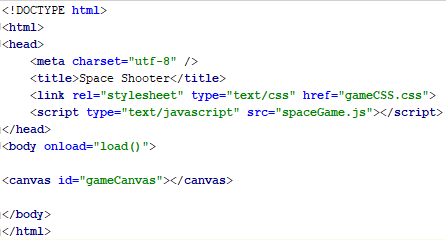
HTML5

HTML is a markup language that is used to create the contents of a web page. This games HTML file is called spaceShooter.html.



The <!DOCTYPE html> declaration will let the browser know which type of HTML to expect and this effects how the webpage is rendered. The DOCTYPE declaration must be declared at the very start of the document. The root element of the document lies between the <html> and </html> tags.

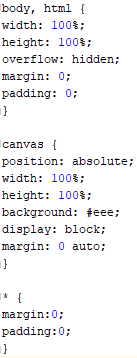
The main information about the document such as title, charset and any external resources is contained within the <head> and </head> tags.

Inside of these <head> tags, the <meta> tags contain a charset which is set to a value of “utf-8”, which specifies the character encoding for the HTML document. The <title> tags contain the title of the project, which in this case is set to “Space Shooter“. The <link> tags are used to link the document to an eternal style sheets. The rel attribute specifies the relationship between the active document and the linked document. The value is set to “stylesheet” which imports a style sheet. The type attribute specifies the content to be linked with the document. This value is set to “text/css” which indicates that the content is CSS. The href attribute specifies the location and name of the linked document, in this case, the value is set to “gameCSS.css”. The <script> tags are used to define an external script file through an src attribute. The type attribute defines the type of script, which it set to “text/javascript”. The src specifies the location of the external script file, which is set to “spaceGame.js”, the main game file.

The <body> and </body> tags contain the content which is visible to the users when visiting the webpage. Inside of these tags, the onload attribute is used to execute a specific script or function. In this program, it is set to “load()” which is referencing the load() method inside the javascript file. Inside of the <body> tags, a <canvas> and </canvas> tags are used to define the rectangular area on the HTML page. The id attribute is used specify a unique id for the element, and is called “gameCanvas”.

CSS

CSS is used to describe the appearance or presentation of the contents on a webpage.



Inside the html element of the CSS, the width and height property are both used to set the width and height of the element. These are both set to 100% to ensure they cover the full percent of the body element. The overflow property defines what happens when content overflows an element’s box. This value is set to hidden which will clip the overflow and ensure that the overflow is hidden. The margin property sets the value of the margin in the document, and is set to a default value of 0. The padding property sets the padding value in the document, which likewise to the margin, is set to a default value of 0.

Inside the canvas element of the CSS, the position property is used to specify the type of positioning element in the canvas. This property value is set to absolute, which ensures that the elements are positioned relative to the previous position elements of the body of the HTML The width and height are both set to 100%. The background property can define all the elements of the background from colour to images. The background is set to a solid colour of grey. The display property specifies the type of box used for the element. The property value is set to block which will display an element as a block element. The margin property is also set to a default value of 0 with an additional property of auto, ensuring that the browser calculates the margin.

Inside of the \* selector, the margin and padding are both set to 0 as they override the remaining elements. They are set to 0 to ensure that no margin or padding is present in any element in the document.

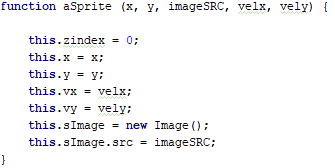
JavaScript

JavaScript is a programming language that runs in the browser which is used to add interactivity and other dynamic features to a website application.

Load Function

As described in the HTML file, the load() function is executed first here. The load() function first contains code to handle the canvas loading. The canvas is loaded first via document.getElementById() and passing through the unique id parameter of the element to be sought, which will be the “gameCanvas” as identified in the HTML id attribute. A context on the canvas is then obtained by using canvas.getContext() and passing through the parameter “2d”. This parameter creates a two-dimensional canvas rendering context. A local storage is then initialised by first checking that the local storage is available by calling the created function called storageAvailable(). If storage is available, then the score is stored by calling the function showScore() and then assigning the localStorage variable to window.localStorage, allowing the game to access the stored documents storage. To handle the different states in the game, a switch statement is used to evaluate the different states in the game based on the current value of the game state. Inside this load() function, the game state is set to the main menu, as the main menu is the first scene the user should be presented with. The init() and gameLoop() functions are then ran and the dimensions of the canvas are set by referencing the canvas.width and the canvas.height.

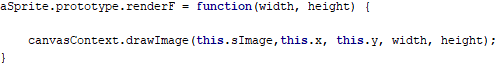
Sprite Objects



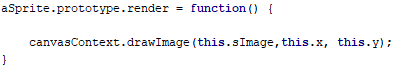
A constructor function called aSprite is created to handle the creation of sprites in the game. This function will be used to create multiple images on the canvas. This function will take in an x, y, imageSRC, velx and vely variable. The zindex identifies which order the object appears at on the screen, this is set to a default value of 0. The x and y value of the object are set to the passed in value. The vx and vy are set to the passed in values of velx and vely, respectively. An image is first created via new Image() and a variable is assigned to this. This variables src value is then set to the imageSRC value passed though to determine the location of the image.

Sprite Functions

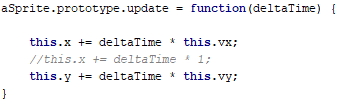
Each function uses a prototype call to inherit properties from the previous constructor and update or change properties depending on the different functions.



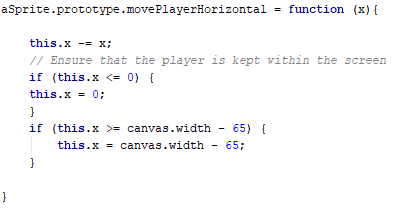
This renderF function is used to render the sprite based on the certain width and height parameters. This function takes in a width and height variable. Once the sprite has been drawn, this function uses the sImage, x and y value inherited from the constructor function in combination with the passed in width and height variable into the drawImage() function to display a new image on the canvas.



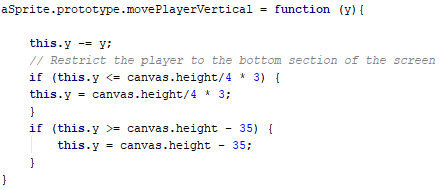
This render function takes in no parameters and uses the inherited sImage, x, and y values to display an image on the screen using the drawImage() function to display and image on the canvas.



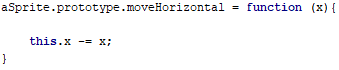
The update function will update the sprite every frame. It will take in a variable of deltaTime which is used to calculate the time from the last frame that was updated. The x value of the current sprite is set to the value of delta time multiplied by the value of the current vx value plus the current x value. The y value is set in the same way, except replacing the x values with the y values of the sprite.

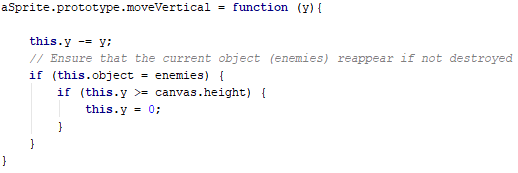


The movePlayerHorizontal function is used to control and restrict the movement of the player along the horizontal axis. This function takes in a value known as x and this value is then subtracted from the current x value to gain a new value of x. There are two if statements in this function; the first one is to check whether the player has exceeded a value of 0 on the x axis, if they have then they are set back to 0. The second one, is to check whether the player has exceeded the current canvas width – a small number of 65 to compensate for the player sprite image size. If the player exceeded this limit, then they will be set back to the canvas width minus a value of 65.

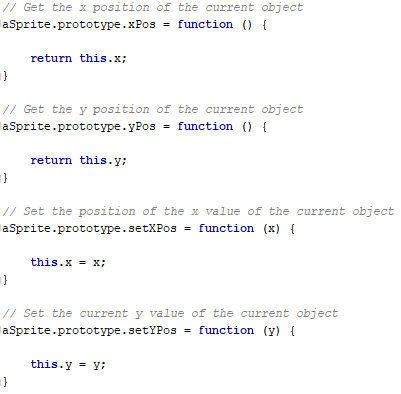


The movePlayerVertical function is used to control and restrict the movement of the player along the vertical axis. This function takes in a value known as y and this value is subtracted from the current y value to gain a new y value. Like the previous method, there are two if statements that will restrict the player to the bottom section of the screen and limit their values on the y axis based on the calculations in the canvas height.



The moveHorizontal function is a simple function that handles the horizontal movement of all the other sprites in the scene. The function takes in a value of x and much like the first line of the movePlayerHorizontal function, will take this x and subtract it from the current value of x to gain a new value of x.

The moveVertical function handles all the vertical movement for all the other sprites in the scene. The function takes in a value of y and subtracts this value from the current value of y to gain a new y value. An if statement is sued to check whether the current object is an enemy, if it is then another if statement is used to check their current position isn’t greater than the current height of the canvas, and if it is, then the y value of the object is set to 0.

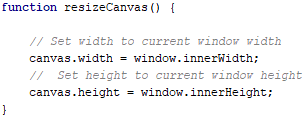


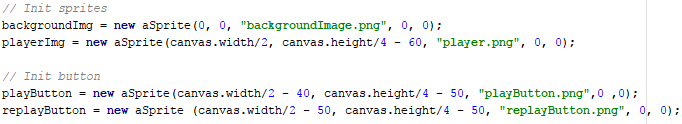
The xPos and yPos functions are getter functions for the objects. These functions return the current x and y value of the current object, respectively. The setXPos and setYPos functions are setters, and take in either an x or y value, and change the object to equal these current x or y parameters.

Init Function

This method handles the initialisation of all the components of the scene. This function is called at the end of the load() function.

This function first starts by checking to see if a canvas context exists with an if statement. If there is a canvas context, then the rest of the code inside of the init function will execute. Inside of the if statement, the event listeners for the window, touch control and keyboard/keypad are called which will attach an event handler to the current document.



The resiveCanvas() function is then called. This function as show below will resize the current canvas to the current window dimensions. The canvas width and canvas height are set to the dimension or the window innerWidth and innerHeight.

Next, the sprites and buttons are initialised. The buttons and sprites both makes use of the constructor function to load an image and use the inherited functions to manipulate their positions. The current lives are then set to 3, and the startTimeMS variable is set to the value in Date.now() which returns the number of milliseconds elapsed.

Game Loop Function

The game loop function first handles the main music in the scene. A check is done to see if there is currently any music playing in the scene, and if not then the main music begins to play via a new Audio() function and passing through the location of the music file. The music will then play with the play() function and the volume will be set to 0.5.

E:\Jack\Documents\Uni Work\4th Year\Mobile Game Development\Documentation Images\Image 13.png

Next, an elapsed time is calculated as shown above. This elapsed variable is then passed through to the update() and render() functions. A check is then done to check if the lives are equal to or less than 0, and if true then the current game state is switched to the game over screen. The 2 collision detection methods are then called and finally a requestAnimationFrame() is called to tell the browser to update the animation for the next repaint.

Render Function

From the game loop, the render function is called. This function is responsible for drawing and rendering all the components on the canvas. Based on the current state of the game, different sprites will be rendered, and different functions will be called based on the current active case in the switch statement. Outside of the switch statements, the background image is rendered using the renderF() function and passing through the current canvas width and canvas height.

The main menu state will first call the styleText() function and pass through the relevant text variables. The fillText() method is then used on the current canvas context to draw and fill the text on the canvas based on the previous style of the text. These functions are used to display the instructions for the game. The play button is then rendered using the render() function.

The main game state will first render the player image by calling the render() function. Next, given that the enemies and bullets are both stored as an array, a for loop is created to iterate through the whole array. A check is then done via an if statement to ensure that the specific array is not empty, and then the specific images are then rendered using the render() function. The styleText() function is used here to display the score and lives at the top of the screen.

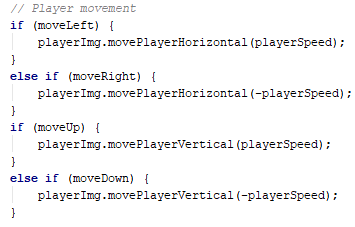
The game over state will contain 3 styleText() functions that will be used to display a game over message, the players score and then the high score of the game. The replay button will also be rendered using the render() function.

Update Function

The update function is used to update the state of the current sprites and objects in the scene. Much like the render() function, this function will use a switch statement to activate and call different functions based on the current game state.

The main menu state has one function; to call the update() function on the playButton and pass through the delta variable from the update() function.

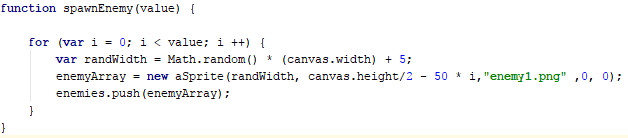
The main game state will first start by incrementally increasing the value of the variable fireRate when this current state is active. AN if statement is used to check if the player is shooting and the fireRate value is equal to or higher than 15. If these conditions are true, then the fireRate is reset back to 0, this will add some delay to the firing of the bullets to improve gameplay and immersion. Next a new sound is created via new Audio() and passing through the filename for the player shooting sound. The volume is then set, and the sound is then played via the play() function. Finally, a bullet is spawned via the spawnBullet() method and passing through the value of 1. Two for loops are then created to handle the movement of the enemies and bullets. The for loops will iterate through the both arrays and use the moveVertical() function on each enemy and bullet in the arrays. The enemies will pass in a method known as getRandomNumber(), and from that will pass in -3 and -5, while the bullets will pass in a value of 5 to the moveVertical() function.



Lastly, the player movement is handled via multiple if statements. If the certain movement variable is set to true, then the relevant moveHorizontal() or moveVertical() functions are called.

The game over state will contain an update() function from the aSprite constructor for the replayButton and will pass through the delta variable from the main update method. Finally, the lives are set back to 3.

Spawning/Despawning Enemies and Bullets Functions

There are 4 methods to spawn and despawn the enemies and bullets; spawnEnemy(), spawnBullet(), despawnEnemies(), despawnBullets().

The spawnEnemy() and spawnBullet() functions will both take in a value parameter. These functions will then both used a for loop to iterate through the number in the passed through value variable. For each value in the loop, depending on the function, either an enemy will be spawned at a random location with the help of the Math.random() function, or a bullet will be spawned at the current players position with the help of the getter functions. Once the bullet or enemy is spawned at their relevant location, they will then be added to the array using the push() function to add a new item to the end of the array and passing through the relevant array variable.

Collision Detection Functions

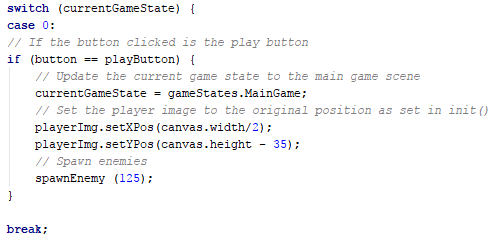
There are two collision detection methods; collision1 and collision2. These collision detection methods are responsible for handling the collision detection between all the sprites in the scene.

Collision1 is responsible for handling the collision between the player and the enemy. A for loop will first be created to iterate through all the enemies in the enemy array. Inside this for loop, an if statement will compare the dimensions of the player image against the specific enemy image. If these dimensions collide with each other, then a hit sound is played via new Audio() and passing through the location of the hit sound file and the play() function. Next, the splice() method is then called to remove an item from an array and the specific enemy is then removed. Lastly, the playerLoseLife() method is called which will take one life away from the current number of lives.

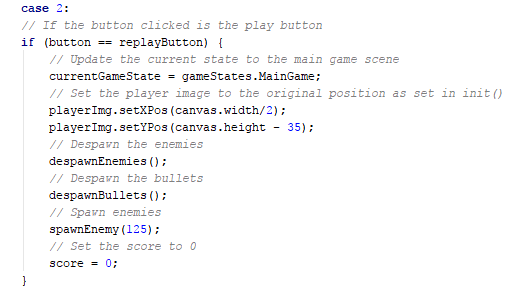
Collision2 is responsible for handling the collision between the bullets and the enemies in the scene. Much like collision1, a for loop is first created to iterate through each enemy in the enemy array. Inside of this loop, another for loop is created this time iterating through each element of the bullet array. An if statement is then used to compare an element of the enemy array with the bullet array by comparing their x and y image length and coordinates. If the images collide with each other, then a hit sound is played using the same method as collision1. Next, the splice() method will be used twice, once for removing the relevant enemy from the array, and again for removing the colliding bullet with the array. Last, the addScore() method is called which will take the current score and add 10 to its current value.

Button Collision Method Function

This method handles the collision between the mouse click and the button. This method contains a parameter which will be used take in a specific button in the scene. Once this button is passed through, a check is done to see if the mouse position collides with the button via the same method used in the main collision detection methods. If the mouse collides with the button image, then depending on the current state of the game and the button click, different functions are executed.



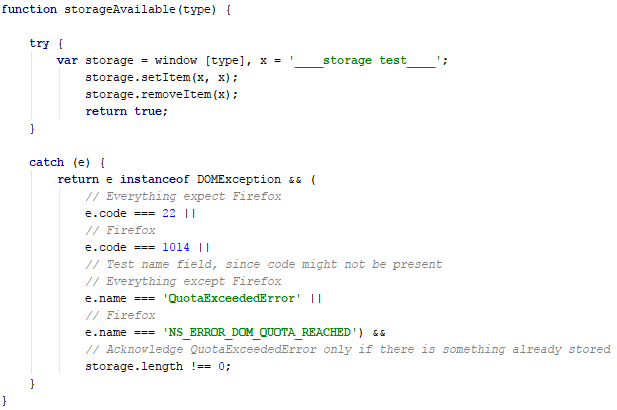
If the current state of the game is in the main menu state, and the current button passed through is the play button, then the current game state will be switched to the main game state. The player position will also be set by using the setter functions in the aSprite constructor to adhere to any window or canvas that may have occurred in the main menu state. Lastly, enemies are then spawned by using the spawnEnemy() function and passing through the number 125.



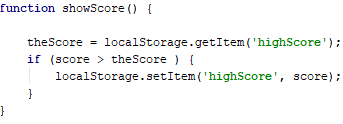
If the current state of the game is in the game over screen, and the current button passed through is the replay button, then current game state is set to the main game. The player position is also set by using the setter functions in the aSprite constructor. The methods despawnEnemies() and despawnBullets() are then called to delete and destroy any leftover sprites from the transition to the main game to the game over screen. Next, the spawnEnemy() function is called to spawn the enemies again and the number 125 is passed into the function. Lastly, the score is reset to 0.

Local Storage Function

This function is used to store the current highest score of the player in the local storage. The Web Storage API is used as the local storage aspect persists even when the browser or device is close and reopened.

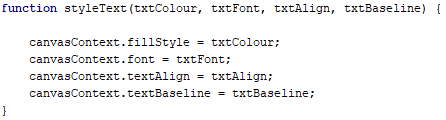


The function storageAvailable() will be used to determine whether the local storage is available or not. A check is first made to the using a try…catch statement. The catch clause will specify what to do when an exception is thrown within the try block. Inside the try scope, a storage variable is created and is set to the current window type along with an x variable is used as the key of the data item. The setItem() function and removeItem() function are then called on the storage key if no exception is caught. If an exception is caught, then an error is displayed.



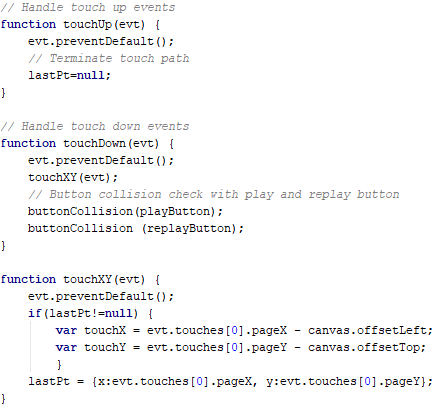
The showScore() function is a separate function that is called in the load() function. This function will retrieve the score from the local storage using the getItem() function and passing in the ‘highScore’ element. The current score is then compared against this high score and if the current score is higher, this score is then set as the high score using the setItem() function.

Style Text Function



This function is used to set the text components. This function will take in 4values as parameters and will be assigned to different text components on the current canvasContext variable. The text colour will be set equal to the fillStyle variable. The text font will be set to the font. The alignment of the text will be set to the textAllign and the baseline will be set to the textBaseline.

Touch Control Functions

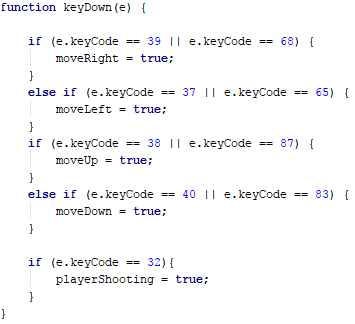


The touch functions are used to handle the touch control in the scene. The touchUp() is used to handle the touch up events. It takes in a parameter of the current event type, and this event will be used to call preventDefault() function which will cancel the event if it is cancellable. The lastPt variable is then set to null.

The touchdown() function is used to handle the touch down events in the scene. It will take in a parameter of the current event type, and like the touchUp() function, will use the preventDefault() to cancel the current event. The touchXY() function is called and the current event in then passed through it. Next, the buttonCollision() function is called twice and the play button and replay button are passed through the function.

The touchXY() function is used to handle the position of the touch. Like the previous two functions, this function will take in the current event type and will use the function preventDefault() to cancel the current event passed through this function. Two variables called touchX and touchY are created and set to the current position on the canvas depending on the different offsets. Finally, the lastPt variable is then set to page position via these previous offsets.

Keyboard Functions



There are two keyboard functions called keyDown() and keyUp(). These functions will take in an event called ‘e’. This event will be passed through and if the keycode of the event is equal to the character code then the specific variables for movement or shooting will be set to false. In the keyDown() function, these variables will be set to true.