Mobile Games Development 1 Coursework: Code Explanation

**Name:** Chun Ting Matthew Wan

**Student number:** S1625394

**Course:** Software Development (Computer Games)

**GitHub link:** <https://github.com/Matt235789/Mobile-Game-Development-1-Coursework>

Disclaimer:

*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*.

*Chun Ting Matthew Wan.*

Functions

By using window.addEventListener the function main() will be called when the page is loaded. This will load the main menu for the game.

*window.addEventListener("load", function ()*

*{*

*main();*

*});*

function main()

This function is used to create a canvas for the main menu. It first sets the variable “gameLoaded” to false to allow other functions to know that the main menu is active and not the main level. The width and height of the canvas is then set. Once this is set, all the UI elements such as the background, score text, font style, font size are all set and drawn onto the canvas. The variable “scoreCookie” will then read the cookie stored on the browser and set itself to equal the cookie. It will then check if “scoreCookie” is greater than “highScore”. If “scoreCookie” is greater than “highScore” will be updated to the value of “scoreCookie”. The background music for the game is also set and played on a loop.

*scoreCookie = document.cookie;*

*if (scoreCookie > highScore)*

*{*

*highScore = scoreCookie;*

*}*

function start()

This function is used to set up all everything needed for the main level such as image files, audio files, multiple different variables and more. The width and height of the canvas is first set. All the image and audio variables are then linked to their file sources. The audio for the helicopter will also begin to play on a loop here. The initial platform used in the game will also be created here.

Once these are all set, all the variables that are used in the game will be reset to its original value. This is so that when the main level is replayed it won’t be using values from the previous time it was played. Values such as the players position, score, missile speed, shield cooldowns and much more will all be reset. The length of certain arrays such as the coin array or the missile array will be set to zero here. This is so that previous elements already created in the array from the previous time the level was played will be removed and won’t be displayed on the canvas. This will also save computing resources as the previous elements from the arrays won’t be used.

The final part of this function will run an if statement to check if “startspawn” is true. If “startspawn” is true, it will begin using timer methods such as setInterval, setTimeout and clearTimeout. The setInterval methods will call a function after a set amount of time and will continue calling that function. For example, it will call the “spawnMissile” function after 1.3 seconds and will also call this function every 1.3 seconds afterwards. The clearTimout method is used to prevent a function from being run from the previous time it was played. The setTimout method will call a function after a set period but it will only run the function once. Once all these methods are called it will also set “startspawn” to false.

*if (startspawn == true)*

*{ console.log("start spawn");*

*intervalCoin = setInterval(spawnCoin, 2500);*

*intervalFuel = setInterval(decreaseFuel, 1500);*

*intervalAddGas = setInterval(spawnGas, 6500);*

*intervalMissile = setInterval(spawnMissile, 1300);*

*intervalMissileSpeed = setInterval(speedMultiplyer, 10000);*

*clearTimeout(clearshield1);*

*clearTimeout(clearshield2);*

*setTimeout(spawnCity, 1);*

*startspawn = false;*

*}*

function update()

This function is used for running the main level of the game and updating all the different values. This function starts of with if statements which is used for the controls of this game. It will check if certain key codes are pressed and update the position of the player or activate an ability. The control for the ability will also check if there is a cooldown on the ability before calling the functions to activate the ability and enable the cooldown.

*if(keys[13] && shieldCooldown == false)*

*{//enter key for shield ability*

*cooldown();*

*enableShield();*

*}*

The control to fly up will also check if the player fuel value is greater than 0 before flying up.

*if (player.y > 50 && fuel > 0)*

*{*

*//how much the helicopter will go up*

*var flyheight = 2;*

*player.velY = player.velY - flyheight;*

*}*

Next all the UI elements are drawn onto the canvas such as the fuel bar, shield icons, score text and more.

A for loop for an image of a city is run. Each time the loop is gone through, it will draw an image of a city at the end of the x position on the canvas. The image will then constantly move on the x axis to the left until it hits the start position of the canvas which will then spawn in another city image at the end of the canvas. This gives an illusion to the player that they are moving past a city.

*for (var k = 0; k < cityBack.length; k++)*

*{*

*ctx.drawImage(cityImg, cityBack[k].x, cityBack[k].y, canvas.width, 600);*

*cityBack[k].x = cityBack[k].x -0.5;*

*if (cityBack[k].x <= 0 && cityBack[k].x >= -1)*

*{*

*spawnCity();*

*}*

*}*

The variable “shieldCooldown” is constantly checked in this function. If there is a cooldown for the shield a red cross will be drawn over the image of the ability.

*if (shieldCooldown == true)*

*{*

*ctx.drawImage(redCross, 65, 89, 50, 52);*

*}*

An if statement is also used to check if the shield ability is being used. If “shield” is true an image of a shield bubble will be placed onto the players current position.

*if (shield == true)*

*{*

*ctx.drawImage(shieldBubble, player.x, player.y, 80, 80);*

*}*

The function animationFrame is called to update the sprite of the helicopter. The updated sprite of the helicopter is then drawn.

*animationFrame();*

*ctx.drawImage(img, spriteWidth\*frameX, spriteHeight\*frameY, spriteWidth, spriteHeight, player.x, player.y, player.width, player.height);*

A for loop for the coin collision is then run though. In this for loop the position of the coins will be constantly updated so that they move at a set speed to the left on the x position. The speed of the coin moving can also be changed here.

coin[k].x = coin[k].x -1.8;

It will then check if the position of the players position is within the position of the coin plus its height and width. If the player has met these conditions, then it will be seen as a collision between the player and the coin.

*if (player.x > coin[k].x && player.x < coin[k].x + coin[k].width && player.y > coin[k].y && player.y < coin[k].y + coin[k].height*

*|| player.x + player.width\*0.5 > coin[k].x && player.x + player.width\*0.5 < coin[k].x + coin[k].width && player.y > coin[k].y && player.y < coin[k].y + coin[k].height*

*|| player.x + player.width > coin[k].x && player.x + player.width < coin[k].x + coin[k].width && player.y > coin[k].y && player.y < coin[k].y + coin[k].height)*

If collided the coins position will then be moved off canvas and a coin collected audio will be played. The score of the player will also be updated to reflect on this coin being collected.

A similar for loop for the collision between the gas can and player is also created. In this for loop the gas cans position will be updated to move across the screen at a set value. Upon collision with the player the gas can position will be moved off the canvas, fuel will be added to the fuel counter and audio will be played for the gas can collection.

A for loop for the missile is then run. Like the coin collision and gas can collision for loop, this will update the missiles position to be moving across the screen. The speed of the missiles can also be changed here. It also runs a similar if statement to determine if the player has collided with the position of the missile. However, there is an additional condition that is checked during the collision. The “shield” variable is checked to see if the players shield is set to true. If false, the missiles position will be updated to moved off the canvas, an explosion audio will be played, score will be deducted, and the player will lose some fuel. The amount of score and fuel that the player loses can also be changed here. If true, the player will not lose any fuel or score.

The final if statement in this function checks if the players y position is less than a certain amount. If the players y position is below this amount it will create the variable “fallspeed” and apply it to the players current velocity which will simulate the player falling. If the players position goes past the specified position it will run the gameOver function.

*if (player.y < 620)*

*{*

*var fallspeed = .8;*

*player.velY = player.velY+fallspeed;*

*}*

*else if(player.y >= 621)*

*{*

*gameOver();*

*return;*

*}*

function cooldown()

This function sets the variable “shieldCooldown” to true and then creates a variable to equal a setTimeout method. This variable uses the setTimeout method to set “shieldCooldown” back to false after a set amount of time. By creating this variable, it allows the setTimeout method to be accessed in another function and cleared which will prevent the function from being run. The timer in the setTimeout method controls how long the cooldown for the shield ability will last.

*shieldCooldown = true;*

*clearshield1 = setTimeout(function(){shieldCooldown = false}, 10000);*

function enableShield()

This function is like the cooldown function but uses the “shield” variable instead. When called it will set the “shield” variable to true. The timer in the setTimeout method controls how long the players shield will last before setting the “shield” variable back to false.

function decreaseFuel()

This function uses an if statement to decrease the amount of fuel. It checks if the fuel is greater than 0 and if it is it will deduct a certain amount of fuel. This function works with a setInterval method that is called in the start function so that every so many seconds this function will be called and deduct fuel.

*if (fuel > 0)*

*{*

*fuel = fuel - 5;*

*}*

function speedMultiplyer()

This function is used to increase the speed of the missile. It sets the speed of the current “missileSpeed” equal to the current “missileSpeed” multiplied by the variable “missileSpeedMultiplyer”. This function also works with a setInterval method that is called in the start() function so that the speed of the missile will gradually increase over a period of time. It will also write the missile speed to the console.

*missileSpeed = missileSpeed \* missileSpeedMultipliyer;*

*console.log("Missile speed: " + missileSpeed);*

function spawnCity()

This function creates an image of a city at the end of the canvas. It pushes a new item to the cityBack array with specified position and image size. It also writes to console each time it creates an image. This function is called in the update function whenever an item in this array reaches the start of the canvas. This creates the illusion that the city goes on forever as the width of this image is also the width of the canvas so whenever the end of this image reaches the start of the canvas a new image is created.

*console.log("spawned city");*

*cityBack.push*

*({*

*x: canvas.width,*

*y: 0,*

*width:80,*

*height:600*

*});*

function spawnCoin()

This function is used to randomly spawn in the coins in the game. It starts by creating a variable called “randomYcoin”. A random number is then generated using Math.random which “randomYcoin” will be equal to.

var randomYcoin;

randomYcoin = Math.floor(Math.random() \* 350) + 150;

Math.random will generate a random number between 150 and 450. As Math.random will generate a random number that includes decimal places Math.floor is used to round down this number to a whole number. A new coin is created in the “coin” array with a set size and x position. However, it uses “randomYcoin” for the y position.

*coin.push*

*({*

*x: canvas.width,*

*y: randomYcoin,*

*width:40,*

*height:40*

*});*

This function works with both the start function and the update function. A setInterval method is used to call this function every so many seconds and the x position of the images are updated in the update function. This means that coins will be randomly generated on the y position and move across the screen on the x position allowing the player to collect these coins throughout the game at random positions that are on the canvas.

function spawnGas() and function spawnMissile()

This function operates just like the spawnCoin function. It creates a variable and uses the Math.random method to generate a random number. It then creates a new item to an array using a set size, x position and the random Y position. It also uses the setInterval method to call this function and update function to update the image position.

function gameOver()

This function is used to end the main level of the game. It is called when the game meets a condition to end the game. When this condition is met the canvas is cleared and a game over screen image is drawn. Text will also be drawn on the canvas to display the score achieved by the player.

*ctx.clearRect(0, 0, width, height);*

*ctx.drawImage(gameOverScreen, 0, 0, width, height);*

*ctx.font = "80px Arial";*

*ctx.fillText(score, 590, 310);*

The clearInterval timer is then called to prevent certain functions from begin run. This prevents the current timers from continuing to run the next time the update function is called.

*clearInterval(intervalCoin);*

*clearInterval(intervalAddGas);*

*clearInterval(intervalFuel);*

*clearInterval(intervalMissile);*

*clearInterval(intervalMissileSpeed);*

A cookie is created which will store the current score. An expiry date for this cookie is not specified so this means that the cookie on default will be destroyed upon the session finishing. The audio for the helicopter is also paused.

*document.cookie = score;*

*helicopterAudio.pause();*

requestAnimationFrame(update);

This method is used to call the update function repeatedly which creates a game loop. This means that at each loop every variable such as the players position that has been updated will be redrawn on the canvas at a new position.

document.body.addEventListener(“keydown, function (e)

This event handler is used to handle the input that is received. It will store the input into array variable keys.

This if statement below checks if the space key is, the players y position is equal or greater than a set position and “gameLoaded” is true. This will then clear the canvas and call the main function to load the main menu. This means that the space key will only work when the conditions that the game is over is met.

*if (keys[32] && (player.y >= 621) && gameLoaded == true)*

*{*

*console.log("space pressed");*

*ctx.clearRect(0, 0, width, height);*

*main();*

*}*

The next if statement is used for the “b” key input. If the “b” key is pressed, “gameLoaded” is false and “instructionsLoaded” is false, the canvas will be updated to paint an image of the instructions image. The “instructionsLoaded” variable will also be set to true.

*if (keys[66] && gameLoaded == false && instructionsLoaded == false)*

*{*

*ctxMain.drawImage(mainMenu2, 0, 0, width, height);*

*instructionsLoaded = true;*

*}*

An else if statement is used to continue on from the previous statement. It checks for the same conditions, however if the variable “instructionsLoaded” is true then it will set “gameLoaded” to true, “instructionsloaded” to false and call the start and update function. This will begin the main level of the game.

*else if (keys[66] && gameLoaded == false && instructionsLoaded == true)*

*{*

*console.log("b pressed");*

*gameLoaded = true;*

*instructionsLoaded = false;*

*start();*

*update();*

*}*

function animationFrame()

This function is used to animate a sprite sheet. It starts by creating a timer for the current time and the time elapsed. If the timer hits 0 it will set it to equal to the max time for a frame. It will then move onto the next sprite on the x position on the sprite sheet. If there are no more sprites on the current row, it will then next to the move row on the sprite sheet. Once the current frame variable reaches the max frame, the variables for the frame, x position of the frame and y position of the frame will be set back to 0 to restart the animation.

*var elapsed = (Date.now() - startTimeMS)/1000;*

*startTimeMS = Date.now();*

*//only update frames when timer is below 0*

*frameTimer = frameTimer - elapsed;*

*if(frameTimer <= 0)*

*{*

*frameTimer = frameTimeMax;*

*frameX++;*

*if(frameX>frameXMax)*

*{*

*frameX = 0;*

*frameY++;*

*//end of row, move down to next row in sheet*

*if(frameY>frameYMax)*

*{*

*frameY = 0;*

*}*

*}*

*frame++;*

*//reset frames to 0 in event that there are empty spaces on sprite sheet*

*if(frame > frameMax)*

*{*

*frame = 0;*

*frameX = 0;*

*frameY = 0;*

*}*

*}*

Twitter follow button

An anchor element is created to add a to add a twitter follow button onto the web page. The attribute href will point to the twitter profile URL you want the user to be directed to. The follow button parameters can be changed by editing the data attributes. The attribute data-show-count is to set true to display the amount of follows the twitter profile has. The attribute data-size is set to large to increase the size of the follow button.

*<a href="https://twitter.com/CaledonianNews?ref\_src=twsrc%5Etfw" class="twitter-follow-button" data-show-count="true">Follow @CaledonianNews</a><script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>*

Sources for artwork:

Shield bubble: <https://subsetgames.com/forum/viewtopic.php?t=27097>

City background: <https://www.dreamstime.com/stock-illustration-seamless-cartoon-city-background-vector-image92796336>

Arrow keys: <https://www.cleanpng.com/free/arrow-keys.html>

Coin: <https://pngtree.com/freepng/cartoon-coin_161124.html>

Enter key: <https://commons.wikimedia.org/wiki/Category:Enter_key>

Red gas can: <https://www.google.com/search?q=red+gas+can+cartoon+png&tbm=isch&ved=2ahUKEwi37-mxibbmAhUW-hoKHQhABmwQ2-cCegQIABAA&oq=red+gas+can+cartoon+png&gs_l=img.3...12397.13483..13822...0.0..0.64.474.8......0....1..gws-wiz-img.v8gdQVqJT_c&ei=xVL1Xff4D5b0a4iAmeAG&bih=941&biw=1920&hl=en_GB#imgrc=EGbG9oPC3dyJjM&imgdii=XgBCEEKxfpYhNM>

Helicopter sprite sheet: <https://www.hiclipart.com/free-transparent-background-png-clipart-xsjab>

Missile: <https://www.clipartmax.com/middle/m2i8Z5b1d3d3d3G6_how-to-set-use-missile-svg-vector-drawing-of-a-missile/>

Shieldbubble: <https://ya-webdesign.com/imgdownload.html>

Shield icon: <https://icon-library.net/icon/shiled-icon-12.html>

Skybackground: <https://clipground.com/sky-cartoon-png.html>

All audio sourced from freesound.org

References:

Twitter follow button: <https://developer.twitter.com/en/docs/twitter-for-websites/follow-button/overview>