**Web game report**

**Game Summary**

The objective of my game is to explore a hazardous environment while avoiding obstacles and enemies and trying to get a high score by collecting items and destroying enemies.

The rules of the game involve avoiding obstacles such as spiky enemies and dangerous platforms which result in a restart and trying to collect coins and get a high score.

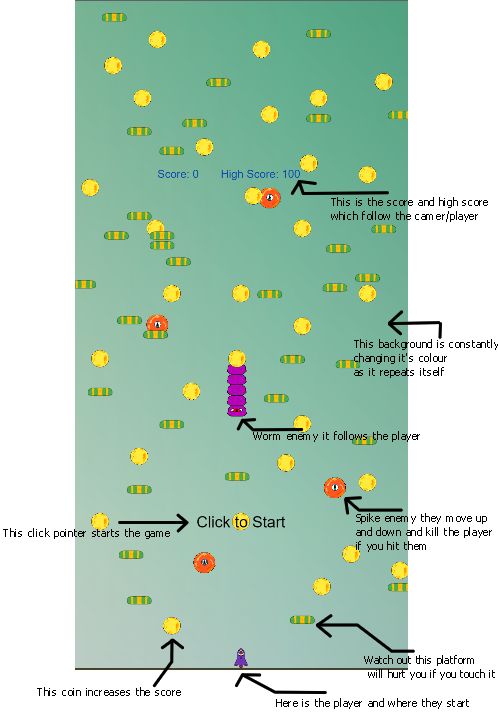
The gameplay involves moving the player character around a level avoiding obstacles by maneuvering around and collecting coins to increase a score and try to get a higher score than previous.

**Research**

First came the period where I researched and investigated similar games that sort of fitted my game vision. One such game is Doodle jump where there is a similar goal of achieving a high score while avoiding enemies, I also was inspired by its cartoony style which would impact how I designed my game’s art assets later. A second game I was inspired by is fancy pants’ adventure which is like Doodle jump is a platformer but with a more fluid and abstract look to it as it is designed like doodles in a children's book or drawings brought to life.

My first piece of research for visual design was looking at the inspiration for my game, looking at different and varied art styles to be inspired by. I decided to go for the 2000s based on urban vinyl toys. The look of said urban vinyl toys I hope to make it look like a figure you could get that was popular in the 2000s or something you would see on a very stylized cartoon in the 2000s and this would make my game stand out or at least nice to look at. Using bright colours and exaggerated looks are the main objectives of all the visual assets in my game.

**Screen/Level Map:**



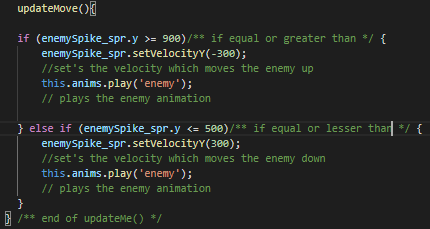
**Functions/Objects**

This function operates the player sprite moving it around the game space affecting its velocity and what animations are playing. Using the cursors of the computer I use the if statement of if certain arrow buttons are down to update the player sprite in the game space.



The updateMove this function moves the spike enemy up and down using an if and else if. Firstly, the first if statement checks if the spike sprite is equal or greater than the y value 900. If this is true it sets the velocity to –300 effectively moving it up and plays the animation from its sprite sheet.

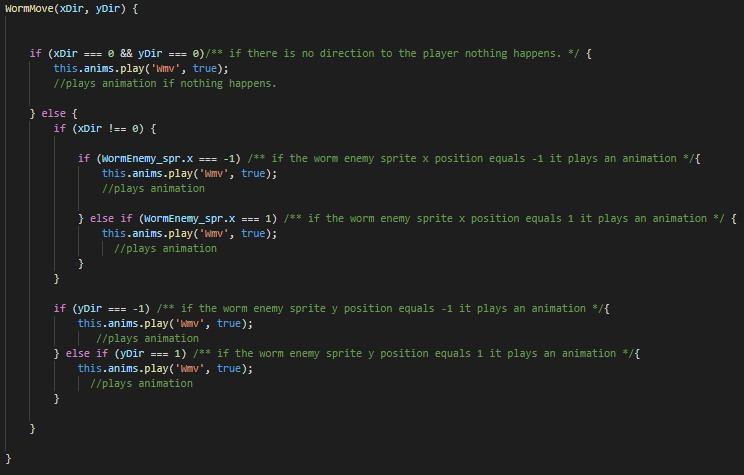
Below is the else if which is the same except if the sprite is equal or less than 500 it set the velocity to 300 moving it down and playing the same enemy sprite animation this creates a loop which results in the spike enemy moving up and down.



Now we have the movement for the worm AI enemy. First, we have a simple if statement where if the two let xDir and yDir equal 0 or in game terms no direction or need to move to the player nothing happens except for playing the worms animation.

Below we have another if statement which contains several if and else if statements, the first one encompasses all the others which uses the if statement of if xDir is not equal to 0. If it is true, then an if and an else if statement occur. The first one occurs if the worm sprite is equal to –1 which plays and animation the same is true with the else if statement except it occurs if it is equal to 1.

Below is the same sort of if statements except for the y axis playing the animation.

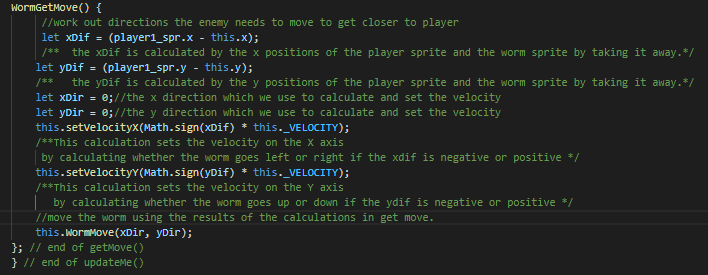


Next following on from wormMove we have WormGetMove which calculates the distance and direction from the player and how to get closer to the player. Firstly, we have the let for the y and x dif which are used to work out the difference of position between the player sprite and the worm sprite by subtracting the difference we get our let value for the xdif and ydif.

Then we have the x and y dir. Which are the values we use to move the worm sprite. The wormgetmove calculates the value for these axes and applies it to wormMove moving it.

Next are two calculations for x and y respectively using math sign with the dif let to see if it is positive negative or null and multiplying it by the velocity of the worm so if the dif becomes negative the velocity is negative so the enemy goes up the opposite is true and thus the enemy will correctly follow the player using the calculations in the get move and follow and kill the player upon touching.

And finally, we update the WormMove using the xDir and yDir.

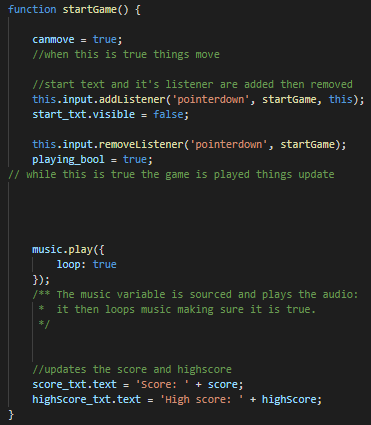


I used the Moon flyer example from week 4 as the template for my cyclops game however through development the final product is drastically different and ended up as more of a platformer but still retains some elements such as a click to start, score system and a revolving background but the original code has now been heavily modified.

In the startGame function it initializes the game by setting up basic things like scores and music. First, we have the canmove which is true this allows movement for the player and other objects in the scene.

Next, we have the listener for the start text which is added and then further down removed when the player clicks it the start text is also removed. And then the playing\_bool becomes true which effectively allows the game to update and things to happen like movement and music.

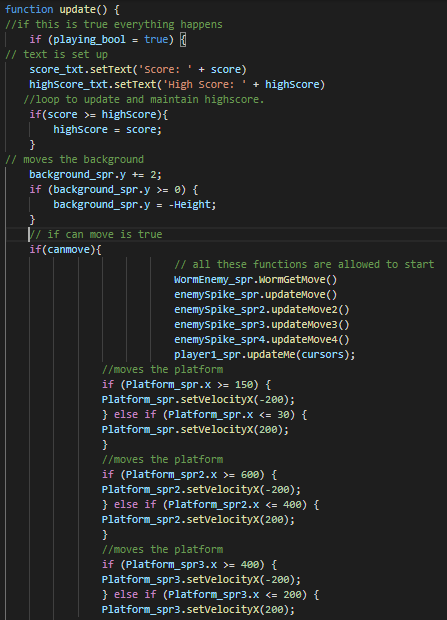
Then we have a loop which plays the music first by finding the variable and its file source and looping it by setting the loop to true. And below we set up the score and high score text using text and variable values from score and highScore.



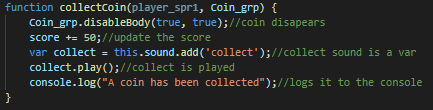
Now we have the update function which everything occurs if playing\_bool is true. After we check if playing\_bool is true we set up the text with their assigned variables as well as an if statement for highScore which will update if the highScore is less than the current score it will copy it and make a new record for the high score.

Below is an if statement which loops the background around making it go down then reset it so to create a seamless cascading look to the background.

Below is an if statement for canmove which if it is true allows everything in it to move. Firstly, we have several functions in separate classes that are allowed to update and occur such as the player and enemies. Below this is if and else if statements for several platforms in the game to move on the x-axis in a set range and a set speed.



The collect coin function updates the scene and the score by removing a coin in the scene and adding points to the score. Firstly, the body of the coin is disabled making it vanish from the scene then a sound is added and then played, and the coin is given a console log.



And now we have the killPlayer function which occurs when the player sprite collides with certain objects in the scene. First, we disable the player sprite then we make sure that the text is visible. Below that we log into the console that the game has been reset and the score is set to 0. We add an audio variable that plays, and we stop the background music from playing and set canmove to false, so everything stops moving. Below this, we have a for loop which resets and replaces any coins that have been collected by the player, so the scene is the same as when the player starts. Then we do a scene restart.

The restart scene I got from a Phaser example which I copied to see how restart worked however that example the restart was triggered by a click from a button while this is caused by player sprite collision, and I only copied the this.scene.restart().



**Code Structure**

My code is divided into several files. Firstly, you have a simple HTML page where all the phaser files are applied to. Next, I have the main js/phaser file Cyclops.js which creates the scene of the game. It also has separate phaser files for the player and enemies. The first of these files is the player JavaScript file which contains information on the player movement and its relation to button inputs by the player as well as how fast it moves when the button press occurs. Then we have two enemy JavaScript files which go over their movement as well as their Ai.

**Assets**

All the sprites in the game I made myself by using Pixilart: using it I developed assets in the game such as the player object enemies and collectable coins and animated them by creating sprite sheets of the character sprites and creating animations in the phaser JavaScript. I Also created the simple moving background by using a digital software called Krita and added some gradients to create a rather simple but dynamic background and make it loop around using some simple code to loop it over and over.

The sound effects I used I got from freesounds.org for example the simple background music and sound effects for collecting coins and dying all have sound effects that play when it happens.

**Implementation Evaluation**

The strengths of my code are the implementation of mechanics in the game. Using functions and especially the update function would help in creating a seamless game experience such as a coin collection system and a high score system and the movement of the player using some if statements to create a control interface helps to create an engaging experience. With Phaser, I can properly and effectively create mechanics that can create a game using a variety of different methods, functions and classes to produce an engaging experience.

A second strength is the use of separate js files to help flesh out certain parts of the game. For example, I have separate files for the player as well as enemies. This helps make the main JavaScript file cleaner and not hard to navigate if, for example, all the information for player movement or enemy Ai was in that file. So, thanks to Phaser we can separate code especially those that focus on a specific part into a separate JavaScript file making it easier to access information separately as well as making it usable.

A third strength is the use of customs assets created by me. By using software such as Pixilart and Krita I created assets mostly pixel art that is unique and give my game a look that is cartoony and I hope is recognizable. Thanks to the straightforward way to implement assets into the Phaser/JavaScript files I can produce and add them to the game very easily with it is easy to modify and edit the assets to my liking such as their size or even change what they look like.

The first weakness of my code is its bloated nature. This is because the excessive and inefficient placement and implementation of things like platforms, the coins and the spiked enemy are not up to size as they take up large amounts of space in the code making it less clean and harder to find information. If JavaScript had the ability to identify and maximize the efficiency of the code and get rid of bloat effectively, it could get rid of all the things that decrease the readability and useability of any code made in JavaScript.

A second weakness is the limited application of Ai. In my game despite implementing very basic pathfinding following enemy ai it is quite simple and limited ignoring the platforms, and, in my game, there is no other Ai even the spike enemy does not have ai, unlike the worm enemy. With time, I think Phaser alongside JavaScript could be more useful and intuitive to develop Ai systems and possibly make it more engaging and advanced and to develop better Ai in my game.

And a third weakness is the design of the game caused by the inexperience in coding. Due to my lack of knowledge when it comes to coding the design of the game is limited in its ability to be even more engaging or interesting. I would want Phaser to provide a bit more when it comes to creating and supporting a basic game design comparable to Unity with its basic templates and design and a more usable user interface would help in creating depth in the game design.

**References**

### **PHASER - EXAMPLES - SCENE RESTART**

**In-text:** (Phaser - Examples - Scene Restart, 2018)

**Your Bibliography:** Phaser.io. 2018. *Phaser - Examples - Scene Restart*. [online] Available at: <https://phaser.io/examples/v3/view/physics/matterjs/scene-restart> [Accessed 17 November 2021].

### **PHASER - EXAMPLES - FOLLOW ZOOM**

**In-text:** (Phaser - Examples - Follow Zoom, 2018)

**Your Bibliography:** Phaser.io. 2018. *Phaser - Examples - Follow Zoom*. [online] Available at: <https://phaser.io/examples/v3/view/camera/follow-zoom> [Accessed 3 December 2021].

‌

### **8 BIT DEATH SOUND BY MENTOSLAT**

**In-text:** (8 bit Death sound by MentosLat, 2018)

**Your Bibliography:** Freesound. 2018. *8 bit Death sound by MentosLat*. [online] Available at: <https://freesound.org/people/MentosLat/sounds/417486/> [Accessed 23 December 2021].

### **8 BIT GAME LOOP 001 SIMPLE MIX 4 SHORT 120 BPM.WAV BY JOSEFPRES**

**In-text:** (8 bit game loop 001 simple mix 4 short 120 bpm.wav by josefpres, 2021)

**Your Bibliography:** Freesound. 2021. *8 bit game loop 001 simple mix 4 short 120 bpm.wav by josefpres*. [online] Available at: <https://freesound.org/people/josefpres/sounds/610506/> [Accessed 23 December 2021].

‌

### **8 BIT DEATH SOUND BY MENTOSLAT**

**In-text:** (8 bit Death sound by MentosLat, 2018)

**Your Bibliography:** Freesound. 2018. *8 bit Death sound by MentosLat*. [online] Available at: <https://freesound.org/people/MentosLat/sounds/417486/> [Accessed 23 December 2021].

‌