AMERICAN UNIVERSITY IN BULGARIA

DEPARTMENT OF COMPUTER SCIENCE

COS 491 Senior Project I

2D Web Game

Sadi Qevani, 100080165

Author: ______________________ , **Date:** _________
signature

Supervisor: __________________ , **Date:** _________
signature

Blagoevgrad, 2014

AMERICAN UNIVERSITY IN BULGARIA

DEPARTMENT OF COMPUTER SCIENCE

Title: 2D Web Game

Title in Bulgarian: 2D уеб игра

Author: Sadi Qevani

Abstract: The main focus of the 2D Web Game, is to build a local co-op game with web technologies. The use of this technologies allow the developer to build high tech, full scalable mobile/web application that can run on almost any device that has a browser implemented in it.

The implementation of the game is done in such a way that the user has to connect to the game and play on a local environment and thus increase the fun/enjoyment while playing.

Todays companies focus more on the online feature, but i believe the true core of gaming is playing with your friends on a room in front of a screen. The amount of user who can participate on this game is unlimited, so there is no limit to the users who can play.

Declaration of authorship:

"The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of John Galletly. All sources, used in development, are cited in the text and in the Reference section."

Author: ___________________________________

AMERICAN UNIVERSITY IN BULGARIA

DEPARTMENT OF COMPUTER SCIENCE

Contents

1. Introduction
   1.1 Overview of web games
   1.2 Meteor Impact
   1.3 Purpose
   1.4 Target Environment
   1.5 Objective and Success Criteria

2. Analysis of the application-domain problem
   2.1 Functional requirements
   2.2 Non-Functional requirements
   2.2.1 Usability
   2.2.2 Reliability
   2.2.3 Performance
   2.2.4 Supportability
   2.2.5 Implementation
   2.2.6 Portability
   2.3 Use cases

3. Design of the software solution
   3.1 Introduction into the design of the app
   3.2 Server Side
   3.3 Client Side
   3.4 Mobile Side
   3.5 Cloud
   3.6 Game Assets
   3.7 Security
   3.8 UML Class Diagrams

4. Implementation
   4.1 Technologies used
   4.2 Testing and debugging

5. Conclusion
   5.1 Difficulties encountered during development
   5.2 Future Development
   5.3 Screenshots

6. References

1. Introduction

1.1 Overview of web games

The web is 25 years old, yes 25 years of increasingly innovation in the field of computers, and if you had the chance to experience the web 25 years ago, you would see that the amount of innovation in this field is enormous. Thanks to the web people now are more connected and we have seen small startups flourish and become the biggest and most powerful companies in the world (Google).

But the way back then was really different form the Web today, at that time people were limited by the technologies they had, and they could not do to many things on the web. The web was slow, not friendly and it was really small compared to todays size, where billions of people are connected through a computer or a mobile device. At that time you could not build great apps, and games, because there were many inconsistencies though browser vendors (Browser Wars, Netscape VS IE). But today is different from back then, today we have a lot of browser competition which is driving innovation, today we have WebGL, Canvas, HTML5, ECMAScript 6 and 7, NodeJS. We have great tools, which allow us to build great products.

One of the major field in the web which will grow exponentially in the next coming years are Web Games. I am confident that the browsers will become really powerful and they will handle games better than any platform. For today we have several softwares that allow us to run 3D games with amazing graphics, the ACMjs project from FireFox, allows you to do amazing computations thanks to its low level API. Brendan Eich in 2011-2012 demonstrated a game build with Unreal Engine in jsConf, and it was a full fledged 3D game with amazing APIs, if he could do that in that period of time, imagine what we can achieve in the next upcoming years.

My motivation to do the Web Game was my passion for the web, and the web technology stack (HTML5, JavaScript, NodeJS). I didn't want to just build a 2D game, i wanted to build something that could show the full potential of the web technologies and i believe i have succeeded in making a game which is cool and at the same time shows the potential of the web.

HTML5 is the successor of the last standard of the markup language in the web, that is HTML 4.01. HTML5 brought many good stuff to the web, such as:

- The canvas element for immediate mode 2D drawing.
- Timed media playback
- Offline Web Applications
- Document editing
- Drag-and-drop
- Cross-document messaging
- Browser history management
- MIME type and protocol handler registration

• Microdata
• Web Storage, a key-value pair storage framework that provides behaviour similar to cookies but with larger storage capacity and improved API.

• Geolocation.
• The Indexed Database API, an indexed hierarchical key-value store (formerly WebSimpleDB).

• HTML5 File API, handles file uploads and file manipulation.
• Directories and System, an API intended to satisfy client-side-storage use cases not well served by databases.

• File Writer, an API for writing to files from web applications.
• Web Audio API, a high-level JavaScript API for processing and synthesizing audio in web applications.

• ClassList API

While all the following technologies are amazing, the problem that we have currently have is the implementation of this technologies in major browser vendors, since IE6,7,8 are outdated versions but still there are people who keep using them, and in a way we have to think about that user base as well, since if we drop the support for this browsers, we might lose clients/users.

ECMAScript refers to the standardized version of JavaScript (JavaScript is a trademark of Oracle, so that is why JavaScript toke the name ECMAScript). With the upcoming of the next version, ECMAScript has some really nice features implemented into the language, that will make Front-end Developers even more productive and will make the code more modular.

NodeJS, is “is a platform built on Chrome JavaScript Runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.” - As taken by NodeJS.org,

What this means is that nodejs is a runtime environment where javascript developers can run their code and use the API provided by this machine to build server side applications. The nice thing for NodeJS is that its really fast and non-blocking, which allows us to build real time applications on the web and which is the major building block for my game, without this piece of technology, i don't think i could have build a real time web game.

1.2 Meteor Impact

Meteor Impact is a game that i wanted to build long time ago, and with the motivation for my senior project, i achieved that. Meteor Impact is not just simply a 2D Web Game, its more than a 2D Web Game, its a co-op Multiplayer 2D web game, where your phone is the controller, and the screen is the computer. I believe this game could not be achieved without the amazing technologies provided by the web and the browser.

The idea behind this game is to use Sockets to control the instance you create on the game, and thous trying to defeat the adversary (AKA your opponent). The way the game works is simple, you open the game from a browser vendor, and the game preloads the assets and does everything that needs to do on the background, then we have a landing screen, which requires the user to click so the world can be created. After the world has been created the user should go to the mobile version of the game, by going to url/mobile, in that moment the game will understand that a user has accessed the controller, and it will instantiate a new Space Ship which only the specific user can control.

There are additional complexities on how the game works, but this should be a basic overview.

The idea that i can control something from another computer with a mobile phone excites me, because there is an infinite amount of potential to the apps and games we can achieve in this way. Imagine having a web app that you can control a sword through your phone and play with another person online, or maybe some equilibrium games. Also we can have consistency throughout our apps as we can send messages to our apps when we do something, the same concept apple has implemented in Yosemite, for safari browser when you want to move your page you are viewing from your mobile phone to safari.

I know this are technologies and they will get outdated sooner, and new stuff will appear, but the idea of controlling apps from your phone to your PC has a great potential for new apps and games, and every day we are coming closer to have more games that interact with us in many more ways than before.

1.3 Purpose

The purpose behind the project was to build a game that would be played locally with friends and that i would have to control the game with a remote controller (mobile phone).

The user should enjoy the game in a local environment with this friends in front of a client machine (PC).

1.4 Target Environment

The target environment which the game focuses to achieve is browser vendors, that have implemented the HTML5 technology and allows the use of Canvas or WebGL.

The target can be:

- Children - Kids between 7 to 13 years old who would like to play a game.
- Teenagers - Teens between 13 to 18 years old, they would like to play against their friends to compete and also show something cool to their friends.
- Adults - People between 18 to 35, who would want to play in work with their co-workers in a competitive game that would not require any console or remote controller except their mobile phone and a pc.

1.5 Objective and Success Criteria

The objective of the game is to provide a good gaming experience and enjoyable time with friends.

The success criteria would be that the game is popular and plenty of people are playing it.

2. Analysis of the application-domain problem

The functional requirements define what features the application (game) has for its users, and what they should expect from the application (game). The application core is sockets and cloud. The game makes use of the sockets which sends and receives messages, so the Internet connectivity to the cloud is of highest importance. Internet connection should be available for the program to work, although, some parts of the program are also cached by the application so they can be used even without Internet connection.

2.1 Functional requirements

2.1.1 The game should run through a browser and the game will start to load the assets needed to run the game. Because the game is using web technologies, we should use some browser vendors that support some specific technologies such s Canvas/WebGL.

2.1.2 The application should provide the user with an interface to allow him to control the object/player class. This is done through the mobile version of the game, which opens a new view and provides him with buttons that sends some messages on the backend, and allow him to control the game.

2.1.3 The application (game) should be able to connect and share data with the online servers. The server of the application is all stored in the Cloud. Application takes advantage of the Internet connection to get the data from the Cloud and also send data to the cloud. The messages that the user sends when he types a button is going through a socket message to the cloud and the server in the cloud replies with another message that is sent back to everyone, so the view can be updated.

2.2 Non-functional Requirements

2.2.1 Usability

The game can be accessed by any browser who supports the specific needs for the game to use such as canvas, WebGL, sockets, and several html5, css issues. But its suggested to not be accessed directly by a mobile or table, since the screen on this devices is to small, and it might affect your experience and usability on this devices.

The game can be made to be accessed by a bigger range of devices, but this will need in-code optimization, shims, additional libraries for them to work.

The user will need a smartphone for him to access the game, the game is suppose to be controlled remotely by another browser, be it a mobile or table or even another computer with internet access.

2.2.2 Reliability

The game makes use of the internet, and thus someone with with a bad internet connection wont have a pleasant gameplay. Because a bad internet connection might cause the users to lag, since the message sent by the users phone might arrive late on the browser.

A browser with a good support of web technologies is needed, so versions of IE 6,7,8 and most probably even 9 wont be supported, since they are outdated browsers and for the optimal performance a good browser is needed.

2.2.3 Performance

The performance of the game depends on the type of the device, if the hosting device has low spec, it might be a bit hard to run a clean 60 frame per seconds. Also internet might affect the performance.

The game needs a continuous connection to the Internet, because the controller and the game talk over the internet with sockets connection. Internet connection is required to retrieve and send data from the Cloud. The datas include messages from the player (direction of the spaceship, ids of the players).

Internet connection is a must for Meteor Impact. Information should be up-to-dated, since the game needs to track where the Users are and their command that's why it needs continuous Internet connection. The game also caches some part of the the assets, like sound, images, but the internet connection is needed to control the player.

2.2.4 Supportability

The game will be supported by a cloud solution platform, called Heroku, which will take care of all the Cloud functionalities, like CDN, URL, Bandwidth, etc.

The game will be firstly released in beta version, for testing purposes and bug reports. This specific games supports multiplayer connections, so we don't know how the game might act after 100 players connect, or after 1000, so we need to do the appropriate debugging.

Feedback will be welcomed, and interaction with users to fix the bugs will be the main purpose of the beta phase.

The app will be lunched online with a specific URL, i will also share the source code, so people who have specific knowledge will fix bugs on their own and push it to me through GitHub.

2.2.5 Implementation

- IDE (Integrated Development Environment)
  - I have used WebStorm as the main IDE to develop the application, i have chosen this specific IDE, since i am comfortable with it, and its an amazing tool to develop applications, its easy, has code sniffing and many other tools that allow me to concentrate on the problem solving.

- User Interface
  - I have used some open source sprites and resource that i found online, the license for the assets are with Common Creative, which can be used in any project without the need to pay or display any copyright notice.

- Cloud
  - The platform i decided to put my system is Heroku. Heroku is a cloud platform that supports a wide range of development tools, starting from Ruby on Rails, to NodeJS, and that is why i decided to pick heroku, another reason was that, it allows me to use a single core instance for free, and i wont need to pay to showcase my application.

- Testing
  - Testing of the game have been doing with Google Chrome Developer Tools, and the mobile testing have gone through the Android Device XPERIA Z3. This tools are easy to use and i could debug them, and they worked flawlessly on this devices.

2.2.6 Portability

The game can run on any supported Browser, which includes the latest and most upcoming browser vendors.

The game can run on mobile and tables, but to achieve a maximum pleasant game, we need to add additional tools, libraries, and optimize the game for mobile plays.

2.3 Use cases

First Use Case

Name of the use case: Starting Game

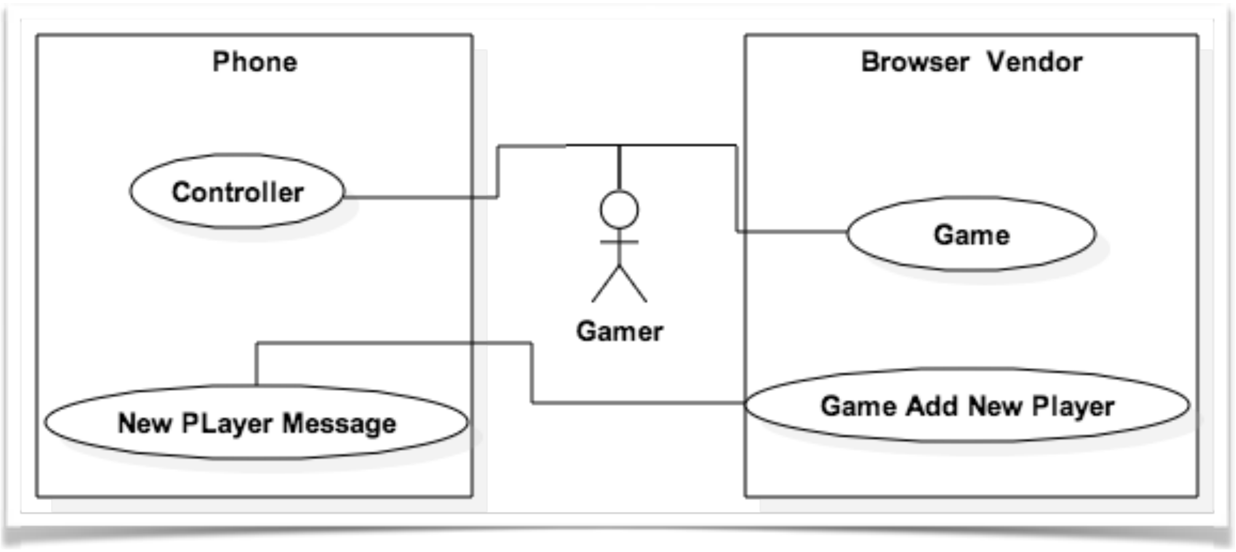
Participating Actors: Gamer

Description:

The user opens the web browser, inputs the URL of where the game is located, and in that moment the Browser parses the code (html, css, javascript) and runs it locally. The user gets an text asking to click. User clicks and the game starts. The browser generates the world, the meteors are randomly generated and placed randomly on the world.

The user takes his phone and open the browser in his phone, and access the mobile link, the moment the user accesses the control on his phone, an new instance is created in the game, the instance is the players ship, which only that player can control.

Use Case Diagram:



<u>Second Use Case</u>

Name of the use case: Ship Control

Participating Actors: Gamer

Description:

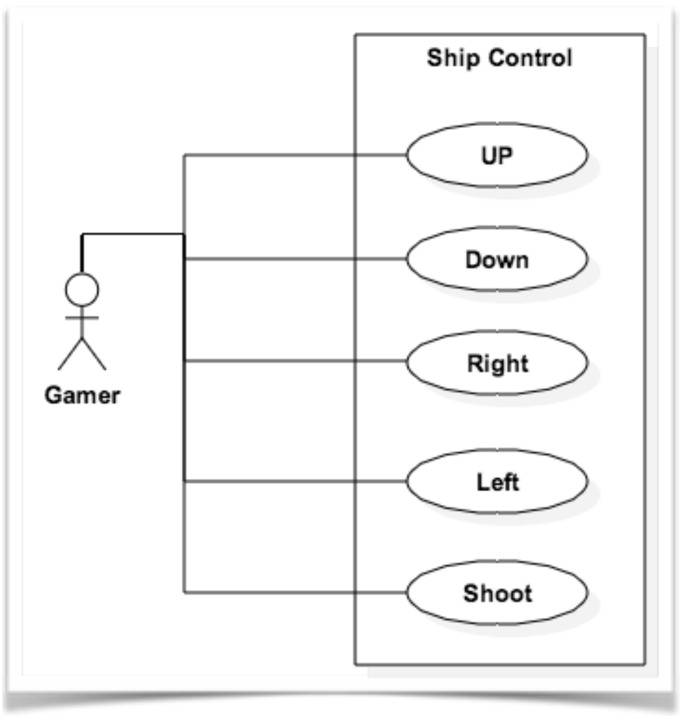
The way that the system is built, is that the player must control the ship through a mobile phone, he needs to access a link, and then the link will generate a new ship on the screen, and the user can control the ship with the commands provided by the control.

The control has 5 basic commands, which are: up, down, left, right and shoot.

The Up command pushes the ship forward.
The Down command pushes the ship downward.
The left, rotates the ship to the left.
The right rotates the ship clockwise. (to the right).

And the shoot button, opens fire, which sends a object from point x to point y, and if in the way it collides with an object it will destroy that object.

Use Case Diagram:



<u>Third Use Case</u>

Name of the use case: Multiplayer Game

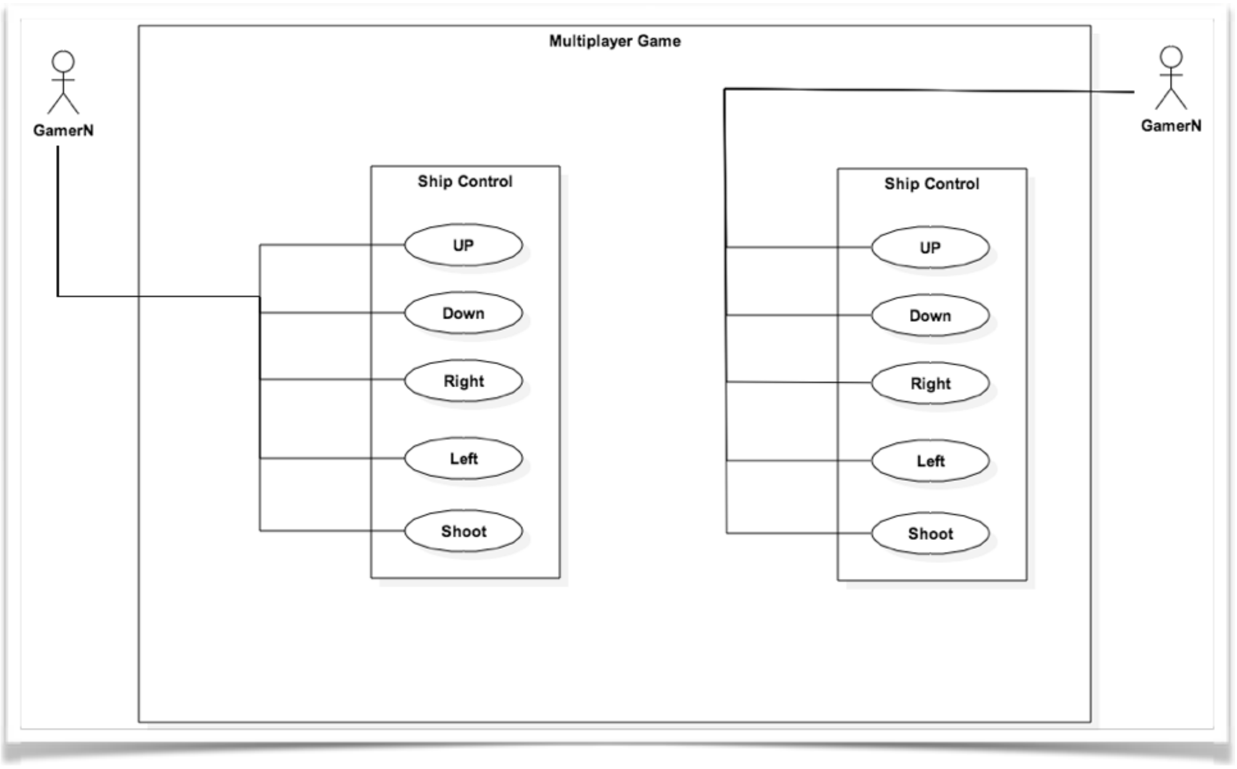
Participating Actors: Gamers

Description:

The multiplayer case refers to the feature the game has to support many devices connected at the same time, and that allows the players to play against each other.

This feature has been achieved with the sockets technology, and for every device that is being inputed a new ship is generated on the screen.

Use Case Diagram:



3. Design of the software solution

3.1 Introduction into the design of the app

Meteor Impact is built entirely on JavaScript. The language of the Web. I decided to build it on JavaScript due to the technologies that are emerging with this languages. I would mention Sockets, NodeJS (server side). Another reason i decided to go with JavaScript, is that this language achieved what Java could not. Write once, and run everywhere. Every device today has a browser implemented, and that is the reason why JavaScript would be a perfect language to write cross platform apps or games. But JavaScript nowadays is not only on browsers, but developers are implementing it in other platforms such as Open Source circuits like arduino, etc. So you could see how much popular and how much variety has JavaScript.

The application makes use of Canvas or WebGL. I have decided to use a framework to help me build a game that would be really performant in almost all devices and platforms. The framework tests for several feature of the browser, if for example one browser does not support WebGL (which is the default) it switches over to Canvas.

Since the app is suppose to be used/controlled by a remote controller (mobile phone), i've had the need to implement several other technologies such as NodeJS, and Sockets.

Like i have mentioned above, NodeJS is a platform where you can write JavaScript on the server side, and NodeJS takes care of providing the assets to the browser, and also takes care of the routing.

Sockets, are a new feature in the browser, though they have been around for much longer than the web. Sockets helped me to send messages from client to the server and from the server to the client. This allowed me to have a 2way messaging system with client-server-client, so i could communicate with the mobile phone and then have consistent communication from the mobile phone to the server and to the client.

The game consist of several states. I call them states because they run independently of each-other. Everyone has a specific feature that needs to accomplish.

- **Boot**: This states is the very first state to be executed, and boots the game.
- **Splash**: This state is suppose to show your game/company images.
- **Preloader**: this states preloads all the assets needed to execute and run the game.
- **Menu**: This state is where the code for the menu is located.
- **Game**: This is the final state where the game runs.

3.2 Serve Side

Server side refers to the code that is located in the server and is hidden from the browser, the browser does not know anything about that code, it only knows that when its sending a request to the server, its getting a reply usually the server passes a string of ascii characters, and then the browser decides what it does with it.

Meteor Impact server side code is entirely on JavaScript. Its running on the V8 Engine from Google, and its really performant. The technology that is making this possible is NodeJS.

Node.js is a platform built on Chrome's JavaScript Runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js provides way more tool that to create a simple http server. nodejs provides tools for IO, DNS, and many others, but this things are no need for us.

The server API on node.js is a bit rough so i am using an Abstract Layer for the server management. The framework that i am using to receive http requests, is called ExpressJS. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Express is only a web framework that provides an API for me to control several complex features that Node has, but its not the core functionality that allows me to pass messages from client to client. The tool that does this job is SocketIO.

Socket.IO enables real-time bidirectional event-based communication.
It works on every platform, browser or device, focusing equally on reliability and speed.

Again SocketIO is an abstract layer on top of Sockets technology, because sockets should work both on the client and server side of the game, i decided to use this framework, since its providing with fall back. This issue falls on the browser vendor issues and their incompatibilities with several new technologies.

SocketIO if it does not detect the socketIO technology falls back on UDP requests, if this does not work goes back to using simply AJAX requests. This is a really convenient tool, since we want the game to work the same way no matter what browser vendor or phone version we have.

3.3 Client Side

The client side, refers to the code that is running on the client side. Differently from out server side code, the client side code is visible to the user, and the user can clearly see the whole code structure.

The client side is the place where our game is running, the assets are loaded in the client, the game is running on the client, and depending on the pc performance, the game might run flawless or might have problems.

There are several ways to build a game on the browser, we can use WebGL, or we can use Canvas. WebGL is used to build 3D games, while the Canvas is a drawing platform, where we can draw stuff on top of each other, but for a 2D game is more logical to use canvas rather than WebGL, since it will be more performant than the WebGL technology.

Since i would like to build a modular approach to the architecture of the game, i decided to go with implementing a tool that allows me to include objects, in a way that you include namespaces in c#. The tool is called Browserify.

Browserify let me code with modularity in mind, browserity separates the code into modular object, and i can include them, with dependency in mind.

3.4 Mobile Side

The Mobile side of the platform refers to the Controller, that, that part of the game is a separate view from the main game location. In fact the mobile view is called mobile.html, and there resides the input control for the game. From that view i am also importing an additional JS file called mobile.js

Mobile.js is the file where the event handles for the buttons reside. I am suing Bean Even handler to perform the events, since events method calls differs from browser to browser, we need a way for the events to work seamlessly on all devices, including smartphones.

When a user touches an button, an event is fired, and the function inside it that has socket method, will execute and send a message to the server. And the server will reply back to the clients screen, and depending on the client event, you will have an effect in the screen.

I have also used CSS to style the buttons so they player can have a nice user experience while playing the game.

I did not use any preprocessor for the CSS, since i wanted to use simply vanilla CSS.

3.5 Cloud

Heroku is a cloud platform that provides PaaS. The reason i chose to go with Heroku is that heroku provides NodeJS Development environment. And another major point is that, for the first core instance, heroku is free. So a student like me can use cloud up to a certain point, and i believe this is a big selling point.

The way we deploy on heroku is through command line. We install the heroku app, which provides an interface for me to push the code through Git.

When i first initialize git, i add the files and add the commits, i will need to add the remote location of the git repo for heroku. When i do that, i just have to push using a simple command “git push heroku master” and then heroku will do its magic for me.

3.6 Game Assets

As a developer i am not someone who can be good at design, and i have no skills in design. But thanks to the open source community i found some really nice looking assets which are provided for free, under the common creative license.

I have used several assets from this resource pack, that including the backgrounds. The sprites for the spaceships. Meteor sprite pack and pullet sprite pack.

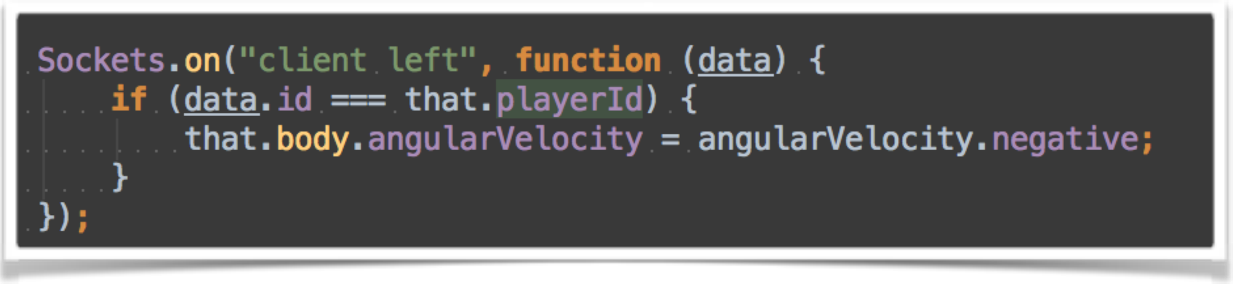
3.7 Security

Since this is a game and has to be played in a local environment, we don't have any data stored in our database, there is no checking for information, nether there is any issue with XSS, nor SQL injection problems.

The only issue that may arise is that people will try to cheat the game, but this would be hard to accomplish, since the environment which the controller works would be only in the phone, or in an environment that the touch should be initialized.

Also since i am using sockets, i am doing some checking when people type the buttons to rotate or to shoot.

Code:



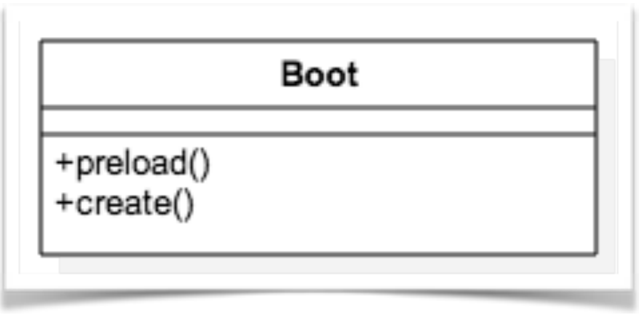
```
Sockets.on("client left", function (data) {
    if (data.id === that.playerId) {
        that.body.angularVelocity = angularVelocity.negative;
    }
});
```

3.8 UML Class Diagrams

3.8.1 Boot

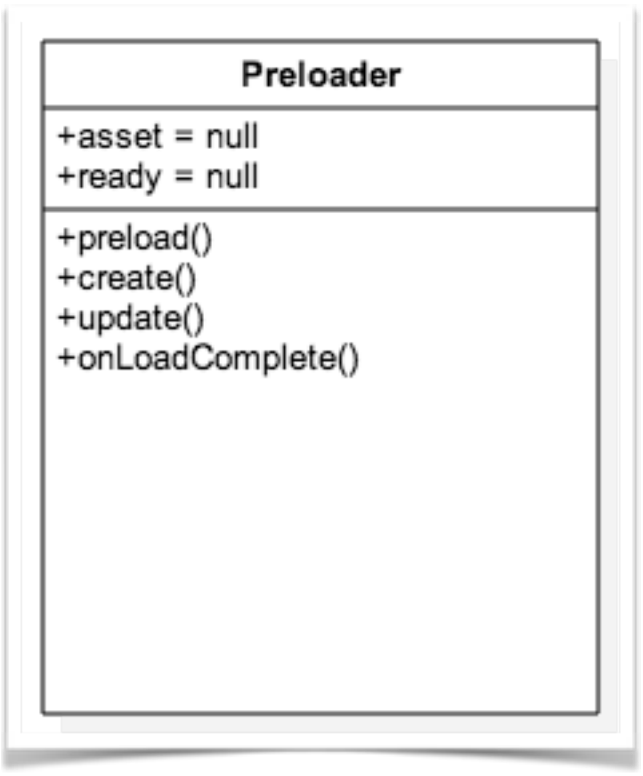
The boot Class, is the most simplest class in our game, the boot class is the class that starts directly when the browser first initialize. The method that is executed when this class is activated is create().

The preload() class preloads some small assets as the initial state of the game.



3.8.2 Preloader

The preloader class, is executed after the gam has booted, and it preloads a list of assets that are suppose to be attached to some objects after the game has been completely loaded.



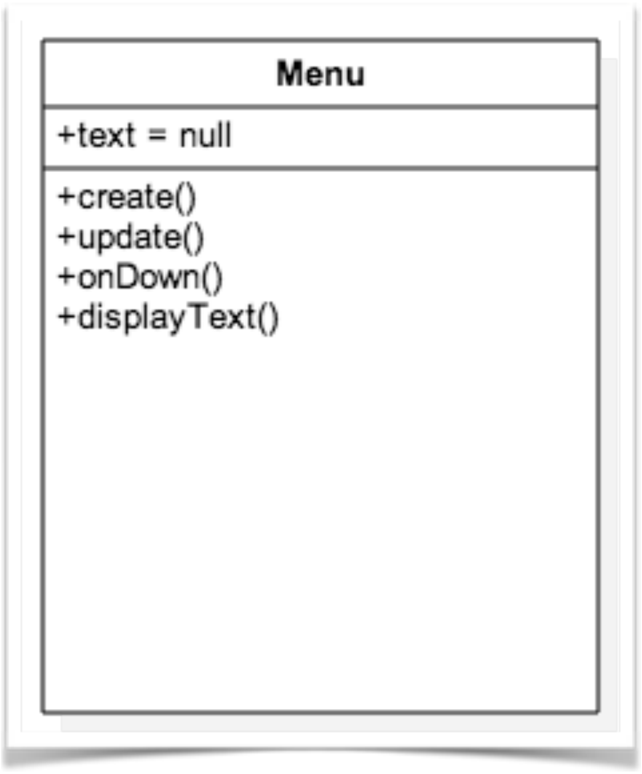
This class will show a loading screen, and it will transfer to the menu state once all the elements have successfully loaded.

The asset property is where the asset object is created. The ready property is initialized false, and once it has been set to true the game will continue to the menu state.

The update() method, is a method that is executed 60 times / second (60 frames per ceond). This is where most of our logic happens when we want to update the view.

3.8.3 Menu

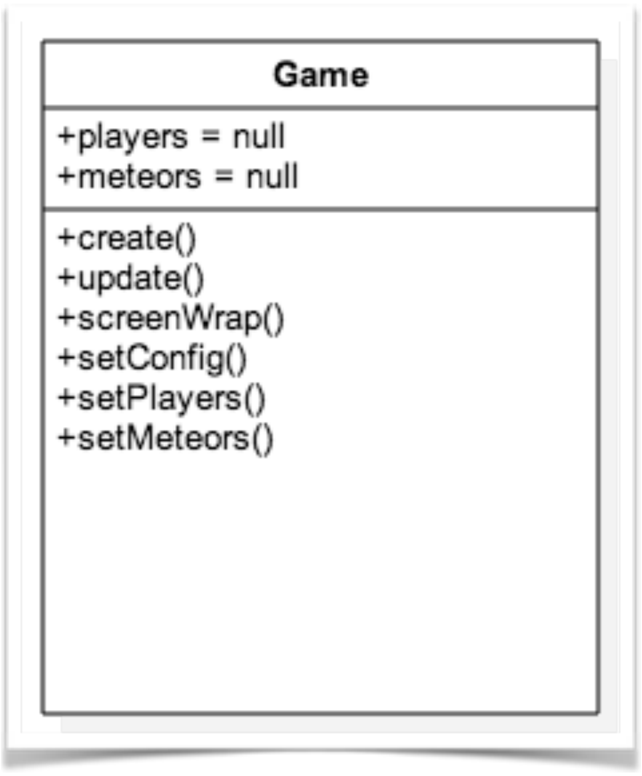
The menu class is executed when the preloaded has finished. In this class we generate a menu, and we attach an event handler that when user clicks on the screen the game will generate the environment and start the game.



3.8.4 Game

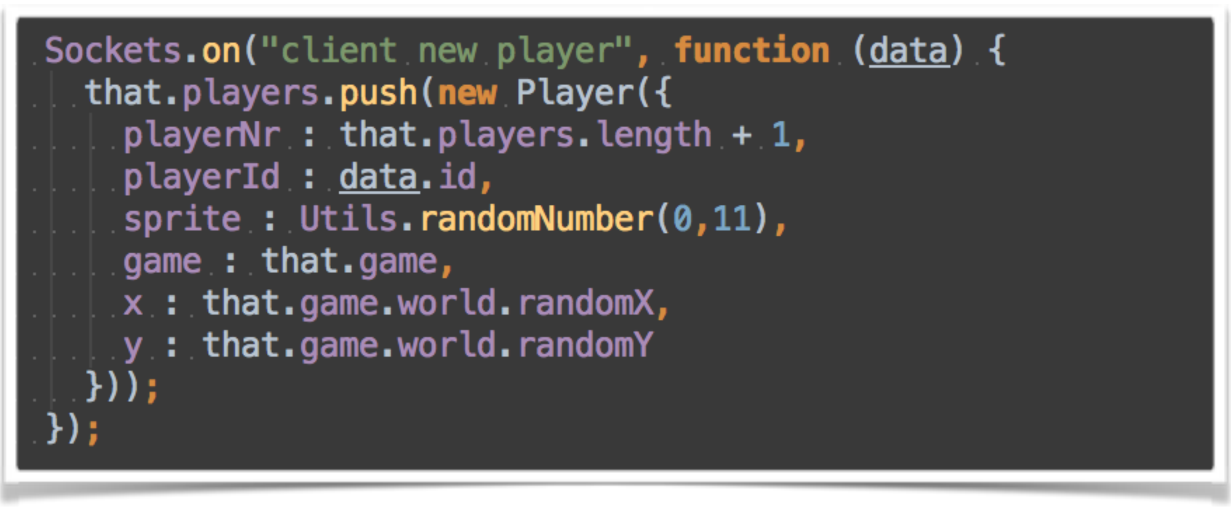
The game menu is the core of the game, everything that is happening when user starts the game is coded up on the game menu.

The game menu has several functions and events. To make the code more modular i have separated the concerns into several functions which all executes in the create() function.



The **setConfig()** method, sets the initial state of the game by setting several options like the background music, adds the background and also sets some other features like pixel foundation which makes the images cleaner.

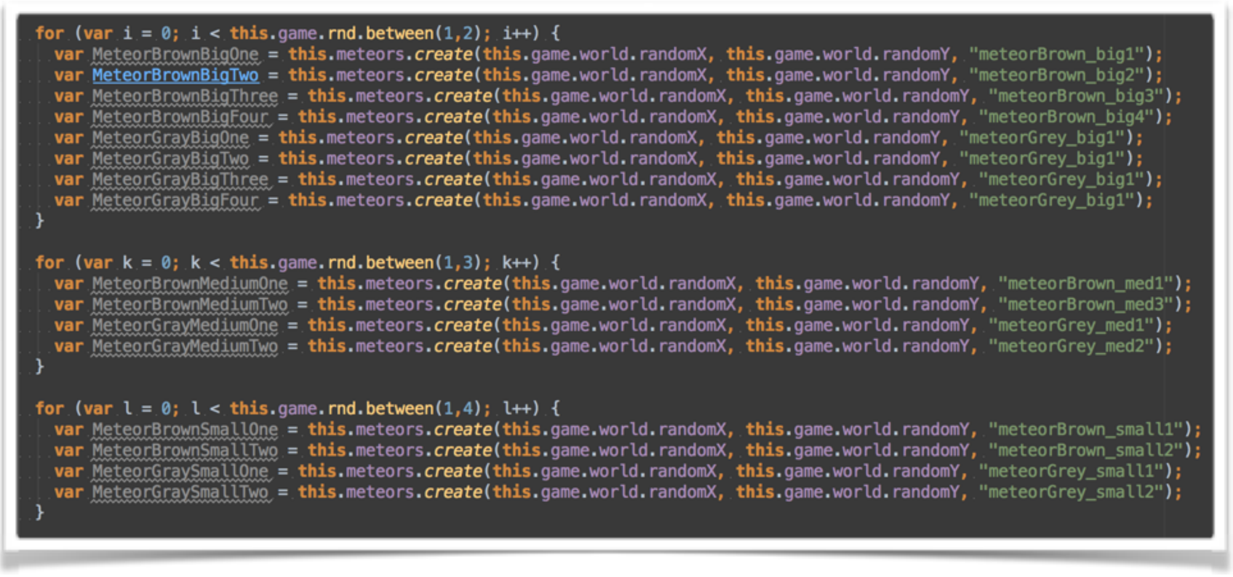
setPlayers() encapsulates the functions where the players are initialized. The way setPlayers work is really interesting, taking it into account that i have to add multiple users. So the players property is an array, which is generated for every user who is being created.



```
Sockets.on("client new player", function (data) {
  that.players.push(new Player({
    playerNr : that.players.length + 1,
    playerId : data.id,
    sprite : Utils.randomNumber(0,11),
    game : that.game,
    x : that.game.world.randomX,
    y : that.game.world.randomY
  }));
});
```

As you can see from the code, we have an socket event, and its pushing a new player object into our array of players. In that initialization is taking several options, like the initial position of the spaceship, the game object, its randomly assigning a sprite and its giving the player an id and number.

The last method is the **setMeteors()** method, which is generating different meteors for our screen. The way i have implemented this is through several loops, since we have different size for our meteors.



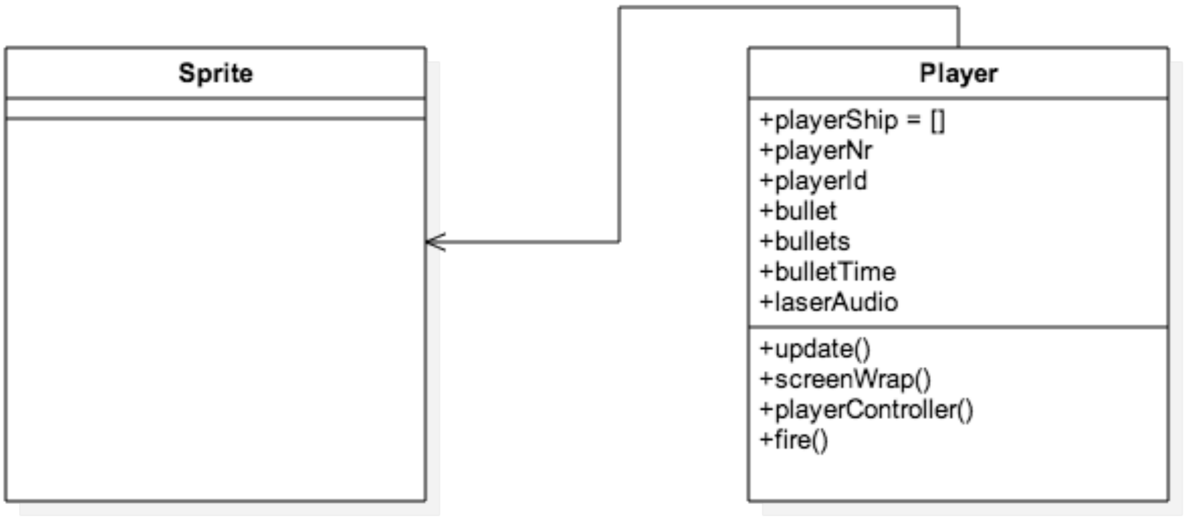
```
for (var i = 0; i < this.game.rnd.between(1,2); i++) {
  var MeteorBrownBigOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_big1");
  var MeteorBrownBigTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_big2");
  var MeteorBrownBigThree = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_big3");
  var MeteorBrownBigFour = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_big4");
  var MeteorGrayBigOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_big1");
  var MeteorGrayBigTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_big1");
  var MeteorGrayBigThree = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_big1");
  var MeteorGrayBigFour = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_big1");
}

for (var k = 0; k < this.game.rnd.between(1,3); k++) {
  var MeteorBrownMediumOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_med1");
  var MeteorBrownMediumTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_med3");
  var MeteorGrayMediumOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_med1");
  var MeteorGrayMediumTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_med2");
}

for (var l = 0; l < this.game.rnd.between(1,4); l++) {
  var MeteorBrownSmallOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_small1");
  var MeteorBrownSmallTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorBrown_small2");
  var MeteorGraySmallOne = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_small1");
  var MeteorGraySmallTwo = this.meteors.create(this.game.world.randomX, this.game.world.randomY, "meteorGrey_small2");
}
```

3.8.5 Player

Now the **Player** class is the most important one, this class is the one responsible for the control of the spaceship, the movements, the firing, screen wrapping and many more.

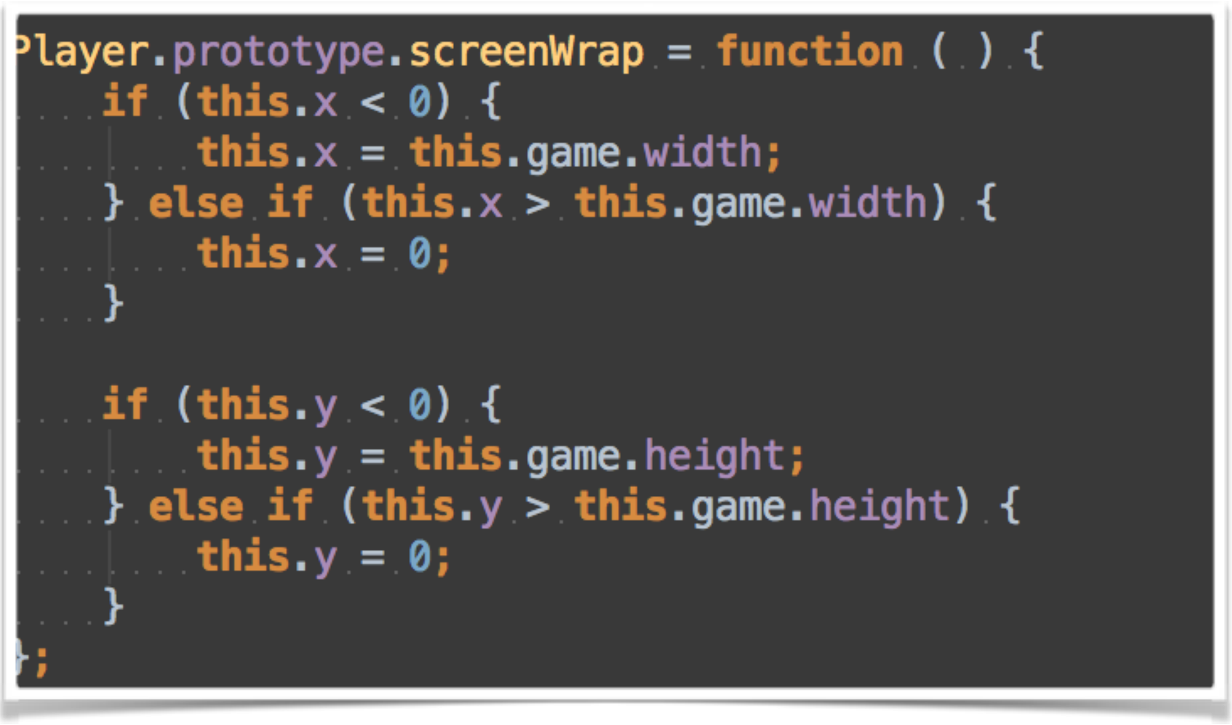
As you can see from the diagram below, the Player class Inherits from the Sprite. The Sprite class is the a global class in the Phaser object, and its responsible to control any object into the world.

The sprite has many methods, to many to insert into an UML diagram, and this class is an object that is provided by Phaser object, which this allows me to control the object, and provides an interface for positioning etc.



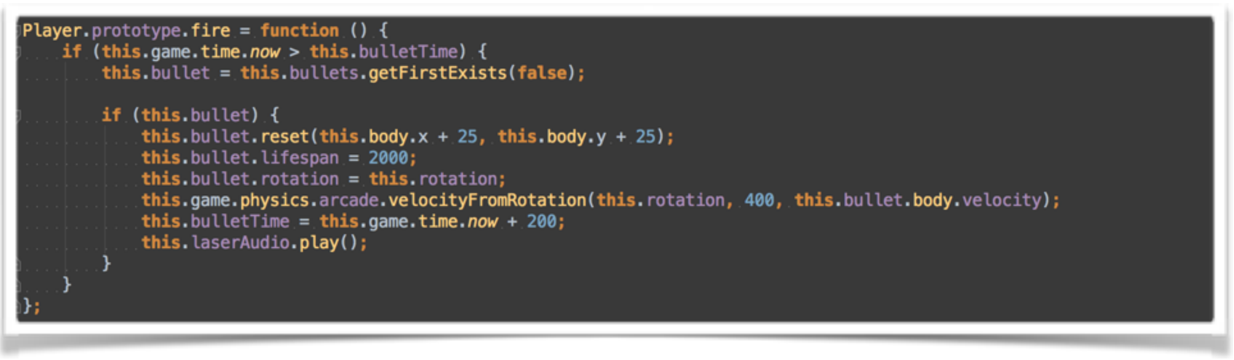
The **update()** method is the same thing over all the classes i have created, is directly referenced through the prototypal inheritance that javascript has.

screenWrap() is an implementation that i have done to move the spaceship on opposite side of the position if it hits the borders of the world. This is a feature that existed in the old game as well, where you would go over and the objects would wrap and pass to the other side. below is a screen of the algorithm which i am using to accomplish this.



```
Player.prototype.screenWrap = function ( ) {
    if (this.x < 0) {
        this.x = this.game.width;
    } else if (this.x > this.game.width) {
        this.x = 0;
    }

    if (this.y < 0) {
        this.y = this.game.height;
    } else if (this.y > this.game.height) {
        this.y = 0;
    }
};
```

The **fire()** method is the method which the player shoots the bullets and they collide with each other. Its designed in a way that it will generate some group of sprites, and then when they reach a specific range they will destroy theirself, also there is an audio when someone fires every time.



```
Player.prototype.fire = function () {
    if (this.game.time.now > this.bulletTime) {
        this.bullet = this.bullets.getFirstExists(false);

        if (this.bullet) {
            this.bullet.reset(this.body.x + 25, this.body.y + 25);
            this.bullet.lifespan = 2000;
            this.bullet.rotation = this.rotation;
            this.game.physics.arcade.velocityFromRotation(this.rotation, 400, this.bullet.body.velocity);
            this.bulletTime = this.game.time.now + 200;
            this.laserAudio.play();
        }
    }
};
```

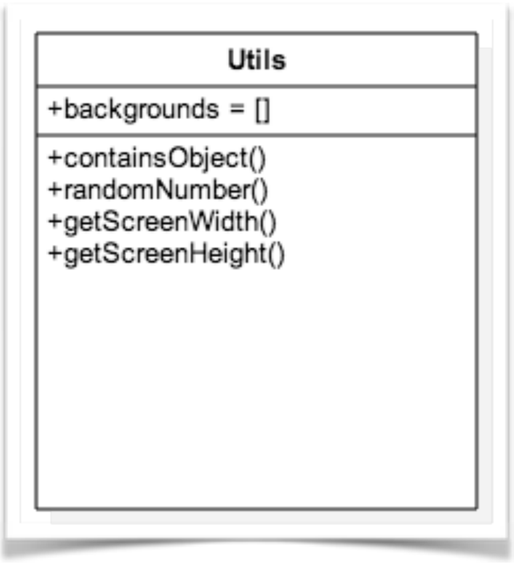
As you can see, we detect the time now, so we wont make the mistake to send 2-3 bullets at the same time, so you can send only 1 bullet per a limited time.

Then we detect if a bullet has been created, and then we give it a position, (usually its where the head of the ship is directed) and then we provide a velocity.

The major issue in this was trying to figure out a way for me how to rotate the bullet when the spaceship would rotate.

The last but not least, its the **playerController()** method**.** This method is the main core functionality to deal with the controllers of the spaceship. It has implemented Sockets and then on specific events the ship should move. All of this functionalities are placed inside the **update()** method, which like i mentioned above is a loop and runs at 60times/s.

3.8.6 Utils

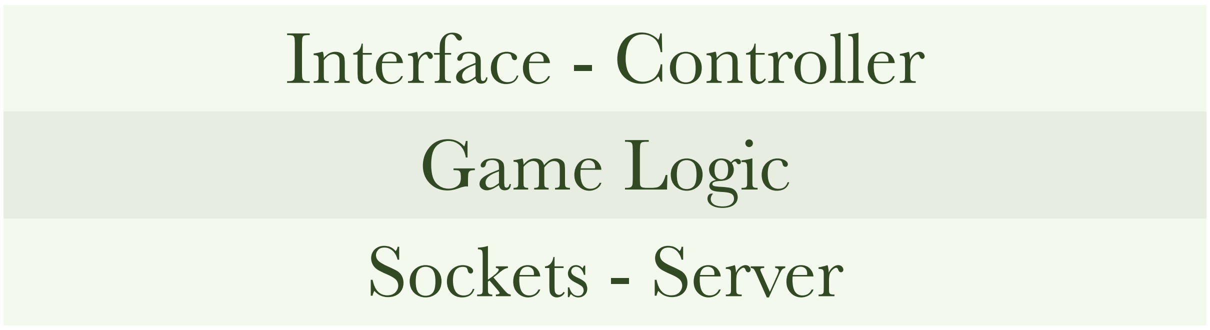


The Utils class, is an utilitarian class, just with the purpose to have some methods on doing some mediocre jobs like random numbers, getting screen width and screen height, containing some arrays to some static datas, etc.

3.8.7 Architecture

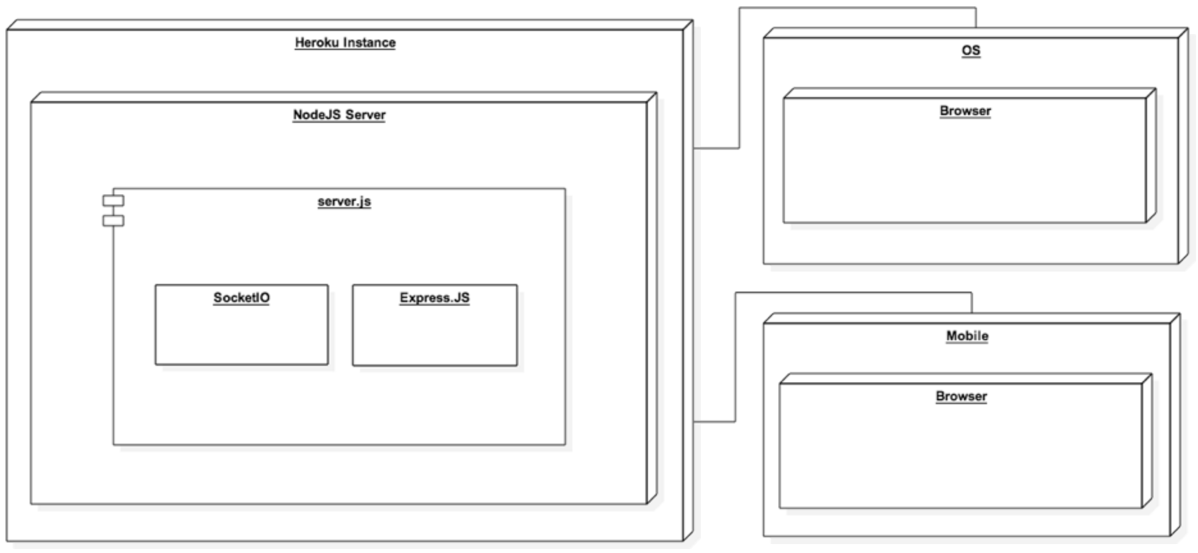
The game has 3 layers, the View, the game Logic and the Server which handles the network.

Below is a graphical representation for the layers.

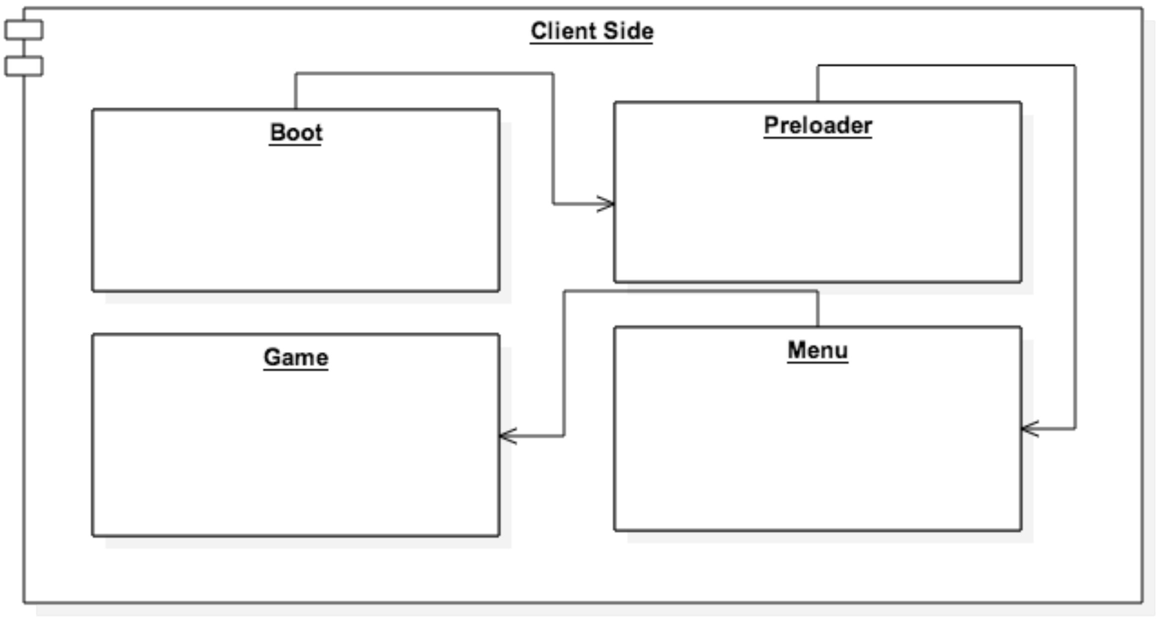


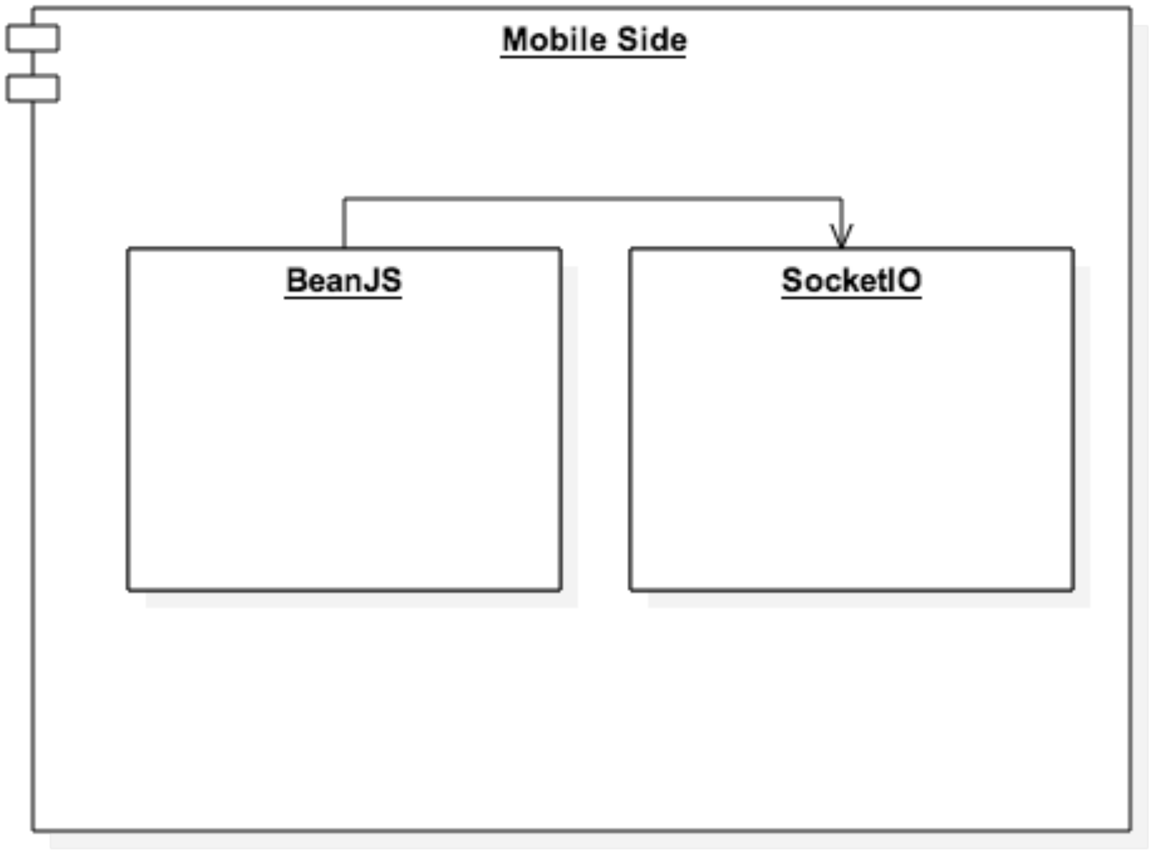
The interface or the controller, is the remote controller which the users access through the phone, and there the controller is interacting with the server which in turn communicate the commands to the game logic.

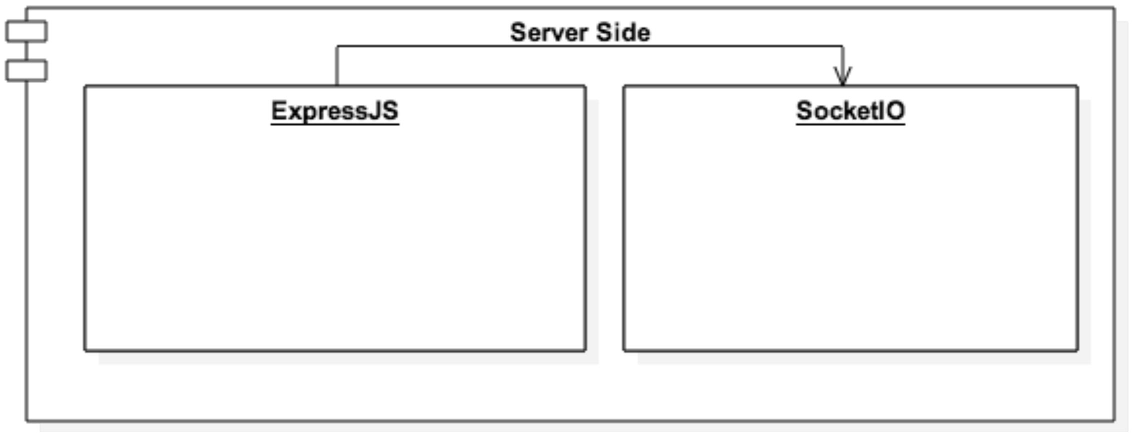
3.8.8 Deployment Diagram



3.8.9 Components







4. Implementation

4.1 Technologies used

I decided to use the technologies that i wanted to learn and understand, that being said my experience on the following technologies was pretty much non-existent. Also the following technologies are really popular among the web/frontend development community, since they tend to decrease the amount of work a developer has to do when he is developing some softwares.

4.1.1 JavaScript

The first thing i decided to go with was JavaScript. JavaScript is a programming language with first class functions, and prototypal inheritance. Its not the standard object oriented programming languages, because it does not have “classes” or overloading, but has a prototypal way of inheriting methods and properties from parents objects.

Also JavaScript might be the most popular programming language in the world. We can find JavaScript everywhere, as long as you have a browser on your device, there is a VM that contains a JavaScript engine.

Also there is a big amount of innovation on the JavaScript world, people are developing libraries, frameworks, and many other tools in daily basis, and some of them are really amazing, and they do a lot of heavy lifting when it comes to building an app or building a game.

Overall JavaScript is great because there are plenty of new things you can use to achieve the result you want.

4.1.2 NodeJS

Like i mentioned in many cases in this page, NodeJS is an instance of the V8 engine, with some specific API to build I/O and network systems. That being said, its not only for I/O or networking. People have built a lot of tools on top of NodeJS, one that i can remember is node-webkit, its a platform that has webkit and node implemented to build windows applications.

An example of such app would be “Game Dev Tycoon”. Game Dev Tycoon, is built on top of node-webkit and web technologies, and it gives a native feeling to it.

I used NodeJS to create a server for my game, so i could communicate from my remote phone controller to the client opened in a PC, i am using sockets, which i will discuss later.

4.1.3 Express

Express is a framework that is built on top of node HTTP library. Its abstracting much of the complications a user has to deal to create a web server.

I needed express just to build a simple routing server, that will redirect an user when he goes to a specific link. And to provide with the adequate page when the user sends that request.

Ex: When user goes to url/mobile, express recognizes and provides the user with the mobile.html page.

4.1.4 Phaser

Phaser on its website refers to its self as “Desktop and Mobile HTML5 game framework”. Framework provides a list of functionalities to game developers that might bring the cost and time down of producing a game.

Phaser comes with several builtin functions, but i haven't used all of them, the functionalities that i have implemented in my game are as follows:

- Preloader - Preloading assets and other resources,
- Physics - The spase physics and collisions
- Sprites - The sprite engine which allows me to create objects.
- Groups - Groups are AI enemies that you need to do something with them in Bulk.
- Sound - Sound system, since sounds are inconsistent throughout the browsers.
- Tilemaps - This are just simple images placed below the game.

4.1.5 SocketIO

SocketIO is a library that abstracts the socket API layer of the browser as well as the Node.js.

I decided to use SocketIO given that this library provides many fallbacks if some specific browsers do not support this kind of API, or if mobile devices do not support this functionalities.

The way Sockets works is with an asynchronous mentality, when a socket is called is pushed to the function call stack and then executed. The function then sends an event, then if a receiving function exists for that event, it will be executed.

4.1.6 Browserify

Browserify is a dependency module manager. It allows me to require() dependencies throughout the system.

The problem is that NodeJS has a require() method, but the browser does not. But how to we include NPM packages inside the browser? Well Browserify solved the issue for us.

In this way we can have manageable dependencies and packages, and our code will be easily managed.

4.1.7 BeanJS

Bean is a small, fast, cross-platform, framework-agnostic event manager designed for desktop, mobile, and touch-based browsers.

I had the need for Bean on my mobile view. Since we are dealing with mobile touchscreen, we need a way for the events to be consistent throughout the thousands of mobile devices.

I have bound the bean events with the buttons i created in the view, so when someone clicks an action, it will send a socket message to the server, and the server will reply back to the client and update the view.

4.1.8 Gulp

Gulp is not a functionality inside the game but more a helpful tool for the developer. This tool can be considered a child of “make”. This tool will compress, move, minify, reorganize and many other functionalities in a command line call.

It is helpful if i want to produce code for the machine, and then push it on the server with the minimum size, given that we want that the game to be as small as possible.

4.1.9 Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

I am a Git fan. Git is a Lifesaver, its a system that allows me to implement stuff without breaking the current code i have, and it does not allow me to mess up my whole project.

Also i need git to push my code into the cloud, as Heroku manages the project based on SCM.

4.1.10 Bower

Bowser is the front-end package manager. With this i am able to download any library i can think of, with just one line.

I use bower in my day to day development, so it would seem unreasonable to not use it on this project as well.

4.1.11 Heroku

Heroku is the cloud platform (PaaS) that i decided to put my files in. The reason was because for 1 single core instance its free, and that it supports NodeJS.

It initially provides you a command line software, that allows me to remotely create instances, push code on the instances i have created, manage cores, and many other cool featured.

4.2 Testing and debugging

There is no software that can be called bug-free, so thats why developers have created debugging tools. This tools allow us to find issues in our software and fix them.

Testing tools vary from environment to environment. In my case, the environment is the browser and the mobile phone. So to figure out bugs and fix them, i have used Google Developer tools, and my actual phone Android XPERIA Z3.

4.2.1 Google Developer Tools

Google Developer tools, is an extension of Google Chrome, and its suppose to help the developer debug its code and to find inconstancies.

Google Developer tools is separated into 8 sections, i will be explaining what this sections are and how they helped me debug my game.

Elements

This section is where the DOM is generated, you can actually see the DOM live in here, you can play with the DOM and see how its reacting when events trigger some aspects of the DOM.

But since my game is based on Canvas, the section Element was to no use of me, but usually when people develop front-end applications, this can solve your problems.

Network

The network Tab, is the place google chrome records the HTTP calls and activities. In this tab you can see what http calls are being made and their replies, at the same time you can see their headers and more. This is helpful when you have a system with an API backend and you would like to see where the problem lies.

The network tab, helped me debug some issues i had with the socket connections, since they are no more than simply network calls, google chrome records them.

Source

This is where all the html/css/javascript files are located. I can say that this was one of the most important features that i used to debug issues.

Since i would have several logical errors or datatype issues my IDE would not detect, these errors were displayed in the runtime and i could debug the issue thanks to the small IDE and sources in the browser.

Timeline

Timeline is a tool that depicts the paints/repaints, events happening in the rendering engine.

With this tool i could detect frame-rate drops, memory leaks, paint issues and many more. With games this tool really comes handy.

Profile

This section, when activated takes a snapshot of your application and displays the events/ function stack calls and how much CPU/Memory they have used. This was helpful as well to detect any issue with my code.

Resources

This section provides an interface for the local storage, database, cookies, caches etc.

This want quite helpful to me, since i am not using any of these features for my game, but in the future this might get helpful, as i would like to make the game playable offline.

Audits

This section does an audit to your app, and it provides you with an overview of the whole app structure and give you some suggestions on how to make the app better, faster.

Console

This might be the most important feature i have used. The reason is that every time i would run the code, i could see most of the errors displayed in the console, and at the same time i could even run standalone code my self.

4.2.2 XPERIA Z3

XPERIA Z3 was a helpful resource to have and to debug the mobile controller, this allowed me to play continuously with the game and detect any issue.

It also allowed me to polish the interface for the controller, since this might have affected the user experience with the game.

I used XPERIA Z3 with chrome browser. Usually the debugging would happen in a local environment, as i would create a static ip out of my computer, enter the wireless local network and then connect to the game.

The speed with the local network was really good, since the lag is minimal in this case, but if we would have a internet connection that would depend on the internet speed.

5. Conclusion

5.1 Difficulties encountered during development

As developers we all face development difficulties, and we have to figure out which one would be the best solution for us to solve. In my journey in developing Meteor Impact i did face quite difficulties in choosing the right tools and right technologies to develop the game.

5.1.1 JavaScript Quirkiness

One of the issues i faced during development is the JavaScript quirkiness. JavaScript is a really great language, but it comes with a really set of bad programming concepts.

this

The “this” problems, is something that affects 99% of JavaScript developers. “this” element is bound to the global scope and differently from other languages that is usually bound to the object.

When used to a function it will bind to the window, and so people might have issue. If for example we would like to access a property that is bound to “this” from an inner function we would have to pass the “this” value to a variable and then use that variable to call the the property/method we need.

But overall i have succeeded solving the issue with the method described above.

Prototypal Inheritance

One of the hardest concepts in JavaScript, is its prototypal system. Because everything in JavaScript is an Object, every object inherits from a single object that is “Object”.

The prototypal inheritance works in a mysterious way. For example, if we create a new object and then pass the Array prototype as a reference, that object will inherit all its methods and properties.

In this way we can even overwrite default methods/properties, giving it a same feature as classical object oriented programming style.

This wasn't such a big issue, in fact this helped me build the UML diagrams, since if i would have developed the system in a functional way, i might have had some problems sketching this diagrams.

5.1.2 Framework Issues

Before i started the game, i had to decide on what kind of framework i should have used, and what would the pros and cons of the specific frameworks be.

In the end i came to the conclusion that Phaser is a well developed gaming framework.

The reason why i chose phaser was because it has a really big community, and had examples of how to achieve specific things. Though the documentation lacked pretty much on explaining several things, and i had to go over some stuff 2-3 times.

5.1.3 Connection Issues

One of the major issues on any online game is the connection issue/lag/ping. I say this because i play a lot of games online, and that if someone has a high ping the game experience is terrible.

Companies, to reduce the lag/ping place servers as close to their user base as possible. But this usually don’t fix the whole thing. They actually code some parts of the game to detect where some people might move, and to prevent the server on overloading, and also to reduce the lag.

In my case the game is located in Heroku, and the server for Meteor Impact is located in USA.

Now lets make some calculation. If we have our servers in the USA and the latency of sending a signal to the server is around 250ms, getting the reply would be the same time 250ms, that means 0.5s to only send a message and getting it back. In terms of apps this would be pretty fast, but in terms of online games this is really big.

The solution for this is to have a distributed system around the world, placing the game on different servers, and serving the datas with the closest server. But this still wont solve the issue, since we cant cover the whole world.

We will have to develop movement prediction, which will guess where the user will go and reduce the latency.

But the solution to this multiplayer issue is beyond our scope of the game. Since our game is mainly focused on local co-op.

5.2 Future Development

There is a series of things that i would want you to look forward in the upcoming updates.

- Database
- User Registration
- User Customization
- Online multiplayer
- Scoring
- Game Modes

This are some future development ideas that i am looking forward to develop in the next iterations.

Below is the list with a more detailed structure of what there is to come.

5.2.1 Database

For now the game focuses entirely on the state of the game, and does not store anything on databases or storage, but this will change in the future, since i would like to implement some features that a database will be needed.

Since i am using JavaScript, i will use a noSQL database, more exactly MongoDB, as they are using the JSON structure type for storing key,value pairs. And NodeJS makes the connection and development with this technologies seamlessly.

5.2.2 User Registration

I would like the players to register on the system, so i can provide them with some features that would be impossible to record on state based games.

The user registration will be developed on the backend and push the datas to the noSQL database.

There would be a menu and registration system, and also a login page, where already logged users will login.

In this way we can keep track of the players and add new features to the game.

5.2.3 User Customization

What every MMO game has is the ability to customize elements of a player. And this is what i would like to bring to the game. The ability of the users to change their ship color, shape, guns, specs etc.

This can only be achieved throughout the registration system, which will be connected with the noSQL database.

5.2.4 Online Multiplayer

For now the players will be able to play co-op, but i am looking forward to make the game playable for everyone online, so everyone can register and play.

The problem which will arise, it will be the synchronization of the AI and objects around the world map. It might be an issue because there are way to many meteors, and all of them will need to send a message to the server and provide them the exact location, so the server will update the view on both clients who are playing in a multiplayer environment.

But i believe there will be some code that will predict the meteor movements, and so it will take around 1-2 messages to make them move, and then if there will be any collision with other meteors or even other players, we will send the message again, and the players view will update seamlessly.

5.2.5 Game Modes

For now, the only game mode that is on this game is the Free-for-all, which everyone has the power to play against each-other and destroy each-other ships.

Some now modes that i have thought would be:

<u>Team death-match</u>

2 groups of players, blue and red, fighting against each-other, the team who got more points in x amount of time wins that game.

<u>Capture the Flag</u>

This mode would be again with 2 groups of players, but there will be a specific items which everyone will need to hold for as much as possible.

If x player is killed, he dies but the flag drops, and anyone will be able to catch it again.

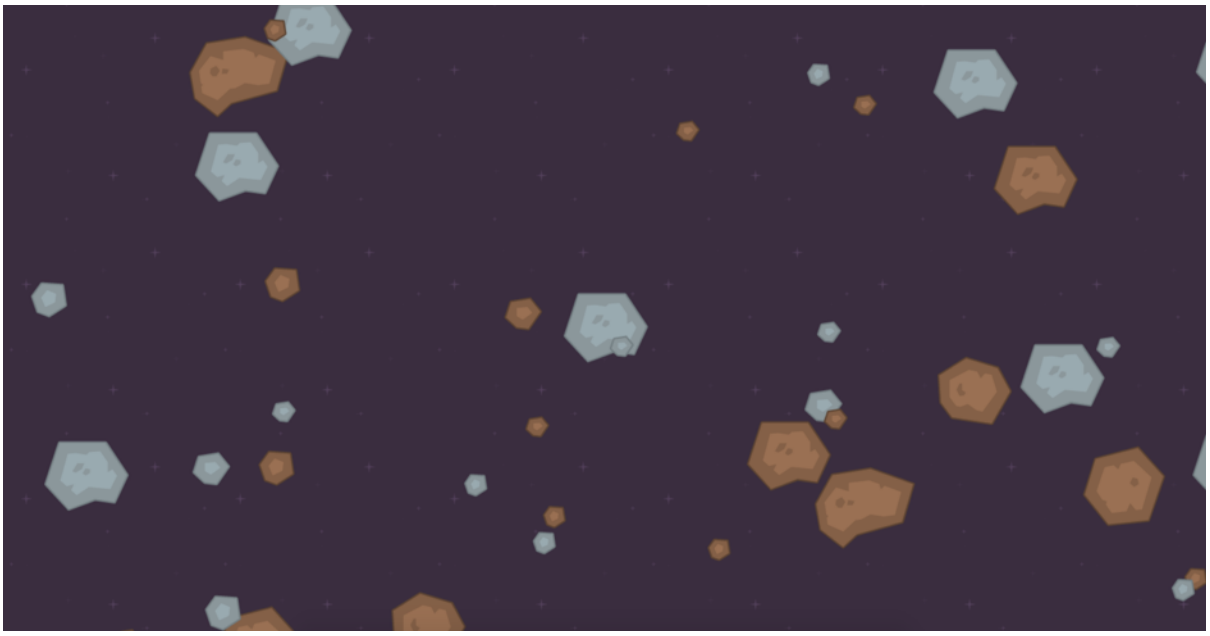
<u>Boss Fighting</u>

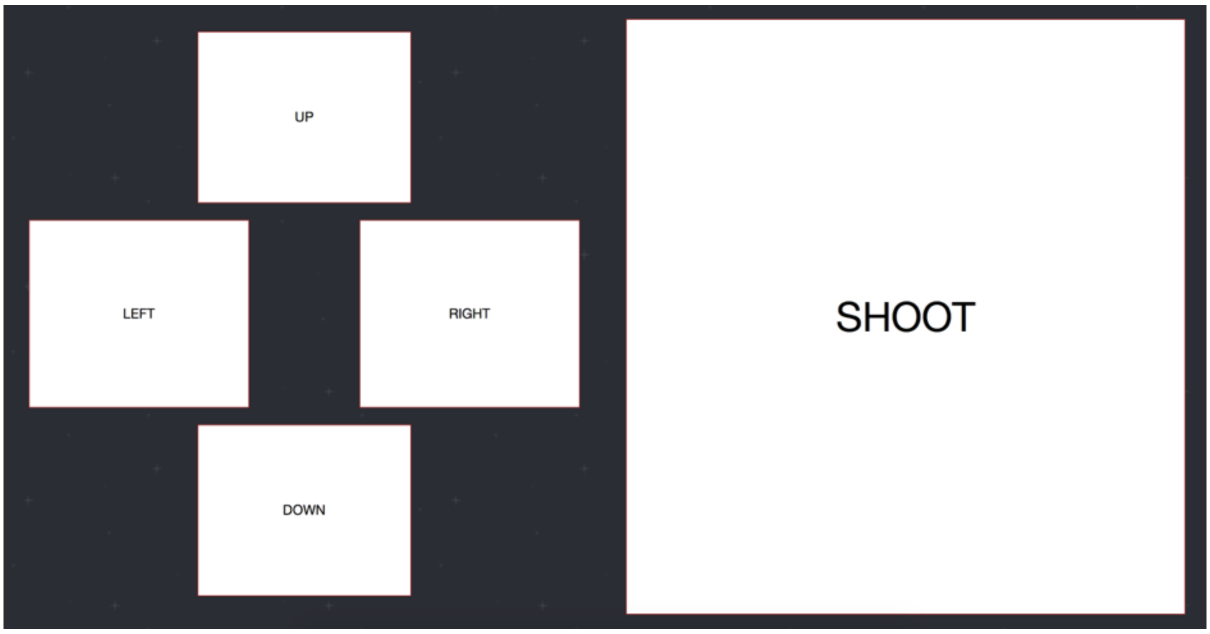
Boss fighting is a mode where out of 5 players, 1 is scaled 2-3 times and has 3-4 more hp points, but he is really slow. The other 4 players need to destroy him if they want to win the game. If the boss player destroys them in the first hand, then he wins the game.

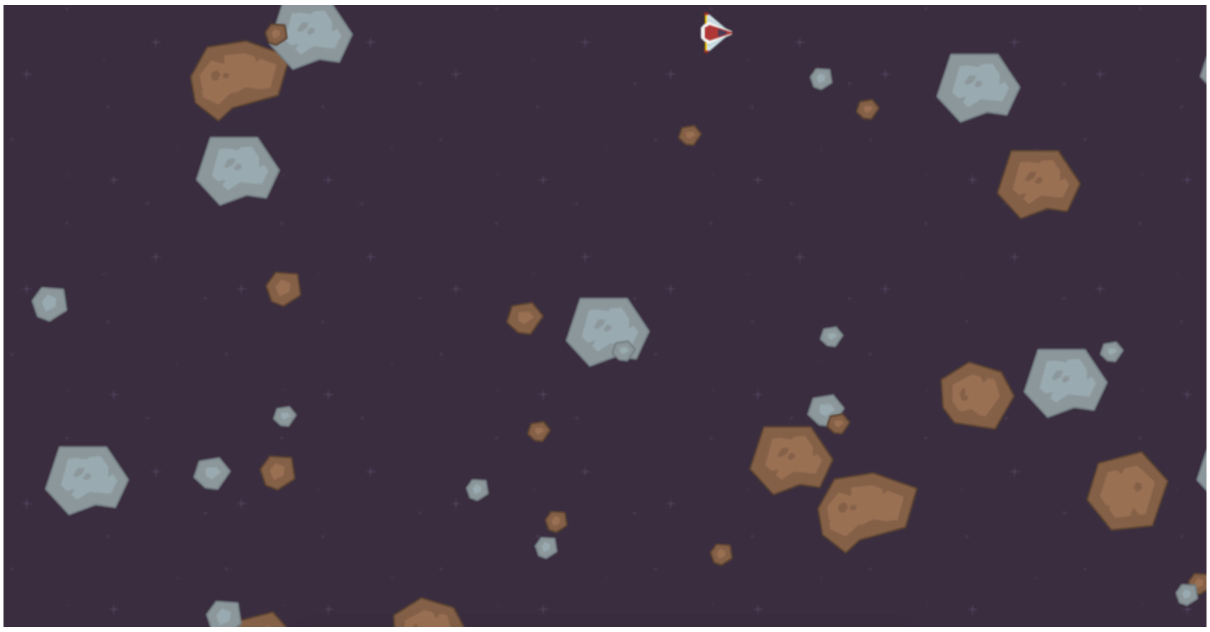
5.2.6 Scoring

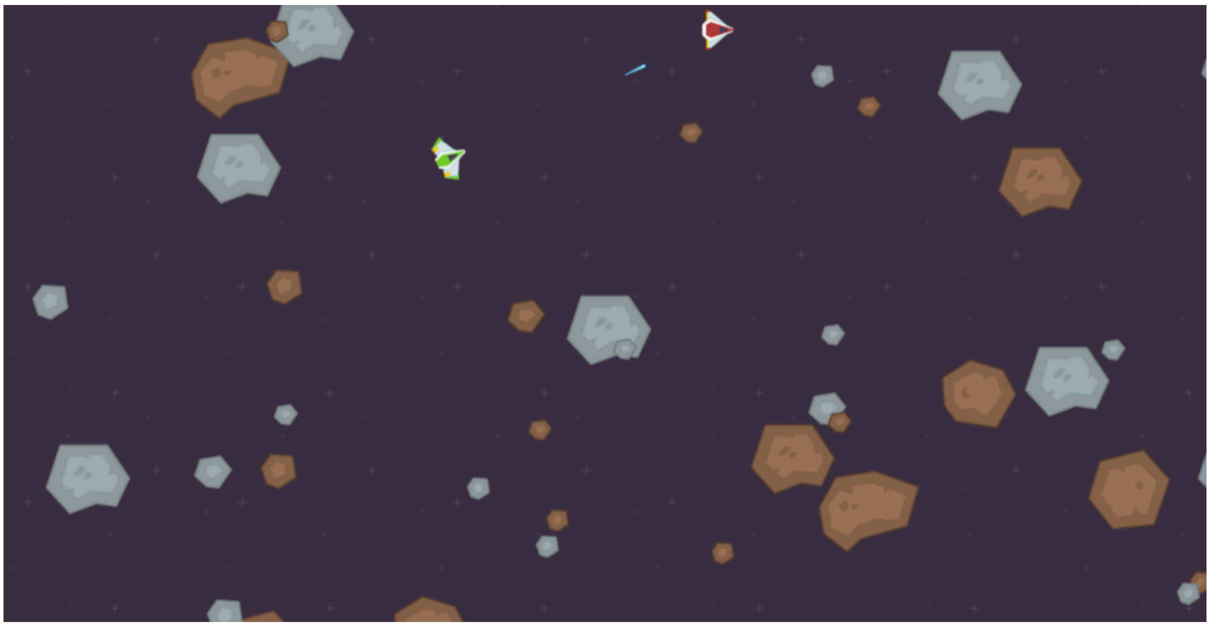
Well, everyone wants to boost about their score online, and show off, so i would like to build a score keeping mechanism. Also a global TOP Scores. The score are calculated by kills and the different modes they are in the game.

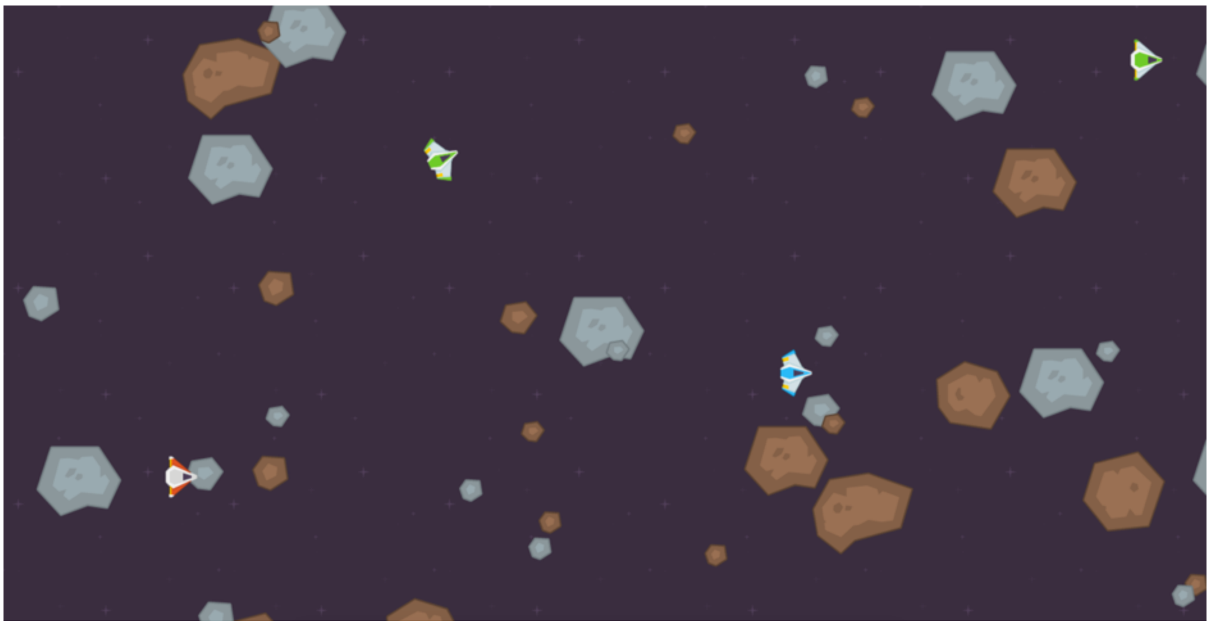
5.3 Screenshots




UP
LEFT
RIGHT
DOWN
SHOOT







6. References

http://nodejs.org

http://expressjs.com

http://phaser.io

http://socket.io

http://browserify.org

https://github.com/fat/bean

http://gulpjs.com

http://bower.io

http://en.wikipedia.org/wiki/Git_(software)

https://www.heroku.com

https://developer.chrome.com/devtools

http://meteorimpact.herokuapp.com

http://www.sonymobile.com/de/products/phones/xperia-z3/

http://staruml.io

https://www.jetbrains.com/webstorm/