

**Mobile Games Development 1 Coursework: Code Explanation**

**2019-2020**

**Name: Peter Alessandro Lunardi**

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

**Matriculation Number: S1636120**

**“***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*.**”**

**Signed by Student:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_16/12/2019\_\_\_\_\_\_**

Contents

[Overview 3](#_Toc27405435)

[Setup 3](#_Toc27405436)

[Game Loop 5](#_Toc27405437)

[DrawBackground() 6](#_Toc27405438)

[DrawPlayer() 6](#_Toc27405439)

[DrawRocks() 7](#_Toc27405440)

[DrawForeground() 9](#_Toc27405441)

[DrawScore() 9](#_Toc27405442)

[DrawFlake() 9](#_Toc27405443)

[DrawSmoke() 11](#_Toc27405444)

[Start and end screens 12](#_Toc27405445)

[References 13](#_Toc27405446)

## Overview

The game consists of a plane on the far left of the screen and multiple rock objects which appear at the far right of the screen and transition towards the plane. The player controls the plane’s vertical position by pressing any key on the keyboard, or by tapping on a mobile device. The player’s score increases when successfully clearing an obstacle. Randomly placed coins are also available to collect and increase the player’s score. A simple particle system is also used to create snow and smoke particles. A start and end screen are included which display instructions as well as the player’s final score and the high score to beat. Different audio is also included throughout the game. This document aims to explain how each of these features were implemented. A screenshot of the game is shown in figure 1.

A close up of a map

Description automatically generated

Figure - A screenshot of the game

## Setup

The game takes place within an HTML canvas, which is defined in an .html document called game.html and is shown in figure 2.

A screen shot of a computer

Description automatically generated

Figure - game.html

This file sets up the html window and canvas which will be referenced within the JavaScript file using it’s unique ID, “gameCanvas”. The width and height are also defined, as well as the .css file which contains very basic information about the appearance of the canvas including a border surrounding it. Two JavaScript files are included here, one containing the game logic, and the other an imported library used for vector maths.

The game.js file contains the logic for the entire game. It begins by getting a reference to the canvas it will be drawing to by calling “document.getElementById('gameCanvas');” and assigning this to a variable “cvs”. The canvas context is then assigned to a variable “ctx” using “cvs.getContext("2d");”. With these two lines of code, the canvas can now be accessed and drawn to within the JavaScript code.

The next step is to load in all the resources used within the game, which in this case is simply image files and audio files. This is done by creating new variables for each asset and assigning a new Image() or Audio() them. Image() and Audio() both contain a .src parameter which points to the asset’s location within the project, and these are set once the variables are declared. The audio loading is shown in figure 3.

A screen shot of a computer

Description automatically generated

Figure - Loading in audio files

Image loading is identical, although there are many more images used than audio files.

Various variables are then declared which are all used within the game which will be identified and described during their relevant functions.

## Game Loop

The main game loop takes place within a function called “loop()”. This function calls various other functions and then requests an animation frame resulting in the game looping. It also contains some logic for when the game should stop looping, or when the game is over. “loop()” is shown in figure 4. This function is then called when the window is loaded using “window.onload = function{ }”.

A screenshot of a cell phone

Description automatically generated

Figure -loop(), the main game loop

All of the game logic takes place within the eight functions called at the beginning of loop(). Each of these functions results in some image being drawn to the canvas in a position defined by each individual function, so the order in which they are called is important since it determines the order in which the images are drawn to the canvas.

A bool was declared earlier to keep track of whether the game is running, called “gameOver” and this is set to true when any conditions are met which result in the game being over. The first of these conditions is when the player flies out of the canvas bounds. The variable “pY” tracks the player’s Y coordinate within the canvas and a simple check is done to see if this value is above or below the bounds of the canvas. The player’s X coordinate does not need to be checked since it remains constant throughout the game.

This function is also responsible for looping the game’s main audio by simply checking if it has ended, and if it has, playing the audio again.

The “gameOver” bool is then checked, and if the game is not over an animation frame is requested for the function using “requestAnimationFrame(loop). If the game is over then one of two functions is called based on whether the game has restarted or not (checked using another bool, “gameStarted”) and these are responsible for displaying the start or end screen.

## DrawBackground()

The first function to be called within the game loop is “DrawBackground()” which is responsible for drawing the background image for the game to the canvas and is shown in figure 5.

A screenshot of a computer

Description automatically generated

Figure - DrawBackground(), responsible for the background scrolling effect

The background is not static and instead has a horizontal scrolling effect. This is done by drawing the defined “background” image to the screen twice, directly next to each other. Since this image is identical in width to the canvas, this can be done by setting the “x” parameter of “ctx.drawImage()” to be bgX (a variable that tracks the background’s horizontal position and is initially 0) for the first image, and bgX plus the width of the canvas for the second image. This results in one image displaying on the canvas, and the other just barely off to the right-hand side. The value of bgX is then decremented by another value, “bgScrollSpeed”, which determines how quickly the background scrolls, which results in both background images moving to the left of the screen. Once the first image reaches the end of the screen, bgX is reset to 0 and the process repeats, giving a scrolling background.

## DrawPlayer()

This function draws the player’s plane image to the screen and is shown in figure 6.

A screenshot of a cell phone

Description automatically generated

Figure - DrawPlayer(), draws the plane image to the screen

This function simply draws the “player” image to the screen using “pX” and “pY” which define the x and y position of the player respectively. Every frame the y value is incremented by “gravity” which causes it to fall towards the bottom of the screen. If the player is flying, tracked via the bool “isFlying” then the y value is instead decremented by “gravity” resulting in the player flying upwards. isFlying is set to true when the player presses any key using document.addEventListener, and set to false when the player releases that key.

## DrawRocks()

This function is essentially one big for loop which draws the rock obstacles to the screen. It loops through each rock in a previously defined “rocks[ ]” array, which initially only has one rock stored in it, and draws the rock based on various parameters. At the end of the loop a new rock is pushed into the array causing new rocks to spawn over the course of the game. Some of the logic used within this method is shown in figure 7.

A screenshot of a cell phone

Description automatically generated

Figure - DrawRocks(), first half of the for loop

Each rock has 4 parameters; x, y, z, and w. X and Y are coordinates and z and w are randomly assigned and determine what style of rock will be drawn as well as whether the rock will be pointing up or down. The rock being processed has it’s x coordinate decremented by “rockScrollSpeed” which determines how quickly the rocks should scroll. When the rock reaches a certain point on the canvas, in this case 600, some new parameters are defined and passed into a new rock, which is pushed to the array of rocks ready to be processed. This repeats over the course of the game giving an endless supply of randomly placed rocks.

The second half of the for loop deals with collision detection between the rocks and the player and is shown in figure 8.

A screenshot of a computer

Description automatically generated

Figure -DrawRocks(), collision detection

For each rock in the array, it’s x and y positions are compared to the player’s, as well as their corresponding image widths and heights in order to detect when the two images cross over i.e. collide. When they do, the gameOver bool is set to true and the game finishes. Finally, an if statement checks if the rock reaches the far left of the screen, or if the player has avoided the obstacle, and increments the player’s score accordingly.

## DrawForeground()

This function is identical to the DrawBackground function described earlier, except this one draws the foreground images at the bottom of the screen. This is called after the background and rocks have been drawing so that they are drawn on top of them.

## DrawScore()

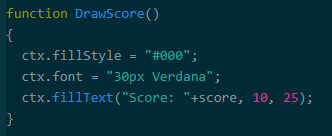


Figure - DrawScore(), draws the playes score to the screen

The next function to be called each frame draws the player’s score to the top left of the screen using the canvas context.

## DrawFlake()

The next function is responsible for drawing the snow particles to the screen, which it does by iterating through an array of “flakes” and drawing each one to the screen. It is shown in figure 10.

A screen shot of a computer

Description automatically generated

Figure - DrawFlake(), draws snowflake particles to the screen

Each “flake” in “flakeArray” has three parameters. A position, a scale and a velocity, represented by “pos”, “s” and “vy” respectively. The function loops through each flake and draws it to the screen based on these parameters. Much like every other entity in the game, the position is decremented every frame causing each flake to fly to the left. When the flake reaches the end of the screen it is sent back to the start with new values for position and velocity. A library called “victor.js” was imported and used for the vector maths required in the particles systems, and vectors were used to model the position and velocity of each flake.

The array “flakeArray” is populated at the beginning of the program shown in figure 11.

A close up of a sign

Description automatically generated

Figure - population of the flakeArray

“flakes” is a simple integer which determines how many particles are present at any time. CreateFlake is an object that stores the parameters of each flake, namely it’s position, scale and velocity, and is shown in figure 12.

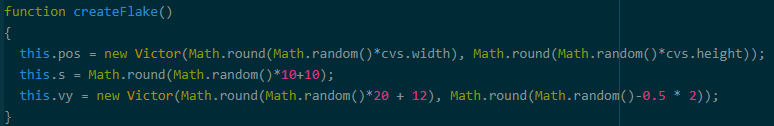


Figure - createFlake(), creates a snowflake particle

The position is just a random point on the canvas and the velocity’s x value is a random number between 12 and 32, since these values produced nice results. The scale is also random and causes each particles to be different in size.

## DrawSmoke()

This is the function responsible for drawing the smoke particles emitting from behind the player. It works nearly identically to DrawFlakes() in that it populates an array of smoke particles and then iterates through each one and draws it to the screen before decrementing its position based on a velocity parameter, shown in figure 13.

A screenshot of a computer

Description automatically generated

Figure - DrawSmoke(), produces smoke particles from the player

The difference with this particle is the way in which its velocity is calculated, and the fact that both the x and y positions are incremented each frame by this velocity. The calculation for the velocity is done in the createSmoke function shown in figure 14.

A screenshot of a cell phone

Description automatically generated

Figure - createSmoke(), creates the smoke particles behind the player

The velocity for each particles is calculated by picking a random point on the far left of the canvas. Using the victor.js library, some simple vector calculations are done to get the vector going from the player, or the particles origin, to a random point on the screen by subtracting these two points. This value is then normalized and scaled to give the velocity of each particles. This results in a cone-like shape emission for the smoke particles.

The final function to be called within the main game loop is DrawCoin(), and this draws the collectable coins to the screen, shown in figure 15.

A screenshot of a cell phone

Description automatically generated

Figure - DrawCoin(), draws the collectable coins

This function does similar things covered in previous functions. It draws an image to the screen based on a variable “coin’s” parameters. The y position is random and the x position is decremented so the coin moves to the left of the canvas. When it reaches the left edge of the canvas, it is transported back to the right edge with new randomised values to give the illusion of a new coin spawning every time. The function also contains some collision detection with the player, and if a collision is detected a sound is played, the coin is transported back to the edge of the screen and the player’s score goes up. The declaration of the coin variable is shown in figure 16.

A screenshot of a cell phone

Description automatically generated

Figure - declaration of coin variable.

## Start and end screens

The start and end screens are created simply by displaying certain text and images to the screen when certain parameters are met, namely the booleans mentioned previously, “gameOver” and “gameStarted”.

The high score is stored in a local storage using localStorage.setItem(x, y), where x is the unique storage id and y is the high score to be placed in the storage. This high score is checked whenever the game is over to see if the player’s score is higher, and if it is, the high score in local storage is replaced with the player’s new score.

## References

Scrolling background: <https://www.geeksforgeeks.org/html5-game-development-infinitely-scrolling-background/>

Particles: <https://www.youtube.com/watch?v=0jfwMEjUIdk>

Game logic: <https://www.youtube.com/watch?v=L07i4g-zhDA>

Images: <https://www.kenney.nl/assets/tappy-plane> , <https://projecthydro.org/developers/hydro-snowflake/>

Audio: <https://freesound.org/people/MrVasLuk/sounds/333203/> , <https://freesound.org/people/sandyrb/sounds/95078/> , <https://freesound.org/people/ProjectsU012/sounds/341695/>