# Gunship Galaxy

## Introduction­­

When we first got together, there were obviously a lot of ideas thrown around the table. Building a WebApp is such a broad specification that it was hard to know where to start. Our first big decision was platform: where did we want people to use our app? We first met the morning after Google I/O 2013, and the demonstration of the new app facilities there (several new APIs, the Android specific IDE based on IntelliJ, to name a few) had a couple of us rooting for Android development. We put that idea to one side, however, as we realised that basing our decision to develop on Android on a flashy presentation was probably unwise.

At this stage we took stock of the devices we used individually. Two out of the four of us had iPhones, and the other two had Android phones. It was then that something else from Google I/O inspired us. We all thought [Chrome Racer](http://www.chrome.com/racer) was pretty cool. So much so, in fact, that it was at least an hour before we did anything productive after we found it. We liked the idea of using Chrome as a kind of ubiquitous platform across devices, and started thinking about how we could do something interesting with it.

After a few iterations, we came up with the idea of using one device as a “monitor” and a number of other devices as “controllers”. We then discussed the potential issue of latency. Firstly, we could design a game where latency would not be an issue in the first place (such as something like Scrabble, where each player could hold their letters on their device, and the “monitor” could show the whole board). However, we realised that this had the potential to be fairly unimpressive.

At the other end of the spectrum, we could design a fast-paced game where latency had the potential to be problematic, but which would be much more impressive when it worked. We decided that the game we ended up with need not be too complex, as successful internet games often are not (see [QWOP](http://www.foddy.net/Athletics.html), or [Boxhead](http://www.kongregate.com/games/SeanCooper/boxhead-the-nightmare-full?ref=stcg)), and that we could find a sweet spot with respect to the latency issue and the pace of the game. Websockets seemed to guarantee a certain degree of speed in connections, and this was a reassurance.

As such our goals were as follows:

1. Using Websockets, design a way of controlling a “monitor” session with a “controller” device.
2. Design a simple game which could be played with it.
3. (Theoretically:) Have multiple potential games to play using this system.

­

## Overview

Figure 2 Gameplay screenshot

Figure 1 – Gunship Galaxy opening menu, unpolished

Our WebApp is a multiplayer 2D arcade style chaotic shooter. Its main appeal – despite the obviously brilliant gameplay – is its retro style, and the ability to play across multiple devices with friends. A basic game session consists of:

* Logging in
* Setting up your game
* Battling against your friends or versus AI.
* Repeat last 2 steps ad infinitum.

## Gameplay

The concept is simple, you pilot a gunship, and the aim is to destroy all the other ships by shooting them enough times. The last one standing wins.

Your ship goes forward all the time, and there is no way to stop it. All you can do is rotate the ship to go left/up/right/down – the only 4 possible orientations – and shoot in front of you. If you reach the edge, your ship automatically turns 180°.

Bullets travel at twice the speed of ships. When they reach the edge, they bounce back and their speed doubles. If they reach the edge again, they are destroyed. If a ship is hit by a bullet, it loses a life. If a ship has no life, it is destroyed.

Bonuses enrich the game, and are what makes it so replayable. Bonuses can randomly spawn and, then be picked up. Examples of bonuses include:

* Invincibility: For 4 seconds, if you are hit by a bullet, you do not lose a life
* Gun disabling: For 4 seconds, all enemy ships have their gun disabled
* Life: Gain one life
* And many more...

When a player wins the game, the data of the game is sent to the database – final rankings, how many times each ship has been hit/has hit somebody. This data makes up the users profile, and in the future, could allow other players to compare stats, and have leaderboards

.

## Group Structure

* Mihai: server side communication.
* Will: client side login, backup/version control.
* Conrad: front end game development, sound design and team coordinator.
* Guillaume: front end game development, graphics.

## Implementation Languages

-Java on server-side: it's the required language for writing servlet functionality on the Tomcat server; it has a wide range of packages to support whatever functionality you could fathom (particularly, it has websocket and database access support); should be easy to deploy the webapp on any platform which has the required support(the cross-platform aspect);

-HTML5 with JavaScript on client-side: JavaScript is the de-facto standard for dynamic functionality of web pages on client-side. It is virtually platform independent, and is supported across desktops, laptops, tablets, mobiles etc. In contrast to this, other technologies/languages, such as flash/silverlight/java applets , are not supported by many mobile devices, particularly ones running iOS, which is one of the most popular mobile operating systems.

## Backup system

We have been using a git repository hosted on github.com to maintain our different versions of the project, and to ensure our WebApp is backed up. However, we still had to make sure any sort of merge conflicts would be avoided ( since that would have probably resulted in unsatisfactory automated resolution by github.com). The approach was mainly to delegate the tasks concerned with different aspects of our WebApp, with prior agreement on what the requirements would be for each component (as to allow easy integration).