Mobile Games Development 1 Coursework: Code Explanation

*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*.

*Signature*. Text, letter

Description automatically generated

https://github.com/ifrift/Mobile-Games-Development-1-Ritchie.git

Contents

[Introduction 4](#_Toc61879228)

[Object Prototypes 5](#_Toc61879229)

[Functions 5](#_Toc61879230)

[macTroid.name; 5](#_Toc61879231)

[macTroid.x; 5](#_Toc61879232)

[macTroid.y; 5](#_Toc61879233)

[macTroid.width; 5](#_Toc61879234)

[macTroid.height; 5](#_Toc61879235)

[macTroid.hp; 6](#_Toc61879236)

[macTroid.ammo; 6](#_Toc61879237)

[macTroid.img; 6](#_Toc61879238)

[macTroid.img.src; 6](#_Toc61879239)

[macTroid.currentFrame; 6](#_Toc61879240)

[macTroid.frameX; 6](#_Toc61879241)

[macTroid.frameY; 6](#_Toc61879242)

[macTroid.startingFrameX; 6](#_Toc61879243)

[macTroid.startingFrameY; 6](#_Toc61879244)

[macTroid.maxFrame; 6](#_Toc61879245)

[macTroid.maxFrameX; 6](#_Toc61879246)

[macTroid.maxFrameY; 6](#_Toc61879247)

[macTroid.timer; 6](#_Toc61879248)

[macTroid.firing; 6](#_Toc61879249)

[macTroid.velX; 6](#_Toc61879250)

[macTroid.velY; 6](#_Toc61879251)

[function start(); 6](#_Toc61879252)

[function update(); 7](#_Toc61879253)

[function init(); 7](#_Toc61879254)

[startScreenRender(); 7](#_Toc61879255)

[render(); 7](#_Toc61879256)

[levelLayout(); 7](#_Toc61879257)

[levelPart1(); 7](#_Toc61879258)

[levelPart2(); 7](#_Toc61879259)

[gamePlay(); 7](#_Toc61879260)

[playerMovement(); 8](#_Toc61879261)

[crouch(); 8](#_Toc61879262)

[stand(); 8](#_Toc61879263)

[flyingMovement(); 8](#_Toc61879264)

[gruntMovement(); 8](#_Toc61879265)

[animations(); 8](#_Toc61879266)

[spwanBullet(); 8](#_Toc61879267)

[colling(); 8](#_Toc61879268)

[colCheck(); 8](#_Toc61879269)

[deathCheck(); 8](#_Toc61879270)

[animationEnemy(); 9](#_Toc61879271)

[animationPlayer(); 9](#_Toc61879272)

[animationPlayerLeft(); 9](#_Toc61879273)

[renderGameOver(); 9](#_Toc61879274)

[Boss Section 9](#_Toc61879275)

[renderBoss(); 9](#_Toc61879276)

[playerBoss(); 9](#_Toc61879277)

[bossMoving(); 9](#_Toc61879278)

[0 9](#_Toc61879279)

[1 9](#_Toc61879280)

[3 10](#_Toc61879281)

[4 10](#_Toc61879282)

[bossAttack1(); 10](#_Toc61879283)

[bossAttack2(); 10](#_Toc61879284)

[bossReset(); 10](#_Toc61879285)

[bossLayout(); 10](#_Toc61879286)

[bossCollision(); 10](#_Toc61879287)

[winScreen(); 10](#_Toc61879288)

[randomNo(); 10](#_Toc61879289)

[randomAtk(); 10](#_Toc61879290)

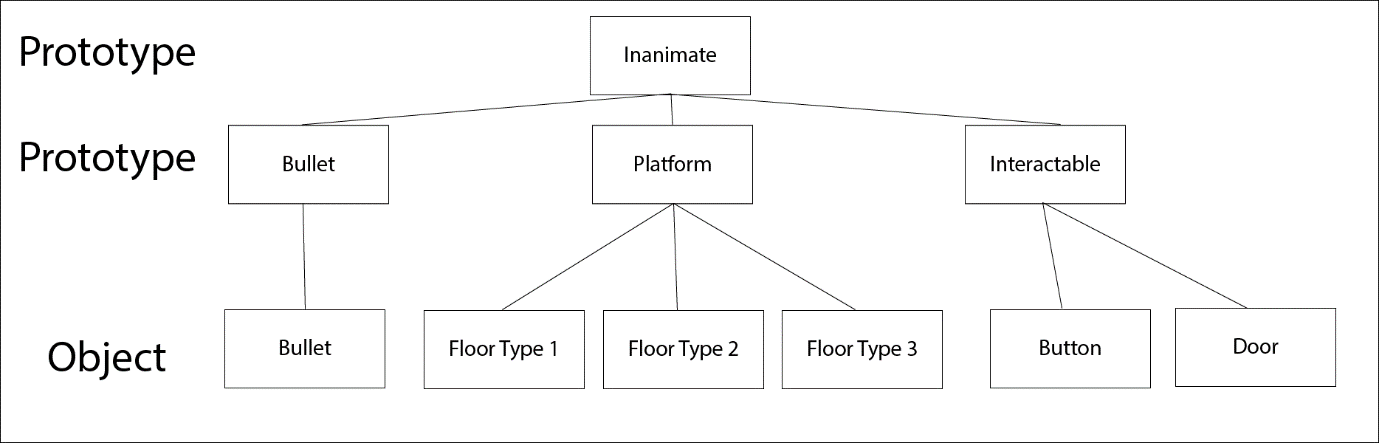
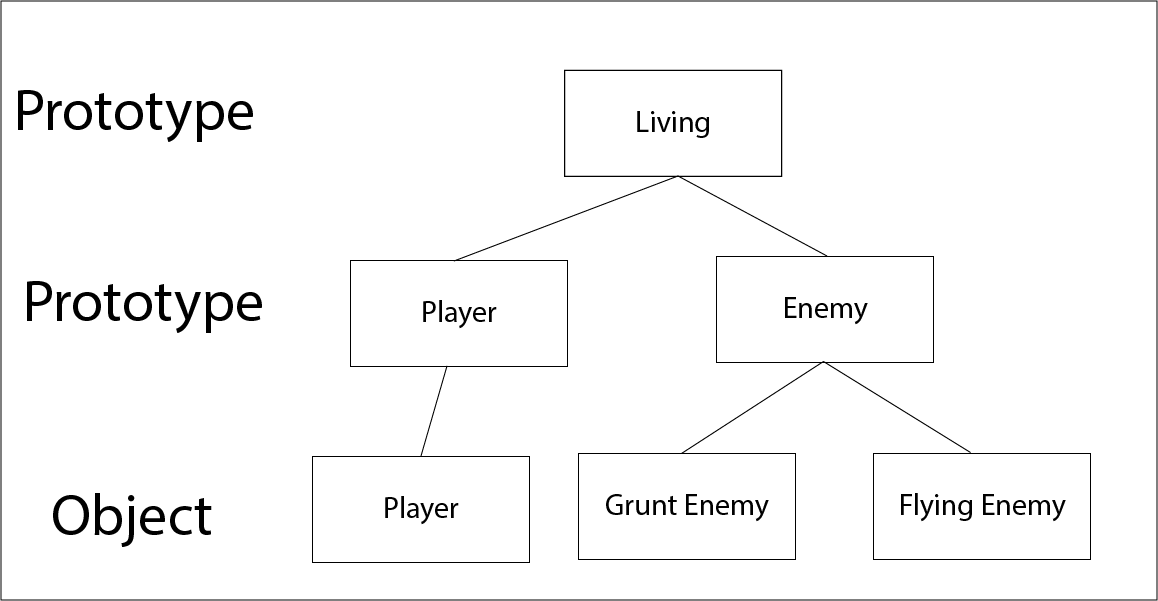
# Introduction

This project aims to create a game using JavaScript. Apart from that there is also a small list of specifications that the game should also meet which are as follows.

* Introduction screen.
* Sounds.
* Player interactivity.
* Good use of game sprites.
* Game background.
* Game UI.
* Game over screen.
* Use of fonts and text.

# Object Prototypes

With my objects I tried to break the down into different categories. The first was to see if the object was alive or whether it was inanimate. I then created prototypes that that inherited from there. From the second line of prototypes, I then created my objects. The below diagrams show be able to illustrate how it is done.



Each object has its own set of variables. Below is a list of variables for the player object and what they are used for.

# Functions

## macTroid.name;

This is the name of the object.

## macTroid.x;

This tracks the x axis position of the object.

## macTroid.y;

This tracks the y axis position of the object.

## macTroid.width;

This tracks the current width of the object.

## macTroid.height;

This tracks the current height of the object.

## macTroid.hp;

This tracks the health of the object.

## macTroid.ammo;

This tracks the current amount of ammo the object has.

## macTroid.img;

This stores the image that the object is using.

## macTroid.img.src;

This stores the source of the image used for the object.

## macTroid.currentFrame;

This tracks the current frame of the animation the object it on.

## macTroid.frameX;

This tracks the current frame position on the sprite sheet relative to the x axis.

## macTroid.frameY;

This tracks the current frame position on the sprite sheet relative to the y axis.

## macTroid.startingFrameX;

This is the starting position of the sprite sheet of the current animation in relation to the x axis.

## macTroid.startingFrameY;

This is the starting position of the sprite sheet of the current animation in relation to the y axis.

## macTroid.maxFrame;

This tracks the maximum number of frames that there will be for the current animation.

## macTroid.maxFrameX;

This tracks the furthers position along the x axis for the current animation.

## macTroid.maxFrameY;

This tracks the furthers position along the y axis for the current animation.

## macTroid.timer;

This sets the speed that the animation will play at.

## macTroid.firing;

This checks to see whether the object is currently firing or not.

## macTroid.velX;

This tracks the objects current velocity along the x axis.

## macTroid.velY;

This tracks the objects current velocity along the y axis.

## function start();

This is the first function that is called when the game is loaded and will only be called once.

## function update();

This is called once every frame. Within it is a switch case that checks to see which screen is currently active, functions are then called specific to that screen.

## function init();

Within this function all the objects are created, this is called once from the start function.

## startScreenRender();

This function is for showing the starting screen to the player. It displays Some images, name of the game, player controls and how to starts the game. Firstly, in this function we clear the screen and then draw all the images/text. As this is called every frame, we need to clear what was drawn last frame so that we are not drawing things on top of each other infinitely.

## render();

This is similar to the prevision function, but this time it is used to render everything for the level that the player plays through (screen 2). Again, like before we clear the screen. Next the level layout of the game is rendered (platforms, enemies etc). Then we draw the players health bar, then the player projectiles (these are positioned offscreen and finally we draw the player.

## levelLayout();

This function which is called early in render(); is to called other functions which draw different parts of the level. The level has been broken into different function to help make development easier. Having the level broken into small manageable sections help when navigating through the code.

## levelPart1();

This draws the first section of the level.

## levelPart2();

This draws the second section of the level.

## gamePlay();

This mainly contains everything that goes on within the level. To help organised and make the code readable it has been broken up into various functions such as

playerMovement();  
flyingMovement();  
gruntMovement();  
flyingDeath();  
animations();  
spawnBullet();

colling();

Apart form that the logic that gives the player a sense of movement is in here. After struggling to fine a way to get the camera to follow the player as it moves. Instead, the game has been set up so that all the other objects move around player. So, the player stays stationary and the surrounding objects move to make it look like the player is moving.

When moving the background image, the rate at which it moves is divided by 10, this give a very simile parallax effect to make the background seem further away.

Also, here we have checks to see if the player is colliding with any of the enemies and if they are then the player will lose health.

Finally, there is a check to see if the players health is less than or equal to 0, if so load the game over screen.

## playerMovement();

Within this we have the logic that allows the player to move, fire a projectile and interact with objects. There are also variables that are set which will influence how the player animations play.

## crouch();

This is a small function which update the players sprite to show that it is crouch and the bounds of the player are also reduced which will have a knock-on effect for collision detection.

## stand();

This works in the opposite way to crouch(); when the sprite is updated and the bounds of the player is increased.

## flyingMovement();

Within this function the flying enemies are drawn and programmed to move from left to right and their sprites are rotated 180 degrees when they change direction.

## gruntMovement();

In this function the “grunt” enemies are drawn and programmed to move from left to right.

## animations();

This is used to call the animation functions of all the enemies.

## spwanBullet();

In this function we are checking several things with regards to the projectiles. Firstly, we have two sets of arrays for the projectiles. One for left facing projectiles and one for right facing projectiles. When a projectile is fired the first one of the arrays is spawned. When the second one is fired it draws the second one in the array etc. When we get the end of the array, we go back to the start of the array and the first projectile is then moved to players position and fired. This has been set up in a way so that no matter how fast you fire the projectiles, they will never be moved until the are off screen. Thus, giving the illusion that the player is firing an infinite amount of projectile when there are only 9. This helps with performance.

The projectiles have their “x” position increased/decreased depending on the direction when fired.

Finally a collision check is done to see if any of the projectiles collide with any of the enemies.

## colling();

Within this function we check to see if the player is colliding with any of the floor/ground objects or the interactable button. This is done by called the colCheck(); function.

## colCheck();

In this function we take in two objects as arguments and we check to see if they are overlapping. We also check to see from what direction is the first object is colliding with the second object.

## deathCheck();

In the deathCheck function we check through all of the enemies to see if their “.hp” is less than or equal to 0. If so, the enemy is killed and moved off screen.

## animationEnemy();

In this function we take in certain enemy components as arguments. We the cycle through different sections of the sprite sheet to animate the specific enemy sprite.

## animationPlayer();

This works just like the above function, but it is only used for the player and only used for the animations that are right facing.

## animationPlayerLeft();

This again is the same as above, but it is only used for the left facing animations.

## renderGameOver();

Like with other screens we clear the canvas and then draw the game over screen. The enemies and bullets objects are all set to null just to clean things up a bit.

# Boss Section

## renderBoss();

Here like in other render sections we first cleat the canvas. We then call the bossLayout(); function. After that we draw the boss to screen and call the playerBoss function. The player movement and spawn bullet function are also called. Then there is a check to make sure that the boss has more than zero health. If the boss is alive, we cann the boss moving and boss collision functions. We player the boss music and set the x and y position on the sprite sheet.

If the boss is dead, we then stop the music and draw lots of explosion animations to show the boss dying and change the sprite sheet x position to show the dead boss sprite. The boss then moves to the middles of the screen and the falls to the bottom. Afterwords we then move to the victory screen.

## playerBoss();

In this function we just draw the player, the bullets, and the health.

## bossMoving();

In this function we have a switch case that selects which action the boss will do. A random number functions is called. Depending on the number the boss will do a certain action.

### 0

The boss will move to the centre point of the canvas in relation to the x axis and move towards 20 on the y axis. Once is reaches either of these points another random number is generated.

### 1

The boss will move to the edge of the canvas in relation to the x axis and move towards 20 on the y axis. Once is reaches either of these points another random number is generated. 2

The boss will move to the centre point of the canvas in relation to the x axis and move towards the centre point in relation to the y axis. Once is reaches either of these points another random number is generated.

### 3

The boss will move to the edge of the canvas in relation to the x axis and move towards the centre point in relation to the y axis. Once is reaches either of these points another random number is generated.

### 4

For this last case, the boss will attack the player. A different random number function will be called. If the number is 0 the attack type 1 will be executed and if the number 1 then attack type 2 will be called.

## bossAttack1();

This contains the bosses first attack. When this is called the boss will move to 100 on the y axis and play a specific charging sound. Then it will move to a specific x position. Afterwards It will then charge across the screen. After it leaves the screen its position is reset.

## bossAttack2();

This is like the last attack function. However, the x and y positions are different. After it has moved to the position it will then fire a laser and move forwards. After it leaves the screen its position is reset.

## bossReset();

This resets the boss’s position.

## bossLayout();

Here we draw the background and the platforms that the players stand on.

## bossCollision();

Here we call collision functions between the player and the boss.

## winScreen();

We draw the victory image to the screen.

## randomNo();

This is used to generate and random number form from 0-4.

## randomAtk();

This is used to generate and random number form from 0-1.

# References

All music from <https://freemusicarchive.org/genre/Chiptune?sort=date&d=0&pageSize=20&page=4>

All assets from <https://www.spriters-resource.com/>

All sound effects form <https://freesound.org/>

<https://www.w3schools.com/js/js_object_prototypes.asp>

<https://www.toptal.com/javascript/comprehensive-guide-javascript-design-patterns>

<https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm>

# GitHub Link

https://github.com/ifrift/Mobile-Games-Development-1-Ritchie.git