****

**Interactive Web Development**

**Game Project**

**By William Moriarty (T00119587)**

**Last edited: 14-12-2015**

Contents

[Overview 4](#_Toc437895613)

[Summary 4](#_Toc437895614)

[Control Scheme 5](#_Toc437895615)

[Research 6](#_Toc437895616)

[Possible Games 6](#_Toc437895617)

[Game Composition 8](#_Toc437895618)

[Narrative and Game Mechanic Research 10](#_Toc437895619)

[Plan and Design 12](#_Toc437895620)

[Development Plan 12](#_Toc437895621)

[Drawing the Canvas and using the Game Loop 13](#_Toc437895622)

[Player and Player Input 14](#_Toc437895623)

[Player Weapons and Weapon Control 16](#_Toc437895624)

[Enemies, Power Ups, and their behaviour 19](#_Toc437895625)

[Collisions 32](#_Toc437895626)

[Stages 35](#_Toc437895627)

[Game Assets 39](#_Toc437895628)

[Finishing Touches 47](#_Toc437895629)

[Tools Used 59](#_Toc437895630)

[Considerations and Conclusion 60](#_Toc437895631)

[Works Cited 61](#_Toc437895632)

# Overview

## Summary



*A screenshot of SWARM at level 6*

Swarm is a 2D Space shooter browser game written in the javascript programming language. In Swarm, you take control of a spaceship to defend Earth against an invading swarm of insectoid aliens. Using your laser turret and supply of torpedoes, you must fire and destroy as many alien ships as possible before they reach Earth.

For each alien you destroy, you score points. At each point threshold the level of the game will increase, bringing with it tougher, faster and more dangerous enemies.

At certain intervals you will have the opportunity to grab power ups, boosting the integrity of your shields, hull and torpedo count, to help you survive against the alien swarm.

If you can survive to level 10 you will face off against the mother ship, the alien entity spearheading the deadly invasion.

Can you defeat the mother ship and save Earth?

## Control Scheme

This game uses the keyboard to navigate and control player functions on a 2D canvas. You will be asked to use the left mouse click to begin the game or to navigate through screens.

**W or UP ARROW to move up**

**A or LEFT ARROW to move right**

**S or Down ARROW to move down**

**D or RIGHT ARROW to move right**

**SHIFT and D or**

**SHIFT and RIGHT ARROW to boost right**

**SPACEBAR to fire laser**

**E or F to fire torpedo**

# Research

## Possible Games

One of the most difficult aspects of any project is gaining inspiration to create unique and novel ideas. Although this project was not necessarily required to be original, there remained the challenge of generating the concept of what it should entail. Before any code could be written there was a lot of research that had to be done on what that concept would be.

One thing that was known however was that the project must be in the format of a javascript based game, so with that in mind it I decided to search various parts of the web for inspiration on what to make.

The first step was to find other games created in javascript to get a rough idea of their scope on what they could do and what interaction the player could have with them.

I visited such sites as webdepotresources.com (Depot) that displayed twenty five examples of iconic video games remade in Javascript. Such titles included Tetris, Solitaire, Pacman, Mario and Bomberman. Although they used libraries and frameworks which I wanted to avoid (as per the requirements of the project), they nevertheless gave me a good idea on what browser based games were capable of.

Upon further research it became apparent to me that javascript and browser based games in general were a lot popular than I thought. An article on jlongster.com (jlongster)gave a good summary on how web games are everywhere thanks to the introduction of the canvas element with HTML5.

The site also gave a playable demonstration of simple 2D shooter and a tutorial on how to make it, including the creation and animation of sprites.



After playing the demo an idea began to formulate about the type of game that I was going to make for the project. I thought a 2d shooter like the demo would be appropriate and enjoyable to make, but with a different theme, so I decided to search to the web for javascript based space shooters. Being a fan of such games at a younger age I realized I now had the opportunity to make one, so I kept it in mind as an idea for the project.

One of the websites I encountered showcasing space shooters (including a myriad of different javascript games ) was www.caiman.us (caiman.us)



*One of the space shooting games on www.caiman.us called Space Raker*

After an enjoyable few hours of playing the games taken from [www.caiman.us](http://www.caiman.us) - all the while taking personal notes of their game mechanics, art, and how I would implement them – I came to the conclusion that this is what I wanted to do for my project.

Although it’s important to note that many of these games weren’t written with javascript , they nevertheless inspired me to incorporate the space shooter as the game’s mechanics and theme. With the theme for the project now selected I looked forward to the challenge of implementing it.

## Game Composition

What makes a game?

What makes a space shooter game?

Each 2D space shooter I played from various sites all shared the same common elements that made them a game. Before I could start the game I had to first understand and document what made them.

* **Objective:** Every game has an objective. This is obviously no exception. For space shooters, it mainly consists of defeating and/or surviving waves of enemy AI that would appear on the screen and attack the player in some form. By this, the player is challenged to survive and to retain their virtual lives to progress to the next “level” - where they will probably face more difficult enemies and environmental dangers. When the play has overcome all levels, the games objective is complete. Usually…
* **Player as Spaceship:** The player would almost always control a spaceship sprite of some kind that could be navigated on the canvas by player input to any x,y coordinate (that wouldn’t clash with an enemy). The spaceship would essential be the players avatar and representative in the game.
* **Player will have weapons:** In the most cases the player would also have an arsenal of some kind, in some instances being a turret that would fire bullets, or in other cases lasers. Special high powered weapons are often the norm too coming in the shapes of bombs or torpedoes that have a charge or set amount that are depleted when used. These weapons would be used conservatively to defeat the enemies of the game or perhaps destroy infrastructure like space stations or satellites.
* **Enemies:** Every game has enemies or a challenger of some kind. In the instance of space shooters they could be alien ships or space pirates etc. They would often come in waves from various edges of the screen, seeking to destroy the player by some means. Some would be programmed to crash into the player and destroy him. Others would have their own weapons that would be used against the player, although this is often reserved for bosses. “Bosses” are often enemies that have unique abilities and pose a far greater danger to the player than normal enemies. They are usually reserved for some sort of final challenge at the end of a level or game.
* **Levels:** Most games have “levels”. Game levels are often indicative of the games current difficulty. They are usually represented by numbers with the first level beginning as Level 1. Conventionally, the higher the level, the more difficult the game. Difficulty is often increased by artificial means as levels progress, such as an increase in size, hit points, and speed of enemies, making the enemy more difficult to kill, hence making the game more challenging. The objective of the game is usually complete when all levels of the game have been conquered.

These are the elements that make up many games, but in particular space shooters. I decided I would incorporate these elements into my own game, albeit with own personalized modifications.

## Narrative and Game Mechanic Research

I decided I would form the games composition using the narrative of an alien invasion of Earth. The invading aliens - along with their mother ship - will belong to an insectoid race from a distant planet, so I will have to search for sprites that fit that category. I have found some sprites from this link that I think will be helpful.

http://opengameart.org/content/top-down-2d-rpg (top-down-2d-rpg)

With humanity being threatened with extinction by these invading aliens the player will be taking the role of the defender – who will have to destroy as many of these aliens as possible before they reach Earth. The player will use a laser gun and torpedoes as his offensive tools. They will be able to move up or down, left or right on the canvas to align themselves with the invader and shoot them (coming from the right hand side of the screen). I have decided to use the player’s spaceship sprite from the same link as above.

I have also decided that the aliens will be designed to spawn at random points of the y axis just outside the screen, to make them unpredictable in their location and harder for the player to line up and shoot them. To accomplish this I will probably use the Math.random function on the CANVAS\_HEIGHT. I must also figure out how to draw more than one alien on the canvas at a time.

I have found an extremely helpful article on [www.html5rocks.com](http://www.html5rocks.com) (Moore)that describes how a function can be used to represent a game object and be pushed onto an array, and then drawn on the canvas. This will be extremely helpful when drawing different instances of the same object. I will also be able to expand this method to include other objects that will require multiple instances such as lasers, torpedoes boss projectiles and possibly asteroids to include an element of randomness.

I also need to find a way to represent objects being destroyed when shot at or crashing into something. I found these objects can be destroyed by using a collision function and blow-up function that is also detailed within the article. The blow up function for example will state the object as inactive (using a Boolean variable) after there is a collision and the array in which it is in will use a filter to remove it from the draw method. The collision function will simply take two arguments -the two object names -and check if their x,y position on the canvas (modified by their width and height) will clash.

The player will shoot at the aliens and asteroids with their bullets and torpedoes and destroy them using the same mechanics. The player will gain points for doing this. I decided after doing some minor testing on an older game that points will be a good way to keep track, increment and implement levels.

I thought it might also fit the theme to make enemies tougher and bigger as each level passed – and how the player came closer to the mothership which spawns them each level. Perhaps it could also fit the narrative to say that the tougher enemies stay behind to protect the mothership. With each level I may decide to boost the health, width and height, and perhaps speed of the enemy traversing the x- axis of the canvas.

The amount of levels I will incorporate is yet unknown but the last level will have a showdown between the player and the mothership – which is creating and controlling the aliens and spearheading the invasion of Earth. The player will have to destroy the mothership in order to save Earth, but it will not be an easy task, as the mothership will be designed to have weapons and abilities that will be deadly to the player – which fit the sci-fi theme of the narrative. I have been considering what abilities to give the mothership and after watching some cheesy sci fi flicks I have decided to give it organic “energy weapons”. These will be implemented in the same way as the players weapons – by using objects and spawning them at certain intervals from the mothership to the player.

I have also been considering the option of the player gaining “Power ups” that will spawn at random parts of the screen and give the player extra lives, ammunition, and health. Power ups are fairly popular in shooter games and can often be critical to a players survival.

If the player loses all his lives a gameover screen will come appear, as conventional to most games. I already know how to do this from a previous exercise in the lab, and with a bit of revision I will implement extra screens, perhaps explaining the narrative of the game and its control scheme.

With the research complete and now having a concrete idea of what I wanted from the game and how I was going to implement it, it was finally time to move onto the code. I just hope that my expectations meet the reality.

# Plan and Design

## Development Plan

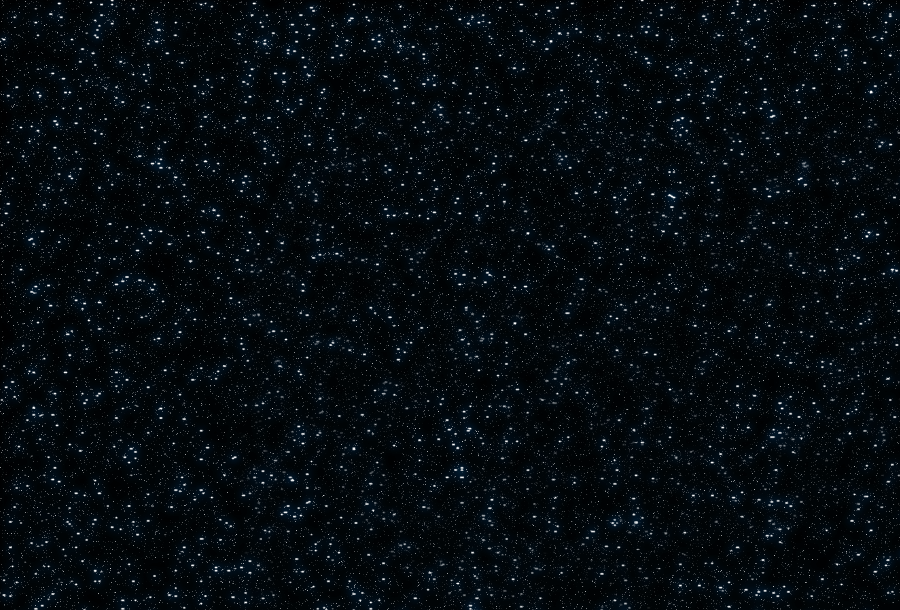
Below is a simple diagram of the design plan that was incorporated into the project. Each element will be explained in detail below on the overleaf. Each step is prioritised in term of its importance and necessity to what makes the game. The highest being the most essential, with each descending step become less critical, albeit all have their importance in making a complete game. Making the game canvas and its background is not included.

### Drawing the Canvas and using the Game Loop

Before the player or any other object could be created, the canvas on which it had to be drawn on had to be made first. I gave it a width of 800px and a height of 650.

<canvas id="gameCanvas" width ="800" height="650" class="background"></canvas>

I decided more or less straight away that I would implement the background with the scrollable css code given to us in class. For the background image I used an image of a star field.



As for the game loop, I was going to stick to convention and use the same method I used for previous projects and what I saw used in other games. The draw function and an update function that would be keeping track of the games updates that would be called in a setInterval function, with its timer set to 1000 to represent a second and have it divisible by a variable set for the frames per second. In this case it would be 30 like most non-pc games.



### Player and Player Input

The first step after the canvas was drawn (and the background added via css) was to incorporate the player into the game. The player or avatar would be represented as a rectangle for the time being until its functionality had been complete.



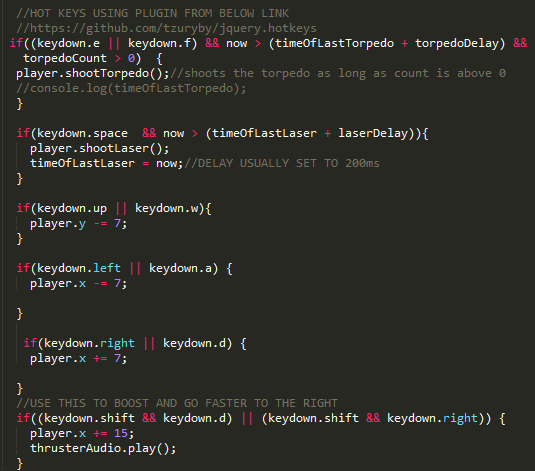
*The player variable with its draw function*

Starting with drawing a simple rectangle onto the screen, it was then decided that it needed movement. After getting it to move using an incremental value via an input function the rectangle could be moved around. However, I found default movement very sluggish so I decided to try use a plugin to make a more effective and fluid control scheme, something that I had overlooked during the research phase of the project.

I downloaded jquery hotkeys plugin from this link on github.

<https://github.com/tzuryby/jquery.hotkeys> (tzuryby)

This was very helpful for not having to pick through keyCodes or keyChars and find their corresponding value. It also allowed me to use non-case-sensitive letters when writing the players navigation, shortening the code.

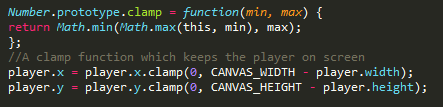


*What some of the player inputs eventually looked like*

The most beneficial thing about it however, was that its script allowed for the handling of multiple key downs and presses, allowing the player to move diagonally. It may seem like a small feature but for a simple game it’s a great addition. I was also able to use modifiers, so I decided to allow the player to go faster by holding down the shift button and press the right arrow or‘d’ key. However, the player could not move diagonal if the shoot key was continuously held. I thought this a good game compromise.

With freedom to move around on the canvas I know had to find a solution to restrict the player from going outside its bounds.

A simple yet elegant function provided the answer.

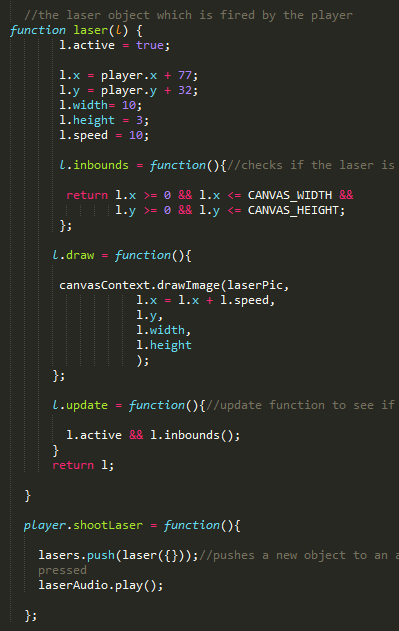


The player was now able to move anywhere on the canvas via keyboard input, but not outside it.

### Player Weapons and Weapon Control

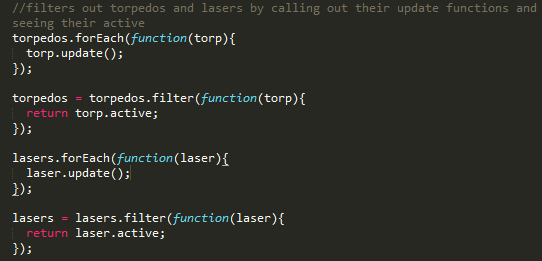
Now that the player (or rather the rectangle) could move around it was now time to fit him with some firepower. I had already decided at this point that the spacebar was going to going to be used to fire lasers and E and/or F to fire torpedoes.

I confirmed these keys to be working by testing them with simple console statements. Now that input code had to be populated with a function of some sort that had to shoot. I decided to do the laser first.



*The laser function and player shoot function*

From my research I had learned how to create multiple instances of an object (loosely speaking as an object in javascript) and push them into an array. The array had to be filtered to only have active instances. It was set to inactive if there was a collision (more on that later).



*Code for using the filter function*

The function was also given sub-functions for checking if it was in bounds, to see if it was updating and most importantly if it was drawing. I used rectangles to draw again before committing to any sprites.

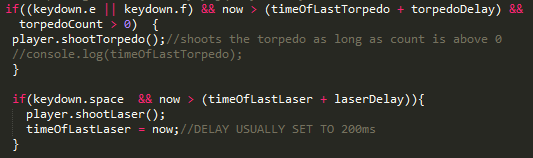
The point at which the laser started drawing was given the players current x,y coordinates to always remain with the player, but it was given a speed attribute that would be summed with its current position (as an argument in the .drawImage function) so that it would give the illusion that it was “firing”.

Declaring an array called lasers[] I decided to use a function (inside the player.shoot function) that would push out a laser instance every time it was called. I then called this function in the update function behind the “if keydown.spacebar” code.

The laser worked but when it first started firing it could remain firing for as long as the spacebar was held down. This was a problem as the laser seemed more like a “stream” of laser rather than an array of bullets. This would also impact the intended difficulty of the game, making it much easier since you could fire lasers without an interval.

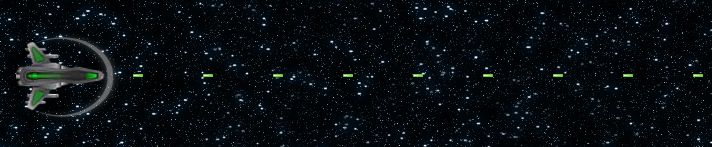
An interval of some sort was needed to handle the timing of the firing rate. I couldn’t use the setInterval method since it made a loop of itself. The timeout function also would only count the timeout for the first time you pressed the spacebar, and it would reset since it was in the update function.

I decided I would make my own timer by getting the current date time and keep having it updated continuously in the update method. Every time I would press the space bar I would then pass the current time into another variable that would keep track of the last time I pressed the space bar. To ensure that the space bar or laser couldn’t be fired continuously I added another if statement alongside the keydown.spacebar checker to see if the current time was larger than the last time the laser was fired with a small additional sum of 200 (held in its own variable called laser delay).



*One of the games many custom made timers*

If the current time wasn’t greater the laser couldn’t fire. If it was larger than the delay you could. Ergo, you could only fire the laser every 2 milliseconds now. You could also set the delay variable to increase or decrease your rate of fire to whatever suited the game.



***The laser firing at its appropriate rate of 2ms***

Although it took me awhile to figure out how to do this these timers became vital in the development of the game and I would later incorporate them all over my project for various functions.

Now that the laser was created the very same methods could be applied to the player’s torpedoes.

### Enemies, Power Ups, and their behaviour

Creating the enemies and power ups was very much the same as creating the laser and torpedoes with one major exception – their behaviour.

There are four major enemies in the game, including the boss. With each passing level they will (excluding the boss) increase in size, speed, health and appear in shorter intervals.

* **DronePod:** The most common enemy that is spawned from the very first level.

The DronePod appears from the right hand side of the screen just beyond the canvas width and at random points of the y axis.

* **Reaper:** Doesn’t appear until level 4. Very much the same as the Dronepod but is much larger and has more health.

Like the DronePod, they appear from the right hand side of the screen just beyond the canvas width and at random points of the y axis.

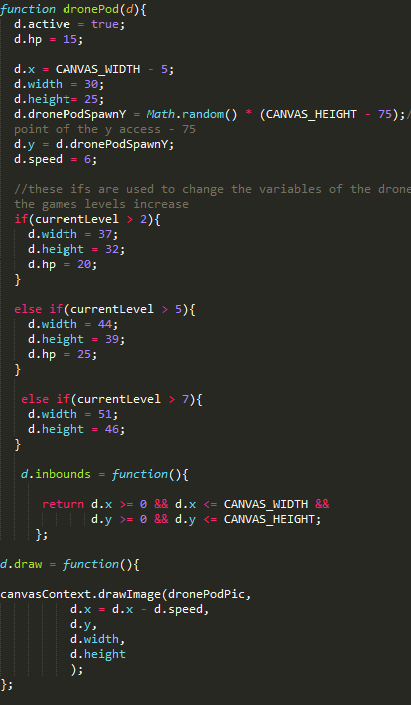
* **Asteroid:** Asteroids were introduced to the game to give an element of surprise and randomness. They spawn at either the bottom end of the canvas (y:CANVAS\_HEIGHT, x:Random) or at the top of the canvas (y:0, x:Random). They go in a random direction when they are spawned but always towards the opposite end of the screen. The player will more than likely spend more time dodging asteroids than they will destroying them.
* **Mothership:** The boss of the game that only appears once at level 10. The mothership has more health than any other enemy (1000 hp). Defeating the mothership finishes the game. The mothership has 3 abilities - Spawning dronelings in waves, which hinder the player and block incoming fire towards her. Energy Orb which is fired at a set interval towards the player. And a laser beam which fires in an arc and is supposed to destroy the player if he cannot find cover in time. However, the laser beam functional as of yet, so it will thought of as a “searchbeam” for the time being.



*In order: asteroid, dronePod, reaper and mothership. NOT TO SCALE.*

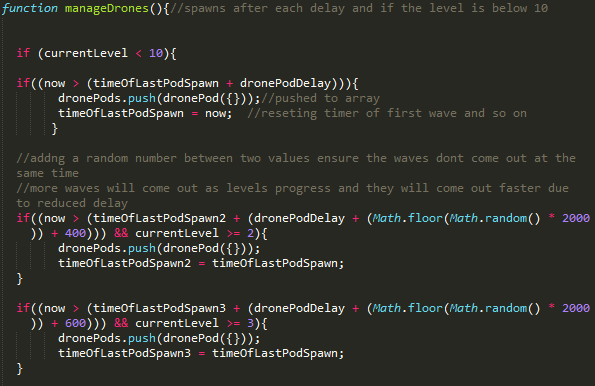
#### DronePods and Reapers

As mentioned earlier creating the DronePod and reaper was much the same as the lasers and torpedoes. Here I will demonstrate the creation and management of a dronePod.



Here we see the dronePods spawn point on the y axis being set to random (within the canvas height). You may also notice that if the levels are at a certain value the width, height and hp change. It is much the same with the reapers.

Before I actually implemented the size increase with levels (which was one of the last things I incorporated since I hadn’t made levels at this stage) I had to set an interval between each time the dronePods were drawn, otherwise they would populate the entire screen. Instead of handling intervals inside in the draw function I decided to make my own custom function that would handle the interval of each spawn, including handling the array filter and the calling for the dronepod’s update sub-function. I would then call that function in the MAIN update function.



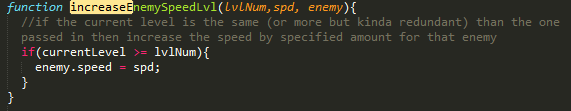
*A sample of the manageDrones function, other enemies manage functions are largely the same in how they manage its spawn rates*

Again, levels were not incorporated into the manageDrones function at this stage. The same timing principle that was used to manage the lasers timer was applied here. The dronePods spawn timer could now be controlled and manipulated depending on how we wanted it to.

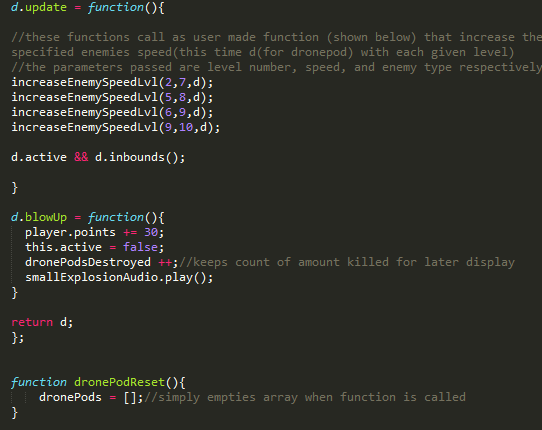
Now it was time to increase its speed. Speed is how fast it travels on the x axis and must not be confused with the speed of how often a new pod is drawn by being pushed to its array.

To increase the speed of the enemies with each progressing level I implemented a function that would pass in the current level, the desired speed and the enemy as its parameters.

I realised I should have done this with health, intervals and other level controlled designs but I simply did not have the time to return and rewrite the code more efficiently.



Below you can see that the function is called four times in the dronePods update sub-function, increasing its speed four times throughout the game, creating a challenge for the player.



The update function also checks if the drone is active and inbounds. I mentioned earlier how an array filter was used to remove anything that was not active.

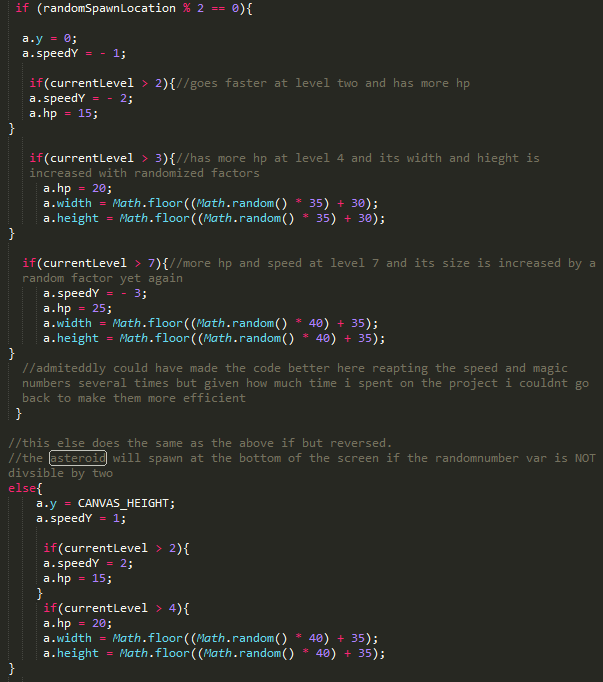
The blowUp function too, which is used in collisions (which I’ll describe later) sets the dronePod to inactive.

That’s all there is to the creation and control of the dronePods and reapers (apart from collisions, which will be described later).

#### Asteroids

Asteroids are created using the same convention as dronePods and reapers. They use the same functions and methodology to control their spawn timers, updates, collisions and size increases.

There is however, a very noticeable difference in their behaviour. Their spawn points and size are randomized with each instance. It is very difficult to predict where an asteroid will appear.

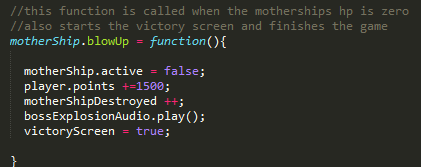


The asteroid function uses a randomized number that is updated continuously to determine where the asteroid will spawn. If it is divisible by two (prime number) the asteroid will spawn at the top of the canvas, otherwise it will spawn at the bottom.

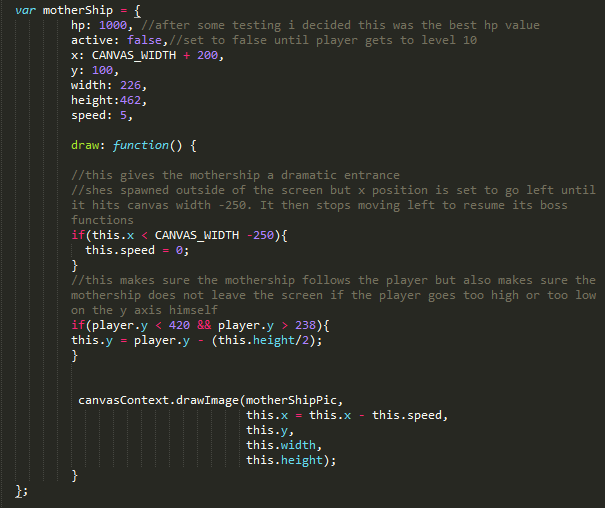
#### Mothership

The big boss of the game.

The mothership will only appear once the player reaches the tenth level. Other enemies will stop spawning when she appears. Although I did not have any levels implemented when I first designed the mothership I had always intended to make her only appear for level 10 as the final encounter. Killing the mothership will reset the game.



The mothership is unlike other enemies because it is unique and does not need multiple instances of itself to be made. The mothership is actually more like the player in terms of its functionality and how it is created, but being an AI its abilities and movements are automatic and predictable. In this regard it shares behavioural patterns with the other enemies.



As you can see form the above snippet the mothership will follow the player between certain areas of the y axis. She is designed not to go off screen (if she does her orb weapon could be exploited). She also makes a dramatic entry as seen by the line

If(this.x < CANVAS\_WIDTH -250){

this.speed = 0;

}

This makes sure the mothership races onto the canvas and stops short to give the player abit of a scare!



*An encounter with the mothership, who is firing her laser beam, also shown are the energy orb and the dronelings which she spawns*

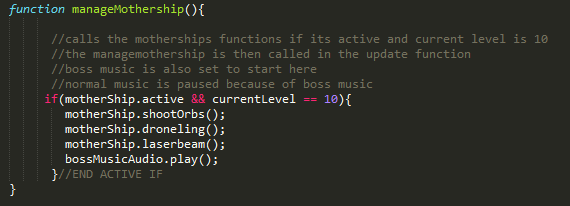
Defeating the mothership will end the game, however the mothership has been given a deadly arsenal to defend herself against the player.

The arsenal consists of dronelings, which are spawned like dronePods and attack in waves and shield the mothership against incoming fire. The energy orb is a massive ball that will reduce the player’s shields to nothing or kill them outright if they are not active. And last but not least is the mothership’s, err, search beam.

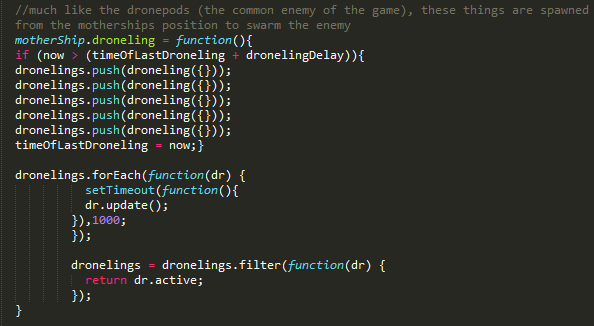
The search beam was originally supposed to be a deadly laser beam but due to time constraints it was not complete, so I will not give an analysis of it.

They are all made as functions that could have instances. Since I have explained how I managed that I will just skip to their behaviour.

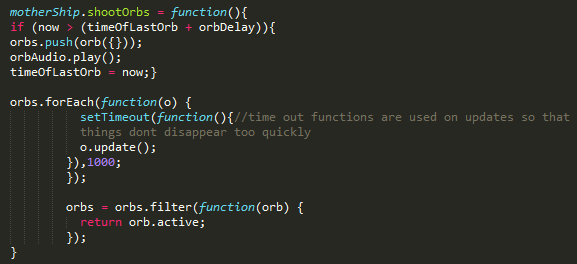
Here we have the code I used to manage all the motherships abilities. This was called in the main update function.



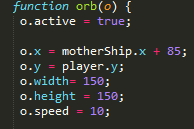
The dronelings were fairly standard to create and manage.



The orbs too were fairly standard and easy to manage.



They were set to be spawned on the same x position as the player was in, challenging the player to get out if its way quickly.



#### Power Ups

I decided to explain Power ups in this section because again, they largely have the same functionality and code as dronePods, reapers and asteroids, including their management of spawn rates, update functions and collision functions.

Power ups are not enemies however.

Power ups are special items that float from random points of the screen to another and can be collected to benefit the player in various ways. Whether the player wants to risk death getting a power up is another decision though.

There are three types of power ups, and they all share a spawn interval. At each interval only one of the three will spawn:

* **Shield Drop:** When a player collects a shield drop he receives an increase to the percentage of their shield.
* **Hull Repair Drop:** When a player collects a hull repair drop there is sufficient repairs made to the hull of their ship. Remember if the hull’s value goes below zero the player dies.
* **Torpedo Drop:** When a player collects a torpedo drop he is awarded extra torpedoes to his arsenal. Torpedoes are powerful weapons and it’s good to have a few handy.

**D:\College\Web Design\Interactive Web Development\Game Assets Files\hd.png**

*In order: a shield drop, a hull repair drop, and a torpedo drop (icons to scale)*

Power drops are similar to asteroids in how their spawn location is determined. They use the same randomly generated number to issue a spawn point. However, to avoid them spawning in the same location of the asteroid they do not check to see if the random number is divisible by two. Rather, they check what number was actually generated, and if it’s a certain value it will spawn in a certain location.

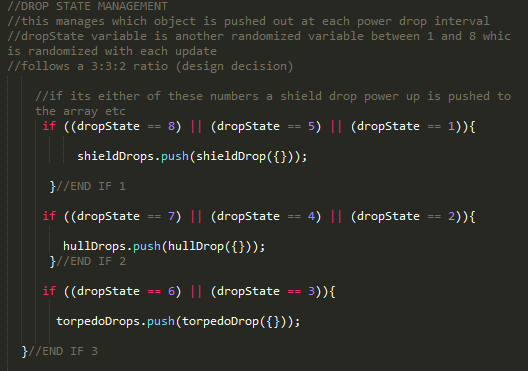
Here’s is some code for the shield function.



The random number can be any value between and including 1 and 8. If 8,5,or 2 is generated, the shield drop will spawn from the right hand side of the screen. If 7 or 4 is generated, it will spawn from the top. If its 6 or 3, the bottom will spawn a power drop. Else, (which is only 1) from the left hand side of the screen. I gave this a lower chance than the rest because I did not want to make it too easy for the player to grab a power up.

In retrospect, I should have made a single function that would handle spawns for each power drop, but again, time was a constraint.

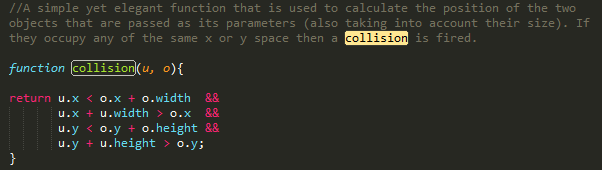
Now let’s take a look at how the start of the drop is determined.



The same principal is used here, using a different variable to push out the power drop.

### Collisions

Collisions were surprisingly easy to manage thanks to this very simple yet elegant function.

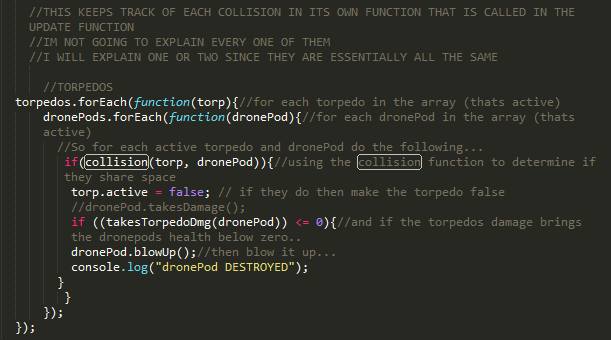


*My personal favourite function in the project*

This collision method would take in parameters of two objects and check their position against each other (compensating for size).

I used a wrapper function to call and keep track of every collision that I wanted to detect in the game. I then called that wrapper function within the update function so that it constantly checked for collisions in the game.

Here is an example of the collision method being used for a torpedo against a dronepod –



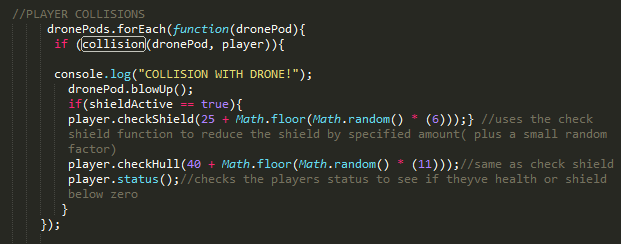
Using the forEach function for both torpedo and dronepod the collision can be made for every instance of both objects in the game. A very efficient function.

For each collision it was only a matter of setting Boolean variables to false through the blowUp method to remove certain enemies’ from the screen.

Using this method I was able to add extra functions that would only blowUp objects if they lost hp or crashed a number of times. I was able to create bounces or pushbacks on collision just by setting the x axis of an object to a minus or plus value when a collision was made.

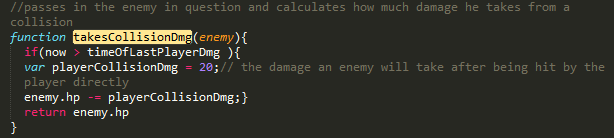
Here is an example of a player collision with a dronepod.

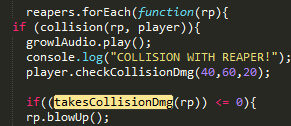
Here, like many other ForEach collisions the players shield and hull is checked against incoming damage before his status is updated in the player status function.



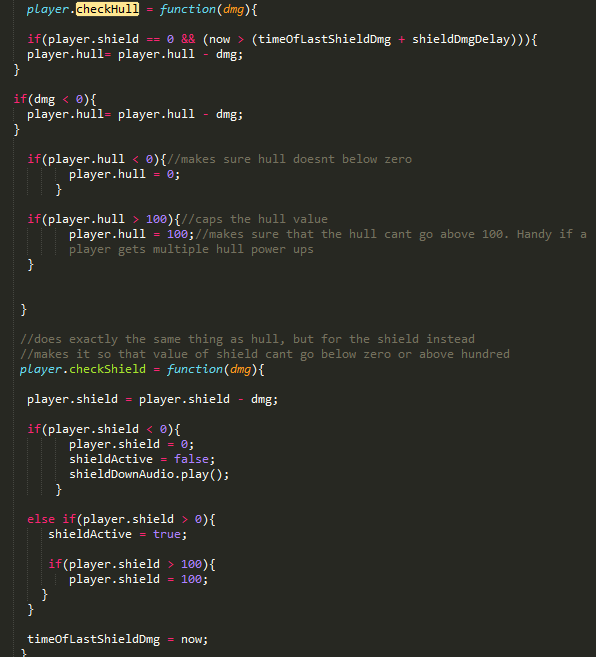
By tracking the collision I could easily manage the fate of both player and AI. In the above ForEach DronePod function the drone pod instantly blows up on contact with the player (design decision). Yet if you were to shoot the dronePod with the laser it would take a number of hits before blowing up.

For the reaper and mothership I made a checkCollision damage function since I didn’t want them to die straight away on impact.



**** I called this function within the reaper and mothership forEach collisions with the player.

*The check hull and check shield functions*



I was able to incorporate a number of functions that would keep track of health, shield and power up drops thanks to the single small collision function.

### Stages

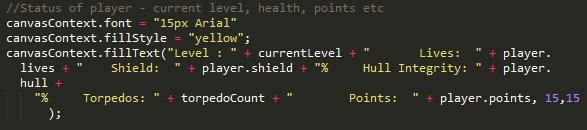
From the beginning of the project I tried to make points and levels go hand in hand, and thankfully it was not as difficult as I thought it would be.

Points and levels can be tracked at the top of the game screen alongside other game information.

Points are located on the far right where levels are on the far left.

D:\College\Web Design\Interactive Web Development\Game Assets Files\lvl6Screen2.png

Here is the code I used to keep track of them, called within the gameScreen in the draw function.



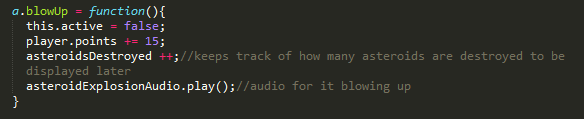
As you can see it calls the variables currentLevel and player.points.

You’re last completed including your points will also be displayed for you when you complete the game or get a game over. The level and points are then reset.



#### Points

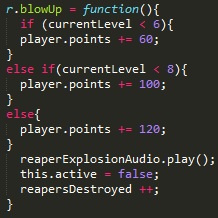
Points are gained by killing enemies or collecting power ups, managed from their blowUp function.



As you can see player.points is increment by 15 whenever this function will be called, so whenever you kill and enemy, you gain points…

The point’s scheme is as follows:

* Asteroids: 15pts
* DronePods: 30pts
* Reapers: 60pts below lvl 6, 100pts for lvl 6 and 7, 120pts for lvl 8 and 9
* PowerUps: 10pts
* Mothership: 1500pts



#### Levels

There are ten levels in the game, starting with the first level as number one. Complicated right?

With each advancing level the difficulty of the game increases. The game reaches its peak difficulty at level ten when the mothership arrives on the screen (the other enemies however do not appear anymore).

The game difficulty is controlled by various “if” statements that are constantly being check by various functions in the update function.

As we already mentioned and explained in the Enemies section, their size, speed, hp increase while the intervals on which they arrive decrease, creating more enemies on screen and making it difficult for the player to keep up with each passing level.

So how do we keep tracks of levels? I decided to do so with points. I thought it best to make point thresholds of some sort. If you get past 200 points for example you will be on level2 etc.



*Simple code showing how the current level is determined by player points* The above image is the final iteration of the point thresholds for levels. You may notice that one the points are over 5000 the current level becomes 10, at which the mothership is made active.

This function is constantly monitoring the player’s points, being called and continuously updated in the main Update function.

### Game Assets

At this stage of development I didn’t actually have many sprites or any audio incorporated into the game –despite the fact that the code I have shown shows images being drawn. They were ways considered the finished or polished product (if such a thing could exist with this project).

#### Sprites

Most of the sprites shown are not to scale.

As mentioned in my research, I found a number of sprites from

http://opengameart.org/content/top-down-2d-rpg

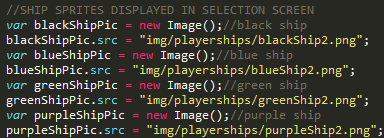
that I said I would incorporate into the game. And I did…



From the images shown throughout this document you may find a few of the above familiar, I have used many of these images for my sprites(including the player ships) and full credit goes to the artist of these sprites and the source they were taken from – which is noted in the Works Cited section.

A sprite would be rendered onto the canvas using the drawImage function and passing it the image variable of choice as one of its parameters.



Declaring a picture variable with the source.

For player ship sprite I made a last minute addition that allows you to select your sprite at the start of the game. So now there are four sprites to choose from.

Torpedo and laser graphics by Kenney Vleugels (www.kenney.nl)(Vleugels)

D:\College\Web Design\Interactive Web Development\SWARM\img\exhaust.pngD:\College\Web Design\Interactive Web Development\SWARM\img\laserRed.pngD:\College\Web Design\Interactive Web Development\SWARM\img\torpedoRed.png

*The Blackbird (with ship exhaust sprite)*

**D:\College\Web Design\Interactive Web Development\SWARM\img\laserBlue.pngD:\College\Web Design\Interactive Web Development\SWARM\img\torpedoBlue.png

*True Blue (source above)*

D:\College\Web Design\Interactive Web Development\SWARM\img\laserGreen.pngD:\College\Web Design\Interactive Web Development\SWARM\img\torpedoGreen.png

*Mean Green(source above)*

D:\College\Web Design\Interactive Web Development\SWARM\img\laserPurple.pngD:\College\Web Design\Interactive Web Development\SWARM\img\torpedoPurple.png

*Purple Rain(source above)*

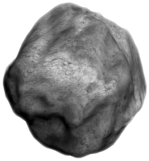
Enemy Ships-



*DronePod*



*Reaper*



*Asteroid (source –OpenGameArts.org)*



*Droneling*



*Motherships energy orb (source - https://i.ytimg.com/vi/\_vyHwlC9FWA/maxresdefault.jpg)*



*Mothership (source - http://orig15.deviantart.net/4056/f/2012/012/5/5/reaper\_comparisons\_by\_kavinveldar-d4m3gan.png)*

**D:\College\Web Design\Interactive Web Development\Game Assets Files\hd.png**

*Hull Drops (source - http://www.gameart2d.com/uploads/3/0/9/1/30917885/2303922\_orig.jpg)*

Banners –(self made)



(shown on startup screen)



(shown on game over)



(shown on victory screen)

#### Audio

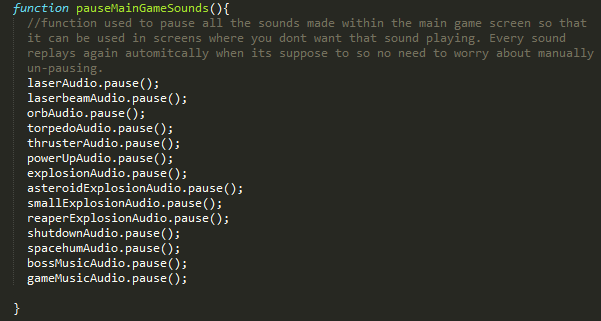
ALL Audio files that I used were downloaded from <http://freesound.org/> while I was logged in with my account.

Importing Audio files into document –



Obviously Audio files can’t be showcased in a word document but they are in the audio folder of the projects files.

Audio files were played in the game using the .playI() function. They could also be paused using the .pause() function, which I did take advantage of numerous times.

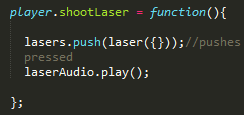


*A function I used to pause all the game sounds that ran during the games main screen. It would be called in the Startup or Victory/Game Over screen*

Audio that was played in the game for the following events:

* When the game started and the banner appeared start up music accompanied it
* Background music for the main game
* Background space hums during the main game
* Music for the game over screen
* Celebration music for the victory screen
* Laser sounds when the player fired their laser gun
* Torpedo sound when the player fired their torpedoes
* A jet engine sound whenever the player used their thrust
* A sound effect for when the player collected a power up
* A “shut down” sound followed by 2 clicks to indicate that the players shield were down
* A different “shutdown “ sound accompanied by an explosion sound with the player died
* A growl sound when you collided with a reaper
* A explosion sound when you killed a reaper
* A different explosion sound when you killed a dronePod or droneling
* A “breaking” sound when an asteroid was destroyed
* Background music exclusive for the boss fight
* A bounce sound that is made when you collide with the mothership
* A “energy hum” sound when the mothership fires an energy orb
* A laser beam sound when the mothership fires her laser beam
* A massive explosion when the mothership is destroyed

Some examples of audio being played –



D:\College\Web Design\Interactive Web Development\Game Assets Files\aadad.png

D:\College\Web Design\Interactive Web Development\Game Assets Files\awd.png

### Finishing Touches

The finishing touches and last minute additions that were made to the game. Since the essentials were complete and there was some extra time available I decided to add a few features that could give the game abit of polish.

#### Extra Screens

***Final iteration of Extra Screens***

*Player begins at intro/start screen*

GAME OVER SCREEN

*IF PLAYER DIES DURING MAIN SCREEN HE IS DIVERTED TO GAME OVER SCREEN*

MAIN

GAME SCREEN

SHIP SELECT SCREEN

CONTROL SCREEN

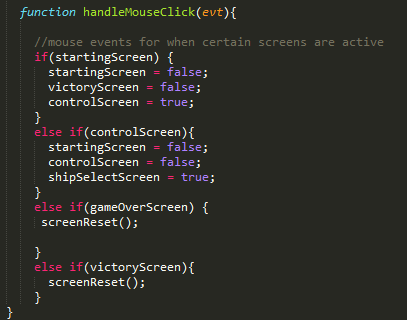
INTRO / START SCREEN

*IF PLAYER DEFEATS LEVEL 10 DURING MAIN SCREEN HE IS DIVERTED TO VICTORY SCREEN*

*PLAYER IS RETURNED TO INTRO/SCREEN AFTER VICTORY/GAME OVER SCREEN*

VICTORY SCREEN

The screens are managed by Boolean variables which are made true or false depending what is happening on each screen. For example, if you click the mouse button on the start screen, the strartscreen Boolean variable becomes false and the control Boolean variable becomes true. When you select a ship from the ship select screen it causes the main game screen to become true and start the game. The draw function is continuously updated in the interval so it also continuously checks which Boolean variable(screen) is true.

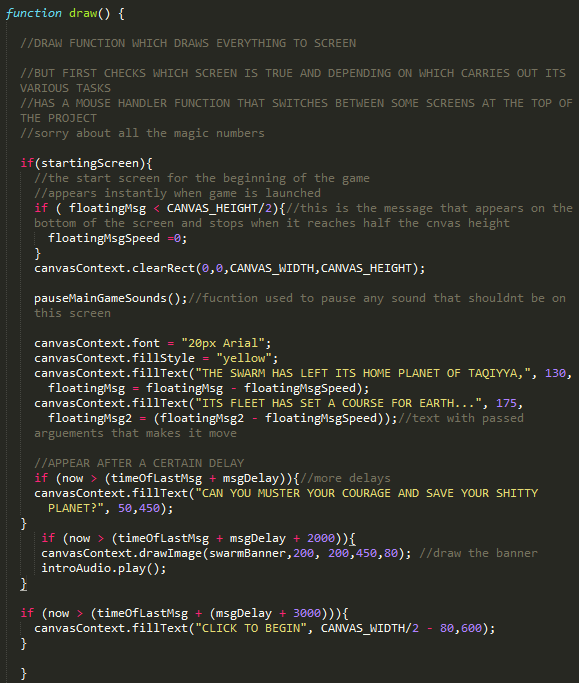


#### Starting screen

This screen acts as the introduction to the game.



The starting screen briefly explains the narrative of the game. After an interval the intro music starts playing and the player is prompted to click the mouse button to begin. It incorporates one of the user made sprites as its banner.



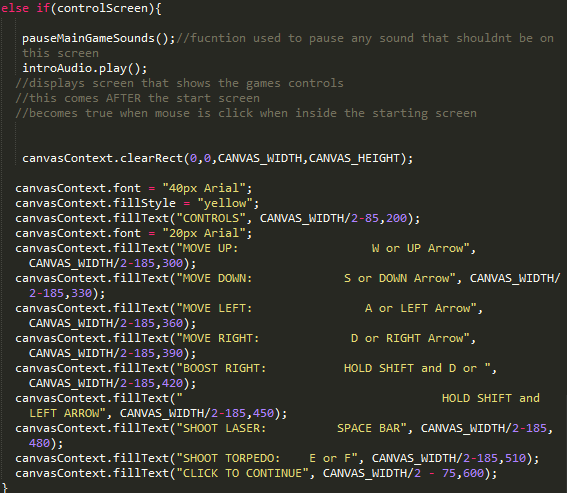
*The code for the starting screen*

#### Control Screen

I thought a controller screen might also be a good addition to give the player better knowledge of the control scheme for the game, as small as it is.



If the intro music didn’t start playing on the start screen (if the player clicked before the message timer was fulfilled) then the control scheme would play the intro music anyway.



*The code for the controlScreen*

#### Ship Select Screen

The last screen that was implemented into the project. When I couldn’t decide which sprite that I would use to draw the players ship (since I liked all of them) I decided I would try and find some way to implement a ship selection function. I thought this would be best used in its own screen, so I decided to make it active after the control screen.



*The ship select screen*

The player can now choose from four different ships to play as in the game. The difference is purely cosmetic and there is no practical differences from ship to ship, including no increase in width or height.

When the player selects a ship after pressing one of the corresponding buttons shown in the above image, the game will begin.



*The code for the ship select screen*

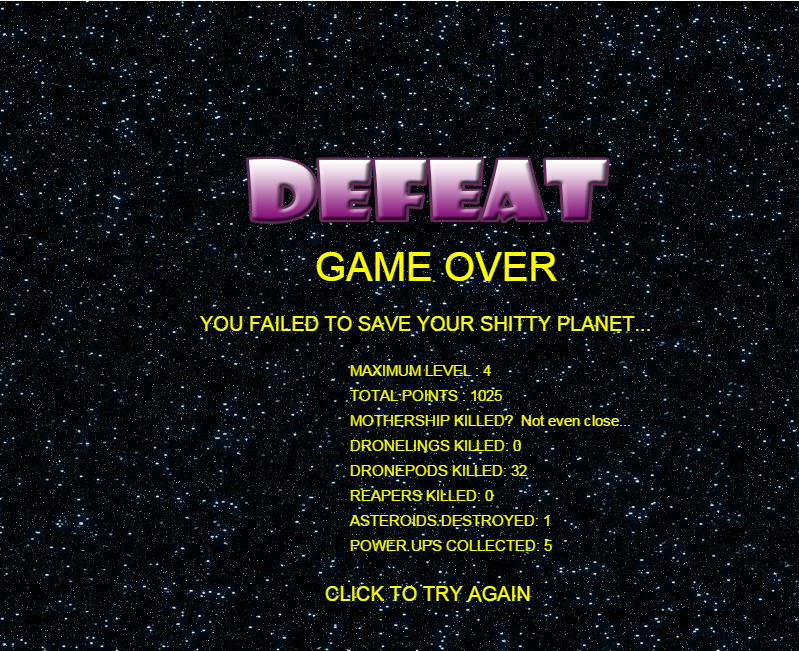


*The Player variable accompanied by the player.shipSelection function*

When the play selects a ship he enters a key. For example if you choose the ship purple rain you must press the “V” key. Pressing the “V” key sets the shipState variable to 4. If the variables shipState is 4 it will call the purple rains associated sprites to draw its image.

#### Game over Screen

The game over screen was the first screen I made for the game. While it initially only had a game over and a game rest function it evolved to play its own audio, draw a special game over banner, add a personal message of condolence and mostly importantly give feedback on the players stats over the course of the game.

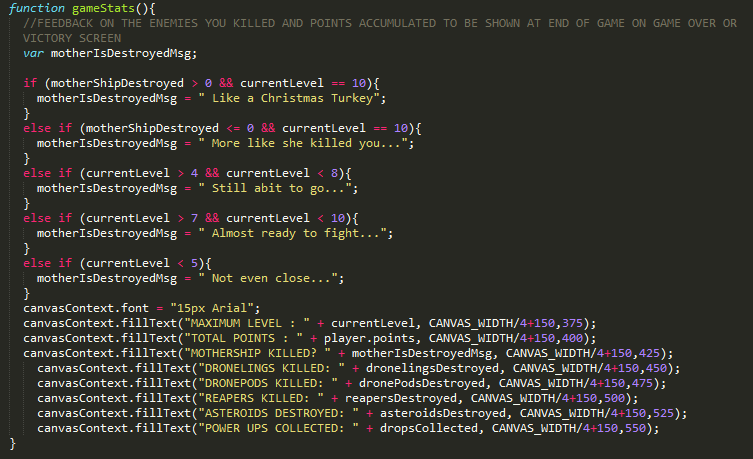


*The Game Over which gives feedback on the player’s game progress*

The player’s statistics such as the level and total points are simply called in a fillText function.

Other statistics however, such as the amount of enemies killed and power ups collected use a variable that is incremented each time an enemy is killed or a power up collected.

They are then passed into the string of the fill Text function. All the fillTexts returning the players statistics are wrapped in its own function so they can be called to the victory screen as well.



*The gameStats function which returns the details of the player’s progress in the game. It is used by both the Gameover AND Victory screen*

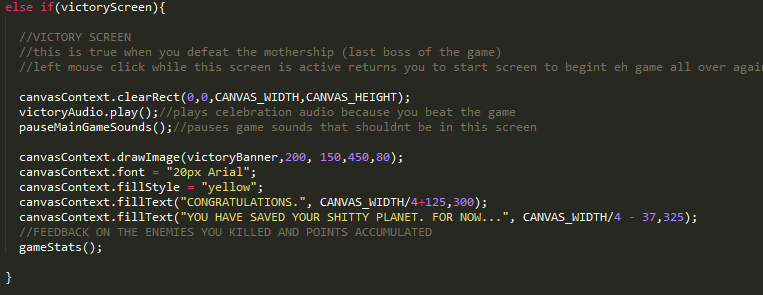
#### Victory Screen

The victory screen is the screen that becomes active when you defeat the mothership and beat the game. Clicking on the victory screen resets the game.

It is very similar in functionality to the game over screen and uses the same gameStat function to display game statistics back to the player.



*The Victory Screen*



## Tools Used

#### Chrome

Although I tested the project in as many browsers as I could Chrome was the one I used the most. I found that things ran smoother on Chrome, perhaps because of the amount of RAM it used from each system. Chrome also has built in permissions for JS which made loading up each time I want to try something much less of a hassle.

I also found the tools that could be used in chrome much more intuitive, including it having a better console log but that is perhaps a personal preference.

#### Sublime

Sublime is a powerful free text editor that I have grown to love over the course of the project. I did the entire project in sublime. Its default colour scheme is very easy on the eyes and I find it easier to spot errors when particular words are colour coded. It also has an intuitive intellisense.

My favourite feature perhaps is its find function which allows you to select all instances of a word and replace them all at the same time with another which saved me a lot of time on the project.

#### Audacity

I used audacity to clip and shorten many of the audio files I downloaded from freesound.org. This is why they all came out as .wav format, because Audacity cannot convert edited files to mp3. I also worked with each sound file in audacity tweaking the decibels in each one to give a lower and more balanced sound. There was a noticeable difference when the sound was being used during the game, it was much more in sync and many of the sounds or soundtrack lost their annoying or piercing edge that they had, but that is again perhaps preference.

#### Photoshop

Photoshop was extremely helpful in the management of sprites. With photoshop I was able to resize, recolour and remove backgrounds from images. I would then save them in png format in the img folder of the project so that they could be called into the game as sprites with no interfering backgrounds.

#### Microsoft Word

Guess what I used Microsoft word for?

# Considerations and Conclusion

Even though I faced many challenges over the course of the project it has become one of the most enjoyable ones I have done during my time at the college. It has also been a long project and even though there is still much I would like to do time has caught up with me.

However, if I had to do it again I would make more use of functions and try not to repeat so much code. If time wasn’t a factor I would definitely revisit the code and refactor it. I would also try to find a way to modularise it so it could easier to handle. One of the biggest problems I had was trying to locate lines and segments of code because the project became so large. Sublime helped me a lot in this regard with its navigation tools.

Proper refactoring of the code would allow much smoother navigation through the project for the programmer and anyone else who reads their code. It would also buy more time to include more features.

To compensate for the haphazardness of the code I including as many comments as I possible could to clarify what I was doing and where certain functions etc were being called and what they were used for.

If the code is refactored sometime in the future I may return more to make several improvements to the games mechanics. One of the most disappointing things that I failed to implement was the motherships laser beam colliding properly with the player.

From playing the game myself I found that the power ups I needed were not the ones that spawned on screen. This was particularly annoying when I was in vital need of a hull repair but received torpedoes instead. There was also instances where too many hull drops would spawn when the player needed torpedoes. I also considered other special drops that would give the player better weapons or maybe extra lives. So this is definitely an area that could be revisited too.

Overall I was satisfied with how the enemies behaved, but I would like to expand the amount of levels and also introduce more boss enemies.

I will be pushing this game to git and hopefully get some feedback from other players on what improvements I could make on the game (AFTER it’s been submitted to the college of course).

In conclusion this project has thought me a lot about the handling of functions in javascript and manipulating the HTML canvas. It has inspired me to do more browser games in the future.

I Hope anyone who plays the game will enjoy as much as I’ve had fun making it.

# Works Cited

caiman.us. (n.d.). */scripts/fw/fspaceshooter.html*. Retrieved from www.caiman.us.: http://www.caiman.us/scripts/fw/fspaceshooter.html

Depot, W. (n.d.). */25-amazing-javascript-games-some-fun-and-inspiration/*. Retrieved 11 11, 2015, from www.webresourcedepot.com: http://webresourcesdepot.com/25-amazing-javascript-games-some-fun-and-inspiration/

jlongster. (n.d.). */Making-Sprite-based-Games-with-Canvas*. Retrieved 11 11, 2015, from jlongster.com: http://jlongster.com/Making-Sprite-based-Games-with-Canvas

top-down-2d-rpg. (n.d.). Retrieved 11 11, 2015, from http://opengameart.org: http://opengameart.org/content/top-down-2d-rpg