool enableControls = true;
    bool pushLeft = false;
    bool pushRight = false;

    public Text completed;
    public Text failed;

    public int damage;
    public float scoreMultiplier;
    public float volumeMultiplier;

        //public Vector2 zeroVelocity;

        // Use this for initialization
        void Start () {
         damage = PlayerPrefs.GetInt("Damage", 1);
```

```
        scoreMultiplier = PlayerPrefs.GetFloat("ScoreMultiplier", 1f);
        volumeMultiplier = PlayerPrefs.GetFloat("Volume", 1f);

        completed = GameObject.Find("Level Completed").GetComponent<Text>();
        failed = GameObject.Find("Level Failed").GetComponent<Text>();
        completed.text = "";
        failed.text = "";
        //zeroVelocity.Set (0f, 0f);
        rb = GetComponent<Rigidbody2D> ();
              playerAnimator = GetComponent<Animator> ();
        scoreText = GameObject.Find("Score number").GetComponent<Text>();
        audioSource = GetComponent<AudioSource>();
        healthBar = GameObject.Find("Health Bar").GetComponent<Slider>();
        //player = GetComponent<SpriteRenderer>();
       }

       // Update is called once per frame
       void FixedUpdate () {
        isOnEnemy = OnEnemy();
        isGrounded = CheckIfGrounded();
        Debug.Log("notDead: " + notDead + ", enableControls: " + enableControls);

        if (pushLeft) {
            pushLeft = false;
        } else if (pushRight) {
            pushRight = false;
        }


        if (isGrounded && !(Input.GetKeyDown(KeyCode.A) && Input.GetKeyDown(KeyCode.D)
&& Input.GetKey(KeyCode.Space)) && enableControls && notDead) {
            //Debug.Log("Forcing player to stop");
            rb.velocity = new Vector2(0f, rb.velocity.y);
        }

        if (Input.GetKey (KeyCode.A) && notDead && enableControls) {
            //MoveLeft();
            //playerAnimator.Play("Player Walk");
            rb.velocity = new Vector2(-forceMultiplier, rb.velocity.y);
                    if (direction == "right") {
                           direction = "left";
                           transform.localRotation = Quaternion.Euler (0f, 180f, 0f);
//changes rotation of player
                    }
              }
              if (Input.GetKey (KeyCode.D) && notDead && enableControls) {
            //playerAnimator.Play("Player Walk");
            rb.velocity = new Vector2(forceMultiplier, rb.velocity.y);
                    if (direction == "left") {
                           direction = "right";
                           transform.localRotation = Quaternion.Euler (0f, 0f, 0f);
                    }
              }
       }

       void Update () {
        if (readyToPlay && notDead && enableControls) {
            readyToPlay = false;
            StartCoroutine(PlayWalkSound());
        }
        if (Input.GetKeyDown(KeyCode.Space)) {
            Jump();
        }
```

```
        if (isGrounded && !(Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D)) &&
notDead && enableControls) {
                    playerAnimator.Play ("Player Idle"); //Plays the animations
            }
            if (!isGrounded && rb.velocity.y > 1f && (!Input.GetKey (KeyCode.A) ||
!Input.GetKey (KeyCode.D)) && notDead && enableControls) {
                    playerAnimator.Play ("Player Jump");
            }
            if (!isGrounded && rb.velocity.y < -1f && (!Input.GetKey (KeyCode.A) ||
!Input.GetKey (KeyCode.D)) && notDead && enableControls) {
                    playerAnimator.Play ("Player Fall");
            }
            if (isGrounded && (Input.GetKey (KeyCode.A) || Input.GetKey (KeyCode.D))
&& notDead && enableControls) {
                    playerAnimator.Play ("Player Walk");
            }
      }

      private void OnCollisionEnter2D (Collision2D other) {
            Debug.Log ("Colliding...");
            if (other.gameObject.tag == "Enemy") {
                  Debug.Log ("Touching enemy...");
                  if(OnEnemy () && notDead && enableControls) {
                        Debug.Log ("On top of enemy");
                        rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
              other.gameObject.SendMessage("Score");
              other.gameObject.SendMessage("EnemyDeath");
                  }
                  else if (notDead) {
              Debug.Log("Punishing player");
              //player.sprite = bonk;
              enableControls = false;
              playerAnimator.Play("Player Bonk");
              if (direction == "left") {
                  //.Log("Force Applied");
                  pushLeft = true;
                  rb.velocity = new Vector2(forceMultiplier, jumpMultiplier);
                  Debug.Log(rb.velocity.ToString());
              }
              if (direction == "right") {
                  Debug.Log("Force Applied");
                  pushRight = true;
                  rb.velocity = new Vector2(-forceMultiplier, jumpMultiplier);
                  Debug.Log(rb.velocity.ToString());
              }

              //rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
              //enableControls = false;
              audioSource.PlayOneShot(hurt, volumeMultiplier);
              healthBar.value -= damage;
              if (healthBar.value <= 0) {
                  failed.text = "Level Failed!";
                  PlayerPrefs.SetInt("CurrentScore", score);
                  if (SceneManager.GetActiveScene().name == "Level 1") {
                      PlayerPrefs.SetInt("CurrentLevel", 1);
                  } else if (SceneManager.GetActiveScene().name == "Level 2") {
                      PlayerPrefs.SetInt("CurrentLevel", 2);
                  } else if (SceneManager.GetActiveScene().name == "Level 3") {
                      PlayerPrefs.SetInt("CurrentLevel", 3);
                  }
                  notDead = false;
                  Debug.Log("Player is dead");
              }
          }
```

```
            }
        if (other.gameObject.name == "Finish Block(Clone)") {
            completed.text = "Level Completed!";
            PlayerPrefs.SetInt("CurrentScore", score);
            if (SceneManager.GetActiveScene().name == "Level 1") {
                PlayerPrefs.SetInt("CurrentLevel", 1);
            } else if (SceneManager.GetActiveScene().name == "Level 2") {
                PlayerPrefs.SetInt("CurrentLevel", 2);
            } else if (SceneManager.GetActiveScene().name == "Level 3") {
                PlayerPrefs.SetInt("CurrentLevel", 3);
            }

            notDead = false;
            Debug.Log("Player Won");
        }
        if (other.gameObject.tag == "Block") {
            if (isGrounded && !(Input.GetKeyDown(KeyCode.A) &&
Input.GetKeyDown(KeyCode.D) && Input.GetKeyDown(KeyCode.Space)) && enableControls) {
                //Debug.Log("Forcing player to stop");
                rb.velocity = new Vector2(0f, rb.velocity.y);
            }
        }
       }

    private void OnTriggerEnter2D(Collider2D collision) {
        switch (collision.gameObject.tag) {
            case "Key":
            Debug.Log("Key");
            //previousScore = score;
            score += Mathf.RoundToInt(50 * scoreMultiplier);
            UpdateScore();
            //StartCoroutine(UpdateScore());
            Destroy(collision.gameObject);
            break;

            case "Food":
            Debug.Log("Burger");
            //previousScore = score;
            score += Mathf.RoundToInt(5 * scoreMultiplier);
            Debug.Log(score);
            UpdateScore();
            //StartCoroutine(UpdateScore());
            Destroy(collision.gameObject);
            break;

            case "Spikes":
            if (notDead) {
                rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
                playerAnimator.Play("Player Bonk");
                audioSource.PlayOneShot(hurt);
                healthBar.value -= damage;
                if (healthBar.value <= 0) {
                    failed.text = "Level Failed!";
                    PlayerPrefs.SetInt("CurrentScore", score);
                    if (SceneManager.GetActiveScene().name == "Level 1") {
                        PlayerPrefs.SetInt("CurrentLevel", 1);
                    } else if (SceneManager.GetActiveScene().name == "Level 2") {
                        PlayerPrefs.SetInt("CurrentLevel", 2);
                    } else if (SceneManager.GetActiveScene().name == "Level 3") {
                        PlayerPrefs.SetInt("CurrentLevel", 3);
                    }
                    notDead = false;
                    Debug.Log("Player is dead");
                }
```

```
            }
            break;
        }
    }

    private bool CheckIfGrounded () {
            if (true/*rb.velocity.y <= 0*/) {
                foreach (Transform checker in groundCheckers) { // BELOW: creates
colliders at each ground checker
                    Collider2D[] colliders = Physics2D.OverlapCircleAll
(checker.position, radius, ground);
                    for (int i = 0; i < colliders.Length; i++) {
                        if (colliders [i].gameObject != gameObject) { //if
collider looked at isn't player
                            maxJumpCount = 1;
                            Debug.Log ("Is grounded!");
                        enableControls = true;
                            return true; //return true if colliding with
object other than player
                        }
                    }
                }
            }
            return false;
        }

        public bool OnEnemy () {
            if (true/*rb.velocity.y <= 0*/) {
                foreach (Transform checker in groundCheckers) {
                    Collider2D[] colliders = Physics2D.OverlapCircleAll
(checker.position, radius, enemy);
                    for (int i = 0; i < colliders.Length; i++) {
                        if (colliders [i].gameObject != gameObject &&
notDead && enableControls) {
                        //previousScore = score;
                        //score += 20;
                        UpdateScore();
                        //StartCoroutine(UpdateScore());
                            return true;
                        }
                    }
                }
            }
            return false;
        }

    public void Jump () {
            if ((isGrounded || maxJumpCount != 0) && notDead && enableControls) {
                //rb.AddForce (transform.up * jumpMultiplier);
                rb.velocity = new Vector2(rb.velocity.x, jumpMultiplier);
                maxJumpCount--;
                //Debug.Log(maxJumpCount);
            }
        }

    public void UpdateScore() {
        if (score < 10) {
            scoreText.text = "00" + score.ToString();
        } else if (score < 100) {
            scoreText.text = "0" + score.ToString();
        } else {
            scoreText.text = score.ToString();
        }
    }
```

```
    IEnumerator PlayWalkSound () {
        if (isGrounded && (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D))) {
            randomVol = Random.Range(minVol, maxVol);
            audioSource.PlayOneShot(walk, randomVol * volumeMultiplier);
            yield return new WaitForSeconds(walk.length + 0.125f);
        }
        readyToPlay = true;
        yield return true;
    }

}
```

ScoreController.cs

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

public class ScoreController : MonoBehaviour {

    public Text[] scoreTexts;
    public Text[] levelTexts;

    public int[] scores;
    public int[] levels;

    int playerScore;
    int playerLevel;

    int tempScore;
    int tempLevel;

    public void DeleteScores () {
        PlayerPrefs.DeleteAll();
    }

    void Start() {
        playerScore = PlayerPrefs.GetInt("CurrentScore", 0);
        playerLevel = PlayerPrefs.GetInt("CurrentLevel", 0);

        scores[0] = PlayerPrefs.GetInt("s0", 0);
        scores[1] = PlayerPrefs.GetInt("s1", 0);
        scores[2] = PlayerPrefs.GetInt("s2", 0);
        scores[3] = PlayerPrefs.GetInt("s3", 0);
        scores[4] = PlayerPrefs.GetInt("s4", 0);
        scores[5] = PlayerPrefs.GetInt("s5", 0);
        scores[6] = PlayerPrefs.GetInt("s6", 0);
        scores[7] = PlayerPrefs.GetInt("s7", 0);
        scores[8] = PlayerPrefs.GetInt("s8", 0);

        levels[0] = PlayerPrefs.GetInt("l0", 0);
        levels[1] = PlayerPrefs.GetInt("l1", 0);
        levels[2] = PlayerPrefs.GetInt("l2", 0);
        levels[3] = PlayerPrefs.GetInt("l3", 0);
        levels[4] = PlayerPrefs.GetInt("l4", 0);
        levels[5] = PlayerPrefs.GetInt("l5", 0);
        levels[6] = PlayerPrefs.GetInt("l6", 0);
        levels[7] = PlayerPrefs.GetInt("l7", 0);
        levels[8] = PlayerPrefs.GetInt("l8", 0);

        if (playerScore > scores[8]) {
            scores[8] = playerScore;
            levels[8] = playerLevel;
        }

        for (int i = 8; i > 0; i--) {
            if (scores[i] > scores[i - 1]) {
                tempScore = scores[i - 1];
                tempLevel = levels[i - 1];
                scores[i - 1] = scores[i];
                levels[i - 1] = levels[i];
                scores[i] = tempScore;
                levels[i] = tempLevel;
            }
        }
        for (int i = 0; i < 9; i++) {
```

```
            Debug.Log("Setting score: " + i);
            scoreTexts[i].text = scores[i].ToString();
            levelTexts[i].text = levels[i].ToString();
        }

        PlayerPrefs.SetInt("s0", scores[0]);
        PlayerPrefs.SetInt("s1", scores[1]);
        PlayerPrefs.SetInt("s2", scores[2]);
        PlayerPrefs.SetInt("s3", scores[3]);
        PlayerPrefs.SetInt("s4", scores[4]);
        PlayerPrefs.SetInt("s5", scores[5]);
        PlayerPrefs.SetInt("s6", scores[6]);
        PlayerPrefs.SetInt("s7", scores[7]);
        PlayerPrefs.SetInt("s8", scores[8]);

        PlayerPrefs.SetInt("l0", levels[0]);
        PlayerPrefs.SetInt("l1", levels[1]);
        PlayerPrefs.SetInt("l2", levels[2]);
        PlayerPrefs.SetInt("l3", levels[3]);
        PlayerPrefs.SetInt("l4", levels[4]);
        PlayerPrefs.SetInt("l5", levels[5]);
        PlayerPrefs.SetInt("l6", levels[6]);
        PlayerPrefs.SetInt("l7", levels[7]);
        PlayerPrefs.SetInt("l8", levels[8]);

        PlayerPrefs.SetInt("CurrentScore", 0);
        PlayerPrefs.SetInt("CurrentLevel", 0);
    }

}
```

scoreSet.cs

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

public class scoreSet : MonoBehaviour {

    public Text text;
    public int score = 0;

       // Use this for initialization
       void Start () {
        text = GetComponent<Text>();
       }

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

       }

    public void UpdateScore (string type) {
        switch (type) {
            case "enemy":
            score += 20;
            text.text = score.ToString();
            break;

            case "burger":
            score += 5;
            text.text = score.ToString();
            break;

            case "key":
            score += 50;
            text.text = score.ToString();
            break;
        }
    }
}
```

SettingsController.cs

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

public class SettingsController : MonoBehaviour {

    public Button button;
    public Slider volume;

       // Use this for initialization
       void Start () {
        Button btn = button.GetComponent<Button>();
        btn.onClick.AddListener(TaskOnClick);

        volume.value = PlayerPrefs.GetFloat("Volume", 1f);
    }

       public void TaskOnClick () {
              switch (gameObject.name) {
            case "Easy":
            Debug.Log("Easy");
            PlayerPrefs.SetInt("Damage", 1);
            PlayerPrefs.SetFloat("ScoreMultiplier", 0.8f);
            break;

            case "Normal":
            Debug.Log("Normal");
            PlayerPrefs.SetInt("Damage", 2);
            PlayerPrefs.SetFloat("ScoreMultiplier", 1f);
            break;

            case "Hard":
            Debug.Log("Hard");
            PlayerPrefs.SetInt("Damage", 3);
            PlayerPrefs.SetFloat("ScoreMultiplier", 1.2f);
            break;
        }
       }

    void Update () {
        if (volume != null) {
            PlayerPrefs.SetFloat("Volume", volume.value);
        }
    }
}
```

SoundController.cs

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

public class SoundController : MonoBehaviour {

	private AudioSource source;
	public AudioClip buttonPress;

    public float volume;

	// Use this for initialization
	void Start () {
	 DontDestroyOnLoad (this.gameObject); //Stays loaded, always
		source = GetComponent<AudioSource>(); //Gets AudioSource component
	}

	public void PlaySound (bool checkIfDestroy) {
		source.PlayOneShot (buttonPress, volume); //Plays the sound
		Debug.Log ("Finished playing sound");
		StartCoroutine(Wait (checkIfDestroy));
	}

	IEnumerator Wait (bool checkIfDestroy) {
		yield return new WaitForSecondsRealtime (0.288f);
		if (checkIfDestroy == true) {
			Debug.Log ("Destroying...");
			Destroy (this.gameObject);
		}
	}

    private void Update() {
        volume = PlayerPrefs.GetFloat("Volume", 1f);
    }
}
```

Bibliography

- **Font 'Pixellife' by William Clifford**
  Permissions: I am allowed to use this font as long as I distribute the game and/or any flies where it's used along with the following file:

  ...

  PUNKROCKTYPOGRAPHY.typeface
  Copyright: 1998-2000 william.clifford
  Website: http://www.williac.com

  License: You, the user, are permitted to use this typeface for your design/word processing purposes. You may use it for both private and commercial causes. You may redistribute this typeface only if you include this file (readme.txt) with it. This typeface does not have a complete character set. You use this typeface at your own risk. If you like this typeface, please tell me. If you would like to remix this typeface, please give me credit. I own this typeface. It is my property. I am letting you use it. My email address is punkrock@williac.com

  ...
- **Game Background by wupto**
  Permissions: https://www.deviantart.com/view/377140556
- **Sprites by finalbossblues**
  Permissions: none needed, https://finalbossblues.itch.io/pixel-platformer-pack
- **Enemy hurt sound by Michel88**
  Permissions: Sampling+ License (https://creativecommons.org/licenses/sampling+/1.0/)
  https://freesound.org/s/76969/
- **Player footstep sound by deleted_user_5093904**
  Permissions: Attribution 3.0 License (https://creativecommons.org/licenses/by/3.0/)
  https://freesound.org/s/268758/
- **Enemy walk sound by LittleRobotSoundFactory**
  Permissions: Attribution 3.0 License, https://freesound.org/s/270421/
- **Player hurt sound by Reitanna**
  Permissions: Creative Commons 0 License
  (https://creativecommons.org/publicdomain/zero/1.0/)
  https://freesound.org/people/Reitanna/sounds/344033/
- **Button sound by Sound Ex Machina**
  Permissions: Standard License (https://www.zapsplat.com/license-type/standard-license/)
  https://www.zapsplat.com/music/buttons-stone-button/