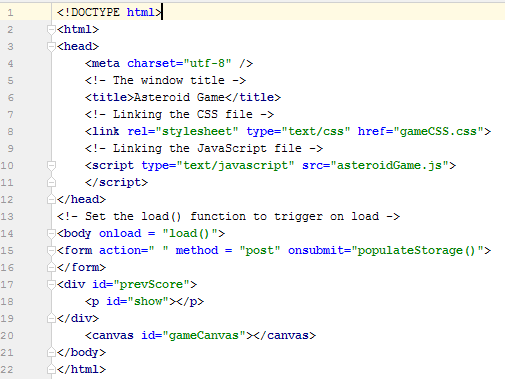
Code Explanation Steven Smith MGD1

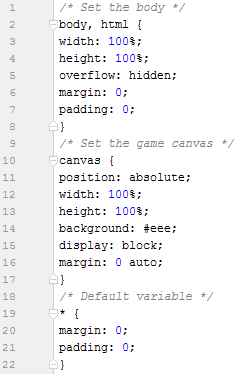
**asteroidGame.html :-**

****

The HTML file is used to describe and define the content of the webpage. The first line <!DOCTYPE **html**> tells the browser which type of HTML to expect which affects how the browser will render the page. Next the root element is opened with the <**html**> tag telling the browser that everything within the tags is relevant to HTML. Next the <**head**> tag is used which tells the browser information about the page. The character set is set and the title is created, after this the eighth line is used to link up the CSS file with the browser. After this, line 10 uses the **type** keyword to set the text being read to JavaScript and also adds the external resource; in this case **"asteroidGame.js"** is being loaded as the external resource and the head tag is then closed with </**head**>.

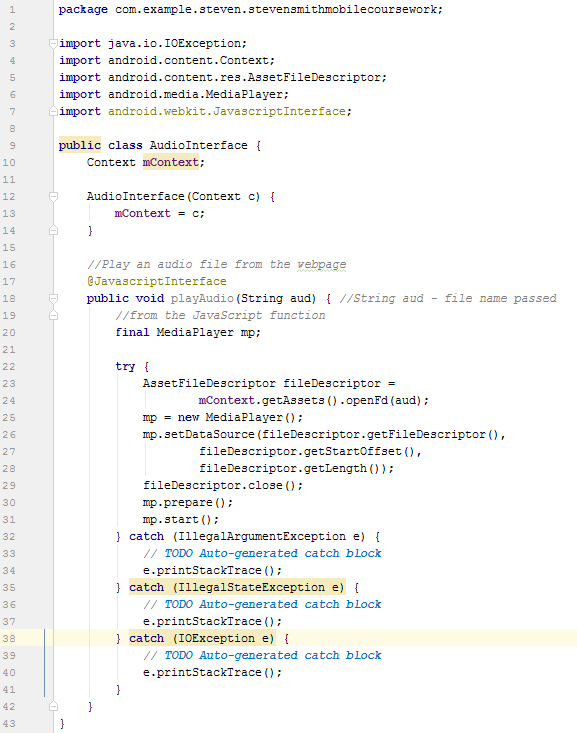
Next the <**body**> tag is opened, this tag contains all content visible to the web user when the page is visited, this includes text, images etc. In the same line the **onload** keyword is used to set which function from the JavaScript file will run when the browser runs the HTML file, in this case **"load()"**. After this the local storage data is handled, the <**form**> is then opened and contains the function under **onsubmit** that will store the data on the form onto local storage. The form is then closed using </**form**>. The <**div**> tag is used to define the local storage and line 19 tells the browser to show the previous saved data, in this case the score. The </**div**> tag then closes this section, after this the canvas from the CSS file is given an id so it can be manipulated in JavaScript. After this both the tags </**body**> and </**html**> are closed.

**gameCSS.css:-**

****

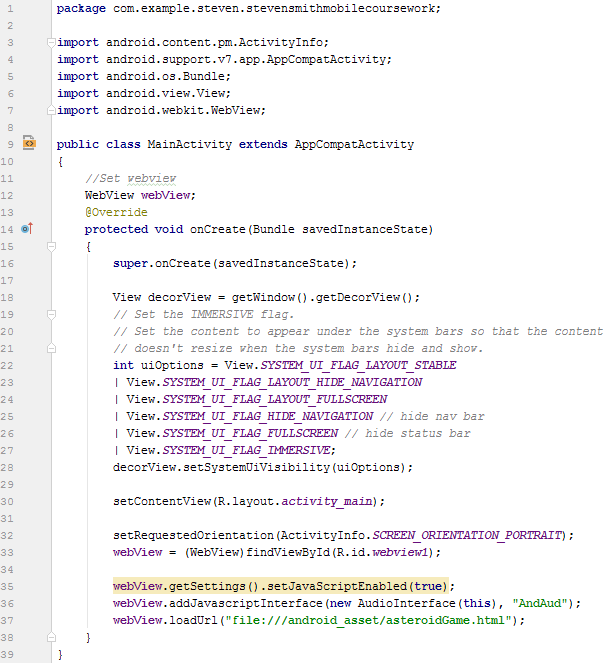
This .css file is used to style the context of the HTML document. The first section from lines 2 to 8 handle the style and content of the HTML body, the width and height are set to take up 100% of the body and any overflow is hidden, the margin and padding are set to zero for aesthetic and practical purposes. From line 10 until 17, the game canvas is set out, the position of the game canvas is set to absolute and the width and height are set to 100% to fill the window it is attached to. The background is then set to grey and the display is set to block, finally the margin is set to 0 auto to ensure the margins are updated if changed. Finally from line 19 to 22 this is the default selector that will pass anything unknown through and prevent errors by assigning them values of 0.

**AudioInterface.java:-**

****

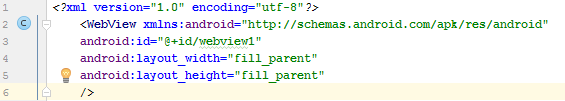
This file was written in an attempt to fix the audio issues with HTMLs <Audio> not working on Android. At the beginning, any external java libraries that are required are imported.AudioInterface is then created as a **public class**, context is then created and set equal to c. After this, code is then written to play an audio file. This is done by creating the method playAudio which takes in a string as an argument. This string is the filename of the audio file that is to be played. A media player is then created and an attempt is made to play the audio while checking for errors. In the end, this workaround required the JavaScript code to be added straight to the HTML document and as an external JavaScript file is being used it rendered this code unusable.

**MainActivity.java:-**



The MainActivity.java code is the actual application file that is executed to run the application. The **import** statements add any additional libraries required for the application. After this the public class MainActivity is created. Within this class the Webview webView is created. A method is then created that houses the onCreate settings. A View is set up and then the UI is changed using uiOptions this ensures that the window created will look the way we want it to look by changing such things as if it is fullscreen or if the navigation bar will show up in the window. After this, the MainActivity.java file is then linked with the activity\_main.xml and the screen orientation is selected, in this case portrait is selected. The webView is then changed to include JavaScript and to add the AudioInterface.java. Finally the HTML file **asteroidGame.html** is selected to load when the window is created which will then lead to the JavaScript file being run.

**activity\_main.xml:-**



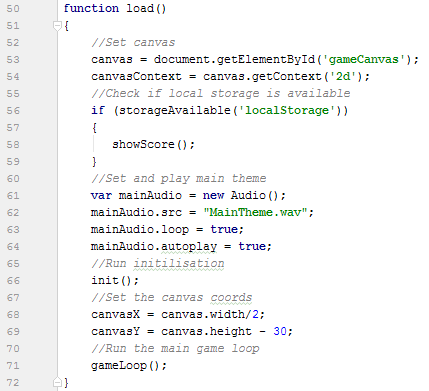
The activity\_main.xml simply contains how the window will be drawn, the first line highlights what xml version is being used and what character set is used. Then the **WebView** is filled in with the WebView id, in this case **"@+id/webview1"**, the height and width are then set to fill the parent window and the WebView is closed.

**asteroidGame.js:-**

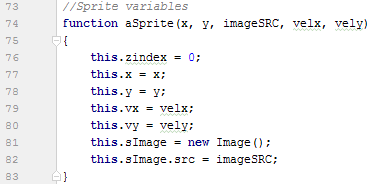


The JavaScript (JS) file shown is used to provide interactivity and functionality on websites, this first screenshot shows all the variables used within the asteroidGame.js file. From line 2 to 6, these variables are used to control, manipulate or add functionality the canvas. The first two variables are simply used to store the canvas and canvas context, after this the canvasX and canvasY variables are used to store coordinates of where on the canvas is being selected; the mouseIsDown variable is used to tell if the user is touching the screen. From lines 8 until 19, these variables are all sprites that are set up at the initialisation stage, the variables on lines 20 and 21 are used to store the current time and to also store the time that the application was launched.

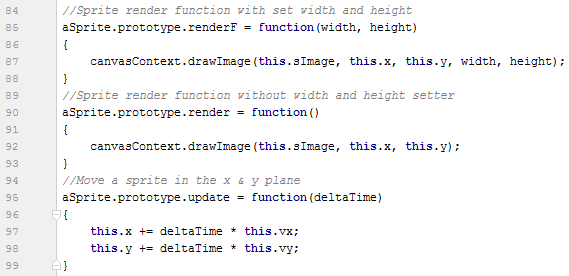
Next, from lines 23 to 29 these variables are all Booleans that are used in different checks throughout the script. From lines 31 to 35 any gameplay integer is stored here such as the state the game is in, ship health, score and the high score that is saved in local memory. From lines 37 to 42, any coordinates are stored, the touchX and touchY variables save where the user has touched on the screen which is almost identical to the touchXup and touchYup variables but these save the coordinates when the user releases their finger off the screen. The last two variables are used for calculating the bullet trajectory of the players’ ships gun. The final variables are used to store the audio files used within the game.



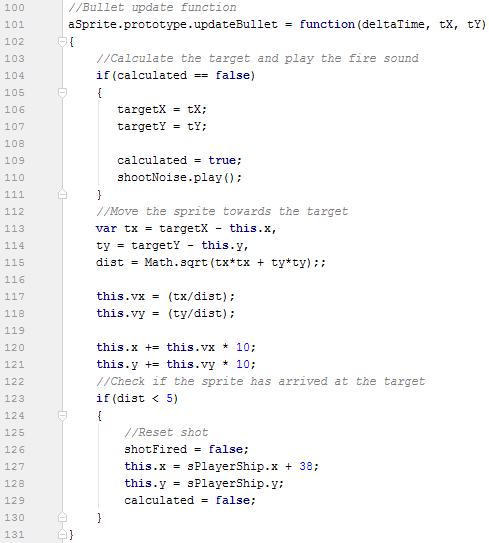
The first function in the JS file is the load() function, this is the function triggered by the asteroidGame.html file and as such is the first function to run. Lines 53 and 54 are setting up the canvas element and canvas context in JS, as the game created is 2D, the canvas context is created as **'2d'**.After this, a check is made to ensure that local storage is available with the storageAvailable() function.If this returns true, the showScore() function is triggered. Next from line 61 to 64, the main audio is set and the theme is loaded then set to loop and auto play. Next the init() function is called to initialise the game sprites. Then the canvasX and canvasY are set and the main game loop is entered by calling the gameLoop() function.



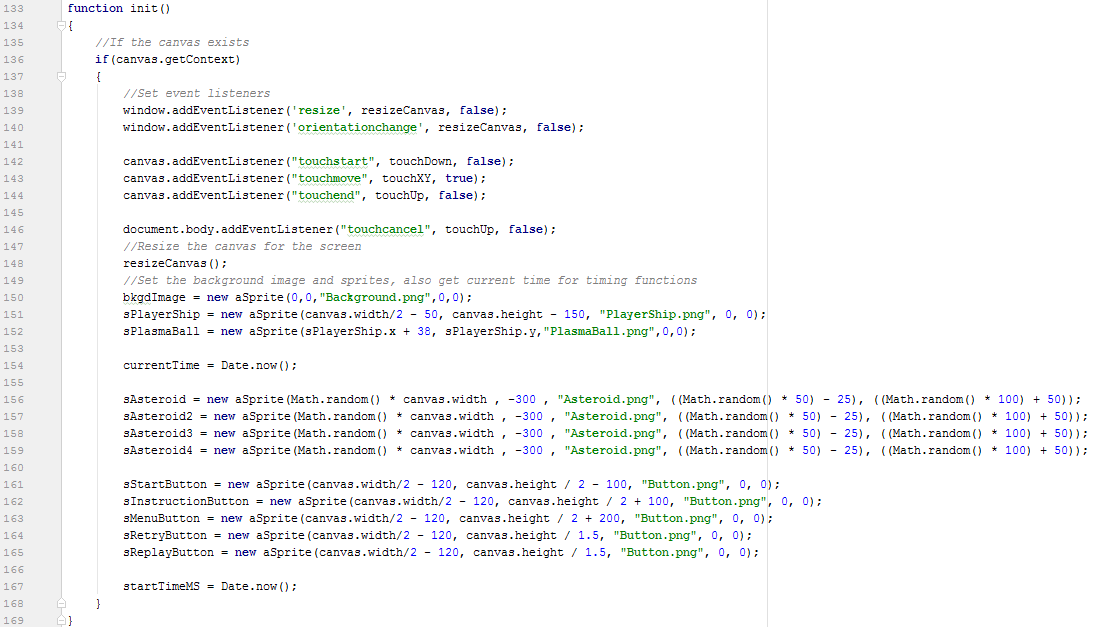
The function aSprite() takes in 5 variables and creates data for a sprite, x and y take in and set the x and y coordinates of the sprite, then velx and vely are used to set the velocity in the x direction and y direction of the sprite. Finally an empty image is created and the specific sprite image that was passed into the function is loaded into the image.



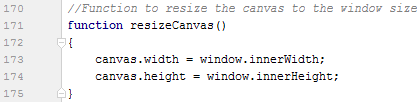
The following prototypes are used to manipulate or provide functionality to the aSprite class. The first prototype, renderF, is used to draw the sprite on the canvas and has functionality to draw the sprite with a set width and height. The next prototype, simply called render this time, is used to draw a sprite onto the canvas without setting the width and height which renders the sprite straight to the canvas using the drawImage() function. Finally, the update prototype is used to change the x and y coordinates of the sprite passed in with relation to the velocity in the x and the velocity in the y direction that the sprite has.



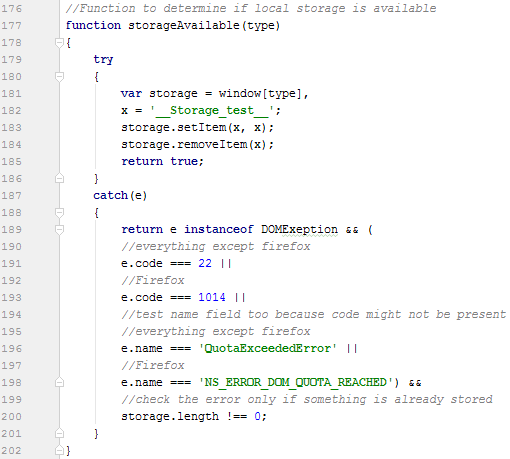
The updateBullet prototype is used when updating the ships bullet sprite when fired. The function takes in the time and target x and target y coordinates, these coordinates are taken from where the user has touched on the screen. Once inside the function, a check is made to determine if the target has been set as the function is called once per frame when the bullet is firing and we do not want the target change from where was first touched on screen. In this check, the targetX and targetY are saved, the check is set to true so it does not run again and the audio for the gun firing is played as this will play the sound only once. Next from line 113 to 121, the distance between the X and Y coordinates of the target and current X and Y coordinates is calculated. This is then used with Pythagorean Theorem to find the distance between the current position and target position. The distance in the X and Y compared to the overall distance is used for calculating the velocity in the X and Y to move the bullet towards the target, then the X and Y coordinates are manipulated with their respective velocities and both are multiplied by 10 to suit the speed of the gameplay. Finally a check is made to obtain if the bullet is within 5 pixels of the target by using the variable storing distance (dist), if this returns true, the shotFired Boolean is set to false which allows for another bullet to be fired, The bullets coordinates are set back to the original coordinates and the calculated check from the start of the function is reset to allow the target X and Y coordinates to be re-entered.



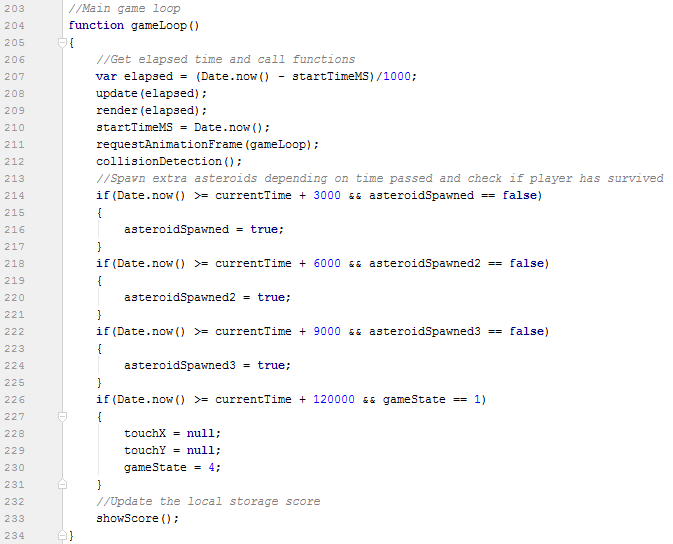
The init() function is the first function called after the local storage check. The first thing within the function is a check to ensure that the canvas exists. If it does, the event listeners that will be used are added here, these event listeners include some window, canvas and HTML body events. Once the event listeners are created, the resizeCanvas() function is called to ensure the sprites being placed will be placed on a canvas of the correct size. Next from line 150 to 165 (with the exception of line 154) all the sprites necessary for the game are being allocated and data on them is being stored. Line 154 and line 167 are two variables that save the time. It is saved twice as startTimeMS is never changed but currentTime is updated on occasion.



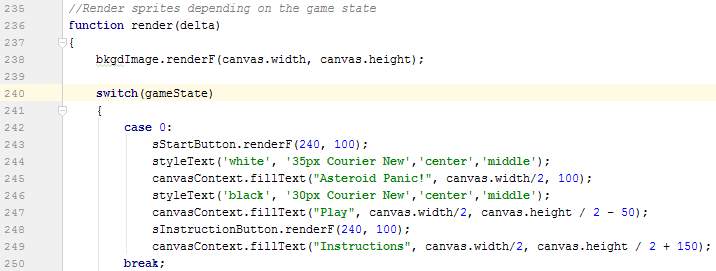
This resizeCanvas() function simply sets the canvas to fit the current window, the first line setting the canvas width to equal the inner width of the window and the canvas height to equal the windows inner height. This can be called to ensure that the canvas fits the window correctly and is called in the init() function before setting the sprites to ensure that when the sprites X and Y coordinates are set they are correctly placed on the canvas.



The storageAvailable() function determines if local storage is available, the function takes in the type of storage being checked, in our case local storage from the load() function. First the **try** keyword attempts to store an item, in this case a string and returns true if it succeeds. While this is happening, the **catch** keyword is used to flag up any errors that happen during the storage test. Error codes for Firefox and other browsers are checked and the error names for Firefox and other browsers are also checked. Finally a check is made to ensure something has been stored which will return the error only if the storage worked.



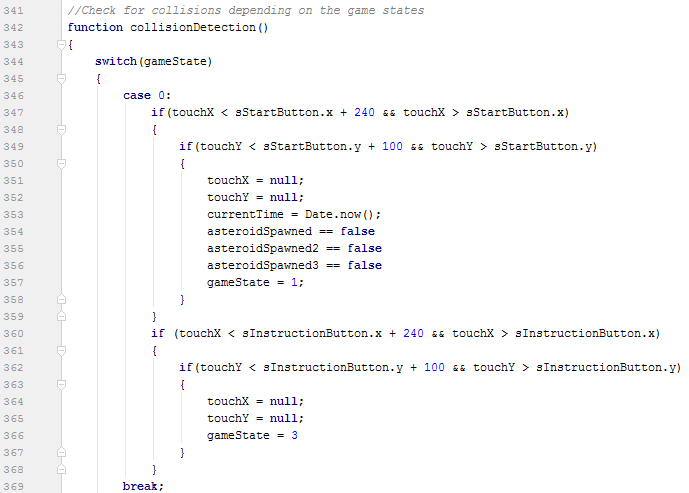
The function gameLoop() contains the main game loop code, the first line finds the elapsed time the game has been running by taking the startTimeMS variable away from the current time. Then the update() and render() functions are called and the elapsed time is passed into them. The startTimeMS is then updated to equal the current time again. The requestAnimationFrame() function tells the browser that an animation is going to be played and updates the browser window, as positions of sprites are being updated and redrawn we want the browser window to also redraw them. Then the collision detection function, collisionDetection() is triggered. Checks are then made with regards to time that has passed to set certain Booleans to true that will spawn additional asteroids, this will increase the difficulty as time goes on, the last time check is to check if two minutes have passed and if they have the win condition is met and the game state changes to the win state. Finally, the showScore() function is called to check if a new high score has been achieved from the one currently on local storage.



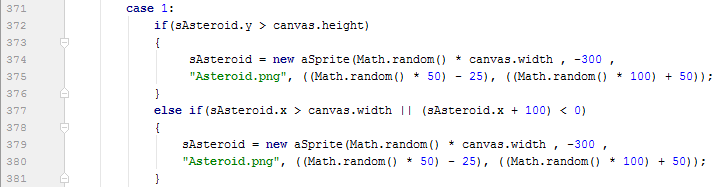
The render() function takes in the time between frame updates and handles the rendering of sprites by passing sprites through the render or renderF prototype and also draws any necessary text. As the background is to be rendered constantly it is not included in the switch statement that renders sprites and text depending on the game state. The game state changing allows for different rendered windows to be switched between easily. The first case when gameState = 0 handles the starting menu and uses the styleText() function with variables passed into it to set the text used and draws the text using canvasContext.fillText() which takes in the string to be drawn and the x and y position. Finally, it also renders the sprites that will appear in this game state by passing them through either the render or renderF prototypes. This is repeated for each individual game state and results in different chunks of sprites and text being drawn depending on the game state.



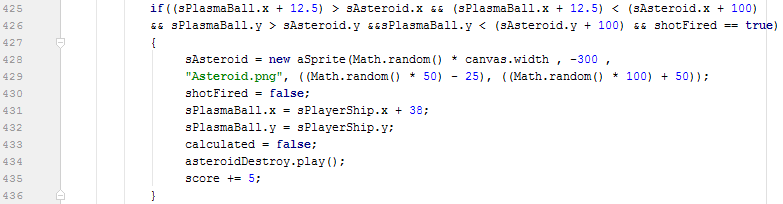
The update() function handles updating any of the moving sprites positions, it will only run when the gameState = 1 which is when the game is in its main play state. The asteroids, which are given random capped velocities in the X and Y are updated depending on if the Booleans checking if enough time has passed are true or not. It also checks if shotFired = true and if so it passes the (now rendered) plasma ball sprite through the updateBullet prototype with the coordinates that were touched on the screen. As sprites only move during the main play state only one case exists for the update() function as the button sprites and other text is all static.



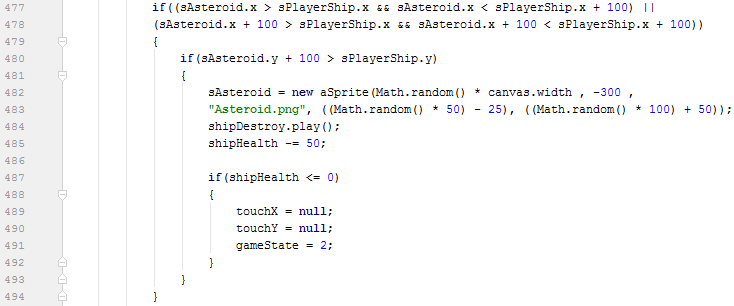
The collisionDetection() function handles all collisions that can occur within the game. It is also affected by the gameState variable as different collisions are checked for depending on what game state is currently active. When gameState = 0 (in the main menu) the collisions checked for are if either the start button sprite or instruction button sprite has been tapped. This is done by comparing either sprites bounding box to the coordinates of the screen touch (touchX and touchY). If either is triggered, the screen tap coordinates are cleared, any initialisation is done and the game state is set to the new game state.



When the game state is in the main play state, three types of collision detection are made. The first determines if the asteroid is still in the confines of the window, if it goes below the window or outside either edge it is respawned at the top of the screen again. This is done for each asteroid.

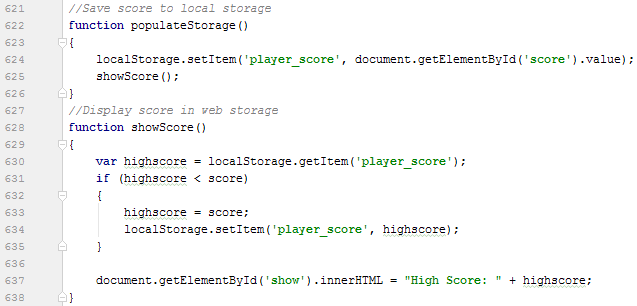


The second collision detection check is to check if the plasma ball is intersecting the asteroid, a check is also made to ensure the bullet exists using the shotFired Boolean. If the collision returns true, the asteroid is respawned and the bullet is reset. An audio file for the asteroid being destroyed is played and the players score is increased by 5. This check is carried out once per asteroid.

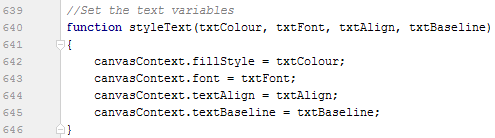


The final collision check is to check if the asteroid sprite and player ship sprites bounding boxes are intersecting, if this returns true it respawns the asteroid and plays an audio file of an explosion, the ships health is also decreased by 50. A check is then made to see if the player has lost all health, if it returns true, the screen touch coordinates are cleared and the game state is changed to the lose state.

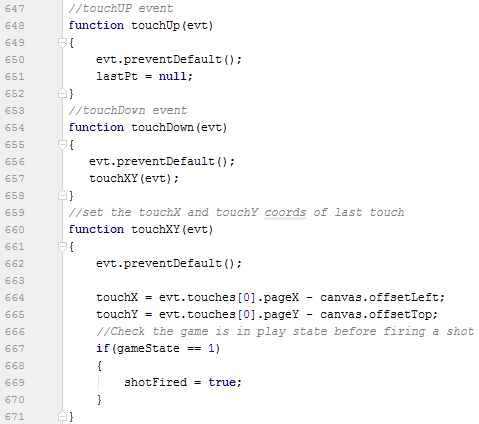
When in states 2, 3 and 4, the collision detection is exactly the same as in case 0. It simply checks if a button has been pressed and if it returns true it handles any initialisation, clears the screen touch coordinates and changes the game state to the appropriate state.



The populateStorage() function is the function tied to the HTML file with regards to updating local storage. This function simply sets the players current score and runs the showScore() function. The showScore() function gets the current score on the local storage and sets it equal to the high score. A check is made to see if the player has beaten the high score and if so the high score becomes the players current score and the current score is saved to local storage as the new high score. Line 637 simply draws the current high score to the inner HTML window.



The styleText() function is a helper function to save time when changing the text, it takes in all necessary variables and sets them accordingly using the canvasContext functions to save repetitive coding when changing the text while rendering.



Finally, the listeners created during the init() function are created, the touchUp function clears the touch coordinates when a user releases their finger from the screen. The touchDown function triggers the touchXY function, when the touchXY function is triggered it saves the touchX and touchY depending on where the user has touched the screen and checks if the game state is in the main play state, if this returns as true then a shot is fired by way of setting the shotFired Boolean to true which will trigger the plasma ball sprite to be rendered, updated and put through collision detection.

References/Bibliography:-

Sprites:-

<http://millionthvector.blogspot.co.uk/p/free-sprites.html>

<https://opengameart.org/content/space-background-01>

<https://opengameart.org/content/asteroid-generator-and-a-set-of-generated-asteroids>

<https://opengameart.org/content/tx-bullet-0>

<https://opengameart.org/content/ui-pack-buttons-and-dialogue>

Audio:-

<https://freesound.org/people/envirOmaniac2/sounds/398941/>

<https://freesound.org/people/nsstudios/sounds/344276/>

<https://freesound.org/people/FlashTrauma/sounds/398283/>

<https://freesound.org/people/sharesynth/sounds/341237/>

Code:-

<https://stackoverflow.com/questions/13849185/moving-an-object-along-a-straight-line-at-a-constant-speed-from-point-a-to-b>

<https://www.codeproject.com/Tips/677841/Playing-Audio-on-Android-from-an-HTML5-File>