Run Sheldon Website

BACHELOR’S THESIS

Author
PAU MONSERRAT NÚÑEZ

Director
ARNAU SANS ALONSO

Tutor
SILVIA LLORENTE VIEJO

AC department

April 17th, 2015

Information Technologies Specialization
Bachelor Degree in Informatics Engineering

FACULTAT D’INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
Abstract (English)

There is almost a billion of websites on the internet, each one of them have a purpose, and most of the time, it’s the only way companies can show who they are to the rest of the world. This is the case of Square Bee videogame studio, they want to showcase their star game to the rest of the world, Run Sheldon.

In this project we mix different internet related technologies to create a game that feels like a website or vice versa. We’ll be using WordPress as the website content management system and Phaser.js as the graphics framework that will help us create the game.

In the WordPress related part, we’ll learn how WordPress can be customized to add custom item types, create and store data on WordPress backend, and retrieve the data to build the website.

In the game related part of the project, we’ll learn how to create a 2D game and adapt it for web limitations.
Abstract (Catalan)

Existeixen quasi mil milions de llocs web a internet, cadascun d’ells amb una funcionalitat específica, i la majoria de cops, és la única manera que tenen les companyies de mostrar-se al reste del món. Aquest és el cas de Bee Square, un estudi de videojocs, que volen mostrar al mon el seu joc estrella, Run Sheldon.

En aquest projecte barregem diverses tecnologies relacionades amb internet per crear un joc que s’assembla a un lloc web, o viceversa. Utilitzarem WordPress com a gestor de continguts i Phaser.js com a eina gràfica que ens ajudarà a crear el joc.

A l’apartat de Wordpress, explicarem com WordPress pot ser customitzat per afegir tipus de ítem personalitzables, guardar les dades mitjançant el panel de administrador de WordPress i agafar les dades per construir el lloc web.

A l’apartat del projecte relacionat amb el joc, aprendrem com es pot crear un joc en 2D i adaptar-lo a les limitacions web.
Abstract (Spanish)

Existen casi mil millones de sitios web en internet, cada uno de ellos con una funcionalidad específica, y la mayoría de las veces, es la única manera que tienen las empresas de mostrarse al resto del mundo. Este es el caso de Square Bee, un estudio de videojuegos que quieren mostrar al mundo su juego estrella, Run Sheldon.

En este Proyecto mezclamos diferentes tecnologías relacionadas con internet para crear un juego que se parece a una web, o viceversa. Utilizaremos WordPress como el gestor de contenidos I Phaser.js como a herramienta gráfica que nos ayudara a crear el juego.

En el apartado de Wordpress, explicaremos como WordPress puede ser customizado para añadir tipos de ítem personalizales, guardar sus datos mediante el panel, y utilizar los datos para construir el sitio web.

En el apartado del Proyecto relacionado con el juego, aprenderemos como se puede crear un juego en 2D y adaptarlo a las limitaciones web.
Contents
1 Prologue......................................................................................................................... 10
2 Introduction.................................................................................................................. 11
  2.1 Formulation of the problem .................................................................................... 11
  2.2 Objectives................................................................................................................ 11
    2.2.1 Design.............................................................................................................. 11
    2.2.2 Guidelines....................................................................................................... 18
  2.3 Project Scope.......................................................................................................... 18
  2.4 Context and stakeholders....................................................................................... 19
    2.4.1 Client............................................................................................................... 19
    2.4.2 Client’s lead website designer........................................................................ 19
    2.4.3 Client’s marketing chief.................................................................................. 19
    2.4.4 Director .......................................................................................................... 19
    2.4.5 Company.......................................................................................................... 20
    2.4.6 Salesperson...................................................................................................... 20
    2.4.7 Project Manager.............................................................................................. 20
    2.4.8 Developer........................................................................................................ 20
  2.5 State of the art....................................................................................................... 20
    2.5.1 Similar Applications........................................................................................ 20
    2.5.2 Project Context: Storytelling websites .......................................................... 21
    2.5.3 Project Context: WebGL games...................................................................... 22
    2.5.4 Project Context: WordPress.......................................................................... 22
  2.6 Risks......................................................................................................................... 22
    2.6.1 Time management........................................................................................... 23
    2.6.2 Graphical power............................................................................................. 23
    2.6.3 Load time........................................................................................................ 23
3 Project management.................................................................................................. 24
  3.1 Methodology.......................................................................................................... 24
  3.2 Plan monitoring..................................................................................................... 25
3.3 Validation method........................................................................................................25
3.4 Project planning........................................................................................................26
  3.4.1 Phase 1 objectives: Choosing a CMS.................................................................26
  3.4.2 Phase 2 objectives: Choosing a graphics framework.........................................26
  3.4.3 Phase 3 objectives: Testing last 2 tasks.............................................................26
  3.4.4 Phase 4 objectives: Creating the world.............................................................27
  3.4.5 Phase 5 objectives: Adding backend features....................................................27
  3.4.6 Phase 6 objectives: Creation of popups.............................................................27
  3.4.7 Phase 7 objectives: Responsive design.............................................................28
  3.4.8 Total time...........................................................................................................28
  3.4.9 Gantt diagram...................................................................................................28
  3.4.10 Plan modifications............................................................................................30
3.5 Software and hardware .........................................................................................30
3.6 Dependencies ........................................................................................................31
  3.6.1 jQuery................................................................................................................31
  3.6.2 Phaser.js............................................................................................................31
  3.6.3 WordPress..........................................................................................................31
  3.6.4 WordPress plugin: Advanced Custom Fields....................................................32
  3.6.5 WordPress plugin: Contact Form 7.................................................................32
3.7 Laws and Regulations ..........................................................................................32
  3.7.1 LOPD..................................................................................................................32
  3.7.2 LSSI....................................................................................................................32
  3.7.3 Licenses..............................................................................................................33
3.8 Budget....................................................................................................................34
  3.8.1 Human resources .............................................................................................34
  3.8.2 Hardware..........................................................................................................35
  3.8.3 Software.............................................................................................................35
  3.8.4 General costs.....................................................................................................36
  3.8.5 Risks..................................................................................................................36
3.8.6 Total cost of the project ......................................................... 37
3.9 Sustainability and Social Responsibility ........................................ 37
  3.9.1 Economic ........................................................................... 37
  3.9.2 Social ............................................................................... 37
  3.9.3 Environmental ................................................................. 38

4 Development ............................................................................. 39
  4.1 Phase 1: Choosing and use of a CMS .............................................. 39
    4.1.1 WordPress structure ....................................................... 40
    4.1.2 Wordpress database structure ......................................... 42
    4.1.3 Theme setup .................................................................. 44
  4.2 Phase 2: Choosing graphics framework .......................................... 47
    4.2.1 Phaser setup .................................................................. 48
    4.2.2 Game directory structure ............................................... 51
  4.3 Phase 3: Phaser Test Environment ................................................... 53
    4.3.1 Problems, decisions and solutions ..................................... 53
  4.4 Phase 4: Development of game features ......................................... 61
    4.4.1 Game’s flowchart ............................................................. 61
    4.4.2 Scene file format and item creation .................................... 63
    4.4.3 Animations .................................................................... 72
    4.4.4 Physics .......................................................................... 74
    4.4.5 Camera .......................................................................... 77
    4.4.6 Inputs ............................................................................. 79
    4.4.7 Sheldon loop and states ................................................. 81
    4.4.8 Scrolling textures ............................................................ 85
    4.4.9 Parallax backgrounds ...................................................... 87
    4.4.10 Game adapt to screen sizes ............................................. 91
    4.4.11 Level of detail texture system ......................................... 92
    4.4.12 Asynchronous load and creation of stages ....................... 96
  4.5 Development of WordPress features .......................................... 99
4.5.1 WordPress backend panel ................................................................. 99
4.5.2 Registering new post types .............................................................. 101
4.5.3 Adding post meta fields ................................................................. 103
4.5.4 Retrieving post data and showing it into the frontend .................... 106
4.6 Phase 6: Responsive design ................................................................. 109
5 Benchmarks ......................................................................................... 110
  5.1 Framerate benchmarks ................................................................. 110
  5.2 Loading time benchmark ................................................................. 110
6 Conclusion .......................................................................................... 112
  6.1 Future improvements .................................................................... 112
7 Figures ................................................................................................. 113
8 Tables .................................................................................................. 116
1 Prologue

My name is Pau Monserrat Núñez, and since October of 2013 I’ve been working at Artesans at Work, a small startup company specialized in WordPress and web technologies. During my time in this company, I’ve been learning a lot on web technologies.

Since I was a teenager, I’ve always been interested in videogame programming and level design, I’ve worked with different communities helping them to create videogame projects, mainly using Source Engine and Unreal Engine.

When I started to work at this company, I was always interested in the visual aspects of the websites, and making them look unique, that’s why when they offered me to make this project I was so excited because it meant to make a website that looked like a videogame.

The participants for this project are my project director Silvia Llorente Viejo, my boss Arnau Sans Alonso, Bee Square website marketing chief Olga Rueda Molina and chief designer Juan Antonio Crespo.
2 Introduction

Barcelona is known for being a city were lots of computer and mobile technologies companies are located. That’s the case of Bee Square, a videogame studio, known for its main videogame release, Run Sheldon.

The videogame consists in an infinite-run side-scroll game, where Sheldon, the main character, need to collect coins and defeat its enemies while collecting power-ups and completing missions.

The game is available for Android and iOS, and has over 100.000 downloads at Play Store and an unknown number at Apple Store. It’s free for both operating systems.

The app income comes from its freemium business model, where you can buy boosters, character clothes, etc. via micropayments.

2.1 Formulation of the problem

Square Bee’s main game is Run Sheldon, so most of the company’s income comes from this game. There is a big update coming up on December 2014, the game is going to be redesigned and will be enhanced with character powers, new enemies, new booster and power ups. The company wants to release the game’s website coinciding with the release of this update as a way of promoting it and increase the game’s player base.

Creating a creative website for the game is also a way of differentiate from competitors and a way for the company to show it on congresses and meetings.

2.2 Objectives

2.2.1 Design

The design of the website consists on different stages of the game, each of them with information about different topics of app’s gameplay.
2.2.1.1 Loading screen

The loading screen will be displayed while the website is loading its assets.

Figure 1 Starting loading screen

2.2.1.2 First stage

The first stage of the website will be displayed once the website stops loading. The main features of this stage are Sheldon that will be controlled by the user, a popup with Sheldon information displayed when the user clicks on him, and a popup displaying the trailer displayed when the user clicks on the big wooden panel.

Figure 2 Game’s first stage
2.2.1.3 Second stage

The second stage of the website show Run Sheldon enemies. If the user clicks on any enemy, a slider is displayed with information about them.

![Figure 3 Game’s second stage](image)

2.2.1.4 Third stage

The third stage main features are Sheldon’s closet, once clicked, a popup will be displayed showing Run Sheldon purchasable costumes.
2.2.1.5 Fourth stage

Fourth stage of the website show the power-up chest. Once clicked, a slider with the different Run Sheldon power-ups will be displayed.
2.2.1.6 Fifth stage

Fifth stage feature is the booster chest that, once clicked, show Run Sheldon boosters in a slide.
2.2.1.7 Sixth stage

The sixth stage main feature is the carrot, once clicked it’ll display a tips and tricks slider. Some optional features are a game ranking and a redeem code received when clicking the treasure chest.
2.2.1.8 Seventh stage

The last stage show a greeting message and a clickable button that displays a contact form to send an email to Bee Square.
2.2.2 Guidelines

Bee Square gave some guidelines on how to approach the project:

- Browsing the site will look like playing a 2D platform game controlled by the mouse wheel, just like browsing a regular site.
- Character will display animations as he runs through the world.
- A texture level of detail system will allow for all kind of devices to load and run the site correctly.
- The site will be built on top of a content management system (CMS) so that the client can edit its content without the need of a web developer.
- The website will be loaded from a json file so that adding new elements can be as easy as editing the scene file.
- The site will be featuring dynamic backgrounds, particles, sounds, etc… to make it feel like a videogame.
- Each stage of the website can be accessed via a navigation menu

Some additional suggestions are made to the client so that they will be included in case of completing the project before the estimated time.

- Website achievements that will look like most of nowadays game archivements. For example, having a quantity of coins collected, completing the website, visiting the website a number of times, etc…
- Showing the Run Sheldon App ranking on the website
- Make the character able to jump
- Adding new animations for coin picking, walking, etc…

2.3 Project Scope

As said previously, the aim of the project is to create a website with similar graphics as the original Run Sheldon game that can be managed via a backend panel.

The first priority is making the client happy with the end result, so all their initial demands should be fulfilled.

The project can be divided into two distinctive parts, creating a game system and integrate a content management system.
The game system will run as a game and it’ll run well on desktop computers, laptops, tablets and smartphones as long as they have enough graphical power.

The management system will have a backend panel where the client can edit the website data that will get automatically updated on the frontend.

2.4 Context and stakeholders

In this section we are justifying the motivations of each stakeholder and, at the same time, justifying the need for this project.

2.4.1 Client

The client of the project is Bee Square. Without the client this project wouldn’t exist. The client motivation is to make their star game, Run Sheldon, popular and thus allow the company to grow.

2.4.2 Client’s lead website designer

The client lead website designer, Juan Antonio Crespo, is in charge of handling all needed material to the developer. He’s also in charge of making decisions along with the project manager so that the website is loyal to its design.

2.4.3 Client’s marketing chief

The client marketing chief, Olga Rueda Molina, is in charge of creating website content and making marketing decisions.

2.4.4 Director

The director of the project, Silvia Llorente Viejo, role is to supervise the author of the project and make sure the project archives its academic goals.
2.4.5 Company

The company where the project is being developed is Artesans at Work. Its role is to deliver the website at time while satisfying the client.

2.4.6 Salesperson

The salesperson, Arnau Sans Alonso, role was to convince the client that the Artesans at Work would make their website come alive satisfying all their needs.

2.4.7 Project Manager

The project manager, Pau Monserrat Núñez, motivation is to communicate with the client, plan and take decisions over the project.

2.4.8 Developer

The developer, Pau Monserrat Núñez, role is take care of the development of the site making sure it follows the design and client needs.

2.4.8.1 Visitor

The visitors are the people that will visit the site once it’s finished. They may visit it for information about the game app, in case they are thinking on installing it, or they already play the game and just want to visit the new website.

2.5 State of the art

2.5.1 Similar Applications

Some websites use a similar king of browsing that the one used on this project using the scroll action to browse the website. Most of them belong to a category of websites called “storytelling” in which scrolling reveals new parts of a story. This style is not used on big corporate websites, rather they are used to explain a theme in particular or advertising a product.
An example is the National Geographic website on their documentary Killing Kenney\(^1\) that narrate John F. Kenny life through different animation effects as you scroll down the site.

Another good example is Boy-Cot\(^2\) website, whit a good use of parallax effects that gives a 3D effect to the site.

The following website uses two characters that follows the user while visiting the site just like Run Sheldon’s website. These are things\(^3\) is the website of two designers and it gives the impression that the characters shown in the site shows us their studio.

The Hobbit\(^4\) newest website is a great example of what an interactive site. Site navigation is done through Middle Earth map, you can follow each film’s character through their journey and visit the places they do on the film. Each location has information about the place and some of them have a 3D mini-game.

The last example is a graphic novel by Peugeot that makes a good use of cartoons and sound to advertise their car Hybrid\(^5\).

2.5.2 Project Context: Storytelling websites

Storytelling websites are not common, and as said in the later chapter, they are commonly used to advertise a product.

There are some libraries that aid the creation of storytelling websites.

Parallax.js\(^6\) is a JavaScript library that helps creating parallax elements and giving depth to the website.

Another library is called Skrollr.js, it helps creating scroll animations so that elements appear on screen as the user scrolls down.

Since we are going to use a graphics framework, we are not going to use these kind of libraries, but they were an alternative before the final decision on using WebGL.

---

\(^3\) These are things. “These are things” [http://www.thesearething.com/](http://www.thesearething.com/)
2.5.3 Project Context: WebGL games

WebGL is being used in some developers and artists portfolios as well as for building games. A great example of the use of WebGL on a regular site is Steven Wittens’ portfolio\(^7\), with some complex 3D animations and shader use. Another example is Aleksandar Rodič’s portfolio\(^8\), with an innovative idea on using the 3D elements as a way of navigation through the website.

WebGL games are simple compared OpenGL and DirectX games, the main reason is they need to be coded in JavaScript, which is multiple times slower than C++, used in most big budget games. Most WebGL games are released as smartphone apps using the Phonegap/CocoonJS frameworks that allow web applications to be converted to apps.

2.5.4 Project Context: WordPress

WordPress is a content management system (CMS) developed in PHP for environments using MySQL and Apache under GPL license and is open source.

WordPress started as a blogging CMS in 2005 and has become the most used CMS over the internet with over 70,000,000 sites using it. The current version, WordPress 4.1, has been downloaded over 26,000,000 times. It has a market share over 60% of websites using a CMS, and over 20% of total internet sites.

The key aspects of its success are its license, ease of use, its features as a CMS and its big community of developers that has helped the growth by publishing free plugins and themes.

Other CMS that compete with WordPress are Drupal\(^9\) and Joomla\(^10\). Although each CMS has their own key features, they can fulfill the same task.

2.6 Risks

As any other project, there are some identifiable risks that should be taken into account before starting the planning.

---

\(^7\) Steven Wittens “Steven Wittens” [http://acko.net/](http://acko.net/)


\(^9\) Drupal. [https://www.drupal.org/](https://www.drupal.org/)

2.6.1 Time management

Due to the uniqueness and complexity of the project, it may not be ready in time. There are some factors that can cause a delay in the release, choosing an inappropriate graphics framework, low fps on most browsers thus making the website unusable, etc. To prevent it, the state of the project will be periodically evaluated.

2.6.2 Graphical power

The project will use large images, some of them displaying at the same time on the screen. To prevent the website to render at low framerate, there will be a level of detail system for textures so smaller versions of them are rendered on smaller screens. Even though there will be a special effort on this point, high framerates cannot be guaranteed on systems without a dedicated graphics card.

2.6.3 Load time

As said before, the website uses large images, the initial website size estimation was around 25 MB, and so big load times are expected. A planned solution is to load the next stages of the website asynchronously while the user browses the current one. This way, only the first stage is needed to let the user start browsing the site.
3 Project management

3.1 Methodology

The chosen methodology for the project is Rapid application development (RAD). RAD approaches to software development puts less emphasis on planning tasks and more emphasis on development.

RAD approaches emphasize the necessity of adjusting requirements in reaction to knowledge gained as the project progresses, this is very important for the development of this project because most of the project is done using an unknown technology (WebGL and Phaser.js).

RAD approaches also emphasize a flexible process that can adapt as the project evolves rather than rigorously defining specifications and plans correctly from the start.

![Rapid Application Development Methodology](image)

**Figure 13** RAD cycle

RAD development cycle consists on the following phases:

- **Analysis and design**: An overview of the project is made and some design guidelines are identified before starting the development
- **Prototype Cycle**: The prototype cycle consists of three phases, once they are satisfied, develop another task:
  1. **Develop**: Develop a task on top of the prototype.
2. **Demonstrate**: Demonstrate it’s working correctly.

3. **Refine**: Refine and optimize the task.

- **Test**: Make the last tests and make sure it’s ready for deployment.
- **Deployment**: Release a new version of the project.

The advantages of RAD are:

- **Risk control**: by evolving a prototype, we can detect early the risk factors and look for alternatives.
- **Projects completed in time**: by focusing on the development of incremental units, catastrophic failures are avoided.

The disadvantages of RAD are:

- **Less control**: one of the advantages of RAD is that it’s a flexible adaptable process. There is an inevitable trade-off between flexibility and control, more of one means less of the other.
- **Poor design**: the focus on prototypes can be taken too far in some cases resulting in a hack and test methodology.
- **Very large systems**: RAD focuses on small and medium projects, the other points made above makes RAD not suitable for large systems where control and design are mandatory.

### 3.2 Plan monitoring

An unexpected delay of one of the phases of the project could result in a delay of the whole project, thus not meeting client’s requests. To prevent this from happening regular meetings are done with the client to make sure the project is on schedule.

In case of a major problem, it will need to be communicated to the client and make a decision on how to solve it: remove functionalities, delay the project, update the design, etc…

Furthermore, the final estimated time for each task was done upwards to prevent from unexpected delays.

### 3.3 Validation method

The project is using Github as a method of code management, so we can track back project versions and go back to another version if the path followed was not giving the expected results.
The client will have access to the project since the beginning of the project, and they'll get no this way we will get regular feedback so we know we are fulfilling their expectations.

3.4 Project planning

The project has been sub-divided in the following tasks, and an estimation of the execution time has been made for every one of the tasks.

3.4.1 Phase 1 objectives: Choosing a CMS

The objectives for the first phase of the project are:

- Choosing the adequate CMS for the development of the project.
- Learning how to use it and integrate it with the project.

Estimated time: 3 h.

3.4.2 Phase 2 objectives: Choosing a graphics framework

The objectives for the second phase of the project are:

- Choosing a graphics framework.
- Learn the use of the chosen framework and integrate it with the project.

Estimated time: 17 h.

3.4.3 Phase 3 objectives: Testing last 2 tasks

The objectives of the third phase of the project are:

- Test in a small scale both phase 1 and phase 2 and look for possible problems that may occur during the development.
- Find a solution for the problems found on the last point.
- If no solution is found, look for alternatives for solving the problem and make the corresponding changes to the project plan.

Estimated time: 65h.
3.4.4 Phase 4 objectives: Creating the world

The objectives of the fourth phase of the project are actually creating all the game related part of the project. It’s the biggest and the most complex.

• Establish a file format so that the scene can be parsed and loaded from this file.
• Parse the file and loads the elements from the file to the website.
• Create the character movement and animations.
• Create texture scrolling on the elements
• Create depth of the backgrounds and parallax optical effect
• Integrate physics for player movement and decoration.
• Deal with all different kinds of screen sizes
• Integrate different levels of detail for textures so smaller screens (tablets and smartphones) use less bandwidth (3G friendly).
• Background loading of game assets so that not necessarily all assets must be loaded before user interaction.
• Add all scene elements and decoration.

Estimated time: 200h

3.4.5 Phase 5 objectives: Adding backend features

The objectives for the fifth phase of the project are creating a friendly panel so that the client can easily edit the information of the site.

• Create a panel for Enemies
• Create a panel for Costumes
• Create a panel for Boosters
• Create a panel for Powerups
• Create a panel for Tips and tricks

Estimated time: 4h.

3.4.6 Phase 6 objectives: Creation of popups

The objectives for the sixth phase of the project are retrieving the data from the last phase and showing it on the website via sliders and popups.
- Create a slider for Enemies
- Create a popup for Costumes
- Create a slider for Boosters
- Create a slider for Powerups
- Create a slider for Tips and tricks
- Create a contact form

Estimated time: 40h.

### 3.4.7 Phase 7 objectives: Responsive design

The objectives for the seventh phase of the project are adapting the site to tablets and smartphones. This phase will only deal with the website HTML/DOM elements, not the game aspect of the site.

Estimated time: 40h.

### 3.4.8 Total time

The total estimated time for the project is:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project planning and client meetings</td>
<td>15</td>
</tr>
<tr>
<td>Choosing a CMS</td>
<td>3</td>
</tr>
<tr>
<td>Choosing a graphics framework</td>
<td>17</td>
</tr>
<tr>
<td>Testing last 2 tasks</td>
<td>65</td>
</tr>
<tr>
<td>Creating the world</td>
<td>200</td>
</tr>
<tr>
<td>Adding backend features</td>
<td>4</td>
</tr>
<tr>
<td>Creating slides and popups</td>
<td>40</td>
</tr>
<tr>
<td>Responsive design</td>
<td>40</td>
</tr>
<tr>
<td>Move to production</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>387</strong></td>
</tr>
</tbody>
</table>

*Table 1 Tasks duration*

### 3.4.9 Gantt diagram

The project dedication time will be 40 hours a week, according to that, the following Gantt diagram shows the previous planning on the real timescale.
Figure 14 Gantt diagram
3.4.10 Plan modifications

Although for the most part of the project we stayed on schedule, some major changes made by Bee Square delayed some parts of the project.

Once all stages of the website were created, Bee Square decided to make them bigger so the world looked more open, opposing to the original design that the world looked narrower. They also added lots of new assets so that the world looked more alive, some of them needing some changes on the asset creation algorithm. These changes added 35 hours to the creation of the world, on the other hand, around the same number of hours got taken off setting up Phaser and testing the environment.

In the end, 23 hours less were needed for the development of the project, the estimation for the responsive design was too high, and 20 hours were saved on this task.

The initial planning predicted the end of the project to be by the end of November 2014, and as initially planned, the project got finished at time. The project moved to production the 19th of December due to the client delaying the date so both app game and website could be released at the same time.

3.5 Software and hardware

The development of project will be done using the following software and hardware:

- **Sublime Text 2**: a sophisticated text editor that will be used for coding the entire site.
- **Photoshop**: the most complete image editor, will be used for process the design so it can be used on the website.
- **Filezilla**: a FTP client that will be used for uploading the project files to the web server.
- **Huawei Ascend 7**: the Android version of the project will be tested on this smartphone.
- **iPhone 5**: the iOS version of the project will be tested on this smartphone.
- **iPad 4**: the tablet version of the website will be tested on this device.
- **Apache server**: the development server where the project will be running.
- **Browsers**: The project will be tested in Chrome, Firefox, Internet Explorer and Opera. The development will be done in Chrome.
3.6 Dependencies

Some external dependencies are needed for the development of the project.

3.6.1 jQuery

jQuery is a free cross-platform JavaScript library designed to simplify client-side scripting. It’s the most popular JavaScript library with over 70% usage on the 10,000 most visited websites.

jQuery is designed to make it easier to navigate a DOM document, handle events, create animations and develop Ajax applications. It also encourages the separation of JavaScript and HTML, and makes the code more clear and understandable.

```javascript
// Javascript query of the element with id "element"
document.querySelector("#element")

// Same query in jQuery
$ (#"element")
```

*Figure 15 JavaScript and jQuery code comparison*

The previously explained points are the reasons we’ll be using the library.

3.6.2 Phaser.js

Phaser.js is 2D JavaScript graphics framework that uses WebGL or HTML5 Canvas renderer. A detailed explanation of the framework is made in a later section.

3.6.3 WordPress

The website will be built on top of WordPress as its content management system. A detailed explanation of the usage is made in a later section.
3.6.4 **WordPress plugin: Advanced Custom Fields**

This free plugin allows the web developer to add different kinds of fields to WordPress items, thus making them more flexible and adaptable for each project. The fields can be visually created on WordPress administration panel and edited for each item. This plugin speeds up the development of most WordPress projects by various orders of magnitude and makes the process of customizing a WordPress theme much simpler. Those are the reasons the project uses this dependency.

3.6.5 **WordPress plugin: Contact Form 7**

This free plugin allows to create customized contact forms that can be easily added to the site. The form can be created on WordPress administration panel without the need of coding knowledge, this way the client can edit the form at any given time.

3.7 **Laws and Regulations**

3.7.1 **LOPD**

The LOPD is a Spanish law that regulates the use of personal information over different media.

Due to the fact that the project doesn’t save any personal data, there’s no need to consider this law.

The only place where the user name and email is asked is on the contact form, but none of the data is saved, it just sends the email and deletes the information.

3.7.2 **LSSI**

The LSSI is a Spanish law that regulates internet use of information. Among other subjects, it regulates the use of cookies.

A cookie is data saved by the browser with the objective of remembering user parameters and information so that the website can offer more customization to the user.

Cookies can be classified in the following categories:
• **Technical cookies:** these are cookies that are necessary to allow the user navigating the site. Order information, login user and password, among other cookies would belong to this category.

• **Customization cookies:** these cookies allow to display the contents of the website depending on the information found on browser cookies. For example, displaying the website on the same language as the browser.

• **Analytical cookies:** these cookies gather statistical information on the use of the website. A good example would be Google Analytics cookies.

• **Advertisement cookies:** these are the cookies that are used to improve the efficiency of advertisement in websites, so that they can be customized for each user. AdSense are Google's advertisement system that uses this kind of cookies.

The law establishes, among other regulations, that all websites that use cookies, with the exception of technical cookies, must warn the user of the usage of these cookies.

The site uses the following kind of cookies:

- Vimeo cookies
- WordPress login cookies.
- Google Analytics cookies

The only cookies not concerned to the law are WordPress login cookies, since they are technical cookies.

The client decided not to follow the law since the website target users are from around the world, specifically from English speaking countries.

### 3.7.3 Licenses

Here the a list of the software packages used on the project and their respective licenses:

- **WordPress 4.1:** License GPLv2
- **Phaser.js:** License MIT.
- **jQuery.js:** License MIT.
- **Advanced Custom Fields WordPress plugin:** GPLv2.
- **Contact Form 7 WordPress plugin:** GNU.
3.8 Budget

After taking into account the different aspects of this project, we can compute the budget. We’ve tried to use free software when it was possible to reduce costs, the priority of the company is to obtain the maximum benefits while offering a quality service.

3.8.1 Human resources

The project will be developed by one person, and managed by two. The budget has been estimated by dividing the tasks into different roles.

The following table show the breakdown of the different roles and budget for each task.

<table>
<thead>
<tr>
<th>Role</th>
<th>Task</th>
<th>Hour estimation</th>
<th>Cost per hour</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesperson</td>
<td>Client meetings</td>
<td>5 h</td>
<td>30 €/h</td>
<td>150 €</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>5 h</td>
<td></td>
<td>150 €</td>
</tr>
<tr>
<td>Project Manager</td>
<td>Plan and viability</td>
<td>50 h</td>
<td>30 €/h</td>
<td>1,500 €</td>
</tr>
<tr>
<td></td>
<td>Choosing a CMS</td>
<td>3 h</td>
<td>30 €/h</td>
<td>90 €</td>
</tr>
<tr>
<td></td>
<td>Choosing a graphics</td>
<td>17 h</td>
<td>30 €/h</td>
<td>510 €</td>
</tr>
<tr>
<td></td>
<td>framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Testing last 2 tasks</td>
<td>10 h</td>
<td>30 €/h</td>
<td>300 €</td>
</tr>
<tr>
<td></td>
<td>Creating the world</td>
<td>25 h</td>
<td>30 €/h</td>
<td>750 €</td>
</tr>
<tr>
<td></td>
<td>Adding backend</td>
<td>1 h</td>
<td>30 €/h</td>
<td>30 €</td>
</tr>
<tr>
<td></td>
<td>features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Creating popups</td>
<td>5 h</td>
<td>30 €/h</td>
<td>150 €</td>
</tr>
<tr>
<td></td>
<td>Responsive design</td>
<td>2 h</td>
<td>30 €/h</td>
<td>60 €</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>113 h</td>
<td></td>
<td>3,390 €</td>
</tr>
<tr>
<td>Web developer</td>
<td>Plan and viability</td>
<td>0 h</td>
<td>15 €/h</td>
<td>0 €</td>
</tr>
<tr>
<td></td>
<td>Choosing a CMS</td>
<td>0 h</td>
<td>15 €/h</td>
<td>0 €</td>
</tr>
<tr>
<td></td>
<td>Choosing a graphics</td>
<td>0 h</td>
<td>15 €/h</td>
<td>0 €</td>
</tr>
<tr>
<td></td>
<td>framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Testing last 2 tasks</td>
<td>55 h</td>
<td>15 €/h</td>
<td>825 €</td>
</tr>
<tr>
<td></td>
<td>Creating the world</td>
<td>175 h</td>
<td>15 €/h</td>
<td>2,625 €</td>
</tr>
<tr>
<td></td>
<td>Adding backend</td>
<td>3 h</td>
<td>15 €/h</td>
<td>45 €</td>
</tr>
<tr>
<td></td>
<td>features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Creating popups</td>
<td>35 h</td>
<td>15 €/h</td>
<td>525 €</td>
</tr>
<tr>
<td></td>
<td>Responsive design</td>
<td>38 h</td>
<td>15 €/h</td>
<td>570 €</td>
</tr>
</tbody>
</table>
### 3.8.2 Hardware

The project requires little use of hardware for its development. We'll be using a desktop computer, a web server and some mobile devices for testing the website once it’s done.

The estimated amortization is based on the 3 month duration of the project.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Price</th>
<th>Useful life</th>
<th>Total estimated amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop computer</td>
<td>620 €</td>
<td>5 years</td>
<td>31 €</td>
</tr>
<tr>
<td>BenQ GW2760HS 27”</td>
<td>199 €</td>
<td>5 years</td>
<td>10 €</td>
</tr>
<tr>
<td>Logitech B100</td>
<td>8 €</td>
<td>5 years</td>
<td>-0 €</td>
</tr>
<tr>
<td>Logitech keyboard K120</td>
<td>13 €</td>
<td>5 years</td>
<td>-0 €</td>
</tr>
<tr>
<td>Logitech headset PC 960</td>
<td>25 €</td>
<td>5 years</td>
<td>-1 €</td>
</tr>
<tr>
<td>Huawei Ascend P7</td>
<td>320 €</td>
<td>3.5 years</td>
<td>23 €</td>
</tr>
<tr>
<td>Apple iPad 4</td>
<td>380 €</td>
<td>4 years</td>
<td>24 €</td>
</tr>
<tr>
<td>Hosting</td>
<td>5.88 € / month</td>
<td>-</td>
<td>18 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>107 €</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.8.3 Software

The project needs the use of some software for its development and use. We’ve used free software as much as possible to reduce the budget.

The estimated amortization is based on the 3 month duration of the project.

<table>
<thead>
<tr>
<th>Software</th>
<th>Price</th>
<th>Useful life</th>
<th>Total estimated amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8</td>
<td>99.95 €</td>
<td>4 years</td>
<td>7 €</td>
</tr>
<tr>
<td>Photoshop</td>
<td>70 €/month</td>
<td>-</td>
<td>210 €</td>
</tr>
<tr>
<td>Sublime Text 2</td>
<td>70 €</td>
<td>3 years</td>
<td>6 €</td>
</tr>
<tr>
<td>Filezilla</td>
<td>0 €</td>
<td>-</td>
<td>0 €</td>
</tr>
<tr>
<td>WordPress</td>
<td>0 €</td>
<td>-</td>
<td>0 €</td>
</tr>
<tr>
<td>Phaser.js</td>
<td>0 €</td>
<td>-</td>
<td>0 €</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>0 €</td>
<td>-</td>
<td>0 €</td>
</tr>
<tr>
<td>Firefox</td>
<td>0 €</td>
<td>-</td>
<td>0 €</td>
</tr>
</tbody>
</table>
### 3.8.4 General costs

In this section we include the generic costs like the internet or the office rental.

The costs of hosting internet service and consumption are not taken into account since they are included in the hosting rental.

The electric cost isn’t computed since it’s included in the office place rental.

<table>
<thead>
<tr>
<th>Context</th>
<th>Consumption</th>
<th>Price</th>
<th>Total estimated amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet service</td>
<td></td>
<td>67 €/month</td>
<td>201 €</td>
</tr>
<tr>
<td>Developer and manager</td>
<td>3 months</td>
<td>170 €/month</td>
<td>510 €</td>
</tr>
<tr>
<td>office place rental</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Salesperson office place</td>
<td>5 hours</td>
<td>170 €/month</td>
<td>-0 €</td>
</tr>
<tr>
<td>rental</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Displacements</td>
<td>4</td>
<td>7 €/displacement</td>
<td>28 €</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>739 €</td>
</tr>
</tbody>
</table>

*Table 5 General costs budget*

### 3.8.5 Risks

As explained in Risks section, there are some risks that can cause a delay during the development of the project. The following table show each risk, its probability, a solution and the added cost of the project to the budget. The costs are computed using the developer price per hour (15 €/h).

<table>
<thead>
<tr>
<th>Risk</th>
<th>Solution</th>
<th>Delay</th>
<th>Probability</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice of an inappropriate</td>
<td>Redo the graphical part of</td>
<td>5 days</td>
<td>5%</td>
<td>30 €</td>
</tr>
<tr>
<td>graphics framework</td>
<td>the project</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low framerate on target</td>
<td>Redo the graphical part of</td>
<td>5 days</td>
<td>5%</td>
<td>30 €</td>
</tr>
<tr>
<td>devices</td>
<td>the project</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low framerate on target devices</td>
<td>Lower graphical quality of the website</td>
<td>1 day</td>
<td>35%</td>
<td>42 €</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Long load times</td>
<td>Create an asynchronous asset loading system</td>
<td>5 days</td>
<td>50%</td>
<td>300 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td>402 €</td>
</tr>
</tbody>
</table>

Table 6 Risk budget

### 3.8.6 Total cost of the project

The following table shows the total budget of the project, which is a sum of the previous tables. It has been decided to add a contingency budget of 15%.

<table>
<thead>
<tr>
<th>Section</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human resources</td>
<td>8,130 €</td>
</tr>
<tr>
<td>Hardware</td>
<td>107 €</td>
</tr>
<tr>
<td>Software</td>
<td>223 €</td>
</tr>
<tr>
<td>General costs</td>
<td>739 €</td>
</tr>
<tr>
<td>Contingency (15%)</td>
<td>1,385 €</td>
</tr>
<tr>
<td>Risks</td>
<td>402 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>10,986 €</td>
</tr>
</tbody>
</table>

Table 7 Total budget

### 3.9 Sustainability and Social Responsibility

#### 3.9.1 Economic

The project is economically sustainable, the material and human resources have been evaluated for the development of the project.

Most of the tools were chosen to be free and open source, thus, reducing the cost and making the product at a competitive price.

The hardware amortization is very cheap, the only way to reduce costs would be reducing the development time.

#### 3.9.2 Social

This project is targeted to the mobile gamer community. Since it’s a public website, everyone can visit it, but it’s only in one language, English, so a big part of potential visitors would not understand it.
Thanks to the website, many people will discover the game and they would be able to see a preview of what’s they’re going to get if they download the game on a smartphone.

It may also inspire some web developers to make websites in a non-conventional way.

### 3.9.3 Environmental

As seen in previous sections, the resources of the project are mostly manpower and computational time. Most of the hardware used is of personal use, so no hardware were bought for the development of the project.

The energy consumption of the desktop computer (300W approximately) used for the development of the project is 2.4 kWh per day. During the development of the project, we used 144 kWh and generated around 100 kg of CO₂. The Spanish yearly CO₂ emissions per capita is 7.4 tones, the project waste equates to the 1.3% of a person’s yearly CO₂ waste. We can say the environmental impact of the project is small.

---

11 http://www.epa.gov/cleanenergy/energy-resources/calculator.html
4 Development

This section will explain the development part of the project. Each of the subsections corresponds to a planning phase explained in the Project planning section.

4.1 Phase 1: Choosing and use of a CMS

Before starting the project we identified client needs, so we could choose the best way to approach the project. As previously said, the client needs to modify some of the content without having knowledge about website coding.

The client has no preference over a CMS (content management system), a custom built CMS can get us out of budget and there’s not really a need to make one from scratch.

The advantages a big CMS offers us are:

- Speed up the development by not reinventing the wheel.
- Basic security is already built in on most CMSs.
- Maintenance of the CMS is done regularly.
- Portability is guaranteed for different systems on each CMS
- Reliability by knowing that tens of thousands of websites are using the same CMS without major problems.

The most common and free to use CMSs for websites are WordPress, Joomla and Drupal.

<table>
<thead>
<tr>
<th></th>
<th>WordPress</th>
<th>Joomla</th>
<th>Drupal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute usage over</td>
<td>23.5%</td>
<td>2.9%</td>
<td>2.0%</td>
</tr>
<tr>
<td>the internet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market share over</td>
<td>60.7%</td>
<td>7.4%</td>
<td>5.1%</td>
</tr>
<tr>
<td>all CMSs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free plugins</td>
<td>&gt;27.000</td>
<td>&gt;7.000</td>
<td>&gt;24.000</td>
</tr>
<tr>
<td>Ease of administration</td>
<td>Easy</td>
<td>Regular</td>
<td>Hard</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>Ease of development</td>
<td>Easy</td>
<td>Regular</td>
<td>Regular</td>
</tr>
<tr>
<td>Price</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Important sites using it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forbes</td>
</tr>
<tr>
<td>SONY</td>
</tr>
<tr>
<td>HARVARD UNIVERSITY</td>
</tr>
<tr>
<td>Linux</td>
</tr>
<tr>
<td>CNN</td>
</tr>
<tr>
<td>THE HILL</td>
</tr>
<tr>
<td>NASA</td>
</tr>
<tr>
<td>WB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Best used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blogs, medium websites, corporate small-sized websites, e-commerce, networking, social</td>
</tr>
<tr>
<td>Fits everything</td>
</tr>
</tbody>
</table>

Table 8 WordPress Joomla and Drupal comparison

WordPress is currently the most used CMSs by a big difference and the most supported by an open community. The previous point, its ease of use, and my previous knowledge of the CMS, are the reasons the project is built in WordPress.

WordPress philosophy is making a lightweight and easy to maintain CMS. That’s the reason it doesn’t come with many basic features like multi-language or e-commerce. Thanks to the active and open community, most of the features can be acquired for free from WordPress plugin repository.

### 4.1.1 WordPress structure

Any WordPress project comes with its load and configuration files in its root directory, and three other directories: wp-admin, wp-includes and wp-content. Each one with a distinctive task.
• **wp-includes**: This folder contains all the files and code that are used by the frontend and backend. Most of the functions deal with querying the database and simplifying the work to be done by the frontend and backend. The contents of this folder should never be edited.

• **wp-admin**: This folder contains all the files and functions that are used by the backend. The backend displays the information of the database and lets the site admin edit the content, install plugins, edit configuration, etc. The contents of this folder should never be edited.

• **wp-content**: This folder contains all the files and functions that are used by the frontend. Wordpress frontend works via themes. Each theme has its own way of displaying the website and they can be completely different from each other.

The following figure shows an example of wp-content directory.
4.1.2 Wordpress database structure

As previously said, WordPress is lightweight and easy to maintain, that’s one of the reasons its database is quite simple. Currently it only supports MySQL database.
• **wp_options**: stores all options under the Administration -> Settings panel.

• **wp_posts**: WordPress data is stored via posts. A post type is always unique, it may be “post”, “page”, “attachment” by default, but additional post types can be created.

• **wp_postmeta**: this table stores all extra data that a post may contain.

• **wp_terms**: a taxonomy term is a way of grouping posts. This table contains the information of a term.

• **wp_term_taxonomy**: this table stores the taxonomy information. This table contains the relation between a term and a taxonomy.
• **wp_term_relationship**: this stores the relation between a post and its terms.
• **wp_users**: Wordpress users are stored in this table. A user has a privilege level, it can be set from a simple subscriber to an administrator.
• **wp_usermeta**: All extra information about a user can be stored in this table
• **wp_comments**: Any post type can have posts. A comment is identified by a post and its author.
• **wp_commentmeta**: All extra information about a comment can be stored in this table.

Querying database information is done via multiple functions that can be found in WordPress Codex site, there is no need to use SQL when creating a WordPress website, although it may be required for creating plugins or adding advanced features.

### 4.1.3 Theme setup

Themes are the way a website is displayed on the frontend. Different themes may share the same database data but display the website in completely different ways.

Themes are stored in *(root directory)/wp-content/themes* in their own directory, and some basic files are required in order to the theme to work.

Some of the basic files are:

• **style.css**: contains the information about the theme (name, version, author, etc…) and the styles of the website.
• **single.php**: the single post template. It’s used when querying a post of type *post*. If this file doesn’t exist, index.php will be used.
• **page.php**: the single page template. It’s used when querying a post of type *page*. If this file doesn’t exist, index.php will be used.
• **home.php**: the home page template by default. If a page is used for the front page, this file will be used to display the blog archive.
• **archive.php**: called on top of all templates. Displays the HTML5 head and opens the body of the website.
• **footer.php**: at the bottom of all templates. Closes the body of the website.
• **index.php**: this template is used when no other files are found.
• **functions.php**: the purpose of this file is to include all theme functions and configurations.

A more in-depth graph can be found below:
Figure 19 WordPress page template decision system
The design of this project is done in a single page. The major part of the site is done in JavaScript, only the templates header.php, page.php, footer.php and functions.php are used.

Underscores theme scratch generator is used so the theme already comes with the basic files, starting from a blank page.
4.2 Phase 2: Choosing graphics framework

The behavior of the project is mostly like a videogame. Doing it with plain HTML5 is inefficient because browsers render regular websites using the CPU. Using the HTML5 canvas element we can generate 2D graphics, but they also are rendered using the CPU. For the GPU to render 2D graphics, the website needs to use WebGL.

WebGL is a JavaScript API that lets use the native implementation of OpenGL ES 2.0. The current support on browsers is the following:

![Figure 21 Browser WebGL compatibility](image)

As seen in the last graph, not all browsers support WebGL, mostly on mobile devices. Since one of the main targets of the website are mobile devices, there needs to be backwards compatibility with HTML canvas.

Starting a project like this from scratch would require creating a graphics framework working in WebGL and HTML5 canvas, a physics engine and all the game logic involved (particles, animations, movement, etc...). It would get us out of budget and time, that’s the reason an already existent framework is needed.

Following there’s a list and a valuation of the most used WebGL frameworks:

<table>
<thead>
<tr>
<th>Name</th>
<th>Graphic capabilities</th>
<th>Documentation</th>
<th>Game capabilities</th>
<th>Physics</th>
<th>Editor</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIXI.js</td>
<td>2D WebGL Canvas</td>
<td>Well documented</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Free (MIT)</td>
</tr>
<tr>
<td>Framework</td>
<td>Support</td>
<td>Documentation</td>
<td>Cross-Platform</td>
<td>Mobile</td>
<td>Price</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>Three.js</td>
<td>2D/3D WebGL and Canvas</td>
<td>Well documented</td>
<td>No</td>
<td>No</td>
<td>Free (MIT)</td>
<td></td>
</tr>
<tr>
<td>Construct 2</td>
<td>2D/3D WebGL and Canvas</td>
<td>Well documented</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>329.99 $</td>
</tr>
<tr>
<td>PlayCanvas</td>
<td>2D/3D WebGL and Canvas</td>
<td>Well documented</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Free (MIT)</td>
</tr>
<tr>
<td>Phaser.js</td>
<td>2D WebGL and Canvas</td>
<td>Well documented</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Free (MIT)</td>
</tr>
<tr>
<td>Babylon.js</td>
<td>2D/3D WebGL and Canvas</td>
<td>Well documented</td>
<td>Yes</td>
<td>Yes</td>
<td>Very basic</td>
<td>Free (Apache License 2.0)</td>
</tr>
<tr>
<td>Crafty.js</td>
<td>2D WebGL and Canvas</td>
<td>Poorly documented</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Free (MIT)</td>
</tr>
<tr>
<td>Box2D.js</td>
<td>2D WebGL and Canvas</td>
<td>Poorly documented</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Free (MIT)</td>
</tr>
<tr>
<td>Cocos2D.js</td>
<td>2D WebGL and Canvas</td>
<td>Poorly documented</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Free (MIT)</td>
</tr>
</tbody>
</table>

Table 9 Web graphic framework comparison

After some documentation, all 3D frameworks are rejected since they add a lot of unneeded overhead that can be critical when displaying the website on old computers or mobile devices.

Phaser.js is built on top of pixi.js, has an open and active community and it’s being used by a lot of commercial and amateur game developers. The well documented codex, its big collection of examples and the good reviews it has are the reasons that the project is using this framework. From now on, the website part related with Phaser will be treated as a game in order for the reader to differentiate between the graphical javascript framework and website HTML elements.

### 4.2.1 Phaser setup

A Phaser game is created once the DOM window loaded event is triggered via:

```javascript
game = new Phaser.Game( gameWidth, gameHeight, renderer, parent );
```

*Figure 22 JavaScript code for creating a new Phaser game*
• **gameWidth**: the width in pixels of games viewport. Set to window width on this project.
• **gameHeight**: the height in pixels of games viewport. Set to window height on this project.
• **renderer**: type of renderer, Phaser.WEBGL or Phaser.CANVAS, Phaser.AUTO automatically detects the best renderer for the browser.
• **parent**: the id of the DOM element where the game will be rendered.

All Phaser modules can be accessed through the game object. Each module adds distinctive functionalities to the game:

```javascript
game.add; // used to add sprites, text, groups, etc
game.camera; // a reference to the game camera
game.cache; // the game cache
game.input; // the global input manager
game.load; // for preloading assets
game.math; // lots of useful common math operations
game.sound; // the sound manager - add a sound, play, etc...
game.stage; // the game stage
game.time; // the clock
game.tweens; // the tween manager
game.state; // the state manager
game.world; // the game world
game.particles; // the particle manager
game.physics; // the physics manager
game.rnd; // the repeatable random number generator
```

*Figure 23 Phaser game attribute modules*

Phaser also allows for different game states. A state of the game provides quick access to common functions like camera, cache, input, sound and more. States help keeping game logic separate for different stages of a game. For example booting the game, preloading assets, main menu, the game, end screen, etc...

Creating a game state is done this way:
function State() = {}; //Create an empty object

State.prototype = {
    init: function() {
    },
    preload: function() {
    },
    create: function() {
    },
    render: function() {
    },
    update: function () {
    }
};

Figure 24 JavaScript of the creation of a Phaser game state

Each stage has a set of functions callbacks executed by the engine at different given times. The most important ones can be seen in the last figure.

- **init**: first method called when entering a state.
- **preload**: preload method is called after init. It’s purpose is to load game assets, objects that require assets that are being loaded should not be created in this method.
- **create**: create method is called after all assets are loaded. The purpose of this method is to create the objects that need the assets loaded in preload.
- **render**: all Phaser objects are rendered automatically without the need to draw them manually. This method is called after Phaser objects are rendered, so any post-processing effects can be added here.
- **update**: this method is called at each game frame. Used to update game logic.

Needed state methods for this project are shown below:

```javascript
//Registers a game state
game.state.add( 'StateName', StateObject );

//Starts a new state and stops updating the current one
game.state.start( 'StateName' );
```

The states required for this projects are Boot, Preloader and Game.
- **Boot**: initializes the game and sets viewport size.
- **Preloader**: game assets would normally get loaded here. This project has some peculiarities, that’s the reason it will load the assets in the Game state while the game is running. It’ll be explained later in another chapter. This state downloads and decodes the game stage json file and passes it to the Game state.
- **Game**: All game logic and asset loading is done in this state.

![Game State Diagram]

Switch state when basic configuration is done

Once scene.json is loaded and decoded, pass scene object and switch state to game (game running state)

### 4.2.2 Game directory structure

All files related to the game are included in the directory assets inside the Wordpress theme project.
The contents of the directories are:

- **atlases**: this folder contains sprite sheets for the different animations and it’s associated json file that defines each frame of the sprite sheet.
- **audio**: audio files for the game in .mp3 and .ogg formats
- **js**: game code
- **RunSheldonGame**: contains the files for each state of the game previously explained.
- **scenes**: contains the json files defining the game’s world.
- **textures**: contains all the graphical elements of the game, in JPEG and PNG format.
- **xhd,hd,md, sd**: they contain different levels of detail for each texture, a more in-depth explanation is done in a following chapter.
4.3 Phase 3: Phaser Test Environment

A test environment is created since the development of games on Phaser is unknown for the participants of the project. In this phase, an exhaustive testing of Phaser capabilities is done in order to make decisions over how to approach the different features the website must have.

4.3.1 Problems, decisions and solutions

During the testing phase, some problems showed up that made us think about alternatives or changing the graphics framework so they could be developed.

This phase also served us to make important decisions before starting to develop the project at its full scale.

4.3.1.1 Movement

The character movement is a major feature of the project, so we needed to decide from the start how the movement system should be approached.

The simplest way of dealing with character movement is fixing its Y coordinate to the ground’s height, and modify its X coordinate as the user moves through the world, but the project demands a more detailed examination.

Firstly, we needed to identify the type of terrain the character was moving: is it flat? Does the ground have bumps? Does the character need to jump? Does Phaser.js allow for complex character movement? Does the character need to collide with other elements?

The kind of terrain the character would be walking on could have bumps and slopes. The following figure show an example of the world’s terrain.
As we can see, the ground isn’t flat, so there needs to be some kind of control over the character Y coordinate.

There were two main ideas, using a physics engine, or creating a fixed path with nodes over which the character would move.
The following table show the pros and cons of each system.

<table>
<thead>
<tr>
<th>System</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physics engine</strong></td>
<td>Flexible system, allows for complex types of movement</td>
<td>Flexible system, unexpected bugs can be found</td>
</tr>
<tr>
<td></td>
<td>Built-in movement tools (forces)</td>
<td>Computationally expensive</td>
</tr>
<tr>
<td></td>
<td>Built-in collisions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scalable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Easy to create a jump movement</td>
<td></td>
</tr>
<tr>
<td><strong>Path node system</strong></td>
<td>Fixed system, avoids most of movement bugs</td>
<td>Fixed system, movement types are hard to make</td>
</tr>
<tr>
<td></td>
<td>Basic movement is easy and fast to develop</td>
<td>Hardly scalable</td>
</tr>
<tr>
<td></td>
<td>Computationally cheap</td>
<td>Handmade collisions</td>
</tr>
</tbody>
</table>

*Table 10 Movement system comparison*

The decision to use a physics engine was made for its flexibility and scalability. Bee Square may want to add features in the future, like jumping or creating a more complex world. The physics system also guarantees a natural looking movement since it uses physics formulas present in the real world.
Phaser.js supports three different 2D physics engines, Arcade.js, Ninja.js and P2.js. Arcade.js and Ninja.js are tile based physics engines, all collision bodies on this system must be rectangular or square. P2.js is a full physics system, it allows convex bodies and all kinds of physical constraints.

Even though P2.js is more computationally expensive, we chose it over the other two because we need convex shapes to create ground slopes.

4.3.1.2 Screen sizes

The website target devices varies from desktop computers to smartphones, there is a need to make an early decision on how to approach different screen resolutions.

There were three ideas for approaching this problem, fitting the world to the screen height, zooming the camera so the world units coincide with screen pixels, and creating a different and simpler website for tablet and smartphones.

The following pictures show an example for each case:

![Figure 29 Game scaling system](image)
![Figure 30 Displaying a portion of the world system](image)

<table>
<thead>
<tr>
<th>System</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fit the world to the screen</td>
<td>No need to change elements of the world</td>
<td>All the world would look smaller</td>
</tr>
<tr>
<td></td>
<td>Fast to develop</td>
<td>Scaling the world may cause measurement unit problems.</td>
</tr>
<tr>
<td></td>
<td>The website would look the same in all devices</td>
<td>Can run at slow framerates on tablets or smartphones</td>
</tr>
</tbody>
</table>

| Zoom the camera       | No need to change elements of the world       | The website would look different depending on every device |
|                       | Fast to develop                               | Smaller screens would only see a portion of the website |
|                       |                                               | Can run at slow framerates on tablets or smartphones |

| Different website version | Customized website version for tablets and smartphones | Development time will get dramatically increased | Not meeting client expectations |

*Table 11 Game adaptation to screen systems comparison*
4.3.1.3 Texture size

The original design of the website’s height was 1,700 pixels, that is the export from a vectorized illustration. Exporting the textures in this size to create the world is unnecessary since most computer screens height is below 1080 pixels. We decided to scale down the original design to 1,200 pixels height to avoid unnecessary weight to the website, speeding up loading time and increase framerate at the same time.

After making the decision to decrease the design’s size, there was still an issue regarding the screen size of the devices. It’s unnecessary for small devices to download and render high quality textures since the details are lost on smaller screens. The only idea that came to mind was to create a level of detail system for the textures, so different sizes of the same texture would get loaded depending on screen size. A more complete explanation is done in an upcoming section.

4.3.1.4 Loading time

Since the beginning of this phase, we realized that there was going to be a problem with the time it takes to load the website. As we began to export textures we noticed some textures weighted more than 1MB. An estimation of more than 30 MB of data in textures was made.

Multiple usability studies\textsuperscript{12}\textsuperscript{13}\textsuperscript{14} determine that as loading time increases, the probability of losing a visitor increases too. Most of the visitors will leave the website before 10 seconds have passed and the website isn’t still loaded. A load time of 4 seconds is considered acceptable.

Since the weight estimation of the website is so high, there needs to be some kind of feedback to show the visitor the loading progress so he thinks twice before leaving the website. The decision of displaying a loading screen is necessary.

\textsuperscript{13} http://www.nngroup.com/articles/response-times-3-important-limits/
\textsuperscript{14} https://blog.kissmetrics.com/loading-time/?wide=1
Having a loading screen helps the visitor know how much he will have to wait before the website finally loads.

Despite the efforts done to give the visitor some feedback, it isn’t really solving the loading time. The objective is to load the website in less than 10 seconds. After some testing we realized it was impossible to load and create the whole world in less than that time. So another loading system needed to be developed.

Since the website design is composed by different stages or sections, we can load them separately and let the user interact with the site while the following stages are loading in the background. This way, only global assets and the first stage are needed for the game to start running.

A more detailed explanation of the loading system is done in an upcoming section.

4.3.1.5 Framerate

Since the project uses large textures, and the game uses all browser viewport, acceptable framerates cannot be warrantied for all devices. A prerequisite for developing the website is set so that for it to run at acceptable framerates the device needs to have a dedicated graphics card.
Despite the prerequisite, an effort needs to be made for obtaining the maximum framerate possible. The following list show some of the planned optimizations to increase the framerate:

- Disable collisions on stages other than the current one being visited.
- Level of detail texture system (explained in an earlier section)
- Only update sprites on the current stage
4.4 Phase 4: Development of game features

In this section we’ll explain the logic behind the game part of the project and the different approaches we took for every game’s feature.

Since most of this section is made in JavaScript, I won’t be covering the intrinsic details on how they were developed, but they will be explained via figures and diagrams so the reader doesn’t need to know the details of the technologies used. However, some code will be shown and explained when needed.

The game part of the project will be using the following technologies:

- JavaScript
- Phaser.js
- JSON
- jQuery

4.4.1 Game’s flowchart

In this section, we’ll explain the game flowchart and game loop. This section should work as a reference for the upcoming sections since it explains the whole game process from start to finish. The game has two distinctive blocks, load and configure the game, and run the loop.

As said in a previous section, the game loads its assets asynchronously, it’s represented in the flowchart as an independent process that runs in parallel to the game loop.

Most of the processes shown in the flowchart will be explained in detail on upcoming sections.
4.4.2 Scene file format and item creation

The scene file contains all the assets and elements shown in the game. Having a scene file lets us separate the game assets and elements from the game code, and abstract the asset loading and element creation.

The scene file format for the project is written in JSON since it’s a common data storing format and jQuery comes with a built-in parser.

```json
{
  "stages": [
    {
      "width": 0,
      "name": "global",
      "locked": false,
      "resources": {
        "images": [
          /* Array of image textures */
        ],
        "atlases": [
          /* Array of spritesheets */
        ],
        "audios": [
          /* Array of audio files */
        ]
      },
      "sprites": [
        /* Array of sprites */
      ],
      "tilesprites": [
        /* Array of tilesprites */
      ],
      "coinGroups": [
        /* Array of coin groups */
      ],
      "sounds": [
        /* Array of sounds */
      ],
      "emitters": [
        /* Array of particle emmitters */
      ]
    },
    /* Second stage */
  ]
}
```

Figure 33 Summary of assets/scenes/runsheldon-map.json
The previous figure show a partial representation of the format used to represent the world scene.

Once the game boots, this file is loaded and decoded using jQuery’s `getJSON` helper function:

```javascript
jQuery.getJSON(game.load.baseURL + "scenes/runsheldon-map.json", function (decodedJson) {
    decodedJsonScene = decodedJson;
});
```

*Figure 34 JavaScript code used to decode a JSON file*

The decoded json is stored in the global variable `decodedJsonScene` that is used for loading and creating the assets later on.

### 4.4.2.1 Stages

The stages determine the assets and elements that are contained inside them and they represent each stage present on the design.

The stage attributes are:

- **width**: stage width in pixels.
- **name**: name of the stage
- **locked**: attribute that controls whether the character can move to this stage

```json
"stages": [
    {
        "width" : 3000,
        "name" : "teststage",
        "locked" : true,
        ...
    }
]
```

*Figure 35 partial definition of a stages element in the scene JSON*

### 4.4.2.2 Resources

The resources are the asset files needed for each game element. This object contains the stage needed textures, sprite sheets and audio files.

The resources attributes are:

- **images**: an array of the needed textures for the stage
- **atlases**: an array of the needed sprite sheets for the stage
- **audios**: an array of stage sounds

```json
"resources":{
  "images":
  // [materialName, textureFile]
  ["transparent","transparent.png"],
  ["dark-overlay","dark-overlay.png"],
  ["sheldon-shadow","sheldon-shadow.png"]
},
"atlases":{
  // [materialName, textureFile, atlasFile]
  ["sheldon", "sheldon.png", "sheldon.json"],
  ["coin", "coin.png", "coin.json"]
},
"audios":{
  // [soundName, audioFile]
  ["testaudio"," testaudio.ogg "]
}
}
```

*Figure 36 Resources array of a stage in scene JSON*

Images, spritesheets/atlas and audio files can be loaded using the following Phaser methods:

```javascript
/* Loads an image */
(game.load.image( keyName , imagePath ));
/* Loads a spritesheet */
(game.load.atlasJSONHash( keyName , textureFilePath , atlasFilePath ));
/* Loads an audio file */
(game.load.audio( keyName, audioPath ));
```

*Figure 37 JavaScript and Phaser.js code to load resource files*

### 4.4.2.3 Sprites

The sprites are the elements that make most of the world. They have position, size and material among other extra attributes.

An UV coordinate determines how the texture show up on the shape. Since sprites have a rectangular shape, they have four vertices, which means, four UV coordinates for each sprite. A UV coordinate is a two-dimensional vector with values comprised from 0 to 1, and they represent the location, in percentages, where the texture should be projected on that vertex.
Sprites will always project the entire material on themselves, that means, its uv coordinates will always be uv1(0,0), uv2(1,0), uv3(0,1),uv4(1,1).

The following attributes comprise the basic definition of a sprite in the scene file format.

```
"sprites":{
    "x" : 4250,
    "y" : 0,
    "z" : 90,
    "width" : 1550,
    "height" : 1230,
    "material" : "stage04-trunk-front"
}
```

- **sprites**: array of sprites
- **x**: horizontal position of the sprite on the world
- **y**: vertical position of the sprite on the world
- **z**: depth position of the sprite on the world
- **width**: width of the sprite
- **height**: height of the sprite
- **material**: name of the material resource it uses

Extra attributes will be explained in upcoming sections.
The following Phaser method creates a sprite and adds it to the world.

```javascript
/* Create sprite and store its object into var sprite */
var sprite = game.add.sprite( xPosition, yPosition, keyTextureName );
/* Store the sprite to the sprites array of the current stage being created */
stages[currentStage].sprites.push( sprite );
/* We assign/modify all the extra sprite attributes here */
/* ... */
```

**Figure 40** JavaScript and Phaser code for creating a sprite

### 4.4.2.4 Tile Sprites

Tilesprites are sprites that are more flexible on the UV coordinates of the material, the UV coordinates can be positioned anywhere on the texture. They also allow for textures to be infinitely repeated over the tilesprite, allowing to project seamless textures.

![Tilesprite](image)

**Figure 41** Tilesprite partial definition in scene JSON file

Tilesprites have the following attributes:

```json
"tilesprites": {
  "x": 1800,
  "y": 0,
  "z": 30,
  "width": 150,
  "height": 1200,
  "material": "stage03-waterfall",
  "tileScale": { "x": 1, "y": 2 },
  "tilePosition": { "x": 100, "y": 200 },
  "scroll": { "x": 0, "y": 800 }
}
```
• **tilesprites**: array of tilesprites
• **x**: horizontal position of the tilesprite on the world
• **y**: vertical position of the tilesprite on the world
• **z**: depth position of the tilesprite on the world
• **width**: width of the tilesprite
• **height**: height of the tilesprite
• **material**: name of the material resource it uses
• **tileScale**: scale of the texture projected on the tilesprite. Default is 1
• **tilePosition**: offset position of the texture from top-left tilesprite’s position
• **scroll**: scroll velocity of the texture. Default is 0

The following Phaser method creates a tilesprite and adds it to the world.

```javascript
/* Create tilesprite and store its object into var sprite */
var tileSprite = game.add.tileSprite(xPosition, yPosition, width, height, material);
/* Store the tilesprite to the tilesprites array of the current stage being created */
stages[currentStage].tilesprites.push(tileSprite);
/* We assign/modify all the extra tilesprite attributes here */
/* . . . */
```

Figure 42 Javascript and Phaser code for creating a tilesprite

### 4.4.2.5 Coin Groups

Coin groups are groups of sprites that represent a coin on the map.

Each coin can detect collisions with Sheldon, and will fire an event if so, making the coin disappear.

![Coin Group Example](image)

The following figure show the format of a coin group on the scene file.
"coinGroups" : [
  {
    "chance" : 0.7,
    "sprites" : [
      {
        "x" : 2100,
        "y" : 410,
        "z" : 70,
        "width" : 79,
        "height" : 91,
        "material" : "coin",
        "animations" : [
          /* Explained on a upcoming section */
        ],
      },
      {
        "x" : 2220,
        "y" : 410,
        "z" : 70,
        "width" : 79,
        "height" : 91,
        "material" : "coin",
        "animations" : [
          /* Explained on a upcoming section */
        ],
      },
      {
        "x" : 2340,
        "y" : 410,
        "z" : 70,
        "width" : 79,
        "height" : 91,
        "material" : "coin",
        "animations" : [
          /* Explained on a upcoming section */
        ]
      }
    ]
  },
],
/* Create coin and store its object into var coin */
var coin = game.add.sprite( xPosition, yPosition, keyTextureName );
/* Store the coin to the coins array of the current stage being created */
stages[currentStage].sprites.push( coin );

Figure 43 Coin group partial definition in the scene JSON file

The attributes of a coin group are:

- **coinGroups**: array of coin groups
- **chance**: the chance of the group to spawn in the world
- **sprites**: array of sprites representing the coins

The coins are created exactly the same as a sprite, since they are sprites, but they are added to the coin groups array instead of the sprites array.
4.4.2.6 Sounds

The sounds of the game appear when some events are fired or as background music.

The attributes of a sound element are:

```
"sounds" : [
  {
    "key" : "loadingScreenAudio",
    "volume" : 1,
    "loop" : true,
    "autoPlay" : true
  }
]
```

*Figure 45 Sound definition in the scene JSON file*

- **sounds**: array of sound elements
- **key**: key name of the sound
- **loop**: should the sound loop?
- **autoPlay**: start playing the sound once audio file is loaded

The following Phaser method creates a sound and adds it to the world.

```
/* Create sound and store its object into var sound */
var sound = game.add.sound( keyName, volume, loop );
/* Store the sound to the sounds array of the current stage being created */
stages[stage].sounds.push( sound );
/* We assign/modify all the extra sound attributes here */
/* . . . */
```

*Figure 46 Javascript and Phaser code for creating a sound element*

4.4.2.7 Particle emitters

Particle emitters are elements that spawn sprites in a given way. They can spawn high numbers of sprites without harming performance by having all sprites precached from the start, this way there is no object creation or destruction while the game is running.
The attributes of an emitter element are:

```
"emitters":
{
  "x": 3000,
  "y": 1400,
  "z": 89,
  "width": 6000,
  "maxParticles": 100,
  "material": "snowflake",
  "lifespan": 9000,
  "particlesPerSecond": 30,
  "gravity": 70,
  "scale": 0.8
}
```

- **x**: horizontal position of the emitter on the world
- **y**: vertical position of the emitter on the world
- **z**: depth position of the emitter on the world
- **width**: width of the emitter where the particles will be spawned
• **maxParticles**: sets a maximum number of particle sprites to precache
• **material**: material of the particles
• **lifespan**: life time of each particle before disappearing
• **particlesPerSecond**: particles emitted per second
• **gravity**: speed of the gravity affecting the particle
• **scale**: scale of the particles relative to its material size

The following Phaser method creates a particle emitter and adds it to the world.

```javascript
/* Create emitter and store its object into var emitter */
var emitter = game.add.emitter(x, y, maxParticles);

/* Store the emitter to the emitters array of the current stage being created */
stages[stage].emitters.push(emitter);

/* We assign/modify all the extra particle emitter attributes here */
/* . . . */
```

*Figure 49 JavaScript and Phaser code for creating a particle emitter*

### 4.4.3 Animations

Animations can be added to any sprite or tilesprite. In the project, there will only be three elements with animations, Sheldon, load bar between stages and the coins.

An animation can be described as a series of frames, that when played at a certain framerate, give the illusion of a moving image.

If a sprite has animation, its texture needs to be a spritesheet, and it need to go with an atlas file with definitions for each frame (name, position and size). The atlas file is in JSON hash format, it’s a standard format for exporting spritesheet animations that most spritesheet editors support.
Animations can be added to a sprite by adding the animations array into its definition:
“animations”: [
  {
    “name”: “run”,
    “frames”: [
      “run00”,
      “run01”,
      “run02”,
      /* etc... */
    ],
    “fps”: 18,
    “loop”: true,
    “autostart”: true
  },
  {
    “name”: “jogging”,
    “frames”: [
      “jogging00”,
      “jogging01”,
      “jogging02”,
      /* etc... */
    ],
    “fps”: 14,
    “loop”: true
  },
  {
    “name”: “idle”,
    “frames”: [
      “standing01”
    ],
    “fps”: 1,
    “loop”: true
  }
]

Figure 51 Animations definition on the scene JSON file

- animations: array of animations
- name: name of the animation
- frames: array of frame names defined in the atlas file
- fps: framerate of the animation
- loop: should the animation loop?
- autostart: automatically start animation when sprite is created

4.4.4 Physics

As explained earlier, we decided to use a physics engine for the movement system of the character.

The initial configuration for physics is the following one:
// We are telling Phaser to use P2.js physics
game.physics.startSystem(Phaser.Physics.P2JS);
// We tell the physics engines to detect collisions
game.physics.p2.setImpactEvents(true);
// Physics restitution is 0
game.physics.p2.restitution = 0;
// Physics friction coefficient is 1
game.physics.p2.friction = 1;
// Gravity in units/s^2
game.physics.p2.gravity.y = 5600;

Figure 52 JavaScript and Phaser code initialization of physics

Once we have initiated the world, we can create physical body. Physical bodies are attached to sprites and they need a collision group to interact with other bodies. Collision groups are a way of optimization, where collisions are only calculated for those collision groups that can be collided with each other.

A static element is an element that has a physical body but will never move. They are computationally cheap since physics calculations are not applied to them.

Adding physics to a body can be done by adding the physics attribute to a sprite:

{  
    "x": -500,
    "y": 0,
    "z": 100,
    "width": 6700,
    "height": 365,
    "material": "transparent",
    "physics": {
        "static": true,
        "collisionGroup": "worldGroup",
        "collidesWith": ["sheldonGroup"]
    }
}

Figure 53 physics definition on a sprite on the scene JSON file

- **physics**: physics attributes
- **static**: is body static?
- **collisionGroup**: collision group this body belongs to
- **collidesWith**: array of collision group this body collides with

The previous configuration will create a rectangular physical body the size of the sprite.
There are some cases where there needs to be a more flexible body shape. For example, we would want to have a circular physical body. The following attribute sets a circular body to the sprite:

```
"physics" : {
  "static" : false,
  "collisionGroup" : "decorationGroup",
  "collidesWith" : ["worldGroup","sheldonGroup","decorationGroup","phaseLimitGroup"],
  "circle" : 45
}
```

**Figure 55 Circle option on the physics definition**

- circle: radius of the circle in world units.

**Figure 56 The circle physics option in action**
There’s still an unsolved issue regarding slopes, a new attribute was created for customizing body shapes.

```json
"physics" : {
  "static" : true,
  "collisionGroup" : "worldGroup",
  "collidesWith" : ["sheldonGroup"],
  "shapes" : [
    [-1, -1.2, -1, 1, 1, 1, 1, -1]
  ]
}
```

- **shapes**: array of shapes. Each shape is an array of percentage coordinates relative to the sprite size [x1,y1,x2,y2,...]

![Figure 57 Physics shapes attribute on scene JSON file](image)

4.4.5 Camera

In this section we will discuss the camera logic and how the user interacts with it.

The camera has the following attributes:

- **position**: x,y position of the camera
• bounding box: a bounding box that defines the visible part of the world

![Figure 59 Camera visible portion of the world](image)

The camera is set to occupy world’s height, and adapt its width depending on the aspect ratio of the viewport.

There are some situations where the camera will get blocked and won’t let the user keep moving forward through the world. The next stage may not be loaded, in this case, the camera clamps its bounds so the next stage can’t be seen. There’s also a situation where the user may need to accept a button to be allowed to walk into the next stage, in this situation, and only if the next stage is loaded, the camera will clamp its position to the next stage start location.

The following figure shows the process of updating the camera for each loop.
4.4.6 Inputs

The only way the user can interact with the game is via inputs. Since the project target devices are also mobile, there needs to be both input systems, mouse/keyboard and screen touch.

The following function initializes game’s inputs by enabling key listeners and mouse/touch callbacks:
/* Variable that stores the horizontal input for each frame */
var iX;

function initInputs() {
    /* Enables left and right arrow keys to be listened and returns a key object */
    arrowLeft = game.input.keyboard.addKey( Phaser.Keyboard.LEFT );
    arrowRight = game.input.keyboard.addKey( Phaser.Keyboard.RIGHT );

    /* Sets touchStart as the starting click/touch callback */
    game.input.onDown.add( touchStart );
    /* Sets touchEnd as the ending click/touch callback */
    game.input.onUp.add( touchEnd );
    /* Sets touchMove as the mouse/touch move listener */
    game.input.addMoveCallback( touchMove );

    /* Sets mouseWheel as the mousewheel listener */
    game.input.mouse.mouseWheelCallback = mouseWheel;
}

function updateInputs(){
    /* If left arrow key is down set -45 as the input amount */
    if( arrowLeft.isDown )
        iX = -45;
    /* If right arrow key is down set 45 as the input amount */
    if( arrowRight.isDown )
        iX = 45;
}

function mouseWheel( input ){
    iX += -game.input.mouse.wheelDelta * MOUSEWHEEL_SCALE;
}

Figure 61 Phaser input initialization

Detecting key press can be done through the key object we saw in the previous figure by checking its attributes in the update function:

Mousewheel input detection is quite simple to do, the following figure show how the callback works:

The touch input process needs to move the camera while swiping on the screen. That means, capturing the initial touch position, and moving the camera according to user movement on the screen. The following callbacks show the process of updating the camera while the user is swiping.
4.4.7 Sheldon loop and states

In this section we will discuss the logic behind Sheldon. Sheldon purpose on the game is:

- Following the camera while the user visits the site.
- Collecting coins found through the scene.
- Displaying a popup message when clicked.

4.4.7.1 Sheldon states

A very simple artificial intelligence system was created to let Sheldon perform different actions depending on the current state he’s in. Each state has a specific set of actions performed while Sheldon’s in that specific state, switching states may also trigger a transitional set of action.

The following list show Sheldon’s states:
- Idle: Sheldon’s standing still. There’s no animation on this state.
- Warming up: Sheldon’s standing still while warming up. There’s a warmup animation played in this state.
- Running: Sheldon is moving following the camera. There’s a running animation played on this state.
- Falling: Sheldon’s on the air falling. No animation is played during this state.

The following diagram show the state diagram and how they are related to each other:

![State Diagram](image)

*Figure 65 Sheldon states*

### 4.4.7.2 Sheldon’s loop

The Sheldon update function has to deal with character AI, movement, collisions with other elements and updating its child elements.

The following flowchart shows the tasks carried out by Sheldon’s update function:
Figure 66 Sheldon update flowchart
4.4.7.3 Sheldon movement

We’ve discussed the different Sheldon states and processes, but we haven’t still explained how he moves through the world.

Since we’ve been using a physical engine, and movement is just horizontal movement, the process of moving Sheldon is quite simple.

The following block of code updates Sheldon position:

```javascript
/* Compute movement direction */
var direction = cameraActor.position.x - sheldonSprite.position.x;

if( direction >= 0 ){
    /* Make Sheldon sprite face right */
    sheldonSprite.scale.x = Math.abs( sheldonSprite.scale.x );
    /* moveRight is a P2.js built-in function that applies a positive X force to a
     * physical body while trying to keep the body on the ground*/
    sheldonSprite.body.moveRight(SHELDON_MAX_VELOCITY);
} else {
    /* Make Sheldon sprite face left */
    sheldonSprite.scale.x = Math.abs( sheldonSprite.scale.x ) * -1;
    /* moveLeft is a P2.js built-in function that applies a negative X force to a
     * physical body while trying to keep the body on the ground*/
    sheldonSprite.body.moveLeft( SHELDON_MAX_VELOCITY );
}
```

Figure 67 Sheldon movement JavaScript and Phaser code

4.4.8 Scrolling textures

There are some sprites on the game that needs to have a scrolling texture, that is achieved by using a tilesprite. As explained in an previous section, tilesprites can use seamless textures by repeating the texture over tilesprite surface. For making it scroll we just need to move its texture position by using the classic physics position formula $x = v \cdot t$. 

85
/* Update tilesprite */
function updateTileSprites(stage){
    var tilesprites = stages[stage].tilesprites;
    for( var i = 0; i < tilesprites.length; i++ ){
        /* ... */
        var elapsedSeconds = game.time.tickElapsedSeconds();
        /* If tilesprites scrolls */
        if(tilesprites[i].scrollVelocity){
            /* Apply horizontal scroll */
            tilesprites[i].tilePosition.x = sprites[i].scrollVelocity.x * elapsedSeconds;
            /* Apply vertical scroll */
            tilesprites[i].tilePosition.y = sprites[i].scrollVelocity.y * elapsedSeconds;
        }
        /* ... */
    }
}

As can be seen in the following figure, the waterfall scrolls its texture down by executing the previous algorithm.

![Scrolling textures in action](image.png)
4.4.9 Parallax backgrounds

Parallax is a displacement or difference in the apparent position of an object viewed along two different lines of sight, and is measured by the angle or semi-angle of inclination between those two lines.  

The following figure show a simplification of the effect.

![Parallax Figure]

Figure 70 Parallax effect example

The effect can also be used to calculate distances since nearby objects have a stronger parallax than further objects.

The effect is strongly related with animal, and in particular, human depth perception. By having one eye next to another, looking in the same direction, we can tell how further away an object is by internally calculating the perceived distance change of that object when the viewpoint changes.

Parallax effect in web design and videogames tries to confuse the brain that the 2D elements on the screen are in a three dimensional world. This is achieved by moving the backgrounds depending on the depth that the element is supposed to be.

The attribute used to add parallax to a sprite or tilesprite is:

```
"parallax" : { "x" : 0.15 , "y" : 0 }
```

Figure 71 Parallax attribute in sprites and tilesprites

---

15 http://en.wikipedia.org/wiki/Parallax
The value indicates how much the object is affected by parallax, higher values will make the object have a stronger parallax, and zero value doesn’t apply any parallax to the object.

The update function applies the following formula to the sprite/tilesprite texture position:

\[
x = x_0 + c_x \times p_x
\]

- \(x\) : sprite/tilesprite texture parallax position
- \(x_0\) : sprite/tilesprite texture initial position
- \(c_x\) : camera current X axis position
- \(p_x\) : parallax amount

If we used the last formula without previously preparing the initial position, we would get incorrect sprite positions. The problem has to do with the fact that the desired sprite position isn’t the same as the initial position on the formula. An example scenario has been prepared to make the issue more understandable.

The sprite S is defined as so:

```json
{
    "x" : 8000,
    "y" : 0,
    "z" : 10,
    "width" : 100,
    "height" : 100,
    "material" : "randommaterial",
    "parallax" : { "x" : 0.3 , "y" : 0 }
}
```

The camera is at position 8000 units.

\[
x = 8000 + 8000 \times 0.3 = 10400
\]

As we can see, the desired position of 8000 units of the sprite is incorrect when the camera it’s at its same position. The sprite position should have its desired initial position when the camera is at the same position. We need to prepare the initial position using the following formula.

\[
x_0 = x_d - x_d \times p_x
\]

- \(x_0\) : sprite/tilesprite initial position post parallax setup
- \(x_d\) : sprite/tilesprite desired initial position
\( p_x \): parallax amount

Now if we use the previous example using the last formula we get the correct positions on the sprites/tile sprites.

The sprite \( S \) is defined as so:

\[
\{ \\
    \text{"x"} : 8000, \\
    \text{"y"} : 0, \\
    \text{"z"} : 10, \\
    \text{"width"} : 100, \\
    \text{"height"} : 100, \\
    \text{"material"} : \text{"randommaterial"}, \\
    \text{"parallax"} : \{ \text{"x"} : 0.3, \text{"y"} : 0 \}
\}
\]

The camera is at position 8000 units.

\[
x_0 = 8000 - 8000 \times 0.3 = 5600
\]

\[
x = 5600 + 8000 \times 0.3 = 8000
\]

Figure 73 Parallax example 2

As we can see, now we get the correct position for the sprite.

The following pictures show the actual parallax effect running in the game. The first two images show a camera displacement of around 300 units, we can see that by looking at Sheldon position just in front the two rabbits.
If we compare both images, we can see that the background doesn’t have a fixed position. The following picture is a multiplication of these two images, by multiplying both images we can see where the parallax effect is taking place.
4.4.10 Game adapt to screen sizes

In this section we will explain how we adapted the game for any kind of screen.

Phaser automatically converts game units into pixels, since the game’s world height occupies 1200 units, that would mean, 1200 pixels of height whatever the screen size is.

The following picture show how we would see the game depending of random chosen screen sizes if we didn’t adapt the game to the screen.
Since it’s the game height what bothers us, we will be adapting the game’s height to the screen height.

We initialize the world scale with the following functions:

```javascript
/* We divide game height (screen height) by world's height (1200px) */
game.world.scale.set(game.height / game.world.height);
```

By executing the last function, we scale the game’s world up or down to fill screen’s height, this way the game will look the same in any screen resolution.

![Figure 77 Game display after scale adaptation](image)

### 4.4.11 Level of detail texture system

As we’ve explained before, there needs to be a texture level of detail (LOD) for two reasons, increasing framerates on low spec devices, and decreasing loading times on mobile devices.

The textures are stored in different directories depending on the quality level. The quality levels supported are xhd, hd, md and sd from higher to lower quality.
The following variables manage the texture LOD system:

```javascript
/* Texture quality level ID */
var TEXTURE_QUALITY_XHD = 0;
var TEXTURE_QUALITY_HD = 1;
var TEXTURE_QUALITY_MD = 2;
var TEXTURE_QUALITY_SD = 3;

/* textureQuality var is initialized to XHD */
var textureQuality = TEXTURE_QUALITY_XHD;
/* The textureQualityArray store the directory name for every quality level */
var textureQualityArray = ['xhd','hd','md','sd'];
/* The textureQualityScaleArray stores the scale in comparison with XHD of each texture quality level */
var textureQualityScaleArray = [1,0.8,0.65,0.45];
```

The following diagram show the texture quality decision process:
The texture LOD system is only used when loading the assets, we use the following code to correctly generate the texture path.

```plaintext
textureURL = websiteURL + "\assets/textures/" + textureQualityScaleArray[textureQuality] + textureName;
```

Figure 80 Texture quality decision flowchart

Figure 81 Texture url generation
Scaling down the original texture isn’t a problem for sprites since their UVs occupy the entire sprite surface, but for tilesprites is a major problem since scaling down a the original texture will modify how the UVs are displayed on the tilesprite surface.

An example of the problem is shown in the following figure, since the tilesprite default texture scale is 1, it isn’t adapting the lower sized textures to its surface.

![Figure 82 Tilesprite bug before fixing its scale](image)

For fixing this issue, the following code is required.

```c
/* Scale the texture depending on the texture quality */
sprite.tileScale.x = 1 / textureQualityScaleArray[textureQuality];
sprite.tileScale.y = 1 / textureQualityScaleArray[textureQuality];
```

![Figure 83 Code used to fix tilesprite scale](image)

At last, we can make a comparison of the total size of website’s textures depending on the texture quality being loaded.

<table>
<thead>
<tr>
<th>Texture quality</th>
<th>Texture scale</th>
<th>Website’s texture size</th>
<th>Website size decrease percentage from original size</th>
</tr>
</thead>
<tbody>
<tr>
<td>XHD</td>
<td>100%</td>
<td>31 MB</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>LOD</td>
<td>Size</td>
<td>Compression</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>HD</td>
<td>80%</td>
<td>20 MB</td>
<td>35.5%</td>
</tr>
<tr>
<td>MD</td>
<td>65%</td>
<td>13.8 MB</td>
<td>55.5%</td>
</tr>
<tr>
<td>SD</td>
<td>45%</td>
<td>7.44 MB</td>
<td>76%</td>
</tr>
</tbody>
</table>

Table 12 Texture LOD size and comparison

4.4.12 Asynchronous load and creation of stages

As explained before, website loading times are critical for attracting visitors and preventing visitors from leaving. The system chosen to decrease loading time, along with texture LOD system, is to load the stages independently and asynchronously while the game is already running.

By developing this system the game can start running after loading the two first stages (global elements and first stage).

The following diagram show the process of loading and creating the world.
Figure 84 Asynchronous loading flowchart

If the visitor arrives to the following stage before it’s loaded, a loading bar is shown with the loading progress of the stage.
The following table show the improvement comparison of having this system.

<table>
<thead>
<tr>
<th>Texture quality</th>
<th>World initialized time</th>
<th>Total loading time</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>XHD</td>
<td>11.5 s</td>
<td>45 s</td>
<td>75%</td>
</tr>
<tr>
<td>HD</td>
<td>5.7 s</td>
<td>25.5 s</td>
<td>77%</td>
</tr>
<tr>
<td>MD</td>
<td>4.9 s</td>
<td>21.5 s</td>
<td>77%</td>
</tr>
<tr>
<td>SD</td>
<td>3.7 s</td>
<td>17.5 s</td>
<td>78%</td>
</tr>
</tbody>
</table>

*Table 13 World initialization improvement table*

Although load time will vary between devices, server cpu load, bandwidth and other factors, we can clearly see that the time before the user can interact with the website is improved by around 75%.
4.5 Development of WordPress features

In this section we’ll explain the key elements of the backend data storing and a brief overview of how the database data can be used to create the HTML5 website.

The website part of the project will be using the following technologies:

- PHP
- WordPress
- HTML5
- MySQL
- CSS
- JavaScript

4.5.1 WordPress backend panel

The WordPress backend panel is the administrator part of the website. It can be accessed via:

http://www.mywordpresssite.net/wp-admin

Figure 86 Wordpress backend for mywordoresite.net

After login in we have full access (depending on user role) to WordPress features, even site’s code can be edited from here.
The Pages menu panel lets us manage posts of type *page* that are mostly used to show website static content (Contact, Terms of use, etc pages).

The page edit panel lets us modify the WordPress default page fields like the title, content, date added, author, featured image, etc.
4.5.2 Registering new post types

Default WordPress posts types are very helpful, but implementing new posts types is mandatory for more advanced websites. For example, an e-commerce may want to have a Product post type to store product data without having to share the same panel with Pages or Posts.

Registering a new post type is done by using the `register_post_type` function.

```php
function register_enemy_post_type() {
    $args = array( 
        'public' => true, //Can be queried
        'label' => 'Enemies' //Post type name label
    );
    /* Register post type with id "enemy" and $args options */
    register_post_type( 'enemy', $args );
}
/* Execute function register_enemy_post_type as soon as WordPress "inits" */
add_action( 'init', 'register_enemy_post_type' );
```
The previous execution of code will add another item to the WordPress admin panel, the following figure show the result.

![Figure 91 New post type panel](image)

The process of registering a new post type doesn’t add any data to the database, it just creates a backend panel to manage the post type. When a new item of the post type is created, then it’s when it’s saved in the database with its corresponding post type column.
The post types used on this project are:

- Enemies
- Costumes
- Boosters
- Powerups
- Tips and tricks

### 4.5.3 Adding post meta fields

By default, WordPress support a set of data fields for all post types (title, content editor, author, featured image, etc).

It may be useful to store extra fields to a post type, for example, and following the ecommerce example, a product may be using the fields price, stock quantity, shipping price, etc.

WordPress doesn’t have a nice and standard way of adding extra fields to the posts, that’s the reason we’ll be using Advanced Custom Fields plugin for WordPress. The plugin has a panel that lets the administrator add different kinds of fields to any element that matches the Custom Field Group criteria.
By saving the previous Custom Field Group, new fields are added to the enemy post type items.
Figure 94 Enemy with backend panel with added custom fields
4.5.4 Retrieving post data and showing it into the frontend

Once we’ve got the data stored in the backend, we may want to display it in the frontend. A possible simple WordPress page template may look like so:

```php
<?php get_header(); //echo html5 head and website header ?>
<div id="primary" class="content-area">
  <div id="main" class="site-main" role="main">
    <?php /* Start the WordPress loop of the main query */ ?>
    <?php /* In this case, a single page */ ?>
    <?php while ( have_posts() ) : the_post(); ?>
      <h1 class="entry-title"><php the_title(); ?></h1>
      <div class="entry-content"><php the_content() ?></div>
    <?php endwhile; ?>
    </div><!-- .site-main -->
  </div><!-- .content-area -->
<?php get_footer(); //echo website footer ?>
```

Figure 95 WordPress example page template

A WordPress loop, loops through all the elements in a given query. Inside a WordPress loop we can use a huge set of functions (the_title, the_content, etc) to retrieve post data of the current post in the loop. In the previous example we explained how it was a page template, so the WordPress main query will only have one element, therefore a single iteration of the loop will be made.

The project will be using a page template similar to the last figure, inside the loop we will be retrieving all items from each post type and creating its HTML5 slider.

Since there’s only one page on the project, the main query element will be the Home page, and there will be only one iteration on the main loop.
There needs to be a custom loop for each slider, it can be achieved by using the following code.

```php
$args = array(    "post_type" => "enemy", //retrieve posts of post type enemy    "post_status" => "publish", //retrieve only published posts    "posts_per_page" => -1, //retrieve all posts );

/* Query the elements and return the result to $enemy_query */
$enemy_query = new WP_Query( $args );

/* If there's any enemy found */
if ( $enemy_query->have_posts() ) :
    
    /* Loop through all enemy items */
    while ( $enemy_query->have_posts() ) : $enemy_query->the_post();
        /* Retrieve fields */
        $picture = get_field("picture");
        $name = get_the_title();
        $alias = get_field("alias_name");
        $secret = get_field("dark_secret");
        /* ... */

        /* Print html and variables to the html document */
        echo "<div class='enemy-slider slider'";
        echo "<div class='enemy-name'">" . $name . "</div>";
        echo "<div class='enemy-alias'>" . $alias . "</div>";
        /* ... */
        echo "</div>";
    endwhile;

    echo "</div>";

endif;
```

Figure 96 Portion of code showing the creation of enemy slider
From here we can keep building the different popups and sliders that the project has. The styles and JavaScript involved for creating the slides isn’t dealt in the document since it’s not the main target of the document to explain the intrinsic details of the coding languages used.
4.6 Phase 6: Responsive design

In this section we give a brief explanation on how to adapt the HTML elements to the screen size.

CSS3 introduced media queries, by which we can test different conditions that resolve to true or false. The only used media queries on the project is `screen` and `max-height`.

```css
/* Above media queries goes the default website styles */
.slider-title {
  font-size: 50px;
}

@media screen and (max-height: 768px) {
  /* The styles inside this media query will only be computed for screens below 769px */
  .slider-title {
    font-size: 30px;
  }
}
```

Figure 98 CSS example showing media queries functionalities
5 Benchmarks

Since one of the client’s requisites was that the website should run smoothly on PCs and high-end mobile devices, a benchmark is needed to verify we met the initial goals.

5.1 Framerate benchmarks

The devices tested range from low to high-end devices and they represent most of the client’s target devices.

The results for the framerate benchmarks were the following ones:

<table>
<thead>
<tr>
<th>Device</th>
<th>Game viewport</th>
<th>Average framerate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop PC, i5-4670k, GTX 760</td>
<td>1903x965</td>
<td>60fps</td>
</tr>
<tr>
<td>Huawei Ascend P7</td>
<td>598x280</td>
<td>26fps</td>
</tr>
<tr>
<td>Mac Book Pro 13”</td>
<td>1440x801</td>
<td>50fps</td>
</tr>
<tr>
<td>iPhone 6</td>
<td>559x375</td>
<td>28fps</td>
</tr>
<tr>
<td>Low-end PC</td>
<td>1903x955</td>
<td>16fps</td>
</tr>
</tbody>
</table>

*Table 14 Framerate benchmarks*

As we can see in the previous table, almost all devices got acceptable framerates apart from the low end PC. The low framerates on this device can be explained because it doesn’t have any dedicated graphics card, so the graphical power is very limited.

In general, we are happy with the framerates benchmark results.

5.2 Loading time benchmark

The results of the following benchmark shouldn’t be taken too seriously since loading time depend to lots of factors, Wifi signal power, 3G signal power, server CPU load, net load, etc. But we can guess from the results if world initialization is too high for a website, thus not meeting common usability guidelines.

The results of loading times were the following ones:
<table>
<thead>
<tr>
<th>Device</th>
<th>World initialized time</th>
<th>Total load time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop PC, i5-4670k, GTX 760 (50 Mb/s Ethernet)</td>
<td>4.5s</td>
<td>18.7s</td>
</tr>
<tr>
<td>Huawei Ascend P7 WiFi</td>
<td>2.9s</td>
<td>9.2s</td>
</tr>
<tr>
<td>Huawei Ascend P7 3G</td>
<td>5.5s</td>
<td>33.3s</td>
</tr>
<tr>
<td>Mac Book Pro 13” (100 Mb/s Ethernet)</td>
<td>5.4s</td>
<td>19.5s</td>
</tr>
<tr>
<td>iPhone 6 3G</td>
<td>10.1s</td>
<td>28s</td>
</tr>
</tbody>
</table>

*Table 15 Loading time benchmarks*

The value that is important to us is the world initialization time, which in most cases, is low for loading a videogame like this one. We are happy with the results since we set up a goal of 10 seconds load time.
6 Conclusion

The project has been a positive experience for me. I’ve always loved videogame development and I’ve been involved in some projects, but this is my first real game that I’ve completed. Since I would want to end working in the videogame industry, I’m sure this project will help me getting introduced into this world.

It has also been very satisfactory for me mixing all kinds of different technologies and concepts and ending up with a working result. The knowledge I’ve acquired during the degree has helped me planning the project, developing it, and delivering the project in time.

Thanks to this project I’ve discovered WebGL and all the power it has. I’m so excited with this technology that I’m planning on making my portfolio using Babylon.js, one of the tools I compared Phaser with.

Bee Square, the client, expressed his satisfaction when they saw the website running.

6.1 Future improvements

There are some optional features that were not included in the first version of the website. The following features could be included in the second version of the website:

- Sheldon jump: the jumping action was left behind because it was causing too many bugs. A little more study on the physics system and we should get it working.
- Website achievements: a fancy feature to mimic videogames achievements and getting rewards for it.
- App game ranking: to show a panel with the real-time ranking of players on the world.
- Add states: adding states and animations for multiple Sheldon actions (picking coins, etc)
- Phaser editor: create a Phaser editor for visually create world assets instead of typing the world by hand.
- Change Sheldon spritesheet when he changes the clothes on the closet.
7 Figures

Figure 1 Starting loading screen .......................................................... 12
Figure 2 Game's first stage ................................................................. 12
Figure 3 Game's second stage ............................................................. 13
Figure 4 Enemy slider that appears when clicking on an enemy ........... 13
Figure 5 Game's third stage ............................................................... 14
Figure 6 Popup that lets change Sheldon clothes ............................. 14
Figure 7 Game's fourth stage ............................................................. 15
Figure 8 Power-up slider that appears when clicking on the chest .......... 15
Figure 9 Game's fifth stage ............................................................... 16
Figure 10 Boosts slider that appears when clicking on the chest ........... 16
Figure 11 Game's seventh stage ......................................................... 17
Figure 12 Game's seventh and last stage ........................................... 17
Figure 13 RAD cycle ........................................................................ 24
Figure 14 Gantt diagram ................................................................. 25
Figure 15 JavaScript and jQuery code comparison ............................... 31
Figure 16 Wordpress root directory .................................................. 41
Figure 17 wp-content directory example ........................................... 42
Figure 18 Wordpress default database structure ................................ 43
Figure 19 Wordpress page template decision system .......................... 45
Figure 20 Custom theme directory structure example ......................... 46
Figure 21 Browser WebGL compatibility ........................................... 47
Figure 22 JavaScript code for creating a new Phaser game ................. 48
Figure 23 Phaser game attribute modules ........................................ 49
Figure 24 JavaScript of the creation of a Phaser game state .................. 50
Figure 25 Run Sheldon website assets directory ................................ 52
Figure 26 Last stage with slopes on the ground ................................ 54
Figure 27 Physics engine system with collision bodies ....................... 54
Figure 28 Path movement system .................................................. 55
Figure 29 Game scaling system ........................................................ 56
Figure 30 Displaying a portion of the world system ............................ 57
Figure 31 Loading screen ............................................................... 59
Figure 32 Game flowchart ............................................................... 63
Figure 33 Summary of assets/scenes/runsheldon-map.json................................................. 63
Figure 34 JavaScript code used to decode a JSON file......................................................... 64
Figure 35 partial definition of a stages element in the scene JSON ........................................ 64
Figure 36 Resources array of a stage in scene JSON............................................................... 65
Figure 37 JavaScript and Phaser.js code to load resource files............................................... 65
Figure 38 UV example............................................................................................................ 66
Figure 39 Partial definition of a sprite in the scene JSON....................................................... 66
Figure 40 JavaScript and Phaser code for creating a sprite.................................................. 67
Figure 41 Tilesprite partial definition in scene JSON file..................................................... 67
Figure 42 Javascript and Phaser code for creating a tile sprite.............................................. 68
Figure 43Coin group partial definition in the scene JSON file............................................... 69
Figure 44 JavaScript and Phaser code for creating a coin..................................................... 69
Figure 45 Sound definition in the scene JSON file................................................................. 70
Figure 46 Javascript and Phaser code for creating a sound element..................................... 70
Figure 47 Picture showing the particle emitter generating snowflakes.................................. 71
Figure 48 Emitter definition on the scene JSON file............................................................. 71
Figure 49 JavaScipt and Phaser code for creating a particle emitter..................................... 72
Figure 50 Sheldon’s spritesheet with all of its animation frames......................................... 73
Figure 51 Animations definition on the scene JSON file...................................................... 74
Figure 52 JavaScript and Phaser code initialization of physics............................................. 75
Figure 53 physics definition on a sprite on the scene JSON file........................................... 75
Figure 54 The last figure definition in action......................................................................... 76
Figure 55 Circle option on the physics definition................................................................. 76
Figure 56 The circle physics option in action........................................................................ 76
Figure 57 Physics shapes attributes on scene JSON file..................................................... 77
Figure 58 Las figure in action............................................................................................... 77
Figure 59 Camera visible portion of the world....................................................................... 78
Figure 60 Camera update flowchart.................................................................................... 79
Figure 61 Phaser input initialization..................................................................................... 80
Figure 62 Update key inputs JavaScript code........................................................................ 80
Figure 63 Mousewheel callback........................................................................................... 80
Figure 64 Touch input system............................................................................................... 81
Figure 65 Sheldon states........................................................................................................ 82
Figure 66 Sheldon update flowchart.................................................................................... 84
Figure 67 Sheldon movement JavaScript and Phaser code.................................................. 85
Figure 68 Update tilesprites JavaScript and Phaser code..................................................... 86
Figure 69 Scrolling textures in action.................................................................................... 86
Figure 70 Parallax effect example ................................................................. 87
Figure 71 Parallax attribute in sprites and tilesprites ................................. 87
Figure 72 Parallax example 1 ................................................................. 88
Figure 73 Parallax example 2 ................................................................. 89
Figure 74 Parallax effect comparison between two different camera positions 90
Figure 75 Picture displaying where parallax takes place ............................ 91
Figure 76 Game display before scale adaptation ...................................... 91
Figure 77 Game display after scale adaptation ....................................... 92
Figure 78 Textures level of detail folders ................................................ 93
Figure 79 Texture LOD system important variables .............................. 93
Figure 80 Texture quality decision flowchart ....................................... 94
Figure 81 Texture url generation .......................................................... 94
Figure 82 Tile sprites bug before fixing its scale .................................... 95
Figure 83 Code used to fix tile sprite scale .......................................... 95
Figure 84 Asynchronous loading flowchart .......................................... 97
Figure 85 Stage loading bar feedback .................................................. 98
Figure 86 WordPress backend for mywordressite.net ........................... 99
Figure 87 WordPress backend panel .................................................... 100
Figure 88 Pages backend panel ........................................................... 100
Figure 89 Page backend panel ............................................................. 101
Figure 90 WordPress PHP for registering a new post type .................... 101
Figure 91 New post type panel ............................................................. 102
Figure 92 Site’s database wp_posts table .......................................... 103
Figure 93 Advanced Custom Field, Enemies group admin panel .............. 104
Figure 94 Enemy with backend panel with added custom fields .............. 105
Figure 95 WordPress example page template ..................................... 106
Figure 96 Portion of code showing the creation of enemy slider .............. 107
Figure 97 Enemy frontend slider ......................................................... 108
Figure 98 CSS example showing media queries functionalities .......... 109
8 Tables

Table 1 Tasks duration ................................................................. 28
Table 2 Human resources budget............................................. 35
Table 3 Hardware budget ............................................................. 35
Table 4 Software budget ............................................................. 36
Table 5 General costs budget ..................................................... 36
Table 6 Risk budget ................................................................. 37
Table 7 Total budget ................................................................. 37
Table 8 WordPress Joomla and Drupal comparison ..................... 40
Table 9 Web graphic framework comparison ............................. 48
Table 10 Movement system comparison ..................................... 55
Table 11 Game adaptation to screen systems comparison ............. 57
Table 12 Texture LOD size and comparison .............................. 96
Table 13 World initialization improvement table ........................ 98
Table 14 Framerate benchmarks ............................................... 110
Table 15 Loading time benchmarks ......................................... 111