

Mobile Games Development 1

Coursework 1.

Code Explanation.

James O’Callaghan.

s1920031

# Declaration

*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*.

*Signature*. James O’Callaghan.

James O’Callaghan | Mobile Games Development 1 | 16/01/2021

Contents

[GitHub Repo. 2](#_Toc61923295)

[Trippz.js 2](#_Toc61923296)

[Input.js 7](#_Toc61923297)

[gameLoop.js 9](#_Toc61923298)

[GAMESTATE 9](#_Toc61923299)

[Case zero. 10](#_Toc61923300)

[Case one. 11](#_Toc61923301)

[Case Two 18](#_Toc61923302)

[Case Three 18](#_Toc61923303)

[TextObj.js 19](#_Toc61923304)

[Vector2.js 20](#_Toc61923305)

[Camera.js 20](#_Toc61923306)

[gameObject.js 22](#_Toc61923307)

[Player.js 27](#_Toc61923308)

[Enemy.js 28](#_Toc61923309)

[Button.js 30](#_Toc61923310)

[AnalogStick.js 30](#_Toc61923311)

[Index.HTML 31](#_Toc61923312)

[Refrences. 32](#_Toc61923313)

# GitHub Repo.

Below you can find the link to the repository stored on GitHub.

GitHub Repo Link

https://github.com/PsyberGames/MGD

GitHubDesktop.

x-github-client://openRepo/https://github.com/PsyberGames/MGD

GitHubZip Link

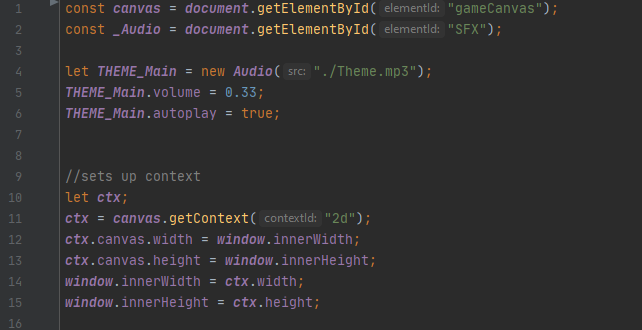
<https://github.com/PsyberGames/MGD/archive/main.zip>

Clone HTTPS

https://github.com/PsyberGames/MGD.git

# Trippz.js

The trippz.js file is our main initializer and is what declares and sets up all our variables for the game so they can be used within the game loop as well allowing input to check for player input and these are two of the functions that are called within trippz.js. We will explain these functions within their respective section of the code explanation but just be aware these give our game most of its functionality. But before getting to that let’s look at the variables we set up and why.



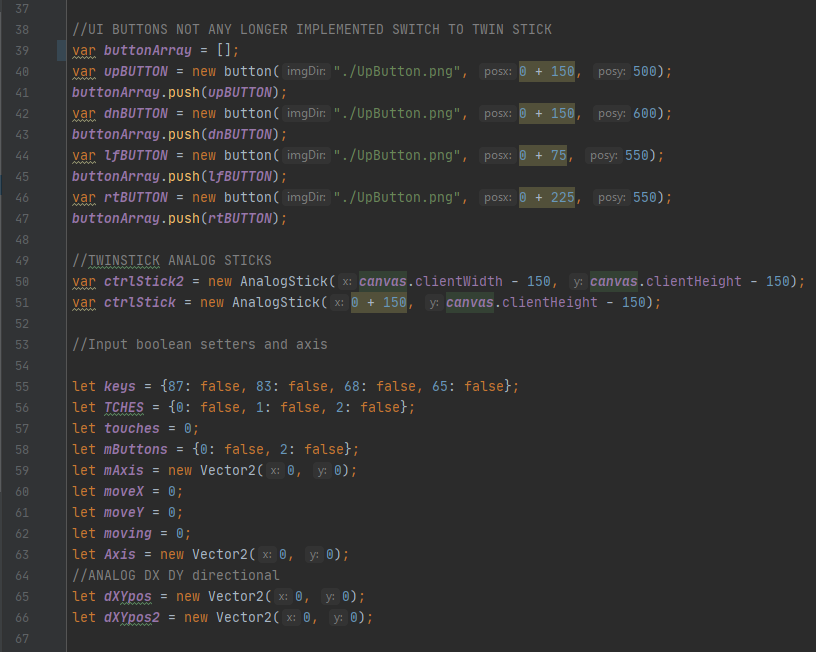
From this image we can see the beginning of our initialization and we begin by setting to const one for the canvas in which we will be using to allow render the game objects onto canvas so can be displayed on screen. Which is handled with our Camera object that has its own class we will be look at later. Then we have \_Audio which is how we add audio to the browser window and these are declared first since everything else after this point will make use of these to compute the combinations of functions to give the player the output from the game.

Now with those in place we set Theme\_Main to be equal a new audio source which allows us to play this track and if we had not set up \_Audio then this would not have been possible since we would have no target location to which we could play our track as Audio(with a directory to file) is a function built into HTML which allows us to load tracks directly from JavaScript. We then set the volume for more of pleasant audio experience rather than an explosion of sound going off in the players ears, furthermore we set this to auto play which allows the audio to play as soon as possible.

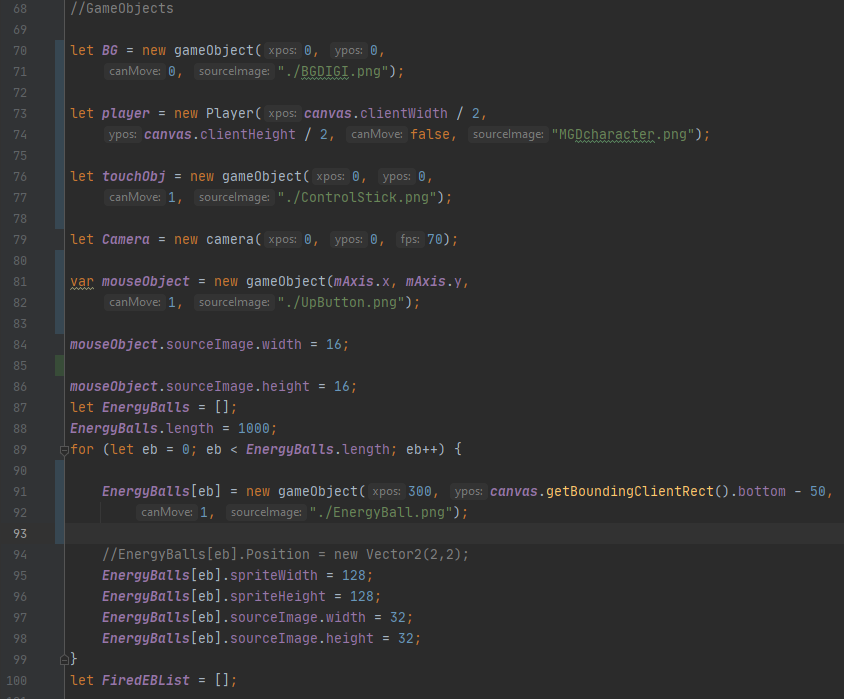
We then move on to context which is actually setting up our canvas to contain like a 2D world in which we can render our game object to screen to allowing the player to visually see them .We set up the context and once we have then with add constraints to the context to allow for the context to be scaled dependent of the play screen size to allow delivery to mobile device not all device have been tested but some device have already had the game displaying on the devices but there has to be more testing to have the game display correctly over multiple devices.



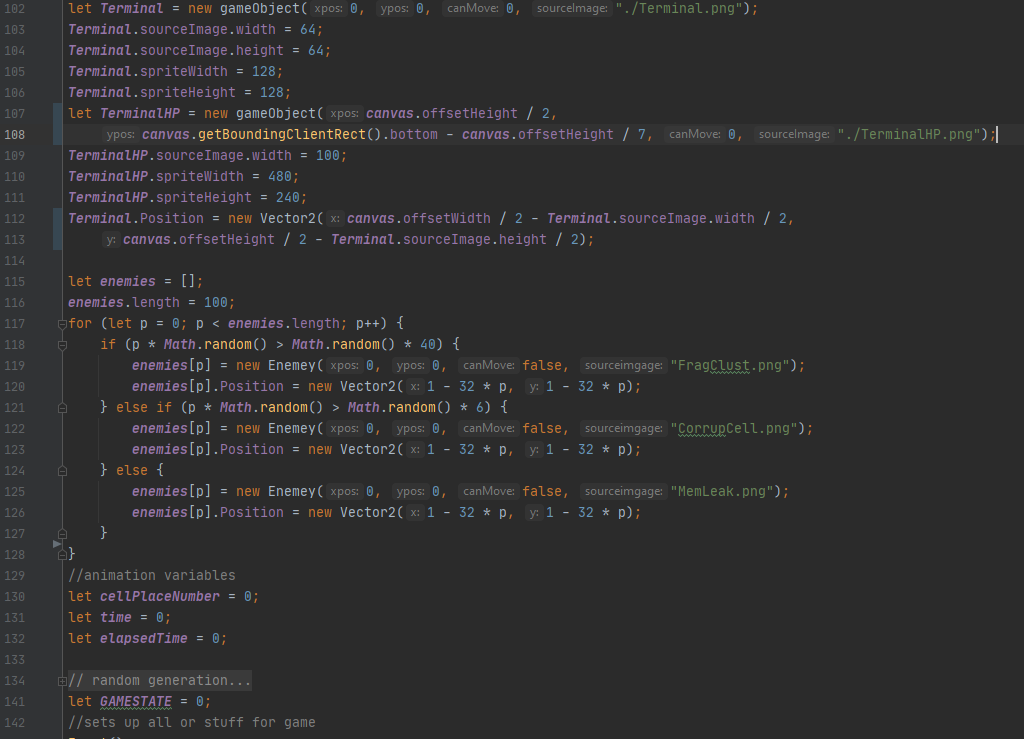
At which point we move onto boundaries for the game so the Player can not leave the screen add this is achieved by creating 4 Game object that actually get moved into place at the left, right top and bottom of the screen with a width and height set which actually correspond with the Background in the game given the player the illusion they are contain within a specific area. Though these objects do not render to screen these are purely declared and used for collision for the player can not leave the screen space. At which stage we declare hSText and tHpText which are UI elements to display text these have their own class that get discussed further in our document. HsText deals with displaying HighScore which is just score but bad choice when it came to naming convention with this one. tHpText which displays the health of the terminal which is what the player needs to prevent the enemies from getting to.



After which we begin declaring our GUI with the touch screen buttons, buttonArray hold all the button we declare like Up, Down, Left and Right which are create new object of type button which is link via HTML to our button.js that is just a small class but let move on to the next virtual controls. After the button declaring and pushing to the array we have the twin stick Analog stick which ended up being the final version that was left implemented to the game thought the directional buttons where left incase in future could use those buttons for something else other than direction. For example, could become buttons that could open a menu, perhaps and inventory or weapon upgrade there several possibilities that could be considered. Leading on from there we have the input Boolean setters and axis declaring we will briefly skim over since we will explain it further once we get into the input.js file. Keys are an array at set value that contain Booleans these Booleans is what switch when the player press a keyboard and for the specific keycode listed seen in the image above. TCHES again similar to the keyboard with the array setup with value containing Booleans that correspond to touches on screen limiting to the max of three, touches is the total number of touches and allows us to limit the touches to be matching the array so we do not get any errors. Then we have mButtons which is again similar set up as previous arrays but this time dealing with the mouse button inputs, then that leave mAxis which sets up the mouse axis as a Vector2 which is class created to deal with position with few other function built in allowing calculations of the Vector2’s again the explanation will be below. MoveX and MoveY is to allow the player input to be dealt with as axis-based system allowing us to implement the player movement that currently in place. Leading us into moving which is a Boolean that allows for us to check and see if the player is moving and this is made use of in the gameLoop.js. Axis is very much the Axis that becomes the player movement velocity that allows the player to move across our canvas. Then finally at the end of this section which again set up another set of axis’s this time it the DXYpos and DXYpos2 both are used to deal with movement of the virtual controllers and store the direction of the stick as a Vector2. The next section is where we start to declare our game object and is little longer than previous so we will split into to images to make easier to read.



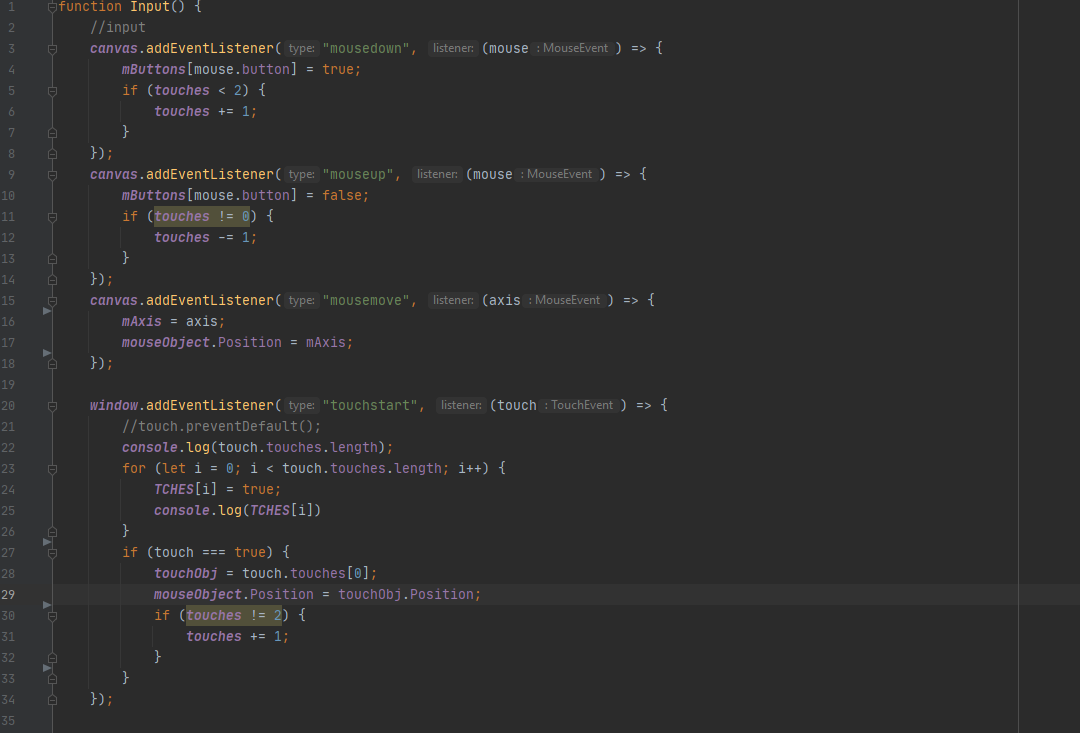
Moving into the actual object that will become our game object that are given functionality to give our game the intractability for the user. And the top and rarely used but still a very important part of our declarations is BG the holds a new game object using gameObject class, that store the information for us to draw our background in order to allow for it to be render to screen. Next we have player that again is it own class called Player but that is actually an extended class and extends gameObject so the initial constructors look a little different when doing this and we will address that upon reaching some class that make use of this. On line seventy-six we have touchObj which is another gameObject but this time this is not render to the screen and allows for us to detect collision with the virtual controllers that has been implemented for use in android devices. Camera now this could have been another class as the name could be misleading since this is what is better known as a renderer class because this is what handles the drawing of the game object to our canvas. That bring us onto mouseObject which is very similar to touchObj but unluck the touch object we do want to render this one to screen to help the player to indicate the direction in which the player wishes to attack. And leading on from the idea of attacking this when we declare EnergyBalls which is an array that will be array of one thousand object initially just a test to see how many could be generated ended up being left in as proof that even with a JavaScript browser game that dealing with thousands of objects is achievable and this why energyBalls never got there own extension of gameObject since this would not allow me to explore multiple techniques of developing with JavaScript. We iterate though the array adding a new gameObject for each entry in EnergyBalls. Finally, we have FiredEBList which is an array that deals with the fired EnergyBalls since the object are pooled and displayed on screen as and ammunition indicator. When they are fired, they move out of the EnergyBalls and into the FiredEBList and we can further explore this upon coming across it in gameLoop.



Moving on we have Terminal and TerminalHp one is a gameObject dealing with the physical object drawn to the screen the player can interact with the other a visual UI element to display the Terminal Signal Health which is what the player is directed to maintain while defeating enemies to increase there high score. The UI for the TerminalHP to decrease based of the player.Healthpool which directly relative to the Terminal Signal Health which in essence is the same. Then it the declaring of enemies using similar techniques as before with declare and array to hold our enemies give it a length and fill the with object type of Enemies which like the Player extends from gameObject. Then what is a game without animation so we declare CellPlaceNumber that deals with the cell number at location of given sprite sheet, time which is what we use to actually time our animations have them display at correct speeds on the player display and then we have elapsedTime as it seem store our game total elapsed time. Finally and one the key variables that really gives the game functionality to handle multiple loop set up so you can have loop dealing with each individual state of your game for example main menu, start, game over quit and pause these are all game state that could potentially declare a change of drawing to the screen and what should be drawn depending on the state of the game.

# Input.js

With all our initial variables set up and initialized then we make a call to our input function for the first which is setup in its own file to allow for easier readability. This file is basically our input function that could have resided else where but as stated previously I wanted to really experiment and see what could be done with JavaScript and the potentiality of the programming language and how it seems to function. As once we have explained one then the events with input are same just set up per event that we wish to call so let look at the image here first.



This is only the beginning of the Input function but as stated it would be pointless to display the whole function rather than explain the function and then explain what event functions are add to our browser window. So, the Element.addEventListener is a HTML function and take a two parameter as seen here. The first is the type of Event we wish to add, and the next parameter is what input we wish to listen to in order to trigger the event. We add this to our elements of choice as you can see here we have it declare in canvas along with window which is two completely different elements but one is contained within the other so for testing purposes and eventually left to clarify findings even though elements are different that make know difference when applying functionality to the game. Although it can create issue if not managed correctly because let say get mouse position if it was a call to the window and not the canvas then your mouse position would be relative to the entire window of the browser, mean while if you declare the canvas to add the event listener to then your mouse position will be relative to your canvas. Didn’t expect to find this but after finding that could potentially because problem thought documenting about it to be a wise decision for future work. So, with that all explained let look at the image little more and what type have we declared and what those specifically do then finish up by informing what other type are contained within the function. Firstly we have ‘mousedown’ and that is the event handler that listens to see if there has been a button on the mouse pressed down, if it does trigger then we can use previous variables declared in Trippz.js mButtons to allow us to set which button is being pressed down and change the Boolean accordingly, straight we have ‘mouseup’ which does the complete opposite from the ‘mousedown’. ‘mousemove’ Is next and this tracks the position of the mouse as stated further up we declared this on our canvas because we want the mouse position relative to the canvas which is our 2D context that we use to render our game world using Camera. There several other event listeners added like ‘touchstart’, ‘touchmove’ and ‘touchend’ these all allow for simulation of touch should the user being interacting using a touch screen or another touch input, allowing the game to have functionality for the mobile market. Lastly within the function we have ‘keyup’ and ‘keydown’ which handle our keyboard input and deal with W, A , S and D that relative to Keys at keycode eighty-seven is equal to W, keycode eighty-three is equal to S, A is equal to keycode sixty-five and D is key code sixty-eight all initialized as false and trigger true should player press key down and false again should they release the key giving the ability to have keyboard function within our gameLoop which we can take a look at now.

# gameLoop.js

This the largest of any of the other components already touched upon but once we have explained how gameLoop function works then the explanation of the classes after that are generally short and efficient. And given the size of this function we will again split it over multiple stages, but one factor is we could split this into sub-sections and describe game states and how each game state changes the function of the game loop.

## GAMESTATE

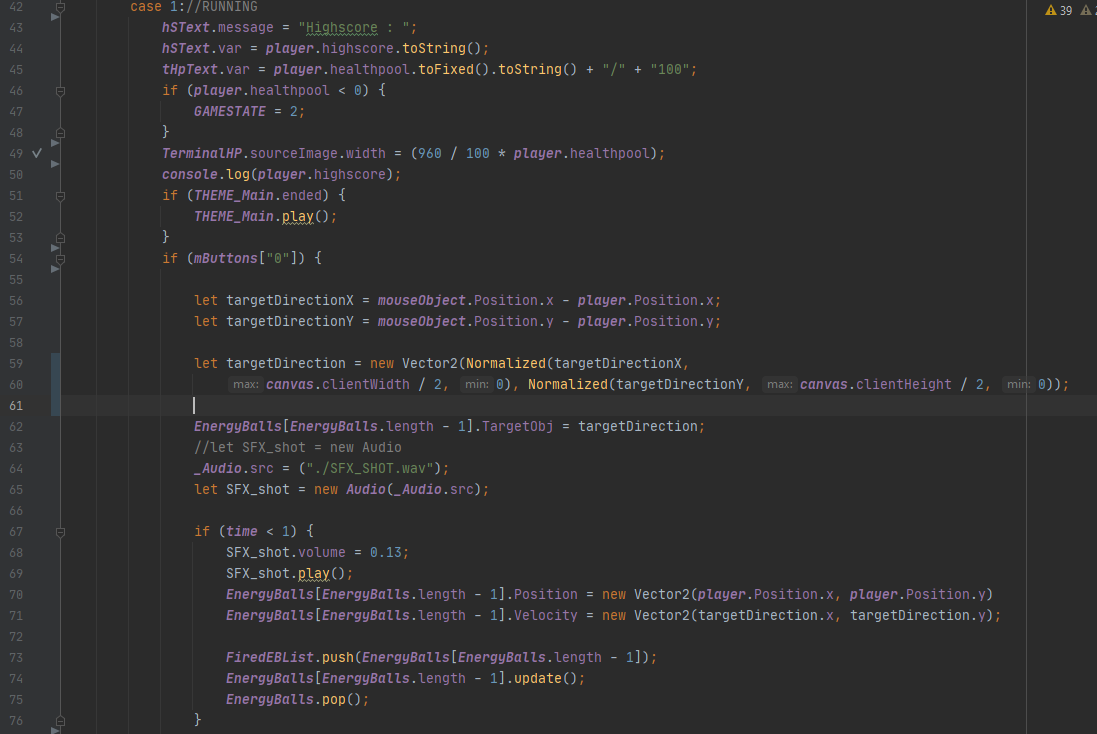
GAMESTATE is handled by our chosen integers and those we will explain later and which each value corresponds to which GAMESTATE. This is all handle through a switch case and the case we are checking against for a change is the GAMESTATE then we do what ever game state is and the gameLoop then is dynamic and allows for multiple states of game to be computed. Below we can explain each of the values and the corresponding function that it runs during game loop.

### Case zero.

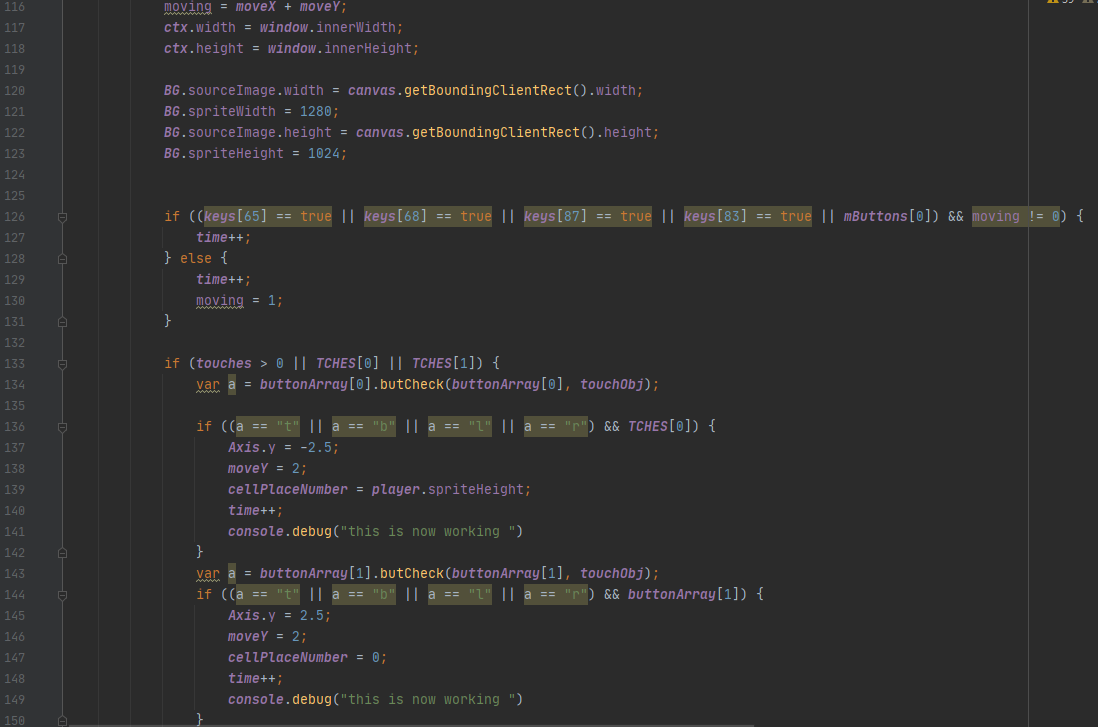


Above we can see case zero and the implementation of code within this state. We maintain the display to begin, then we change the BG.sourceImage.src which is what file we wish to use to display our BG remember BG is our background variable that allows us to render the background to screen for user interaction. By changing the src directory we give the ability to have the renderer to render the corresponding background depending on the gameState potentially this could even be used to progress through a game world which exceed the given screen size with the size of the world and each section of the case would deal with a specific section of the game world. And would render the state appropriately to the image that is appropriate to the specific scene. After all the background has been change accordingly we tell this state of the game loop to begin by check if the mButton[0] which is left mouse button is pressed down and if it is then play the main theme music that we declared previous in Trippz.js. Then because the mouse has been click music is going to begin to play and we want to change the background image src again but this time we are changing it to the running state of the game background which displays the game world for the player to be render to and changes the gamestate to correspond to one excuse the console log I humor myself during development to increase productivity. That is if the mouse button is pressed but if the mouse button is not pressed then it tell the Camera that it still in start and calls Camera.Start which basically renders start we can look at later when explaining the Camera class for now just now we call Camera.Start() because we are still in state of state which is equal zero. After we call window.requestAnimationFrame(gameLoop), which is used to continue to loop the gameLoop function change the game render depending of GAMESTATE. Next, we have case one and this is our running state for our game that runs most of the game functionality and allows the player interaction with the game world so this next state will be split into section to help explanation of the code.

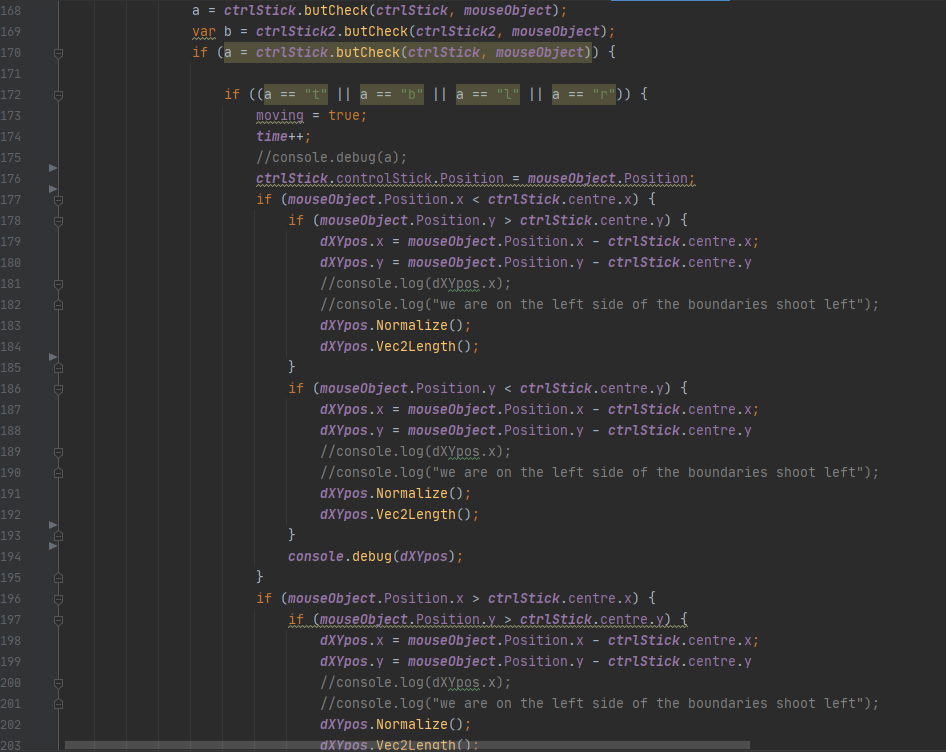
### Case one.



To begin right at the top of GAMESTATE when equaling one we has hSText variable being used to display a UI element for high score and set the variable accordingly as seen in the image above and this is a TextObj and this allow use to draw game UI to screen and if we choose then we could even give it interact ability but this will make more sense upon looking into the TextObj.js file later for now lets focus on case one since it massive. After declaring set up for the UI gameObjects we do the first if check of state one and that correspond to potential state change, and this if the player.Healthpool is less than zero then the player is dead and should initiate GAMESTATE to now equal to changing the game state to be equal to the game over state and value of two. Though if the not less than zero then continue into the rest of the gameloop for case zero. Now making the UI element for the terminalHP to be equal to width of sprite divided by one hundred multiplied by player.Healthpool to allow for the UI element to simulate the decrease of the UI indicator for the terminalHP. For all the function that are self-explanatory we will mention them but not go into depth too much to allow for more explanation to better expanded upon. For example, like next function if the theme music end then replay it, moving on to the mButton[0] and if it pressed down the shoot projectile function now this, we will explain a little more than the replay audio. We declare two variables here to be local and this allows for tracking of the mouseObject position relative to the player and we need to have both the x and y respectively. After we declare another local let to be targetDirection and this equal to new Vector2 with normalized targetDirectionX and normalized DirectionY with base being and centre of the canvas. This Given the direction of the mouse based from the direction of the player, allowing the player to fire projectile in three hundred and sixty degrees and by grabbing the last element in our EnergyBalls array we gain access to one of the game object that we declare in trippz.js that deals with the energyBalls and the projectiles for the player. At line sixty-two you can see making use of that and changing the projectiles TargetObj that we will explain a little later but it the object at which the player is point specifically the mouse position on screen. When we fire a shot we want to play a sound but in order to play the sound and have the sounds playing correctly we only play the shoot sound at the beginning of every time when time is equal to the frames of the animations for the game so while the game frame are less than 1 so at the start then button mouse left is down we will play a shot sound effect and fire a projectile in the given direction of the mouse position. Removing the projectile from EnergyBalls array by popping from the array and pushing it to the FiredEBList array this allows us to make sure we do not fire a bullet that has just been fired and act as you expect a game to act each shot fired pulls from our object pool and fires adds them to pool the deals with the active gameobject and they then become de-active after hitting an enemy or when they leave the screen space at which point they return back to the array dealing with inactive object ready to be fired.

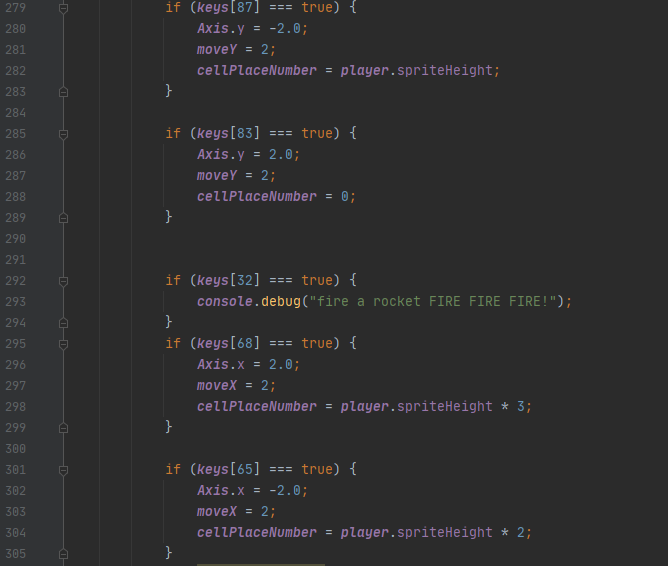


Next section with the case seen in the above image we go and give moving that was originally a Boolean thought which now take the MoveX add the MoveY and this allows us to check if we are moving we don’t do nothing with as of yet we just make sure it ready to use. Allow the BG which is the background to remain relative to the user screen which step in a if statement that does a lot of this or this or that check against the keys corresponding to the keyboard input keycodes and if you are true then increase time if not also increase time and fake movement for animation purposes. Next is what handle the touch input for our game uses touches to check and see if greater than zero if TCHES[0] or TCHES[1] is triggered then check and see if any of the touches are colliding with our virtual controls there are the buttons array at first which deals with original directional movement of the play but decided to remove and replace with the virtual joystick which seem a lot more robust and more modern look to the game interaction. Although for explanation purposes left the original code in since the direction was also explored but moving on and we can take a quick look at the image below that specifically deals with the virtual joystick call within gameLoop.



This section we can see begin by declaring two variable a and b which are variable holding the information return back from our collision call with butCheck which check at which direction you make collision with and then once the collision is true we step in to set the control stick position. The control stick position for the left controller to be more explicit this where we make use of our DXYpos and we normalize the position and then grab the length of it so we can make the calculations to get the location relative to the position of the player from the position of the touch on screen based on the direction of the controller that follows the position of our touch object that is never rendered but detects collisions for our touch functions. And if we then release to touch reset the control stick position and reset the relative DXYpos for the specific control in use. Control stick one being the left controller allows player to move omni-directional in the game world. While the collision for the right controller stick which is controller two will in the future allow for the player to fire in the given direction of the controller stick two position and this will be relative to the player given similar directional fire as the mouse does.

Moving on we can now have look at the keyboard input and how we handle that within case one have look at the image below before we begin to explain what is going on.



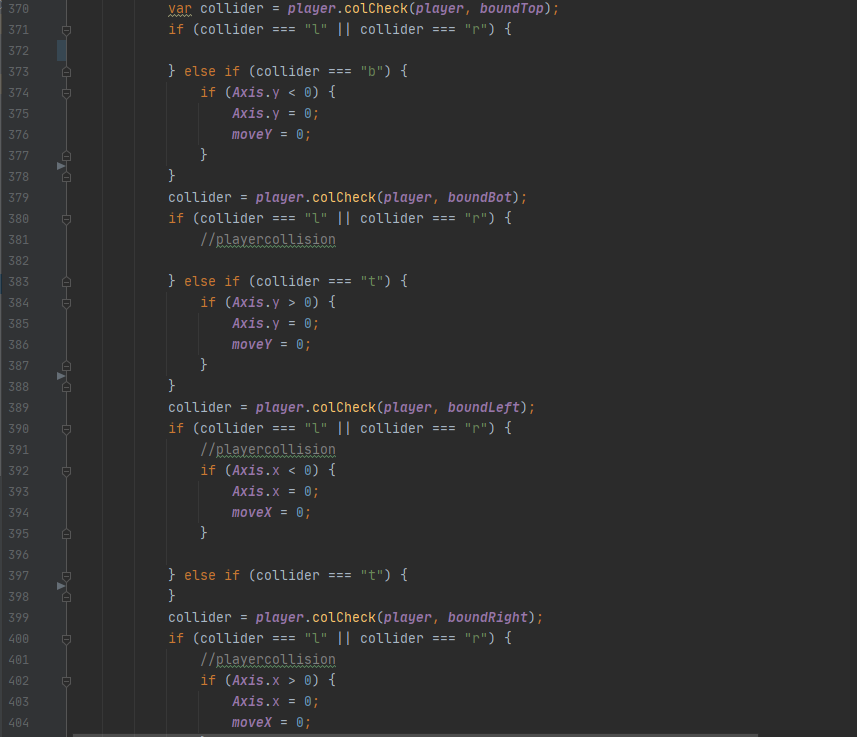
This isn’t to bad to explain so from what can be seen from the image above we check each of the keycode values within keys and should they be true then we make the relative changes to Axis and MoveX or MoveY depending on the direction of the movement. At which point we set the cellPlaceNumber for specific animation that will become more easier to understand once we look at the player and gameObject classes. But for now, it easy just to know that this deals with the cell location for the first frame of the animation specific in a directional manner to render the player to look like he rotates as he moves.



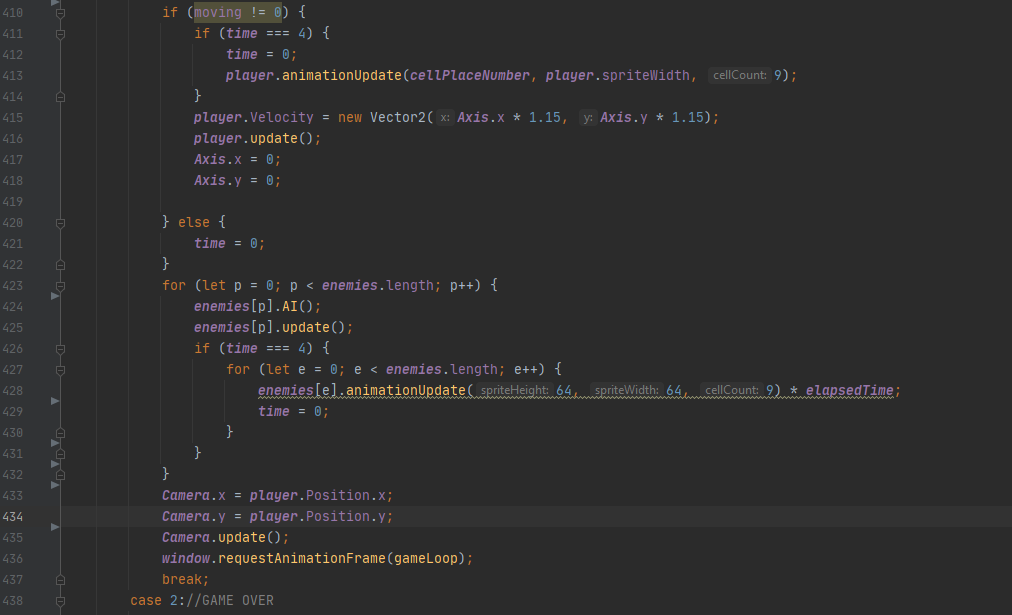
From the image above we can see that at this point of gameLoop we are dealing with the collisions for the terminal with the player and with the enemies. To make a check with the player to determine if the player is colliding with the terminal game object and if it does then we want to freeze the movement of the player this collision on terminal isn’t specific great it is a really basic collision and is buggy during game play I have left it again as I want to speak about how this took me ages to locate this specific bug and change everything else before actually find if we get top collision freeze the negative side of the y and opposite for the bottom and then with left right being similar though on the x axis it is handle soon with the boundaries of the game. Though felt I should point out how many could overlook such small problem which in turn could lead to a lot of days debugging. Then onto the enemies and what happens when they collide with the terminal well, they make a call to a function within gameLoop.js called terminalHit() and can see that on the image below.



It is a simple function and removes point one from the player.healthpool which what controls the Terminal Signal Ui element. Though it is a fraction here once applied to our player variable and it has passed through the math’s then it removes one for each enemy successful in reaching the terminal. After which we check for the player and if he tries to leave the play area boundaries and we can have look at the image below and then explanation will become more apparent.



And as seen we are taking the player and running a collision check for each of the boundaries surrounding the player area and checking each correspondently. So, if top boundary checks the bottom for collision and only allow movement in the opposite direction, with the bottom boundary it the complete opposite. If we take the right boundary and the player hit it, we check and if it true that the player object is colliding only allow movement in the opposite direction similar the same for the left boundary but in the opposing manner. Finally, this next image is the end of our GAMESTATE equal to one.

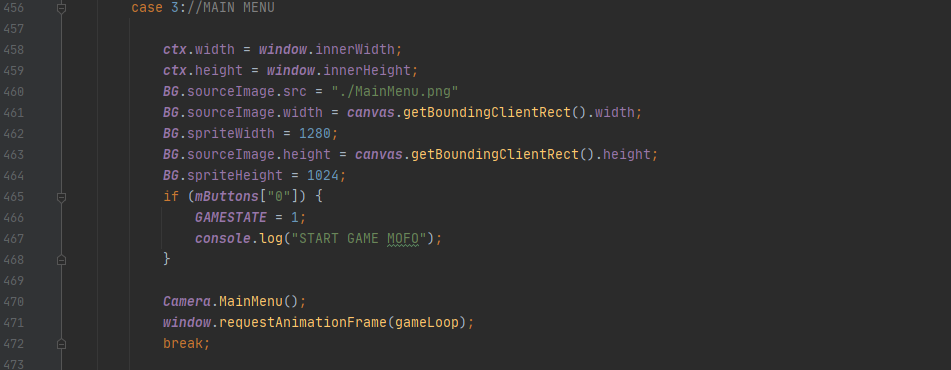


So, in this section it is the last section out of the state, and this is we handle a lot of the movement that goes on from the players or the enemies. As seen if moving is not zero then we assume we are moving, and we must do something. Well if time reaches four then set time zero and update the player animation update the render to screen of the player. After we take care of time, we set the player Velocity to new vector using the previous Vector2 Axis that was declared in Trippz.js and which then has the varying value applied when input is pressed on keyboard or touch control are active. Then we update the player through the players gameObject update and reset the Axis to finally be zero again because this is the end of the game loop before we re-render the screen and apply movement so reset the Axis so it can go through the same process again while the GAMESTATE remains the same little like a while loop but not! Then if not moving irrespectively of the value of time set back to zero to begin the animation time again since we use time to cycle through our sprite sheet cell locations. After we use a for loop to do similar actions for the enemies but because we don’t have input to deliver the movement to the enemies the enemy class hold really basic Ai that target the Terminal and only the terminal paths to the terminal and then what seem to the player like enemy dies or deals damage to the terminal and disintegrates off screen assimilation to the terminal system and corruption of the core. But really, we do little magic inside the enemies’ class itself and we can touch on that when we get to the enemy class. After doing all the enemy AI calls and enemy update calls check time like the player animate the enemy and reset the time to zero. We set the camera x and y to the player position x and y then update the camera which renders our context to screen respective of the screen. Once we have finished all we make a HTML window call similar to the previous state last line requesting animation frame which act like render update and calls loop again. Now from the end line we can se we can assume Game Over is GAMESTATE equal to two, meaning case three must be for Main Menu we can see both case states below respective of case number but the do the same as state zero but the camera calls the state specific renderer function display the states correctly.

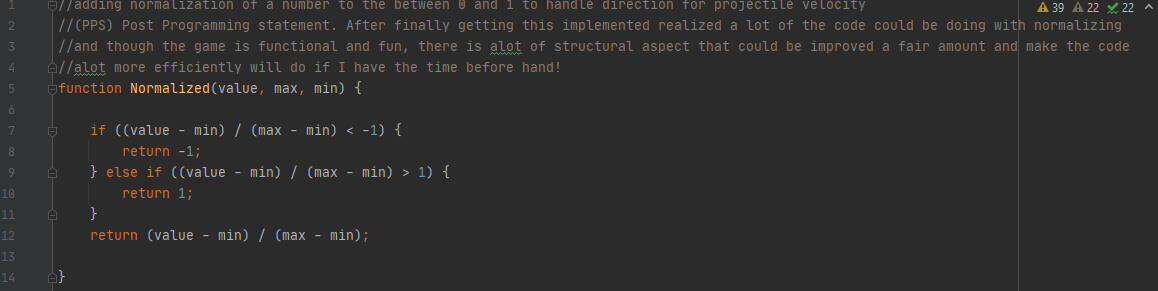
### Case Two



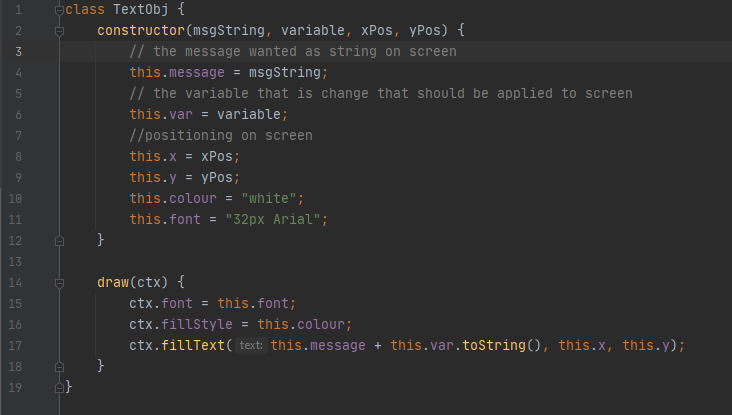
### Case Three



And that ties up the GAMESTATE and how implementation was achieved with JavaScript, not quite finished with GameLoop.js yet though since there is another little function that is part of gameLoop.js but not a actually function running within game loop unless needed and that is Normalize(). And due the intensive comment already within comment we will finish with gameLoop.js with this final image of the Normalize function.

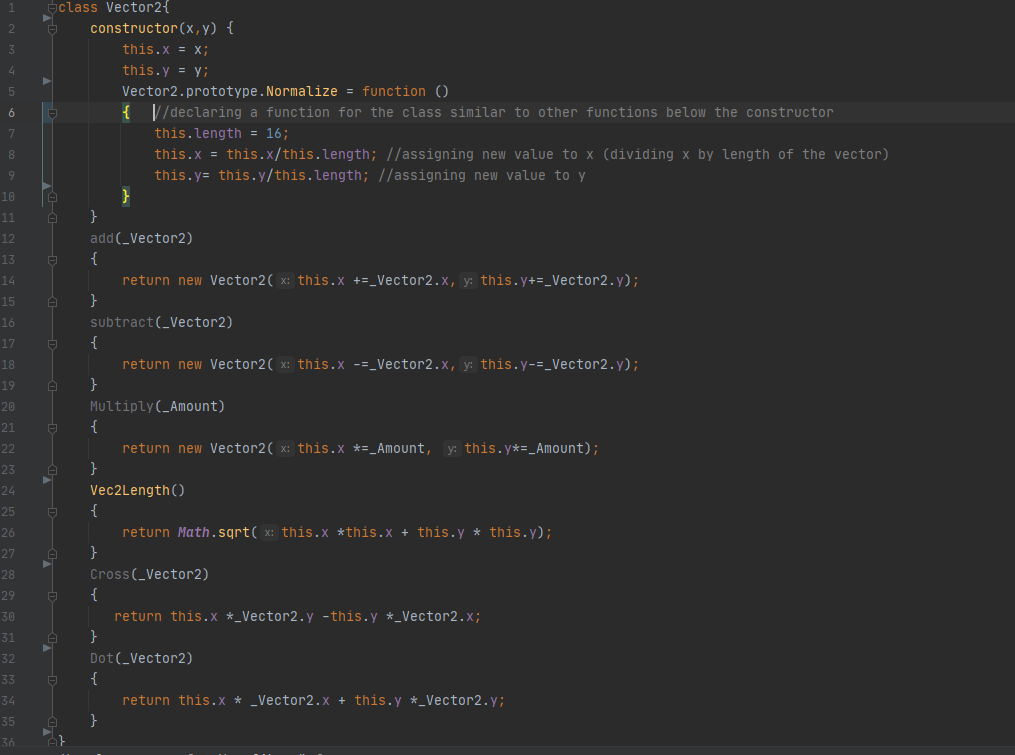


# TextObj.js



This is the class the allows for creation of UI gameObject to render them to the screen after passing in the particular variables to the constructor parameters we setup the variable for Text object respectively via this dot function and then the variable name for the class. After which we leave the constructor happily set up the corresponding variables to allow for renderings text objects to screen. At this point you see we declare a new function within the TextObj class and call it draw and that take a context as a parameter with corresponds back to the context we declared back in the trippz.js. And by using the context make use of the local variables and set the font the colour and fill the context with specific text and x and y location with string containing the information to be displayed to screen.

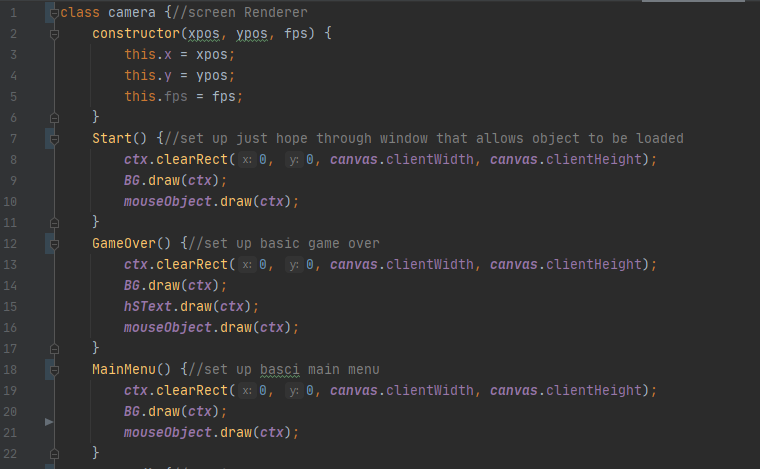
# Vector2.js



Another of the classes is the Vector2 class which is as stated give our game the ability to have x and y locate under on variable allowing us to implement some of the movement code that other wise would have been less smooth. Not much else to it has an x and a y and it also has access to few function explicit to a Vector2 like Normalize a vector two, add vector 2 to the local vector two, subtract, amount, Vec2Length giving the length of a vector two particular useful when dealing with directions and velocity. The cross product of vector two and the dot product of a vector 2 most of the function implemented are not made use of but were implemented for future use. And that concludes the Vector2.js file allowing us to move onto Camera.js.

# Camera.js

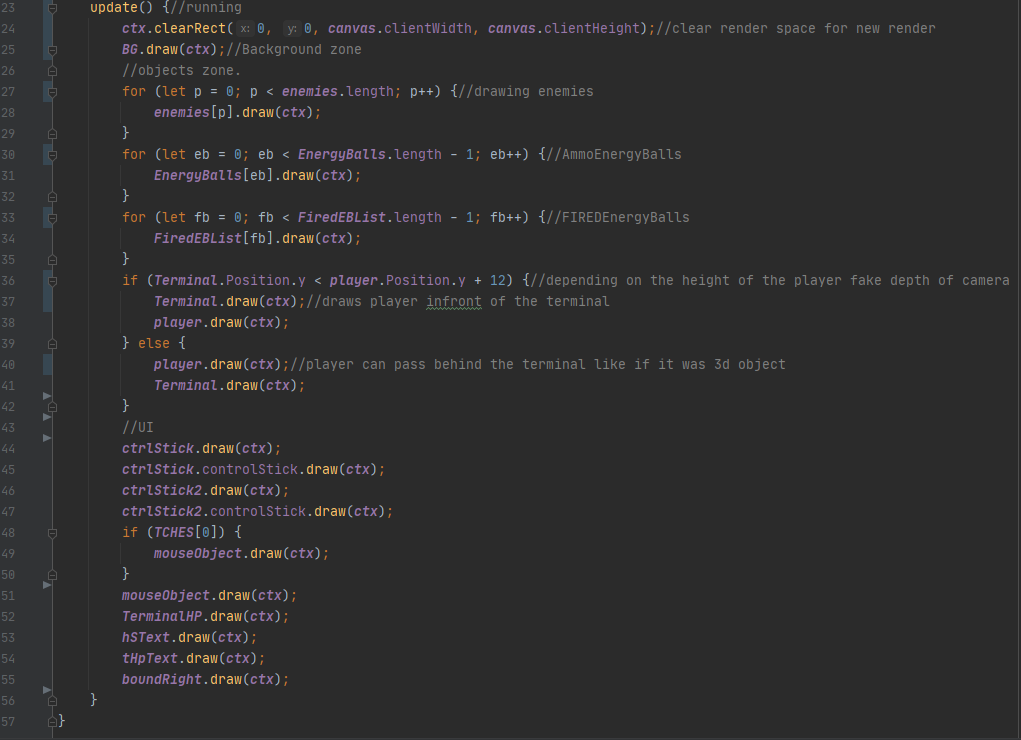
With the Camera.js class slightly more than the name implies this class handles our is really our renderer and which render our game objects to screen have a look at the first part below.



As per the image displays this class set up x and y and fps which is declared but never used as soon of the advance mechanics planned were still not fully implemented for example there was going to be a specific class that would have dealt with animation and would allow each animation to be independent which is really not one of specified deliverables for this module though originally over scoped the project this was one of the thing that was able to be cut back on since animation were working just not independent per object.

After the construct we see we have function all doing their own render of a specific gameState all doing pretty much same thing clearing the context and redrawing it depending on the specific call be called from gameLoop.js. Though simplistic effective at what it is doing and has some interesting possibilities I plan to explore in the future a good deal further.

This next Image displays the Update function within Camera which is what runs our main Renderer function that updates the running of the gameLoop when GAMESTATE equal one and is considered running.



With all the draw function it call them and draws from bottom up meaning what ever is last in the function call will be first on screen reason being why all the UI is drawn at the bottom of the function since the UI should be above all other drawn game object. And one really smart part of this is the point of drawing the player and terminal and how the player when playing the game looks like he passes behind a object which is actually only 2 dimensional image but because of the height of the player y position we can say that if he is higher and the object is the terminal and it can be consider able to walk behind then we drawn in this fashion to display the possibility to adding depth even for a 2D context. Other than that, pretty much same clear rectangle for render area / draw area and draw the game object respectively.

# gameObject.js

Beginning this class off due the size we going to start the constructor and then explain each function within gameObject separately to allow for a more coherent explanation.



In the constructor we have four parameters coming in and this is the x and y position for the gameObject that allow the renderer to draw the gameobject respectively of the screen space. The next in coming parameter is a Boolean and allows to check in the object should or should not have animation not implemented but the base for the system was. And finally, the sourceImage is the directory of the sprite sheet or image for the specific game object. After declaring within the constructor, we go head and set the base values that allows for animation to be handled further down in the AnimationUpdate() function. After which the base variables are set up for our gameObject and we can move onto the next function within the class called colCheck() which we will explain the function as overview and explain what further is going on with specific checks. Look at this Image below before we move on.

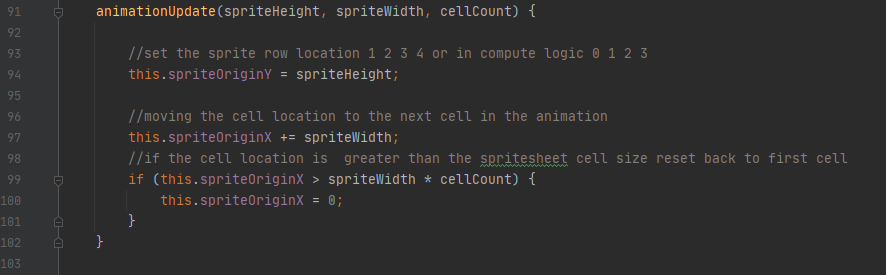


As seen from the image it making use of the spriteWidth and spriteHeight attach to specific objects and check them against the other shape spriteWidth and spriteHeight at the specified division rate that is usually the half extents of the sprite since the measurement for the collision originates at the origin of the object which is actually offset to the top left which corresponds to how canvas draw image from left to right top to bottom. By check these half extents or what we are considering our extent for collision then we compare them to find if the collision are intersecting and if they are from which point did we intersect from and we do this by then testing the origin of the x and y and return the direction of the collision respective of the location based from both origins.

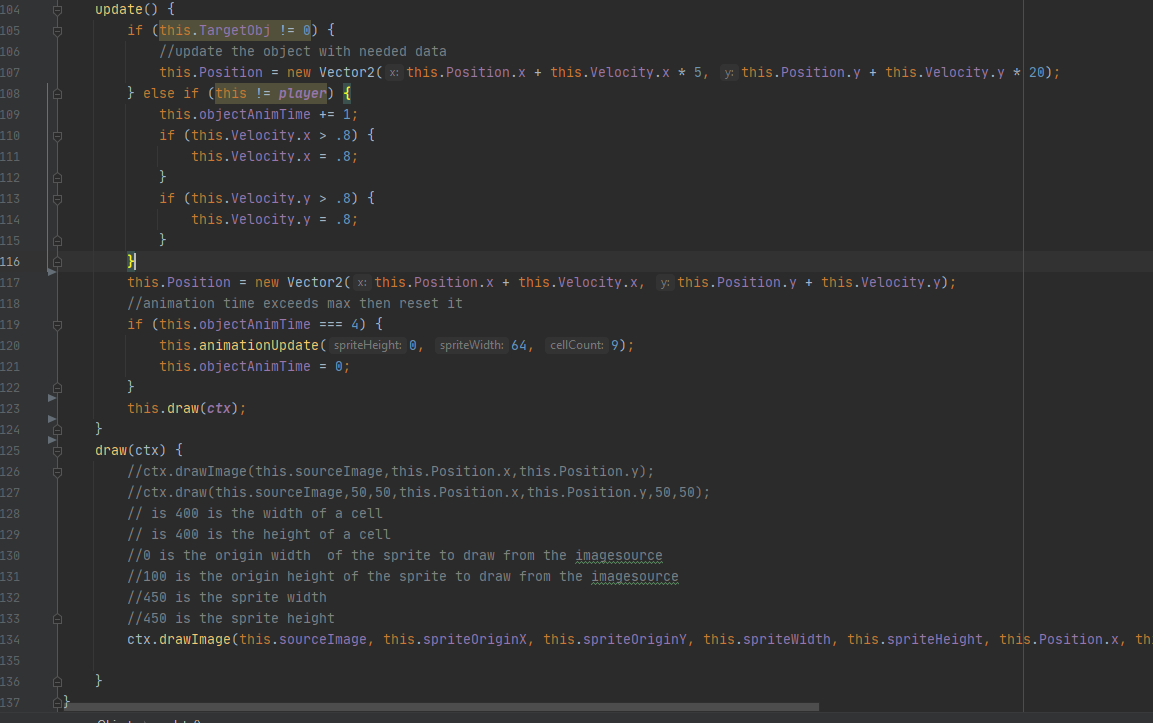
This first is specific when the ShapeA is equal to the terminal and specifically used in the gameLoop.js when the player collides with the terminal and stop the player passing through what was decided would be a solid block. Then have and else which deals with every other object in the game though as stated this is not optimal way to efficiently implement it but they work and as this was an experience exploring JavaScript a lot of the time I got lost experiment in my code far too much. Below we can se the else part of the if statement.



After this it is AnimationUpdate function which allows for the gameObject should we choose to can have Animation or not.

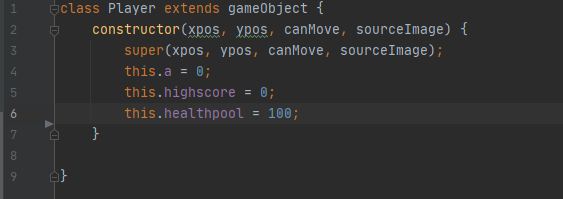


As we see it isn’t a large function at all and allowed the original approach of having a class to deal with animation not being implemented since this so well for the giving requirement of the project. SpriteOriginY is the height at which we can locate the cell of the first frame for the animation. SpriteOrginX is the location of the width on the sprite sheet for the first cell in the animation. Now in order to progress through the animation we want to move the width of sprite origin to equal the origin now plus the sprite width of the gameobject in question. And should we exceed the spriteOriginX by more than the spritewidth multiplied by the cellCount then we reset the spriteOrginX creating a looping animation. The explanation for the code is much more than the code itself but this is due the original set up of gameObject and then from the parameter being passed into the animationUpdate function.



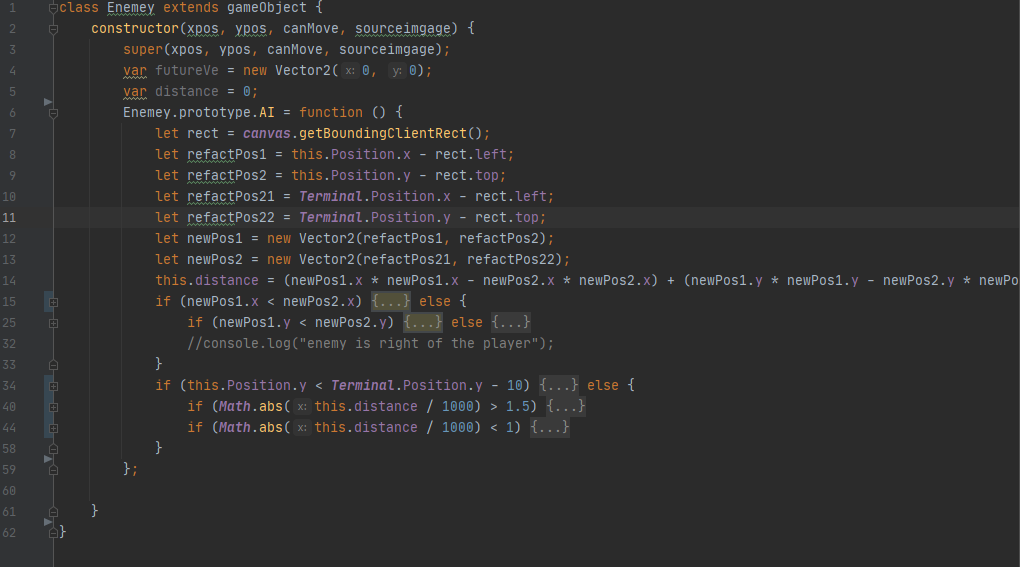
Within the image above we can see the process of updating a game object and basically we got if statement that varies between what is the gameObject decided what should be done for the update this could be move for sure and dealt with within each object own class but was not realized before it was too late. So the update within gameObject deal with which object are which and then if it is a player object animate it with positive one time and if the velocity is greater than the point eight in x or y then it limits the player velocity in simpler sense. Update the position of the object and should the object animationTime ever get great than four reset it. After this is the final execution update and this is the gameobject draw function contain locally per object. Which can be seen in the image as well take ctx which is our 2D context and passes the image to draw with the gameobject store variables, at which is before the Camera update in gameLoop.js which is actually what finally does the rendering to the screen though the image is drawn in the game object it isn’t finally rendered unless declared to draw within the camera.

## Player.js



To be honest I was surprised during this code explanation how short player class seemed to be but that is because a lot of the functionality revolves around the player and though functionality could have been push specifically into the player object it has remained in the gameLoop.js for another discussion point because on the other the following class will be enemies and that as different approach again to setting up the JavaScript class and specifically how we set up the functions within the enemies. But Player as seen extends gameObject which inherits all functionality and variables of gameObject. But allows for us to also add other carriable like this dot a that equal zero, this dot highscore and this dot healthpool which I assume are named efficiently particular for ease of reading. These variables have been touched on already within gameLoop and help display UI TextObj to display the highscore and healthPool. Which give the player the need variables and functions in order to have intractability with the game object and render the appropriately to the screen.

## Enemy.js



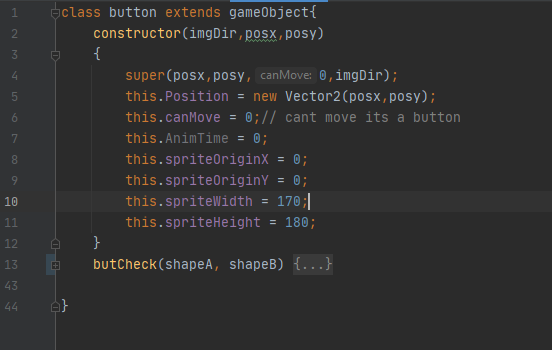
The initial set of enemy is pretty much the same as the player though we declare a function within the constructor of the class by accessing the Enemy being our class and dot to gain access to the JavaScript prototype dot and that allows us to declare a function name in this case is = function() what function the function incase in the brackets. So I purposely closed off the function main body for snippet purposes but we will expand and take look into the function a little closer but first explaining the first seven lines of the function. All the variable declares locally for use per object specifically enemies, rect is the bounding rectangle that the game screen resides. With that we can Triangulate the location and get the distance of the enemy from the target object specifically the Terminal which enemies are trying to attack in order to corrupt the system and crash the core. Now let take a deep look at what happens with newPos1x and newPos2x and same with the newPosy and newPos2y and this where we check to see if the enemies location is above the terminal below the terminal to the left or the right of the terminal to help determine in which direction the enemy needs to move in order to reach the terminal. Further down is specifically the area in which we will expand and have deep look with the image below.



Here is what have as being the movement part of the AI that after determined it location from the Terminal then it can make the decision at which velocity has to be applied for the enemy to reach it destination and just to be explicit this would include the direction of travel due how you work out the velocity of an object.

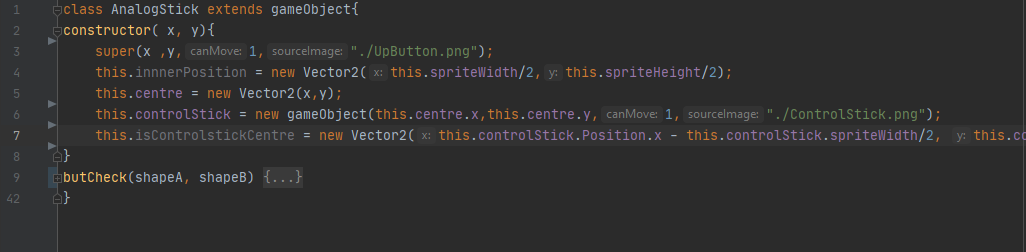
If the object has reached it continue to move towards the terminal upon the point it get close enough and by doing so we move the object of screen and the player loses some health point because the enemies have managed to reach the terminal and cause damage at the core. And the pathing for the enemy I am not even going to begin to explain the math behind it but we want the enemies to path relatively smart but then upon being inline make a bee line for the terminal there it a lot of experimental work and theoretically it can all be explain but in order to do so would excessive so we will leave with so really bizarre math happening that allows the enemies paths to be pretty smooth. But that is the small amount of code added to the Enemy class that work well considering the number line code to achieve the desired effect.

## Button.js



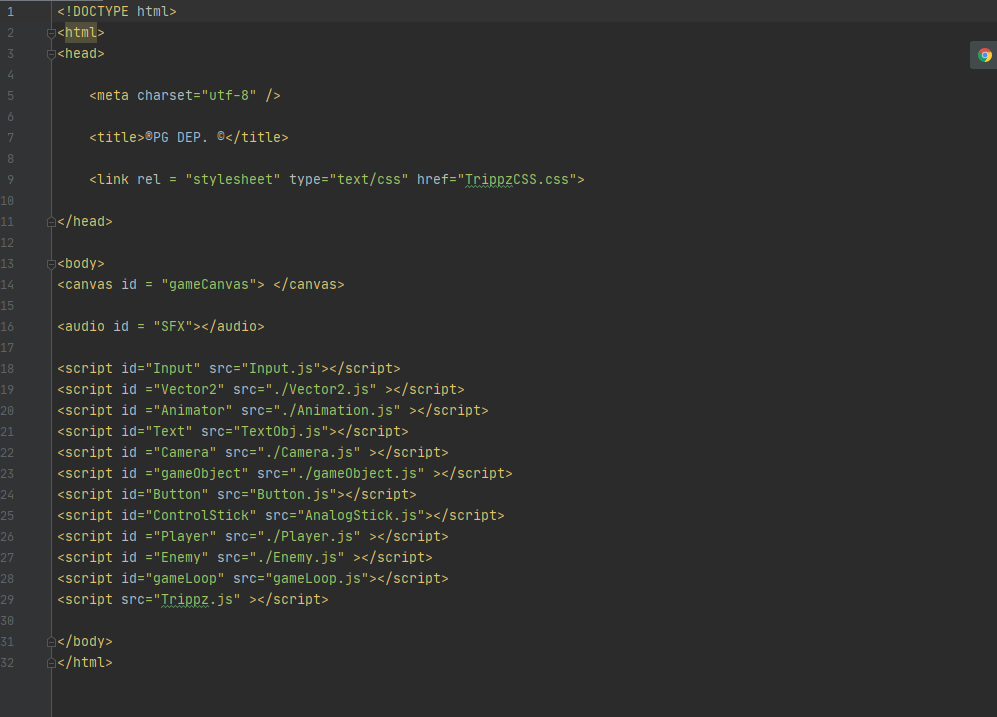
Now the base of gameObject and we know basic set up the class pretty much constructed the same as previous inherited classes inheriting from gameObject but this time this a virtual button so we declare new function called butCheck which is similar to the other colCheck of the gameObject but this is for virtual buttons to check and see if the touch object is interacting with the given button that is all another simple class but yet effect for it purpose.

## AnalogStick.js



Like Button.js thought with little more functionality that allows the simulations of the analog stick on screen as a virtual controller device that can be interacted with touch. Assuming the variable are name fairly well centre is the centre of the object, controlStick is new gameobject residing inside the containing gameObject which what display our analog stick with it restricted area that constraints the analog stick leave that space allowing for somewhat true virtual joystick experience using simulate axis similar true to how a analog joystick would work with hardware.

# Index.HTML



This has been left to the end as it was glue all those part previous discussed together and give the webpage the functionality by linking all the components together and allowing JavaScript to access the HTML5 elements and make use of them within our JavaScript.

# Refrences.

JORDAN HARBAND&Kevin Smith., 2020 *ECMA International.* [viewed 12/11/2020]. Available from: <https://262.ecma-international.org/11.0/index.html#sec-scope>.

JSCONFPHILIP ROBERTSed., 2014. *What the heck is the event loop anyway? | Philip Roberts | JSConf EU.* N/A: JSConf.

MDN CONTRIBUTORS., 2021 *MDN Web Docs.* [viewed 13/01/2021]. Available from: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>.

PHILLIP ROBERTS., 2014 *Loupe.* [viewed 01/01/2021]. Available from: <http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDUwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D>.

Wonderstruck., 2020. *Polycraft.* <http://polycraftgame.com/>: , December 17,.