

Mobile Game Development 1(MHI622947-17-A)

Coursework Code Explanation

Fraser McFarlane

Matriculation Number: S1434566

Computer Games (Software Development)

*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.*

Fraser McFarlane

Table of Contents

[HTML File – ‘gameHTML.html’ 3](#_Toc500589072)

[CSS File – gameCSS.css 5](#_Toc500589073)

[MainActivity.java 6](#_Toc500589074)

[JavaScript – gameJS.js 7](#_Toc500589075)

[Global Variables 7](#_Toc500589076)

[Objects 9](#_Toc500589077)

[Initialising Objects 13](#_Toc500589078)

[Initialising the Game 15](#_Toc500589079)

[Rendering 16](#_Toc500589080)

[Updating 18](#_Toc500589081)

[Touch Input 19](#_Toc500589082)

[Button Clicks 21](#_Toc500589083)

[Collision Detection 21](#_Toc500589084)

[Web Storage API 22](#_Toc500589085)

[Game Load and Loop 24](#_Toc500589086)

[Bibliography 25](#_Toc500589087)

[Asset References 25](#_Toc500589088)

# HTML File – ‘gameHTML.html’

The HTML file for this game contains all the values that describe the contents of the web page that is being displayed.

<!DOCTYPE html>

This tag tells the browser the type of HTML document that is expected to be prepared in this file. This then determines the way in which the browser renders the page elements.

<**html**></**html**>

This tag portrays to the browser that this file is a HTML document and acts as a container for all the HTML elements. This tag also represents the root of the document.

<**head**></**head**>

These tags contain all the informational and descriptive values of the page. The elements described in these tags will tell the browser information about the page including the title of the page and metadata. This will contain the elements that accesses external resources such as CSS and JavaScript files. The following code will be contained within the ‘head’ tag.

<**meta charset="utf-8"** />

This line uses the meta tag which holds metadata about the page. The charset metadata is set to “UTF-8” which tells the browser the type of character encoding that it should be using when preparing the webpage.

**<title>**FRASER MCFARLANE COURSEWORK PROJECT**</title>**

This line tells the browser the title that should be used in the browser window or tab.

**<link rel="stylesheet" type="text/css" href="gameCSS.css">**

The ‘link’ tag is used to include external resources that is going to be used within the page. The ‘rel’ attribute specifies the relationship between the HTML document and the linked document, in this case it is describing a stylesheet. The ‘type’ attribute describes the Internet media type of the document being linked. For this file it is a text file in the format of Cascading Style Sheet (CSS). The ‘href’ attribute defines the location address of the linked document therefore for this document it will be locating the gameCSS.css file.

<**link href="https://fonts.googleapis.com/css?family=Courgette" rel="stylesheet"**>

The google fonts API is linked here, specifically the Courgette font is referenced by the address provided by the Web Fonts service. The relationship if in the form of stylesheet.

<**script type="text/javascript" src="gameJS.js"**></**script**>

The script tag is used to include external resource file. This uses the same ‘type’ tag as used previously for linking CSS files. For this link it is a tax file in the format of JavaScript. The ‘src’ tag represents the source file of the external resource. In this case it is a JavaScript file following the same naming notation as the HTNL and CSS files – ‘gameJS.js’. This line is the last element defined in the <head> tag.

<**body onload="load()"**>

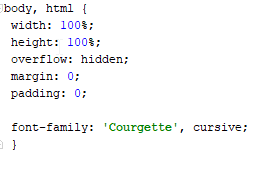
The body tag contains all of the content that will be displayed when the page is loaded. In this case, the gameJS.js file will handle what is displayed. To access the game code the onload event is called which access the JavaScript’s ‘load ()’ function which executes the game code.

<**canvas id="gameCanvas"**></**canvas**>

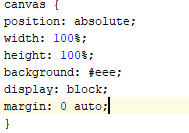
The canvas element is used for drawing graphics on the screen using JavaScript. This creates a container for the canvas element. The ‘id’ attribute is declared so that the JavaScript can access the canvas to allow graphics, texts and imagery to be rendered and manipulated.

# CSS File – gameCSS.css

This CSS file is used to style the HTML elements or tags in a lower level. In this file the HTML and canvas elements are styled accordingly. The body of the CSS file contains 2 selectors; ‘html’ and ‘canvas’. The selectors define the area of the document that is going to be accessed. From the selectors, the properties and property values can be accessed and manipulated. This section of the documentation will be split up by each selector.



This html selector is accessed in the start of the body of the CSS file. The width and height properties are both set to 100% so that the page takes up 100% of the windows spacing for both width and height. The overflow property is set to hidden. The overflow decides what happens to html contents that overflow an elements box. By selecting hidden will set the overflown elements to be hidden from the page. These values can alternatively be set to visible, scroll, auto, initial and inherit. The margin property is set to 0. This is so that there is no spacing around the html element on the page. The value that it is set to is in ‘px’, so the value declared will be how many px wide the margin will be. The padding is also set to 0 which allows no space to be made around the html content which is inside a border. The font-family property is used to define the Google Fonts API font that was linked in the HTML file. The first value is to define the family name. The second value is the generic family that the font is defined from. This is used as a fall-back font that can be used if the font family is unavailable.



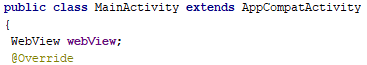
The canvas selector is accessed for low level styling of the canvas element. The position property describes the positioning type used for the canvas element. The value is set to absolute, this means that the position is set to the first position of the HTML element and cannot be moved. The width and height properties are both set to 100% like the HTML selector. As the page will run a game that is being made from the canvas, it is appropriate to set the dimensions to take up the full page. Th background property is set to a hex format colour. The display property which is set to ‘block’, is used to specify the type of box being used for the canvas element. The margin property with this element is also set to 0. However, it also takes in the ‘auto’ value which centres the element horizontally within the container.



The ‘\*’ selector is used to override all remaining selectors to ensure that there is no margin and padding in any element.

# MainActivity.java

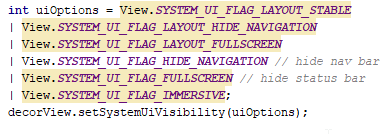
The main activity is the entry point to the android applications. In this file the UI elements, application view and settings will be prepared to host our HTML file when the application launches. This section will break down the Main Activity java class and explain how the android wrapper is being set-up.



The Main Activity class uses to ‘extends’ keyword which is used to create a sub class of the extended class ‘AppCompatActivity’. Within the class scope, a new WebView object is declared named ‘webView’. This object is a View type that displays web pages or online content. This is used so that our HTML file can be used on the android view.



The ‘super.onCreate(savedInstanceState)’ method is used to initialise the activity. The savedInstanceState saves the state of the activity when the orientation changes or a force quit of the application takes place. This way when the activity is started the previous activity state is resumed. A new View object is declared as decorView. The object is defined as the windows decor view by calling getWindow (). getDecorView ().



An integer variable named uiOptions is declared and defined as a series of flags for setting UI elements. The flags set the following attributes respectably, hiding the navigation bar, activating full screen and setting immersive mode. The uiOption is them passed into the decorView objects setSystemUiVisibility(uiOption) which activates all these flags.



The activity content is set from a layout resource xml file from setContentView (). The orientation of the application is set to portrait by passing the Activity Info class’ screen orientation flag into setRequestOrientation () method. The webView object is then set to webview1 id to set the webView layout attributes using findViewbyId (). The return value of this method is casted to WebView.

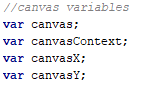


The webView settings are then accessed to enable JavaScript code on this WebView object using the about method calls. The webView is then used to load the gameHTML.html file by using the loadUrl () method and passing in the android asset folder location followed by the file name.

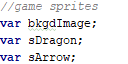
# JavaScript – gameJS.js

This file handles all the interactivity, rendering, updating and game logic involved for creating the game. This section will first go over each global variable that is used within the code then each method used when the script is executed.

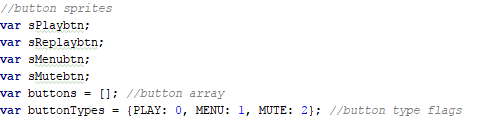
## Global Variables



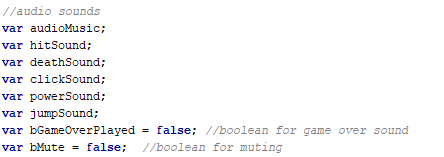
These variables are for handling all the canvas elements that are required for the creation of the application. The ‘canvas’ variable is used for accessing the declared canvas element in the HTML file. The ‘canvasContext’ variable is for accessing the 2D canvas context from the ‘canvas’ object that will be defined in the ‘load ()’ method. The 2 canvas X and Y variables are for storing the x and y coordinates of the canvas element.



These three variables will be defined as new Sprite objects within the ‘initSprite ()’ method that will be called within the ‘init ()’ method. ‘bkgdImage’ will be used as the background of the game that will be drawn over the full width and height of the canvas. The sDragon sprite will hold the player character sprite and the sArrow will be for holding the arrow sprite that will be spawn over multiple instances and act as an obstacle for the player.



These variables are for creating the buttons on the Menu and Game over screen. The first four variables are for the play, replay, menu, and mute button respectively. The ‘buttons’ is an array that will hold these buttons once instantiated. The ‘buttonTypes’ holds 3 enum flags that will determine the type of button each are. This will be used to decide the action that will be executed when the button is pressed.



This group is for all audio related actions. The first six variables are for holding the appropriate sound effect and music of type ‘aSound’. As the game over sound is only required to play once, the Boolean variable ‘bGameOverPlayed’ is declared and initially defined as false. This will be used as a check to see if the sound has been fired and then reset back to false. The ‘bMute’ variable is used to determine if the mute button has been activated or deactivated.



The gameStates enum flags are similar to the button type flags although these are used for determining the different states that the game can be in at a given time during execution. The current state variable is used to determine what elements should be rendered and updated for that given state.



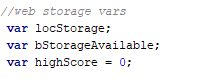
For arrow spawning, the number of arrows that will be spawned is defined by arrowNum and is set to 25. These arrows will be recycled during the game loop once out of frame of the player to mimic infinite arrow spawns. The ‘arrows’ array will be used to hold these arrows. theArrow variable will be used for manipulation and rendering of the chosen arrow in the array and arrowVelx holds the initial x velocity of the arrows.



These three variables are used for creating basic gameplay. The dead Boolean value determines if the player is dead and executes the appropriate code. The score and lives are float and integer values that keep hold of the players score and the amount of lives they have left before game over.



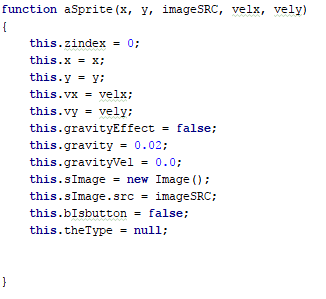
The startTimeMS float variable will hold the elapsed time of the program in milliseconds which will be used for updating elements within each frame. The lastPt holds the last touch location on the screen for both x and y coordinates.



For accessing web storage API, a variable to hold the localStorage object is needed for access. A Boolean is declared to determine if the local storage is available in the current browser window. A high Score float variable is also declared that will hold the score local to the script.

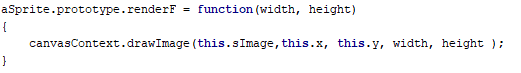
## Objects

**Sprite Object**

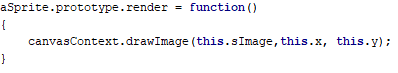
****

The sprite object is used for rendering various images on to the canvas. The Sprite object’s prototype takes in a value for the x and y coordates, an image source file for the sprite and a velocity value for both x and y dimensions. The zindex variable is used for rendering on to the z axis and will act as a layer ordering value this is default set to zero. The x and y values are set to the passed in parameter values from the prototype. This goes the same for the velocity variables. Some objects ay be affected by gravity so a Boolean that is set to false on default is declared and defined. The ‘gravity’ force value that will be used calculate the ‘gravityVel’ is set at a 0.02 value. The ‘gravityVel’ will be set at 0.0 by default. The Sprite object declares and defines a Image variable as a new Image () object. The Image objects ‘src’ is defined as the passed in imageSRC parameter. As some sprites may be buttons a Boolean ‘bIsbutton’ is set to false which will determine the Sprites relationship with UI. The button type is also set to null as this will be defined when the object is instantiated.

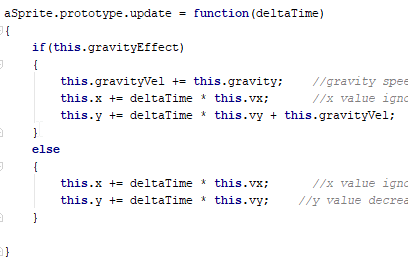
**Sprite Prototype Functions**



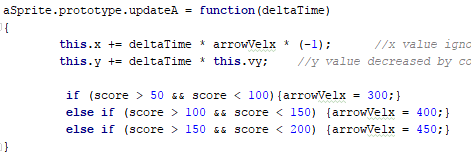
To render a sprite requires a specific width and height when rendering, a render prototype that overrides the canvas context ‘drawImage’ function is created. This prototype function ‘renderF’ takes in two parameter values – width and height. Then when the drawImage function from the canvas context object is called that passes through the image object, the x and y coordinates and finally the desired width and height variables that were passed into the method parameters. This allows a sprite to be drawn onto the screen with specific dimensions.



A simpler render prototype function named ‘render’ is used that will render the sprite with the dimensions of the source file. This render function is used more frequently than the latter. This takes no parameters and passes through the Image and x y coordinates which were defined during initialisation of the object.

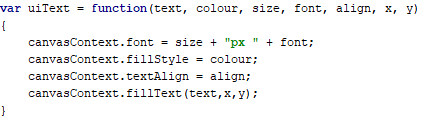


For updating sprite objects positioning every frame a function named ‘update’ takes in a float variable named ‘deltaTime’ which is the time that has elapsed from the last frame update. Within the scope of this method an if statement is first made that will check if the current object is under the effect of gravity. If so then the gravityVel variable is set to equal its value plus the ‘gravity’ force variable. As this method is called every frame then this will create a effect if uncreasing speed when falling. The x coordinates of the sprite is then set to its current value plus the deltaTime value multiplied by the x velocity that was declared. The same code is implemented for the y coordinate expect the multiplication of delta time and velocity is added by the gravity velocity due to gravity effecting the y position of the sprite. If there is no gravity effecting the object, then the same x and y positional increase code is implemented without the gravity modifiers.



Another update prototype function is defined specifically for the arrow sprites. This function is names ‘updateA’. The scope of this function includes the same x and y positional code but the line for the x is modified by the ‘arrowVelx’ velocity variable multiplied by -1. This is so the arrow travels at its appropriate speed and direction as the -1 multiplication makes it fly towards the left of the screen. The if-else statements bellow that code determine the speed of the arrow based on the players current score. This is so give the game an element of increased difficulty and enjoyment.

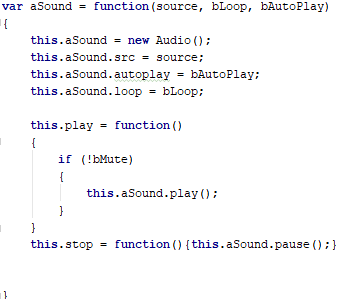
**uiText Object**

****

The uiText object is used for accessing the canvas’ font and text features for rendering text onto the screen. The object constructor parameters take in a text variable which is the string of text that is required to be printed. The colour is a colour value in either RGB (0,0,0) or hex (#eee). This will determine the colour of the font. The size parameter is a size in pixels (px) that the font should be rendered. The font takes in a font family name in the form of a string. The align value is a enum flag which determines the positional alignment of the text e.g. centred, end, start etc. Finally, the x and y are the positional coordinates where the font will be rendered.

Within the scope of the constructor, the canvasContext object is used so that in canvas’ text and font functions can be utilised. First the canvasContext font variable is set to equal the size parameter plus the font as these will determine the size and style of the text. The fillStyle variable is then set to equal the colour parameter. When rendered the text will be styled according to the colour that was passed through. The textAlign value is set to the alignment parameter value. To finish the constructor function the fillText () method is called. This takes in the text value and the x y coordinate values. This function then uses these parameters and the values defined above and renders the text on to the canvas element.

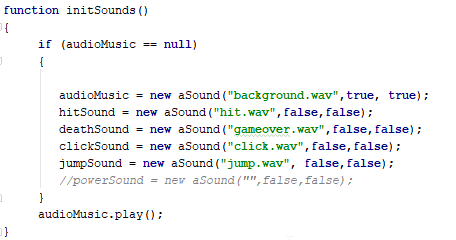
**Sound Object**

****

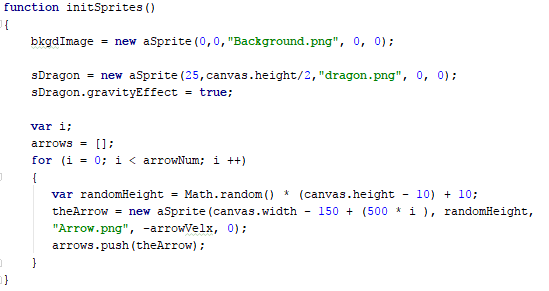
The sound object is made for handling sound firing and looping for both sound effects and theme music. The constructor function takes in 3 variables. First a source file name for the audio clip, then two Boolean variables for looping and auto playing. The scope takes the sound object and defines it as type Audio from the built-in window audio API. The sound object’s ‘src’ is then set to the parameter source value to obtain the clip. The sound object’s auto play and loop Booleans are set to the parameter Boolean values. Two functions are declared and defined within this constructor for both playing and stopping the audio. The play () method first checks to see if the bMute variable is not set to true. If so then the Sound object’s play function is called. Similarly, the stop () method calls the objects pause () function.

## Initialising Objects

This section will go over the initialising methods used for creating all the objects when the main ‘init ()’ function is called. These were split up into individual methods for organisational purposes. The functions that will be discussed are ‘initSprites ()’, ‘initSounds ()’ and ‘initButtons ()’.

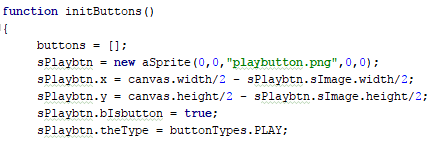


InitSounds takes in 0 arguments and returns void. The scope first consists of an if statement that checks that the audio music variable is equal to null so that the audio is not duplicated when the game is reset. If this is true then the music, hit, death, click and jump sounds are initialised with their appropriate source file and auto play and loop Booleans. The only sound object that is required to auto play and loop is the music sound. Outside of the ‘if’ scope the music object is then set to play.

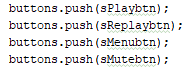


The init sprites method handles all the initialisation of the sprites and images within the game. The method also takes in 0 parameter values and returns void. First the background and dragon sprites are defined with their appropriate positional and dimensional values. The dragon sprite’s gravity effect bool is set to true to allow gravity to affect this object.

For initialising the arrow sprites, the arrow array is first cleared by re defining the array as empty. A ‘for’ loop is then started that iterates through the arrow number variable. A random height variable is declared and is set to equal the return value of the Math.random () function, which returns a random float from 0-1, multiplied by the canvas height. Arbitrary numbers are added and subtracted to some of the values to tweak the positions. The ‘theArrow’ variable is defined, and the width is set to the canvas width plus an arbitrary number (500) times the ‘i’ iterator value. This allows the arrows to appear in order and not all at once. The appropriate height and velocity is also passed into the Sprite constructor. The initialised arrow is then pushed onto the end of the arrow array by calling the array’s ‘push ()’ method.



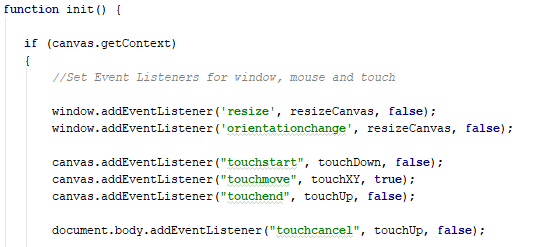
The init buttons method is like the init sprite method. The buttons array is first cleared by setting it to equal an empty array. First the play button is constructed, then the x and y values are set by using the canvas width and height, and the image width and height so that the centre of the sprite is at the centre of the canvas. The button Boolean is then set to equal true and the type variable is set to the PLAY flag. The following code is repeated for all the other buttons with their desired positioning and types.



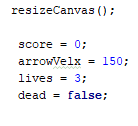
At the end of the function all the buttons are pushed onto the buttons array using the same method as the arrows in the previous init function.

## Initialising the Game

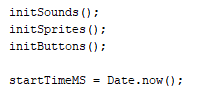
To initialise the game the init () function is called when the load () function is called on execution.



The init function first checks if the canvas context exists as this is required for rendering sprites and checking touch events. Within this scope, all the event listeners are added to the window, canvas and DOM objects. The window will call the resizeCanvas function when the resize or orientation change events are called. The last bool parameter is set to false for both these listeners as the event is only required to be executed in the ‘bubbling’ phase. Event listeners are added to the canvas for touch start, touch move and touch end. The handlers are methods that handle touch events that will be discussed later in this document. The DOM object receives an event listener that calls the touchUp button when the touch cancel event executes.



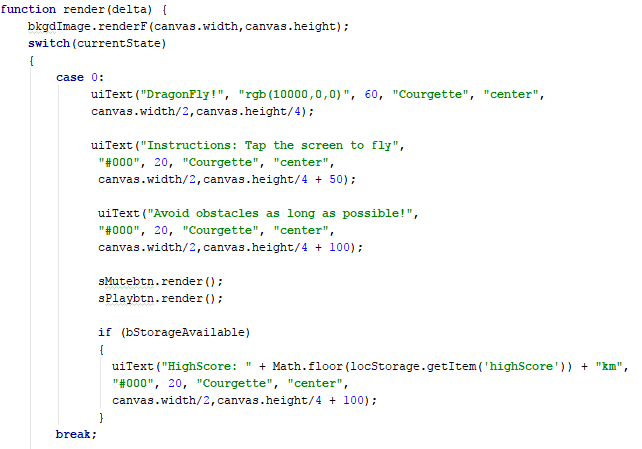
The resize canvas method is then called which sets the canvas dimensions variables to equal the inner width and height of the window object. The score, arrow velocity, lives and dead Boolean are all set to equal their default value. This is for replay purposes within the game.



All the init methods for sound, sprites and buttons are then called and the startTimeMS is then set to equal the elapsed time using Date.now () static function.

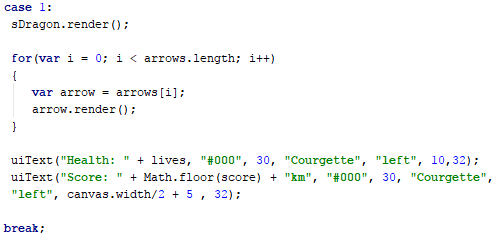
## Rendering

The main render function that handles all of the drawing of objects and elements consists of a switch case for determining what to draw based upon the current state the game is in. The section will mostly make calls of functions that are already describes in previous sections of the document.

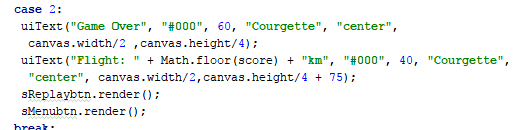


The render takes in the deltaTime variable as an argument as it will be called each frame. Within the scope of the render function the background image called the render prototype function and is set o the width and height of the canvas as this needs to span across the full screen.

Within the switch statement, the first case that is tested upon is 0 which corresponds to the MENU enum flag. For text rendering, the title and instructions are rendered at their appropriate size and style. The Mute and Play buttons are then rendered using the standard ‘render ()’ sprite prototype function. An if statement is made to check the storage available Boolean. If this is true, then the latest high score is rendered upon the canvas.



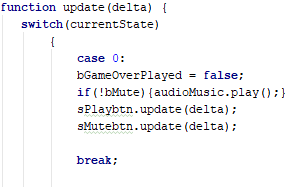
For the ‘GAME’ state. The dragon is rendered first. Then a for loop iterates through the length of the arrow array, rendering each element. The score and health text elements are rendered also. The score variable is used to display the score, which is rounded to the largest integer value using the Math.floor () function.



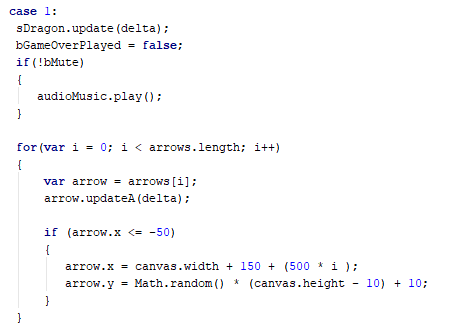
For the game over state. The end screen message is displayed along with the players score. The replay and menu buttons are then rendered.

## Updating

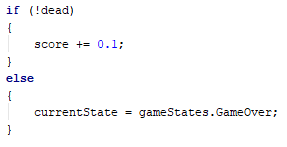
The update function is a similar layout to the render function, using the current state as a switch statement case.



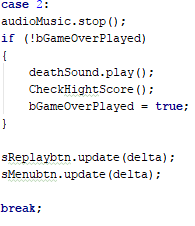
For the menu state, the game over bool is set back to false. The music sound is then set to play if the mute button is not activated. Both mute and play buttons are both set to update from the deltaTime argument.



Within the game state, the dragon is first updated using the update function prototype. The arrow array is iterated through using a for loop and each element in the array is updated using the updateA () method which is was described previously. A check is then made on the position of the arrow. If the arrow’s x positional value is less than or equal to -50 then the appropriate code for resetting the position is made (same code as the arrow init function).



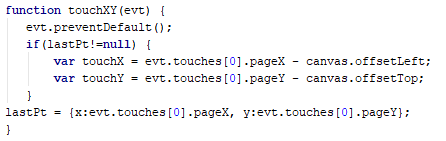
A simple if-else statement is made for increasing the players score by 0.1 each frame is the ‘dead’ Boolean value is not true. If not, then the current state is changed to the game over state.



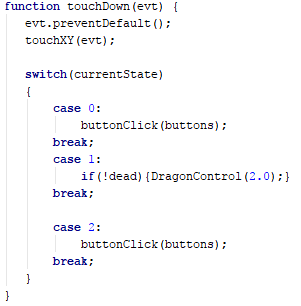
The game over state stops the main theme using the object’s stop () function. If the bGameOverPlayed Boolean is false, then the death sound’s play function is called. The score is check against the saved high score using the CheckHighScore () method. The bGameOverPlayed is then set to true so that it cannot be fired again in this state. The replay button and menu button update functions are called.

## Touch Input

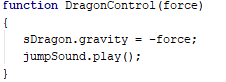
The touch input is handled by three functions ‘touchUp’, ‘touchDown’ and ‘touchXY’. These methods are handlers for the event listener related to touch events. All these methods take in one parameter value which is the event.



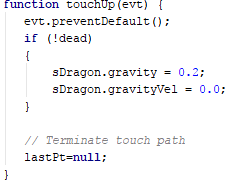
The touchXY method first calls the events’ preventDefaults function which ensures that the default action of an event is not triggered. The lastPt variable is then checked that it does not equal null. If so then two X and Y variables are declared and defined from the last touches x and y positions on the page minus the offsets of the canvas. The lastPt variable is then defined of having and X and Y value flag which is set to equal its corresponding page position.



The touch down function also calls the prevent defaults function. The touchXY function is then called so that the touch position data is recorded. A switch statement is made to handle what methods are called when the screen is touched depending on the current state of the program. The game is at the menu or game over state then button clicks are checked by calling the buttonClicks(buttons) function which takes in the buttons array as an argument. This function will be discussed later in the document. If the game is in the game state then if the player is not dead then the dragon control function is called that takes in a force parameter which is set at 2.0.



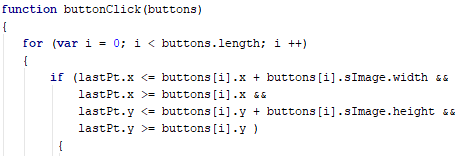
The dragon control function simply sets the gravity to equal the minus value of the passed through argument and the jump sound is set to play.



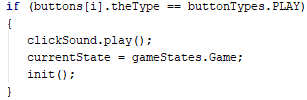
Th touch up function resets the code called in dragon control by setting the modified variables back to their original values. The lastPt is set to null so that the touch path is erased.

## Button Clicks

The buttons are checked by the buttonClick(buttons) function which takes in the array of buttons. This function is called whenever the player touches the screen in bothe the menu and game over screen.



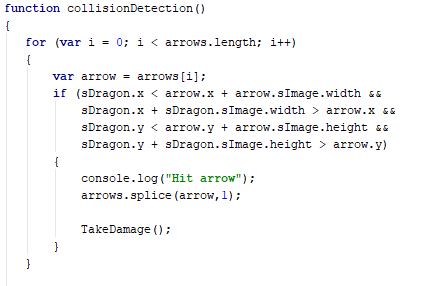
The button array is iterated through using a ‘for’ loop. For each button in the array, the last touch position is tested against the button position and its dimensions to ensure that the touch has been clicked inside or on the button sprite.



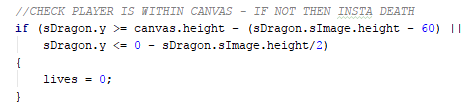
If so then the current button’s type is checked. Depending on the buttons flag type, the appropriate code will be called. The same checks occur for each button type. This ensures that each button carries out specific tasks when clicked.

## Collision Detection

Collision detection must be checked between the arrow sprites and the dragon, or the dragon and canvas boundaries, so that the player can lose a life whenever they touch. This is handled within the collisionDetection () method which handles these checks. The collisionDetection method is called within the gameLoop () method.



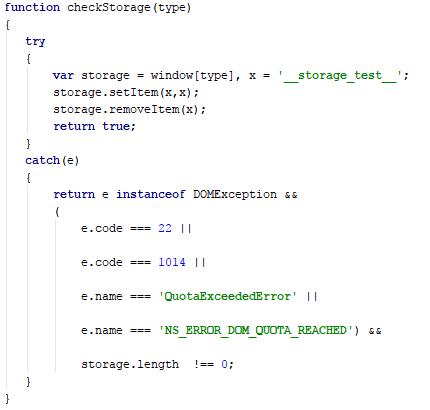
The collision detection method first iterates through the arrow array, the position and dimensions of the dragon are checked against the arrow’s. The they are overlapping then the arrow is removed form the array by calling the splice () function from the arrows array. This stop the arrow from being updated and rendered.



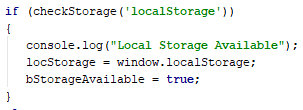
Another check is made for the dragon being within the canvas dimensions. This check the dragon’s y position does not increase further than the canvas height. If so then the lives are set to zero causing instant death.

## Web Storage API

The high score of the player is required to be stored locally on the device they are using. The Web Storage API is used for this. A check is first made on the storage to see if it is available on the current browser the game is running on. A ‘type’ argument is taken in this function as the storage can be local or session type.



A try – catch statement is made which first creates a local storage object and is set to equal the type. A key is also given, illustrated by the x value. Data is then stored on the storage object using the setItem method. Which takes in the key as both the data’s key value and the data itself. The item is then removed from storage by calling removeItem () from the storage object. If no exceptions have been thrown then the function returns – true. If an exception is caught, then an error is returned from error codes from Mozilla and other browsers shown above.



Within the load () function an if statement is made that calls the check storage function. Here, local storage type is needed so that is what will be checked in the parameters. If true is returned, then the ‘locStorage’ variable is set to equal a local storage object. The Boolean for storage available, ‘bStorageAvailable’, is set to true.

When the game over state is reached, the CheckHighScore () function checks if the score is greater than the return of localStorage.getItem(‘highScore’), if so then the setItem method is called that sets the highScore data to the score variable.

## Game Load and Loop

For the game to be loaded when the HTML is executed, the load () function is called. This function sets up all the required elements of the game. Most method calls have already been discussed, however, it is in these methods that they are executed.

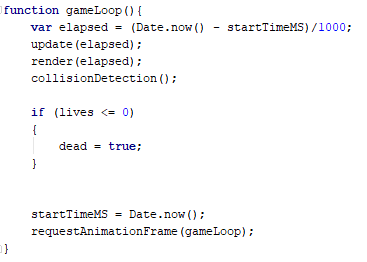


First the canvas variable is set to equal the gameCanvas element that is identified in the HTML file. This is taken by calling getElementById and passing through the id. The canvasContext variable is then set to the canvas objects’ 2D context.

After this, code for checking and initialling local storage is called. This code is stated in the previous section.



The init () function then initialises all elements and the currentState is set to menu. After this the game loop begins by calling the gameLoop () function.



The game loop starts by creating the elapsed time variable. The update and render functions are called, and the elapsed time is passed through as an argument which acts as the functions deltaTime. The collision detection is also called. An if statement checks that if the lives is less than or equal to 0 then the dead Boolean should be set to true. The startTimeMS is then set back to Date.now (). Finally, the browser is told that the program wishes to perform an animation by calling requestAnimationFrame ().

# Bibliography

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

<https://www.w3schools.com/graphics/game_sound.asp>

<https://developer.mozilla.org/en-US/docs/Learn>

<https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial>

<https://stackoverflow.com/questions/tagged/javascript>

## Asset References

**Sprites-**

<http://legendarywars.wikia.com/wiki/File:Arrow-fire-hd.png> - arrow

<https://mobilegamegraphics.com/pvpaterno/GIF/dragon_flying.gif> - dragon

<https://i.pinimg.com/736x/c9/8b/b5/c98bb5f15ddbc0b3e1d2150166eb046a--wallpaper-iphone-phone-wallpapers.jpg> - background

<https://pngtree.com/freepng/cartoon-mobile-gaming-buttons-vector-material_821997.html> - buttons

**Sounds –**

<https://freesound.org/>