

Connor Andrew McSheffery

S1714052

Mobile Games Development

Github Link - https://github.com/CMCSHE201/Mobile-Game-Development-CMCSHE201

I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.

Connor McSheffery

Contents

[Background 3](#_Toc61285244)

[DungeonScroller.js 4](#_Toc61285245)

[Render.js 10](#_Toc61285246)

[Touch.js 16](#_Toc61285247)

[Player.js 20](#_Toc61285248)

[Enemy.js 27](#_Toc61285249)

[References 28](#_Toc61285250)

# Background

The game is a dungeon scroller game where the player’s objective is to achieve the highest possible score in the time limit given.

The player will gain score by jumping on the heads of the enemies which will remove them from the game and reward the player with 100 score. If the player collides with the enemy head-on then they will lose a life and will be sent back to the beginning.

The player has 3 lives to start the game with and the game will end if they run out of lives.

There are 3 enemies on the screen during each wave and once all 3 enemies are defeated, a new wave will spawn allowing the player to achieve more score.

The controls for the game are quite simple. A virtual gamepad has been added to make playing the game much easier. The player moves by moving the joystick left or right depending on the desired direction.



There is also a button on screen which is used to make the player jump. This is used to defeat the enemies or escaping danger.



If the player wishes to pause the game then they can press the pause button situated in the top right of the game which will lock all movement, animation and pause the timer from running out.



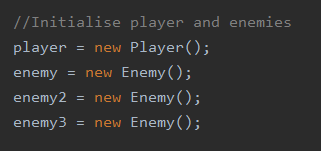
All the sprites used within the game were created independently using Aseprite.

The sounds featured within the game were either taken from online which will be referenced or were recorded independently.

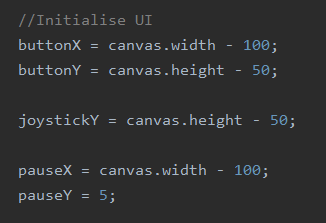
# DungeonScroller.js

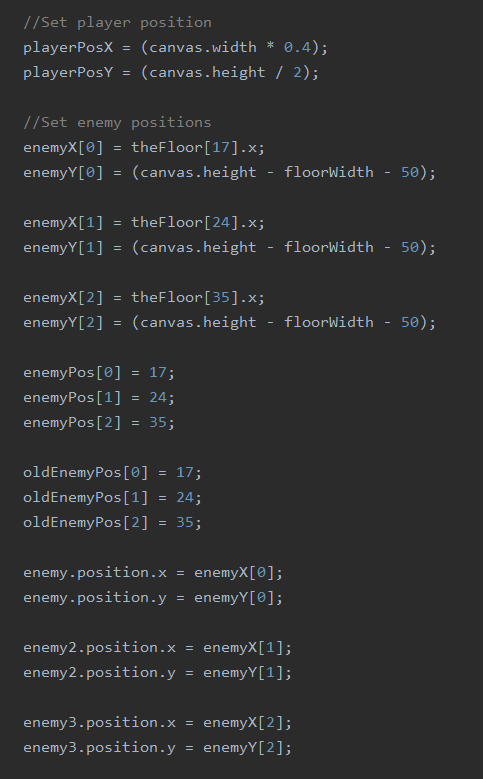
The DungeonScroller file deals with the general creation of the game and the movement of the player.

The file starts by creating instances of the player and enemy classes for within the game.

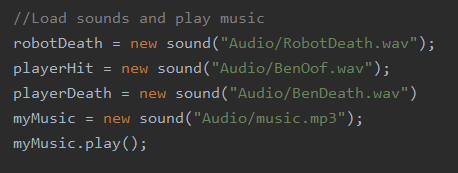


The UI positions are then initialized and set. The player and enemies’ positions are then also set.

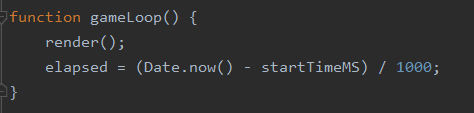




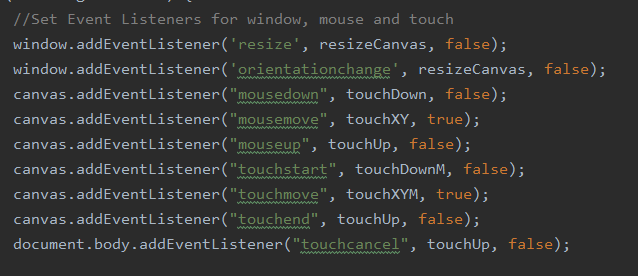
The sounds are loaded in and the music is played on loop throughout the game.



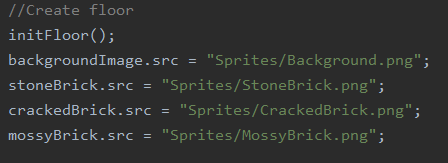
When the game is in its playing state, the gameLoop function will call the render function to keep everything appearing on screen and will calculate the frame time which can be used for other calculations within the game.



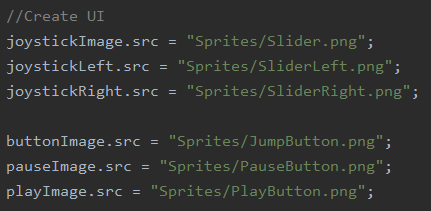
The event listeners are added to the game so touch and other input can be recorded and will then call their respective methods to take the data it gives and put it to use.



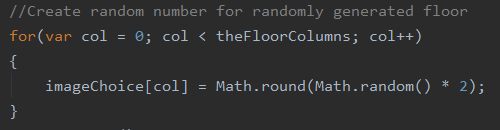
The floor creation method is called which will create all the positions of the floor tiles and then the sprites used for the background and floor will be loaded in.



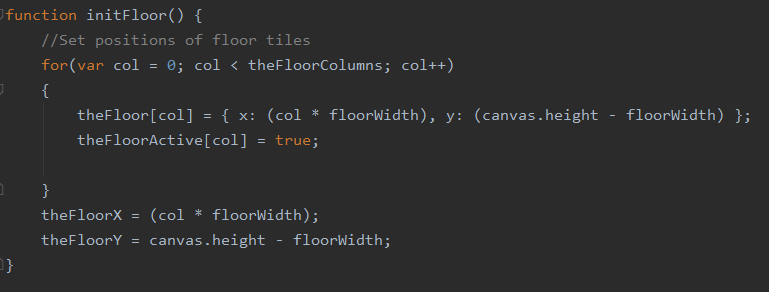
The UI images are also loaded in for use in the render function.



To allow for each type of floor image to be used and to add some randomly generated levels, the Math.random() function is used within an array to choose which floor image will be used when creating the floor. This means each game will have a different level design as the floor tiles will change through each game.



When initializing the floor, the floor’s tiles positions must be set, this is done incrementally through a for loop. Their active state is also set to true as all tiles are in use.

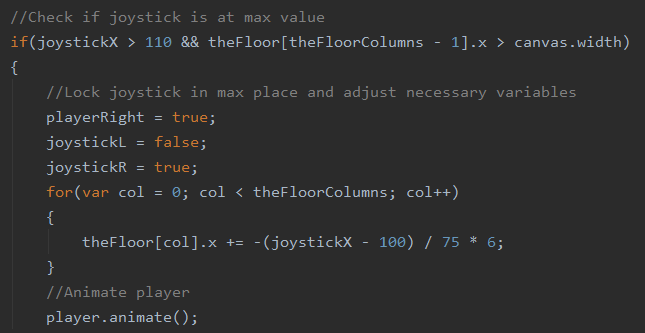


When the player moves right and they are not at the end of the level, boolean variables are adjusted to show the player moving right. The playerRight variable is used when rendering the player so the game will know to use the sprites where the player is facing to the right.

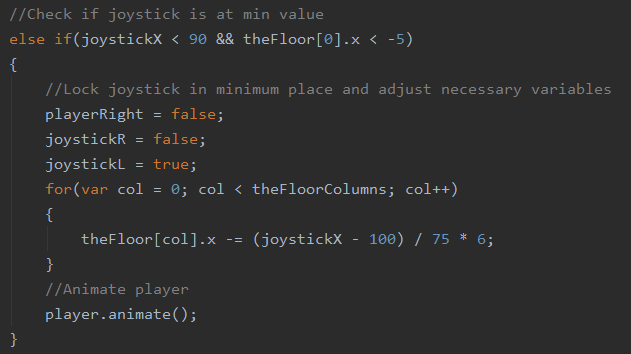
The joystickL and joystickR variables are also used for rendering so the joystick sprite will change to give feedback on the direction the player is moving.

To give the illusion of the player moving without moving the camera, the floor is moved in the opposite direction of which the user wishes to move.

The player is then animated which also helps with the illusion of the player moving.



When the player moves left, the same variables are adjusted but their values are the opposite of when the player moves right.



The sound used within the game is all dealt within a single function which was taken from <https://www.w3schools.com/graphics/game_sound.asp>

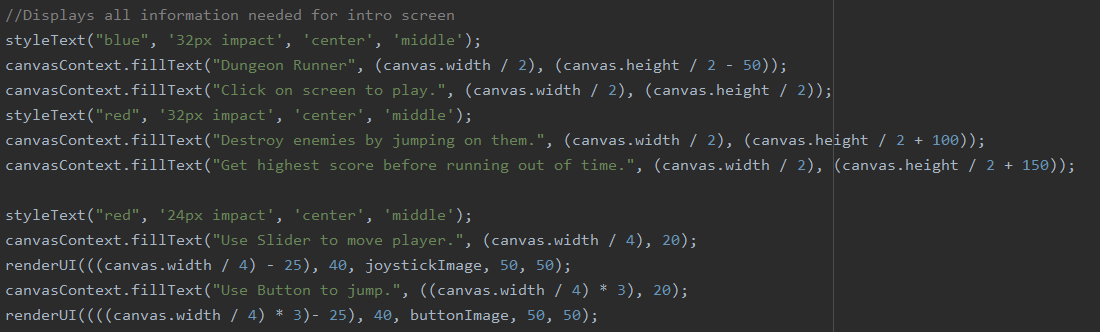
This function allows both sound effects and music to be played through the map with no issues.



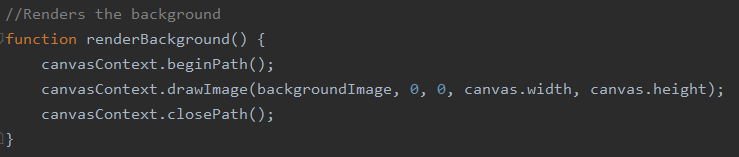
# Render.js

The render file deals with rendering all the entities within the game apart from the player and enemies as they are done within their separate classes. The game states are also dealt with within this file.

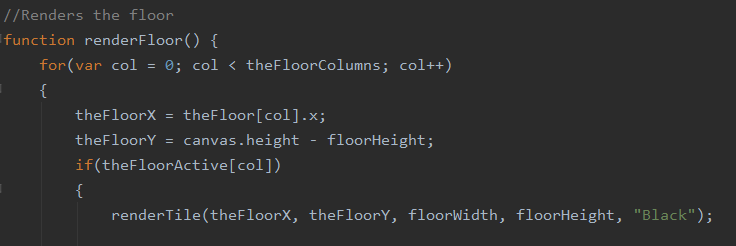
When the game starts, the player is shown the intro screen which also shows the instructions for the game and also lets the player know the premise of the game and how to achieve score.



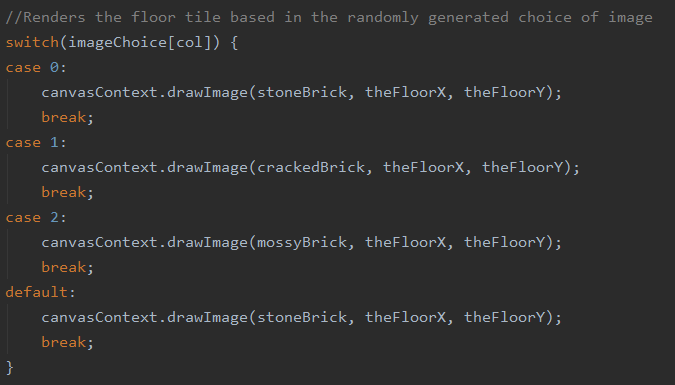
The background is rendered throughout the game as it is used in all game states. The dimensions are based on the canvas width and height as it takes up the whole screen.



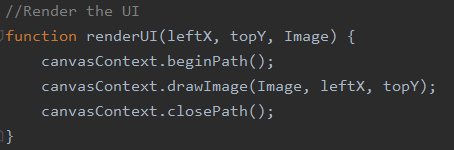
The floor is rendered using a plain black colour until the sprites are generated. The floor is created incrementally so it is placed perfectly. The method also checks if the floor is active and if not, the floor will not render in.



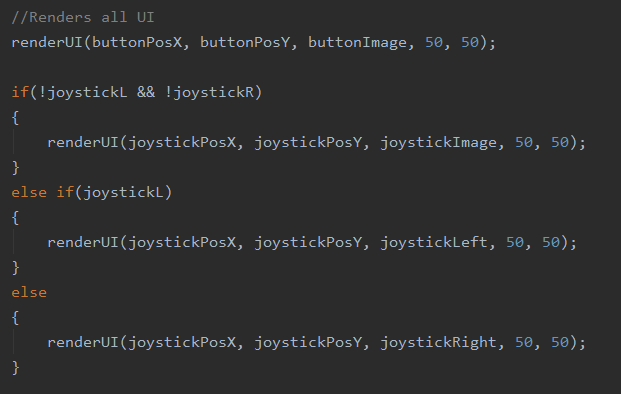
To generate the floor images, a switch case is used to render in the sprites based on the value within the imageChoice variable.



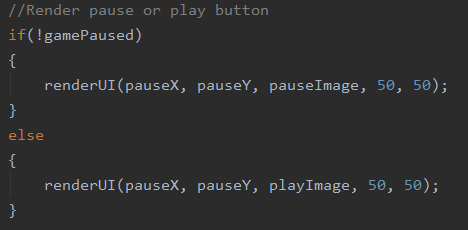
The UI render function takes in the position and image and will then render the image, this is used for all UI such as the joystick, jump button and play/pause button.



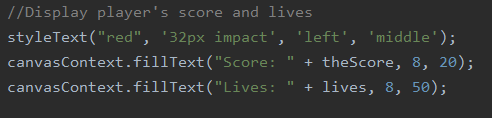
When the player moves the joystick, the image on the joystick changes to show the direction the player is moving. When the joystick is in the neutral zone, the image shows no discernible direction as the player will be stationary.



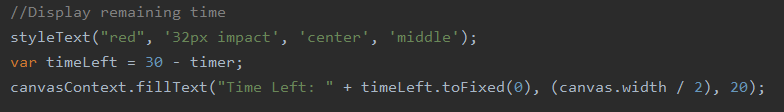
The play and pause button are placed in the same position and either will be rendered depending on whether the game is in it’s pause state or playing state.



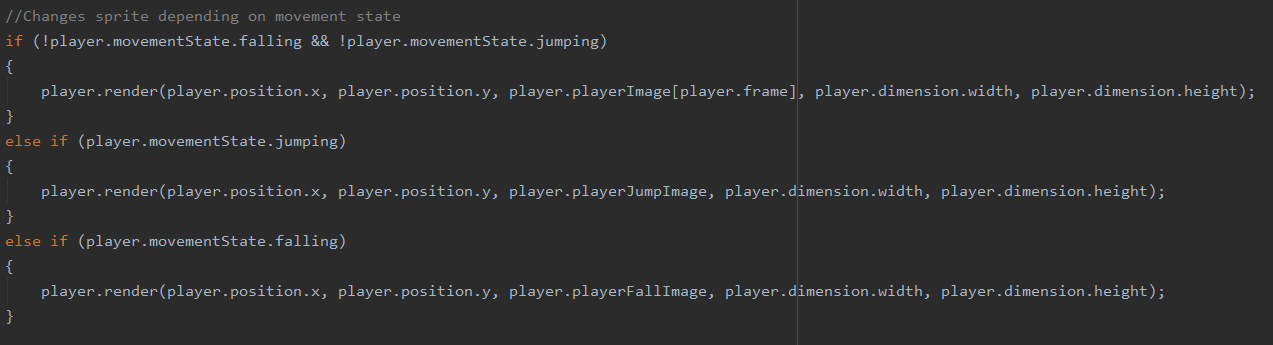
When the game is in it’s playing state, the player’s score and lives are displayed in the top left of the game to let the player know how well they are doing.



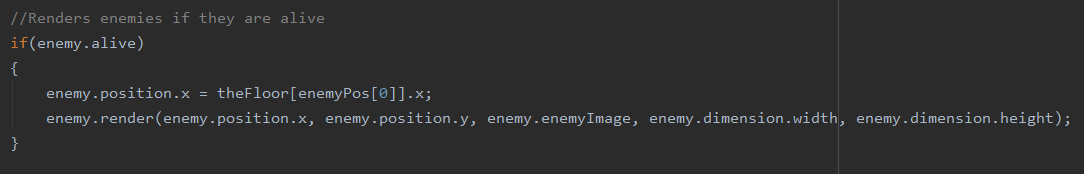
The time remaining is calculated using the timer of the game and is then displayed to let the player know how much time is remaining in the level.



The player render function is called depending on the movement state the player is in as there are different sprites for the player when in these states. The general state of the player also uses the player.frame variable as this allows for the player’s animation to show.

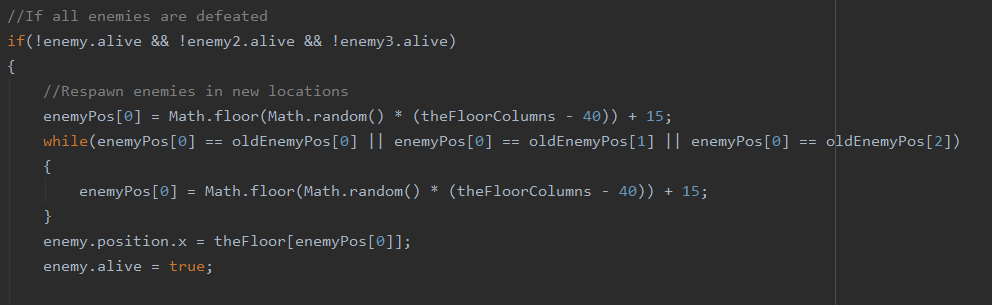


If the enemies are alive then their positions will be kept locked to the tile space they are allocated and is then rendered in.

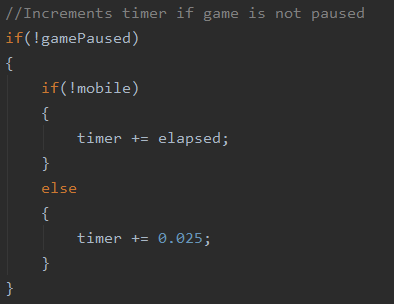


The render file also deals with resetting the game when all the enemies have been defeated.

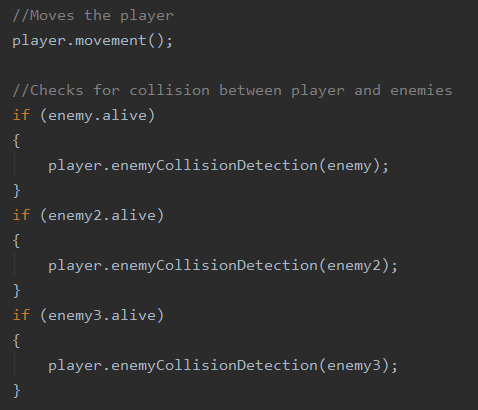
When this occurs, the enemies are given new tile spaces to set their position to. The method also makes sure not to spawn any enemies in spaces there were just previously occupied and also makes sure no two enemies are in the same space.



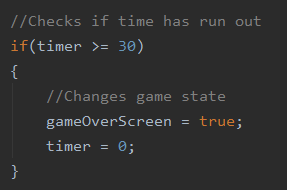
When the game is played, the timer is incremented using the delta time created within DungeonScroller.js.



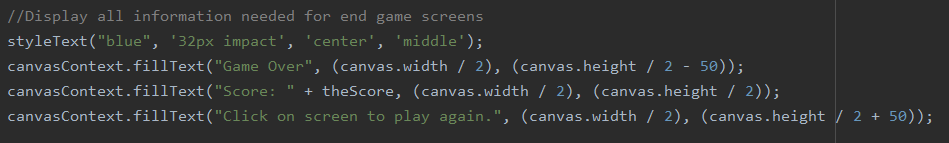
During the game, the player movement is called to allow the player to jump and fall. The collision between the enemies and player are also called here if the enemy is still active.



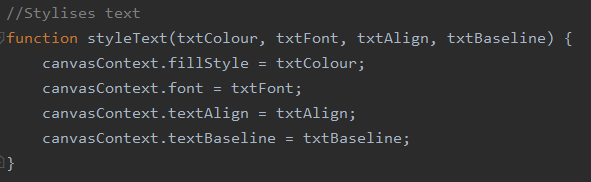
When the time runs out in the game, the timer is reset and the game will be brought to the game over screen.



When the game has ended, the player’s score will be displayed and from there, the player can choose to play again by pressing on the screen anywhere.



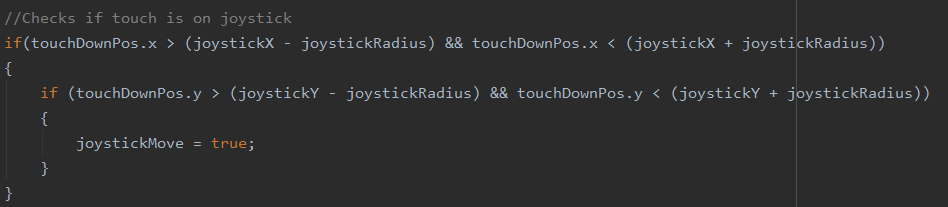
The render file also holds the function to stylize the text within the game, this allows for the colour, font alignment and baseline to be altered which will add more variety within the text on screen.



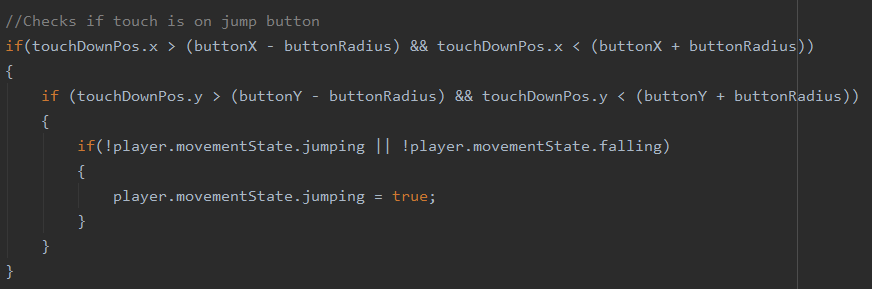
# Touch.js

The touch file deals with all the input that comes from the touches registered on the mobile or all the mouse clicks when the game is played in a browser.

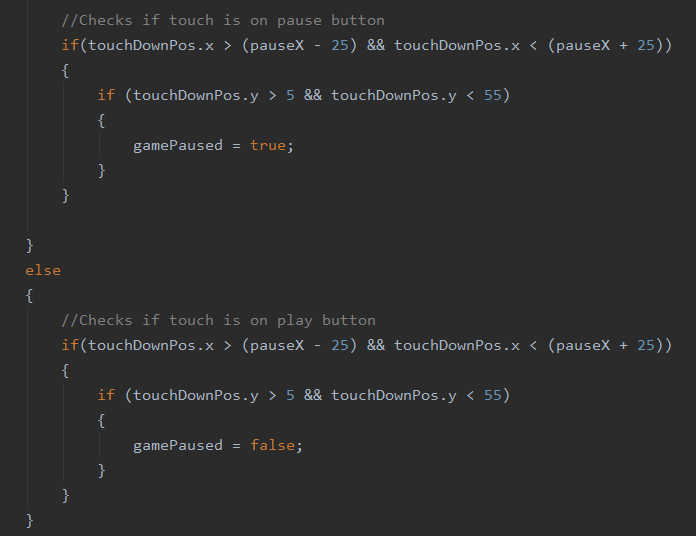
When the game is played and a touch is registered, the file checks the position of the touch and if it is in the vicinity of the UI. If the touch is within the joystick button, then a boolean variable will be set to true which lets the game know the player is trying to move the joystick.



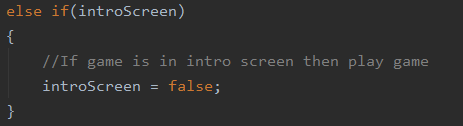
If the touch is registered within the jump button then the player’s jumping movement state will be set to true, making the player jump. There is a nested if statement inside to check if the player is already jumping or falling so the player must wait until they are grounded to jump again.



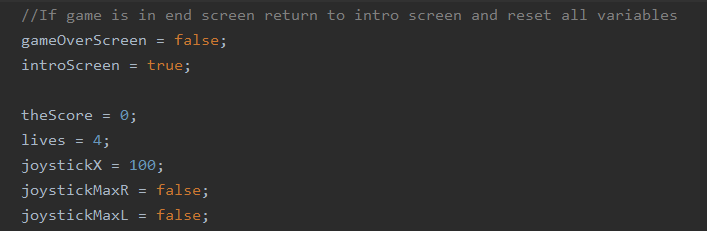
If the player’s touch is within the pause button, then the game will be paused. If the game is already paused then the play button will appear and when the click is registered there, the game will resume.



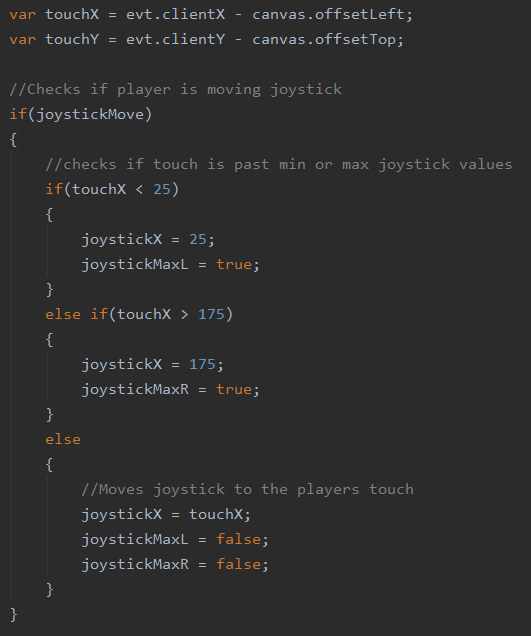
To start the game, the player can just click anywhere on the screen whilst in the intro screen and the game will start. This is done by registering a touch when the game is on the intro screen, once the touch has been registered it will deactivate the intro screen which will start the game.



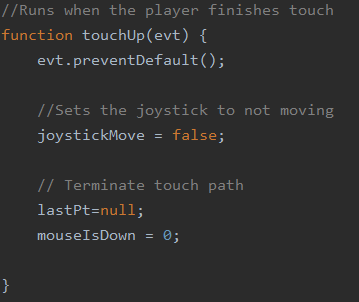
When the game has finished and the player wants to play again, the player can return to the intro scene using the same method as starting the game. When the game is restarted, the necessary variables are also reset in order to try again.



When the player moves the joystick, it will match the position of the touch, unless it is past the slider boundaries, in which case, it will be locked in the maximum or minimum position



When the player removes the touch from the mouse or browser, the file will then be told the joystick is no longer to be moved and the previous touch position is cleared.

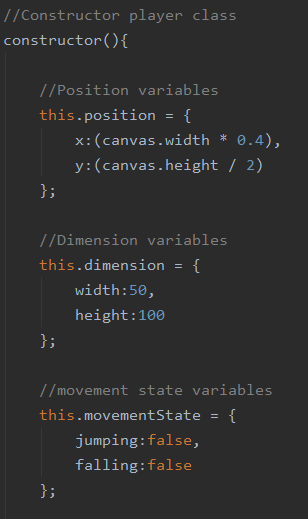


# Player.js

The player class deals with mostly everything relating to the player. It holds all necessary variables, deals with the rendering, movement, collision and animation of the player.

The constructor holds all the variables needed for the player to render, move, and animate.

The variables are set into subsections like the player’s position, dimensions and movement states such as jumping and falling.

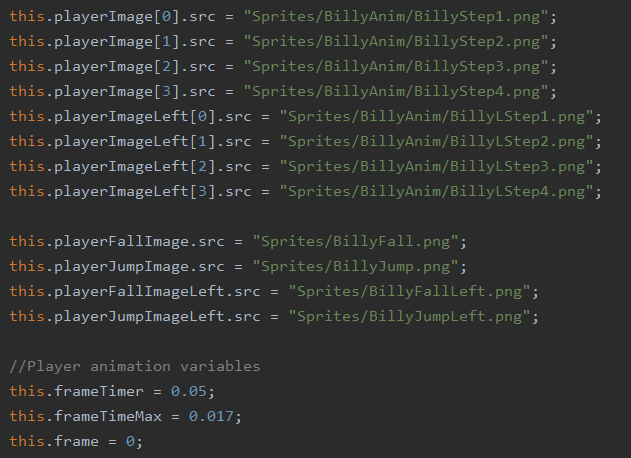


The constructor also holds the images used for rendering the player and the animation frames.

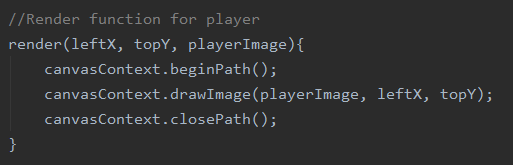


There are 2 separate player image arrays as the image will face the right when moving right and facing left when moving left.

The animation variables are also initialized and their values are inputted for when the animation will change.



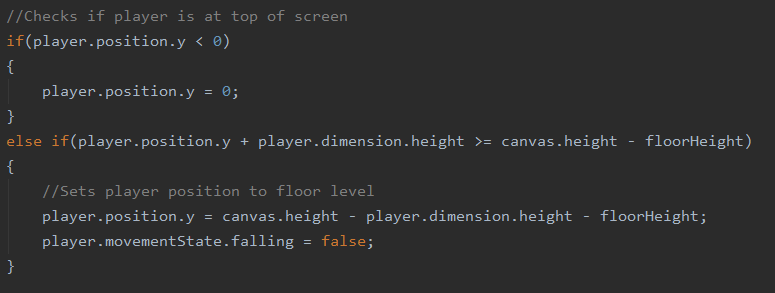
The class holds its own render function so locating or modifying it can be made much easier.



The player class deals with the y-position of the player as the x-position should not be being altered in any way.

The class has some restrictions incase of any bugs, if the player were to jump higher than the screen can show then the player will be brought back into frame.

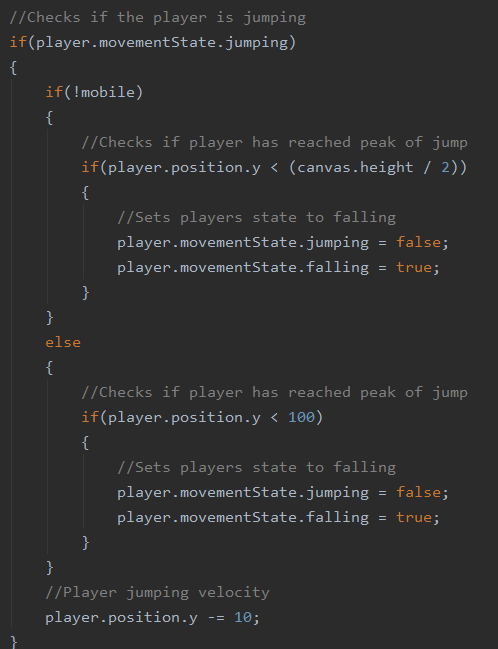
The player will also be locked to the floor position when they are neither falling or jumping.



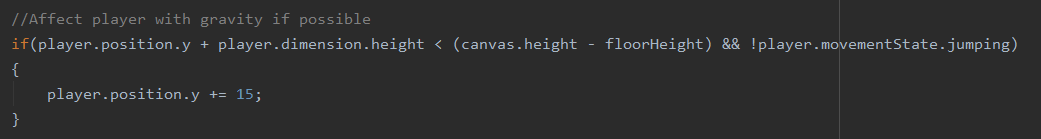
When the player is jumping, if they user is using the browser, the player’s jump height will be the half of the browser’s height which when testing was a good height to have.

However, when using half the canvas height on mobile, there was not enough height to damage enemies so a y-value of 100 was set as the jump height as this allowed better height in the jump and allowed the user the same height freedom as the browsers.

To make the player jump, their y-velocity is altered each frame until they reach their peak in which case their jumping movement state is set to false and the falling state is set to true.

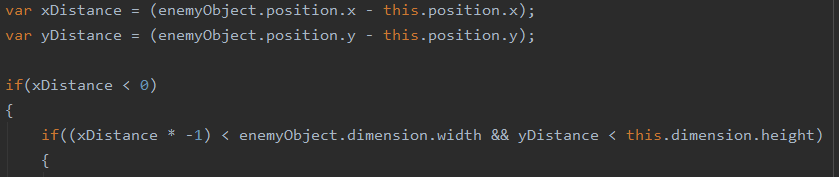


When the player is falling, their y-velocity is altered in the opposite direction as the jumping does. This makes the player fall down until they reach the floor.



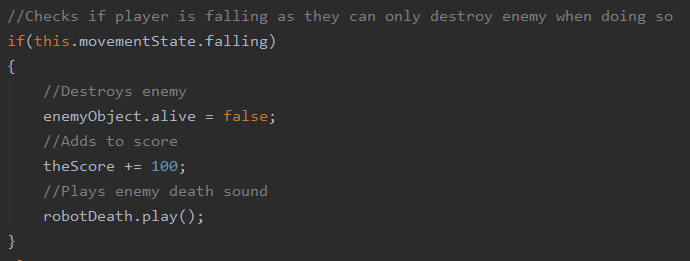
To check for collisions between the player and the enemy, the distance between their x-positions and y-positions are calculated and if they are smaller than the width of the enemy or player then a collision will have occurred.

The collision detection also checks if the distance is positive or negative which allows the method to know which side of the enemy the player is on.



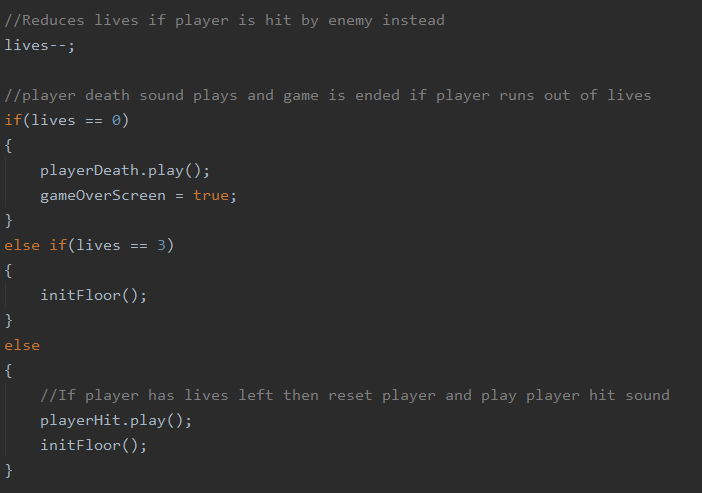
When a collision is detected, the method first checks if the player is in it’s falling state as the player can only destroy the enemy by jumping on their head.

If this is the case, then the enemy will be destroyed, the players score will increase and a sound will play to give audible feedback to the player.

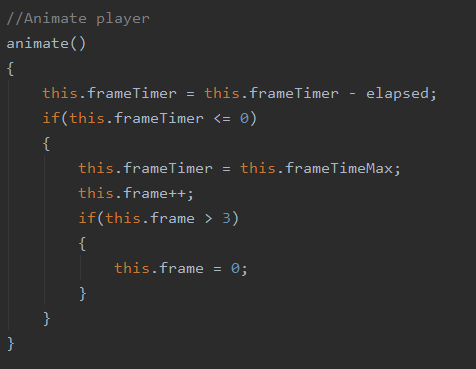


However, if the player is not falling, then that would mean the player has run into the enemy which will cause the player to lose a life. When this happens, the method will check if the player has run out of lives. If this is the case, then audible feedback will play and the game over screen will appear.

If the player still has some lives remaining, then a different audible feedback will play and then the player will be reset back to the start point.



To animate the player, the frame timer is decreased by the delta time calculated in DungeonScroller.js. When the frame timer reaches 0, the animation frame will change, and the timer will reset. The frame is linked to the render function which will change the sprite to render based on which frame is within the variable, therefore animating the player.

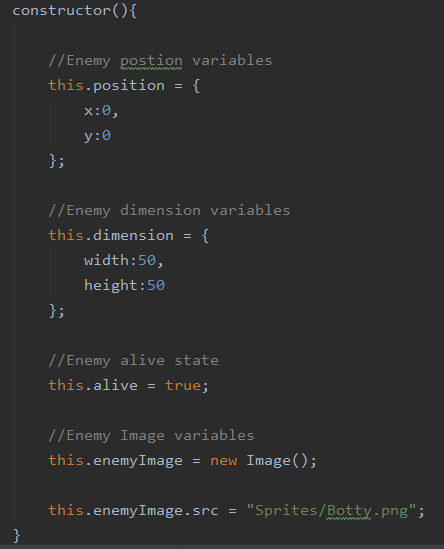


# Enemy.js

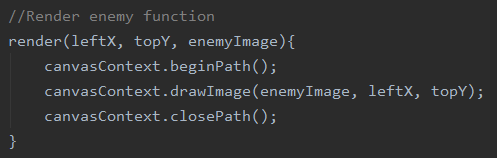
The enemy class holds all the necessary data for the enemies used within the game.

The constructor creates the entity and stores all the variables within it.

The values needed for the enemies are the positions, dimensions, whether the enemy is alive and the images used to render them.



The enemy class also holds its own render function to be called when needing rendered in.



# References

Enemy death sound - <https://freesound.org/people/FiveBrosStopMosYT/sounds/539602/>

The music within the game - <https://freesound.org/people/BloodPixel/sounds/509938/>

Code :-

<https://stackoverflow.com/>

<https://www.sitepoint.com/>

<https://www.w3schools.com/>

<https://www.codegrepper.com/>

<https://www.cssscript.com/onscreen-joystick/>

<https://gamedev.stackexchange.com/>