UNIVERSITAT DE BARCELONA

**Treball final de grau**

**GRAU DE MATEMÀTIQUES**

**Facultat de Matemàtiques i Informàtica**
**Universitat de Barcelona**

---

# Simple Space: 2D game design and development

---

**Autor: Oleksandr Danylenko**

**Director:** Dra. Beatriz Remeseiro
**Realitzat a:** Departament de Matemàtiques i Informàtica

**Barcelona,** 21 de juny de 2017

# Acknowledgments

First of all, I would like to thank Beatriz for helping me with this project.

Then I would like to thank my Mom and Grandmom for not letting me starve to death during the development.

And thank you Feliu for providing me a valuable feedback during the game development.

# Abstract

Nowadays, video-game industry is one of the biggest and fastest growing around, providing jobs to many people and having a very big market. From the 1950s until now, many different video-game genres were created suiting different people tastes. Among them, Shoot 'em up games mainly consist of a player trying to complete the game while evading different obstacles which can include enemies, environmental objects, or different types of projectiles.

During my whole life, I have spent countless hours playing a great variety of video-games. I was always interested in developing one by myself, which led me to consider making this project: a 2D video-game in shoot 'em up genre, which I have named Simple Space.

By making Simple Space, I have learned how the video-games can be designed and developed by a single person. This process includes the usage of different tools for graphic and audio design such as Inkscape, Audacity and Bfxr, as well as the Unity engine for combining them into a seamlessly working game.

Using Inkscape, I have designed a variety of graphic objects for this game which include different space-ships made of simple geometrical figures, a big and complicated Boss ship and many miscellaneous objects used in level design and user interface. Regarding Audacity and Bfxr, they were used to create sound effects that give the player an audio feedback when its ship gets damaged or an enemy ships explode. With respect to Unity, I have used it to design and develop a single working level with a careful crafted game logic, different particle effects and a complete menu system. For this purpose, I have utilized many different game creation tools that Unity offers in combination with the graphical and sound effects created before.

Additionally, I have used a special software called Color Oracle to take into an account how people with colorblindness disability will be able to enjoy the game. I have also considered the ability to play the game on older computer systems by benchmarking and testing the game with tools such as MSI Afterburner. As a result, I have obtained a fully working game, gained a lot of experience and feel more motivated than ever before in continuing making games.

## Keywords

Game development, Game design, Unity 2D, Graphic design, Audio design, Inkscape, Audacity, Bfxr, Shoot ’Em Up genre.

# Contents


**1 Introduction** **1**

1.1 Video-games: industry and market . . . . . . . . . . . . . . . . . . . . 1

1.2 Shoot 'Em Up . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Motivation and objectives . . . . . . . . . . . . . . . . . . . . . . . 5

1.4 Hardware and software . . . . . . . . . . . . . . . . . . . . . . . . . 8

1.5 Memory structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**2 Graphic and sound design** **9**

2.1 Graphic design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1.1 Workspace . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.1.2 Player . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.1.3 Enemy1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.1.4 Enemy3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.1.5 Enemy4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.1.6 Enemy5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.1.7 Boss . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.1.8 Miscellaneous objects . . . . . . . . . . . . . . . . . . . . 17

2.2 Sound design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.2.1 Bfxr . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

2.2.2 Audacity . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

2.2.3 Sound effects development . . . . . . . . . . . . . . . . . 21

2.2.4 Music . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

**3 Game design and development** **23**

3.1 Unity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.2 Project creation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.3 Camera . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.4 Particle systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.5 Shaders and materials . . . . . . . . . . . . . . . . . . . . . . . . . 30

3.6 Game controller . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

3.7 Moving Background . . . 33
3.8 Base classes . . . 34
3.9 Projectiles . . . 34
3.10 Lasers . . . 35
3.11 Player . . . 37
3.12 Enemies . . . 37
3.13 Miscellaneous objects . . . 40
3.14 Boss . . . 41
3.15 User interface . . . 42
3.16 Level design . . . 45

**4 Accessibility and colorblindness 49**
4.1 Types of colorblindness . . . 49
4.2 Accessibility: implicit vs explicit . . . 50
4.3 Color palette . . . 51

**5 Methodology 53**
5.1 Development life cycle . . . 53
5.2 Task management . . . 54

**6 Results 57**
6.1 Graphic design . . . 57
6.2 Colorblindness testing . . . 57
6.3 Game performance . . . 58
6.4 Game controls . . . 59

**7 Conclusions 63**
7.1 Future work . . . 64

**A Diagrams 65**
A.1 Class diagrams . . . 65
A.2 Flowchart diagrams . . . 70

**B Task calendar 77**
B.1 Sprint 1: 21/02/2017 – 07/03/2017 . . . 77
B.2 Sprint 2: 07/03/2017 – 21/03/2017 . . . 77
B.3 Sprint 3: 21/03/2017 – 04/04/2017 . . . 78
B.4 Sprint 4: 04/04/2017 – 18/04/2017 . . . 78
B.5 Sprint 5: 18/04/2017 – 02/05/2017 . . . 79
B.6 Sprint 6: 02/05/2017 – 16/05/2017 . . . 80
B.7 Sprint 7: 16/05/2017 – 30/05/2017 . . . 81

B.8 Sprint 8: 30/05/2017 - 20/06/2017 . . . . . . . . . . 81

**Bibliography** **83**

# List of Figures

1.1 Global Games Market from 2012 to 2017 . . . . . . . . . . . . . . . . . 2
1.2 Age and gender of game players . . . . . . . . . . . . . . . . . . . . . . 2
1.3 Spacewar! on a PDP-1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 Examples of shoot ′em up games . . . . . . . . . . . . . . . . . . . . . 4
1.5 Ingame screenshots from some CAVE games . . . . . . . . . . . . . . . 6
1.6 Screen shoots from some games of 2000s . . . . . . . . . . . . . . . . . 7

2.1 Interface of the Inkscape editor . . . . . . . . . . . . . . . . . . . . . . 10
2.2 Design of the workspace inside Inkscape . . . . . . . . . . . . . . . . . 11
2.3 Design of the Player . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.4 Design of the Enemy1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.5 Design of the Enemy3 . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.6 Design of the Enemy4 . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.7 Design of the Enemy5 . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.8 Final design of the Boss . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.9 Miscellaneous graphic objects used in this project . . . . . . . . . . . 18
2.10 Interface of the Bfxr software . . . . . . . . . . . . . . . . . . . . . . . 19
2.11 Interface of the Audacity software . . . . . . . . . . . . . . . . . . . . 20
2.12 Graphic representation of different sound effects . . . . . . . . . . . . 22

3.1 Difference between orthographic and perspective projections . . . . 24
3.2 Different 2D colliders inside Unity editor . . . . . . . . . . . . . . . . 25
3.3 Common computer graphic pipeline . . . . . . . . . . . . . . . . . . . 26
3.4 Unity default project creation dialog . . . . . . . . . . . . . . . . . . . 27
3.5 Unity editor with an empty scene . . . . . . . . . . . . . . . . . . . . 28
3.6 Camera setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.7 Code of the event for object destruction . . . . . . . . . . . . . . . . . 29
3.8 StarField movement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
3.9 Zoomed in Explosion Particle at the span of one second . . . . . . . 30
3.10 Boss spawn particle animation at the span of one second . . . . . . . 31

3.11 Boss laser charge animation time line . . . 31
3.12 Zoomed out view of all the objects in the Moving Background . . . 33
3.13 Movement code for Moving Background . . . 33
3.14 BaseShip properties . . . 34
3.15 Shortened version of the code used for blinking objects . . . 35
3.16 Different types of projectiles in Unity . . . 35
3.17 Game screen shot showing different projectiles . . . 36
3.18 Different types of lasers in the game . . . 36
3.19 Player object controlled by the user when playing the game . . . 38
3.20 Player properties and values inside Unity editor . . . 38
3.21 Enemies inside the game . . . 40
3.22 Player going around different Laser Barriers . . . 41
3.23 Game screen shot showing the Boss . . . 42
3.24 Navigation path of actions in the Main menu . . . 44
3.25 Different game menus . . . 45
3.26 Interface with the current user information . . . 45
3.27 Level1 sliced, parts from 1 to 4 . . . 47
3.28 Level1 sliced, parts from 5 to 9 . . . 48

4.1 Color palette used in this project . . . 51

5.1 Development cycle of a game object . . . 54
5.2 Screen capture showing part of my Kanban on GitHub website . . . 55

6.1 Example of screen shots as seen in case of colorblindess . . . 58
6.2 Default Xbox 360 game pad controls . . . 60
6.3 Default keyboard controls . . . 60

A.1 Class diagram of the BaseShip class and its children classes . . . 66
A.2 Class diagram of the BaseProjectile class and its children classes . . . 67
A.3 Class diagrama of GameController and BossController . . . 68
A.4 Class diagram with the miscellaneous classes . . . 69
A.5 Flowchart diagram of the Main menu . . . 71
A.6 Flowchart diagram of the Player Projectile . . . 72
A.7 Flowchart diagram of the Enemy Projectile . . . 73
A.8 Flowchart diagram of the Player . . . 74
A.9 Flowchart diagram of the Enemy . . . 74
A.10 Flowchart diagram of the Game Controller . . . 75

# List of Tables

3.1 Folder structure inside the Assets folder created by Unity . . . . . . 27
3.2 Balancing values of both Player and Enemies . . . . . . . . . . . . . . 43

6.1 Game performance on different hardware configurations . . . . . . . 59

# Chapter 1

# Introduction

Video-game industry is one of the fastest growing industries, allowing people from all around the world to play video-games of many different genres. There is a video-game for every person: from first person shooters to racing games, from fighting games to sports games, from strategy games to puzzle games, etc. Among them, this project is focused on shoot 'em up genre, which is one of my favorite ones and whose origins are explained in this chapter.

## 1.1 Video-games: industry and market

Nowadays, the video-game industry is one of the biggest industries around and it still grows very fast. It provides many jobs to people with different talents and produces a big amount of video-games each year, thus resulting in big revenues for the companies that make them. These games are played worldwide by millions of people on a variety of different platforms, including Microsoft Xbox One, Sony Playstation 4, Nintendo Switch, different mobile platforms and, of course, the biggest and best PCs.

From 2012, the global game market revenue went up over the 30% with different mobile platforms, taking up to the 34% of its market share in 2017 as can be seen in Figure 1.1. Due to its continued growth, software developing companies started making more applications for game development and, at the present time, there are many different tools that allow a single person or a very small team of people to successfully develop and sale video-games, which accelerates the market growth of this industry even more.

Games are played by people of all ages and genders alike, as depicted in the charts included in Figure 1.2. This fact provides an easy way to connect parents and children, or different people around the globe. Taking into account this information, it is easy to see why game development is a good industry to get into.

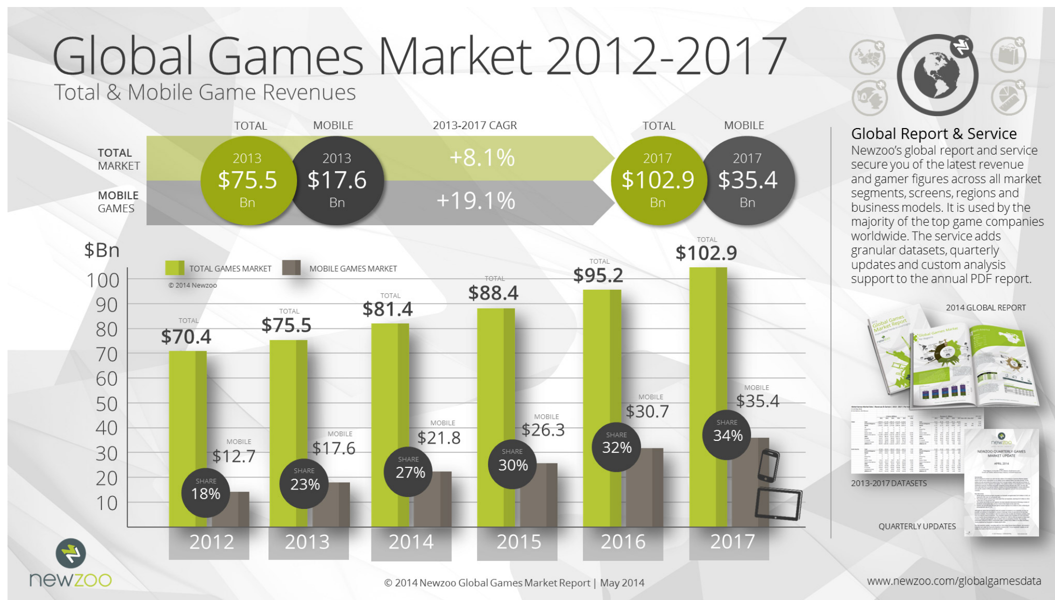



Figure 1.1: Global Games Market from 2012 to 2017 [18]

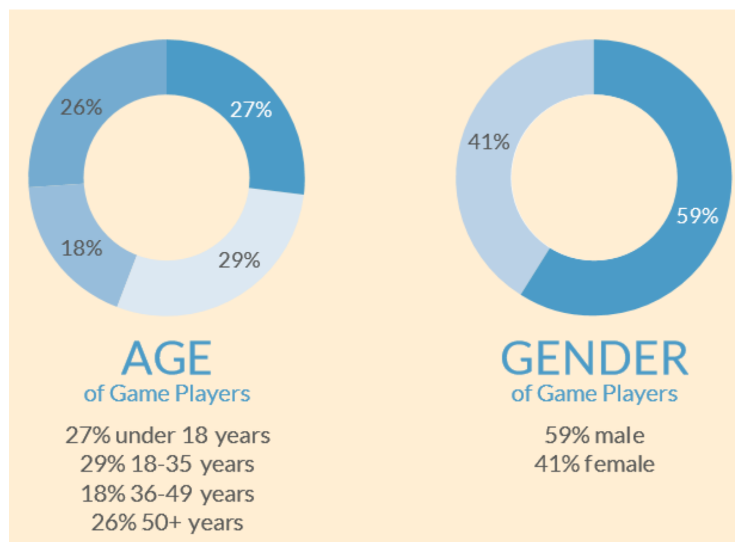



Figure 1.2: Age and gender of game players [7]

## 1.2 Shoot ’Em Up

Shoot ’Em Up, also known as shmup, is a video-game genre in which the player controlled character engages in a battle against hordes of enemies while at the same time having to dodge all incoming attacks which can consist of different

types of projectiles, map objects and enemies themselves [4, 16].

Origins of shoot 'em up genre can be traced back to one of the earliest computer games ever created, **Spacewar!** (see Figure 1.3). Developed in a computer lab at the Campus of the Massachusetts Institute of Technology (MIT) in 1961, the game was running on a DEC's PDP-1 computer. It consisted of two players controlling spaceships, and trying to destroy another one by shooting projectiles and to avoid being sucked by the gravitational field of the sun.

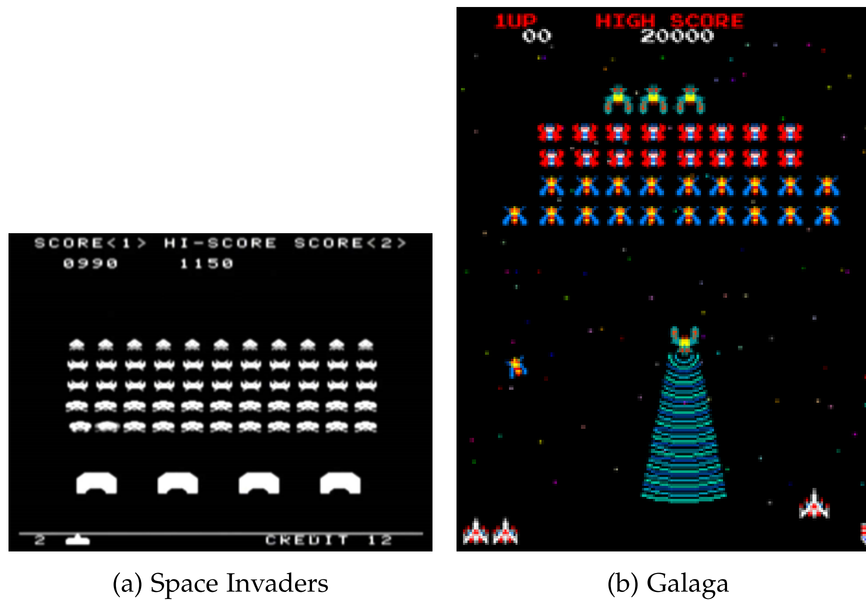
Figure 1.3: Spacewar! on a PDP-1

Almost 20 years later, in 1978, the Japanese company Taito launched **Space Invaders** (see Figure 1.4a), which became a worldwide blockbuster and is considered the true father of the shoot 'em up genre. The game created a high score functionality that saved the players' scores inciting them to play more and trying to up it in the following sessions. In addition, it introduced the concept of giving multiple lives to players and it was also the first game that made the enemies shoot at the player. It had a top down perspective and consisted of a player trying to shoot horde of aliens advancing from top to bottom.

After seeing its success, more and more companies started to release shoot 'em up games. One of them was **Galaxian** 1979, released by Namco. It introduced mechanics such as enemies that try to kill the player in a kamikaze style. Its sequel **Galaga** 1981 (see Figure 1.4b) became one of the most popular shoot 'em up games at the time by having more detailed alien design, bonus rounds and a very elaborated audio. This game sprung many sequels which continue to be released at the present day in a variety of different platforms.

At the same year of Galaxian release, there was released **Asteroids** by Atari, Inc, another iconic shoot 'em up which became as famous as Space Invaders. Its game-play had a triangular ship controlled by the player, able to rotate both left and right and to shat at the direction its facing. The objective was to destroy as many asteroids as possible while evading them. This simple game sold more than 70.000 copies, and it is still considered as one of the best games of all times.

1980 was a start of golden age of games in shoot 'em up genre. Many new




(a) Space Invaders (b) Galaga

Figure 1.4: Examples of shoot 'em up games

games were released every year, that had different mechanics and so they began to spawn many new sub-genres [3, 13]. Some of most significant games of the 1980s included:

- **Defender** 1981: it was one of the most difficult games of its time.
- **Scramble** 1981: it was the first to offer multiple distinct levels.
- **Tempest** 1981[3]: it was the first to incorporate a 3D perspective into the genre, and it is considered to be the game that inspired a sub-genre of rail shooters.
- **Zaxxon** 1982: it was the first isometric scrolling shooter.
- **Xenovious** 1982: it became the most influential vertically scrolling shooter.
- **Time Pilot** 1982: it became the first multi-directional shooter.
- **Robotron 2084** 1982: it was a very frenetic multi-direction shooter, inspiring bullet hell genre.
- **Thunder Force** 1983: it allowed the player to scroll in any direction.
- **Hover Attack** 1984: it later inspired the famous **Bangai-O** 1999.

- **Gradius** 1985: it allowed the player to have different weapons, and it introduced the need for the player to memorize the levels in order to achieve a greater chance of success, which became very important part of this genre.
- **Commando** 1985: it became one of the first games to use human characters in this genre.
- **R-Type** 1987: it created a system which had flying power-ups that changed in colors to represent different weapons. This mechanism also became a very important part of this genre.
- **Contra** 1987: it introduced a cooperative game-play between two players and a multi-dimensional aiming.

Games like **Commando** and **Contra** inspired arcade hits that include **Ikari Warriors**, **Gun Smoke** and **Metal Slug**.

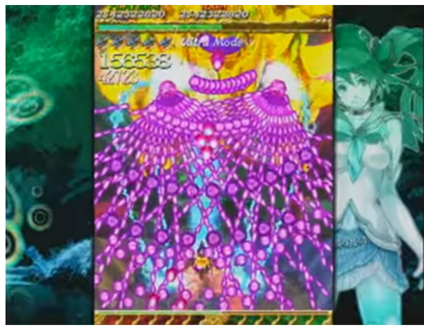
At the start of 1990, the major mechanics of the genre were already established and shoot 'em up genre became the most popular action genre for arcade games. At this time, there were released very popular shooters such as **Raiden** 1990, including many sequels of **Gradius** and **R-Type** [20]. The 1990s were also the era in which bullet hell shooters became popular, with titles like **Toaplan's** and **Batsugun** 1993. In this sub-genre, the player had to evade so much bullets that they filled almost the entire screen. After **Toaplan's** ankruptcy in 1994, its employees created **CAVE** (Computer Art Visual Entertainment) and, until today, they are the biggest and most influential game developers of bullet hell shoot 'em up games.

Some of the CAVE's most popular titles include: DonPachi 1985, Espgaluda 2003, Mushihimesama Futari 2006 (see Figure 1.5a), Deathsmiles 2007 (see Figure 1.5b), DoDonPachi Resurrection 2008 (see Figure 1.5c), and Akai Katana 2010 (see Figure 1.5d).

From the 2000s to the present, there are continuous releases of shoot 'em up games, including: Ikaruga 2001 (see Figure 1.6a), Jets 'n' Guns 2004 [9], Triggerheart Exelica 2006, Syder Arcade 2012 (see Figure 1.6c), Sine Mora 2012 (see Figure 1.6d), Geometry Wars series 2005 (see Figure 1.6b).

## 1.3 Motivation and objectives

During my whole life, I have spent countless hours playing video-games. As I grew older, I often found myself analyzing the games and started asking different questions like: how did they do that?, could I do it by myself?, could I do it better? Little by little, I started developing mini games and found out that game development is an infinitely big world that provides many exciting things to learn.



(a) Mushihimesama Futari

(b) Deathsmiles

(c) DoDonPachi Resurrection

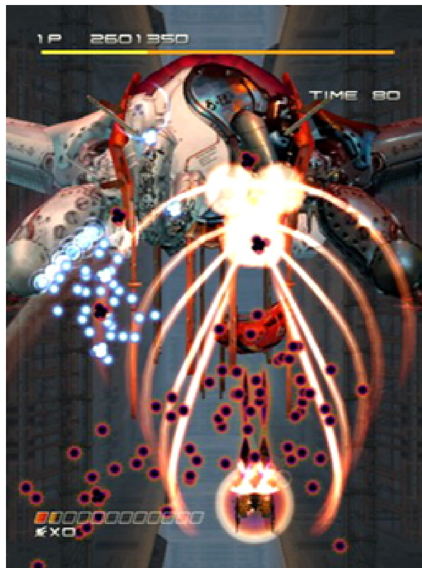
(d) Akai Katana

Figure 1.5: Ingame screenshots from some CAVE games

Doing it as my TFG provided me a great opportunity to face it in a more serious way. Otherwise, I would probably delay it for a long time which would surely be spent working at completely different projects.

Therefore, the main objective of this project was to entirely design and develop a 2D video game in shoot 'em up genre, entitled Simple Space, while at the same time learning different disciplines and tools needed for game development. In order to be able to achieve this main objective, I have split the project into a set of different targets from which the specific tasks were later created. These general targets include:

- Creating a player controlled character and giving it the ability to shoot different things inside the game level.
- Making one playable level with different enemies, map objects and a final




(a) Ikaruga

(b) Geometry Wars: Retro Evolved 2

(c) Syder Arcade

(d) Sine Mora

Figure 1.6: Screen shoots from some games of 2000s

boss character, and give some of them the ability to shoot back at the player or do some other kind of damage.

- Making different sound effects to be heard when the user is shooting or doing some other things.
- Creating visual effects such as explosions.
- Making the level have parallax scroll like effect for movement.
- Making the game be able to be played by colorblind people.
- Making the game style have neon light like characters.
- Including some type of high score functionality.
- Creating some type of menu system.

- Learning as many things as possible while developing the game, which personally it is the most important part of this project.

## 1.4 Hardware and software

The hardware and software elements used in the realization of this work are detailed as follows:

- CPU: Intel Core i7-5930K 3.50 GHz, RAM: 16GB 2800MHz DDR4, GPU: NVIDIA GTX 980 TI
- Windows 10 Pro: operating system.
- Unity 5.51f1: game development engine.
- Microsoft Visual Studio Community 2015: integrated development environment.
- Inkscape 0.92.1: vector graphics editor.
- Bfxr: basic sound effect generator.
- Audacity 2.1.3: cross-platform audio software for multi-track recording and editing.
- TexStudio 2.11.0: integrated environment for LaTeX documents.

## 1.5 Memory structure

This document is organized in seven chapters. The first one is the current introduction. The second chapter explains in great detail how the graphic and sound objects were designed, as well as the tools used for this purpose. The third chapter explains how the game logic and the game objects were designed and developed. The fourth chapter entails the importance of taking into account colorblindness issues in video-game design and how to use it in this project. The fifth chapter explains the development methodology used for this project and some details of task management. The sixth chapter includes the results that I have obtained after completing the project. Finally, the last chapter presents the conclusions and the future work.

# Chapter 2

# Graphic and sound design

This chapter presents detailed explanations about how the graphic and audio elements of this game were designed. It also includes a brief description of different programs and tools used for this purpose, in addition to step by step actions taken when designing graphic and audio elements as well as their uses inside the game.

## 2.1 Graphic design

One of the main targets in the design part of this project was to avoid low quality images when scaling them to different sizes. For this reason, vector graphic images were considered and, particularly, the format SVG (Scalable Vector Graphics). Since Unity, the game development engine used, does not support SVG, the images were designed using SVG format and then exported to PNG (Portable Network Graphics).

Inkscape [11] is a free open source professional vector graphics editor for Windows, Mac OS X and Linux; and it is the software chosen to design the SVG images. Figure 2.1 illustrates the interface of the Inskcape editor that, like any other editing software, has the menu items at the top, the general workspace in which the images are shown at the center, different editing tools following explained at the left, and specific options for each tool at the right.

The Inkscape tools used in this project are detailed as follows:

**Select and transform:** It allows to select and transform objects (change size, rotation, skew).

**Edit path by nodes:** It allows to create, modify and delete the points (nodes) of an object (path).

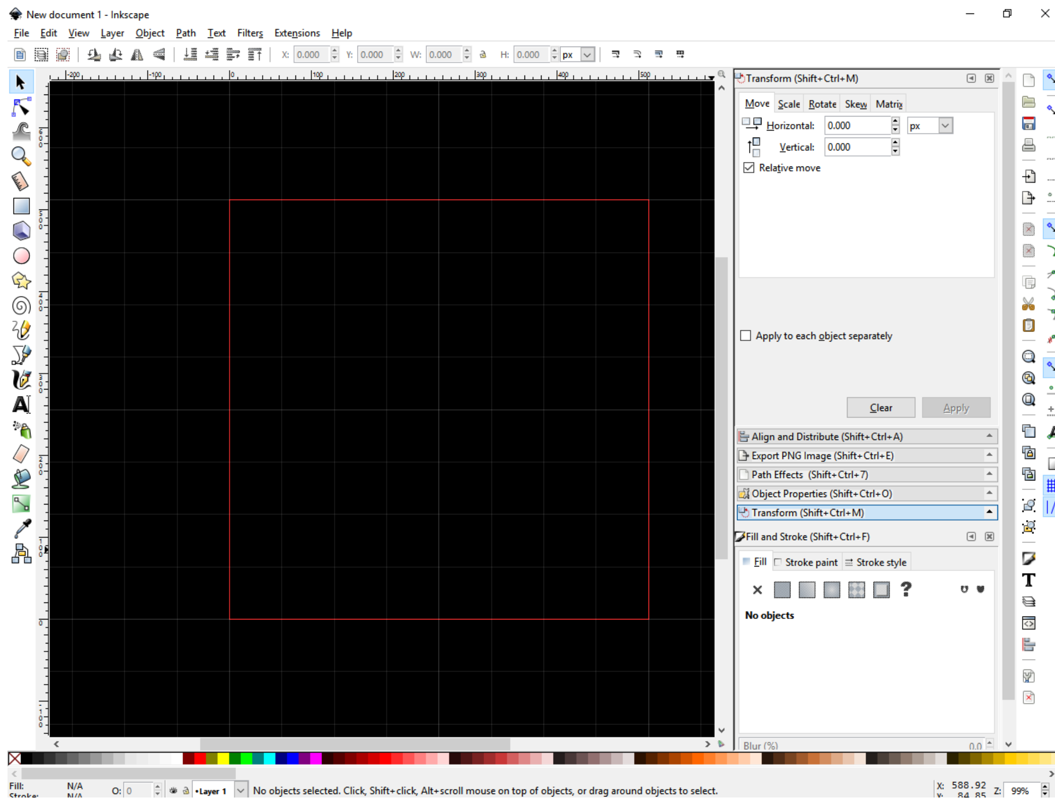



Figure 2.1: Interface of the Inkscape editor

**Create rectangles and squares:** It allows to create rectangles and squares of different sizes.

**Create circles, ellipses and arcs:** It allows to create circles, ellipses and arcs of different sizes.

**Create stars and ploygons:** It allows to create stars and polygons of different sizes with different number of corners.

**Align and distribute:** It allows to order and align objects by different axes and sides.

**Fill and stroke:** It allows to fill objects with different colors, including opacity and blurring.

**Export:** It allows to export an image in different image formats, including PNG.

In the following sections it is explained how the different elements of the game were designed using Inkscape. These elements include the workspace, the player, a total of four enemies, the boss, and other miscellaneous objects.

### 2.1.1 Workspace

The first element created was the workspace, in which all of our graphic elements are included. Figure 2.2 depicts the workspace, which has the following characteristics:

- Dimensions: $512 \times 512$ pixels.
- Rectangular grid: $64 \times 64$ pixels.
- Major grid line every four lines.
- Background of black color.

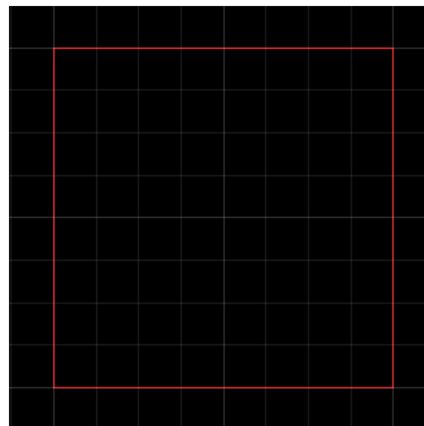


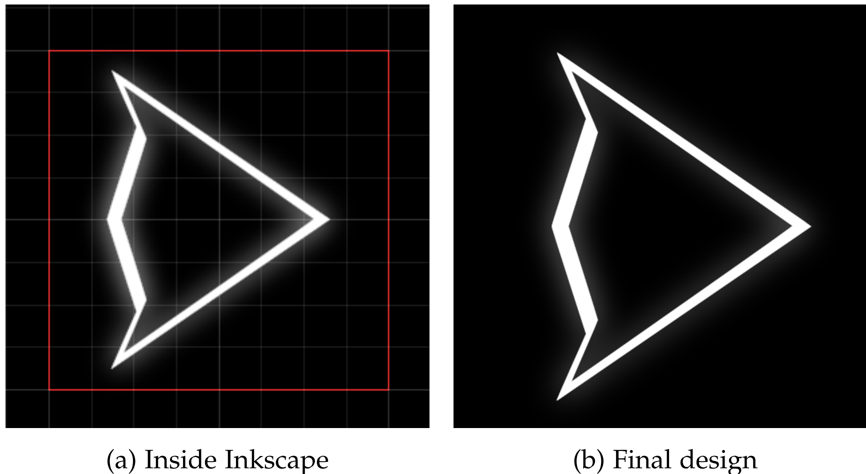
Figure 2.2: Design of the workspace inside Inkscape

### 2.1.2 Player

The first object designed was the Player, which will be controlled by the user inside the game. Figure 2.3 shows the Player, designed according to the next steps:

1. Draw a basic design on piece of paper to use it as model.
2. Create a basic triangle shape and rotate it to face right.
3. Align the basic shape to the center.
4. Edit the basic shape path by adding more nodes and adjust their positions.
5. Duplicate the basic shape and scale it down to create the inner part.
6. Join the two shapes.

7. Fill the resulting shape with white color, and make the middle transparent.
8. Duplicate the object and apply a 10% of blur on it to achieve a glow effect.
9. Combine the glow and the shape into one object.
10. Export the final object to PNG format and test different sizes in Unity to find the one that fits best.



(a) Inside Inkscape (b) Final design

Figure 2.3: Design of the Player

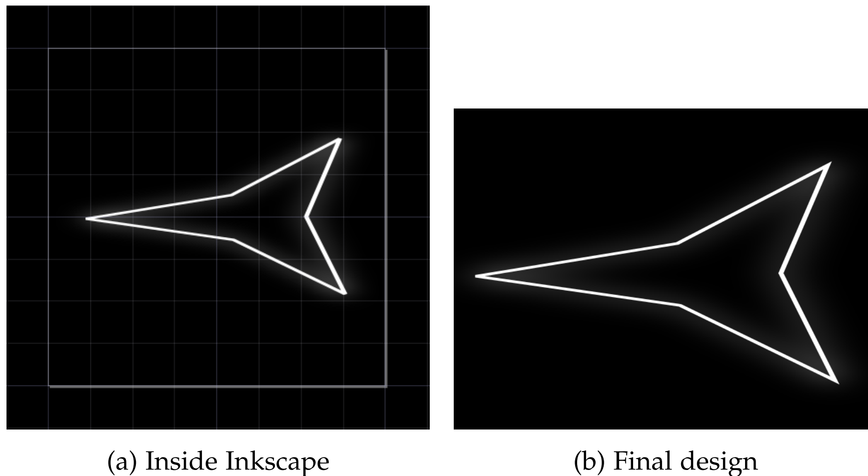
### 2.1.3 Enemy1

The second object designed was the Enemy1, which shoots projectiles at the Player. Figure 2.4 shows the Enemy1, designed according to the next steps:

1. Draw a basic design on piece of paper to use it as model.
2. Create a basic triangle shape and rotate it to face left.
3. Align the basic shape to the center.
4. Edit the basic shape path by adding more nodes and adjust their positions.
5. Make the stroke (outline) width 5px.
6. Fill the shape stroke with white color, and make the middle transparent.
7. Duplicate the object and apply a 10% of blur on it to achieve a glow effect.

8. Combine the glow and the shape into one object.

9. Export the final object to PNG format and test it with different sizes in Unity to find the one that fits best.



(a) Inside Inkscape (b) Final design

Figure 2.4: Design of the Enemy1

### 2.1.4 Enemy3

The third object designed was the Enemy3, which shoots a projectile from each side and rotates around its center while moving. Figure 2.5 shows the Enemy3, designed according to the next steps:

1. Draw a basic design on piece of paper to use it as model.

2. Create a basic circle shape.

3. Align the circle shape to the center.

4. Create a basic triangle shape.

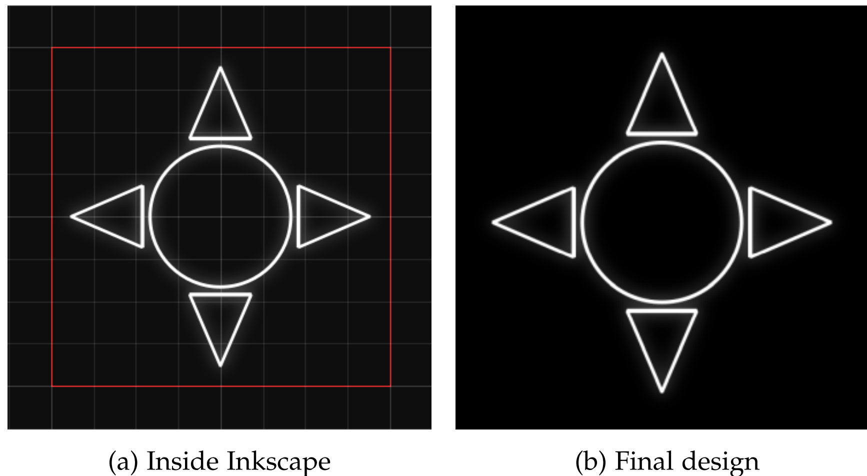
5. Resize the triangle shape.

6. Duplicate the triangle shape three times.

7. Rotate each triangle shape to face four different directions (top, down, left, right).

8. Align each triangle on their respective axis.

9. Group all the objects into a main object.
10. Fill the main object stroke with white color, give it 1px width, and make its middle transparent.
11. Duplicate the object and apply a 5% of blur on it to achieve a glow effect.
12. Combine the glow and the shape into one object.
13. Export the final object to PNG format and test it with different sizes in Unity to find the one that fits best.



(a) Inside Inkscape (b) Final design

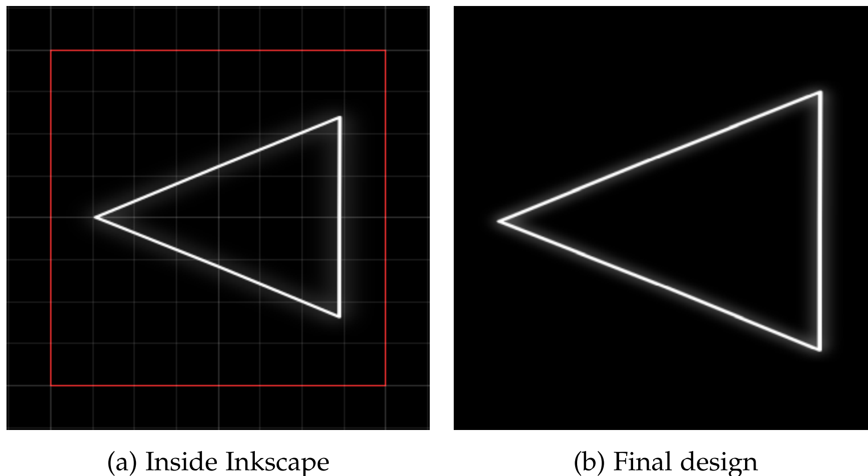
Figure 2.5: Design of the Enemy3

### 2.1.5 Enemy4

The fourth object designed was the Enemy4, which is difficult to destroy due to its small size and its fast movement speed. Figure 2.6 shows the Enemy4, designed according to the next steps:

1. Draw a basic design on piece of paper to use it as model.
2. Create a basic triangle shape.
3. Align the triangle shape to the center.
4. Resize the triangle shape.
5. Fill the triangle stroke with white color, give it 5px width, and make its middle transparent.

6. Duplicate the object and apply a 10% of blur on it to achieve a glow effect.
7. Combine the glow and the shape into one object.
8. Export the final object to PNG format and test it with different sizes in Unity to find the one that fits best.



(a) Inside Inkscape (b) Final design

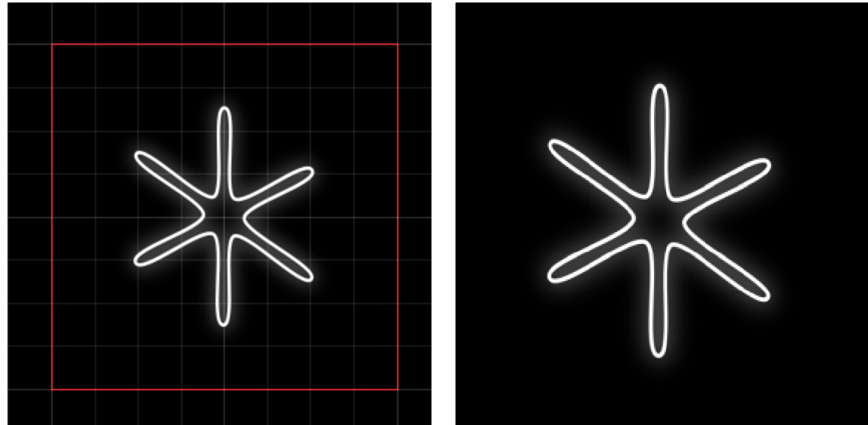
Figure 2.6: Design of the Enemy4

### 2.1.6 Enemy5

The fifth object designed was the Enemy5, which will pursue the Player to death. Figure 2.7 shows the Enemy5, designed according to the next steps:

1. Draw a basic design on piece of paper to use it as model.
2. Create a basic star shape.
3. Align the star shape to the center.
4. Resize the star shape.
5. Make the star have six rounded corners.
6. Fill the star stroke with white color, give it 5px width, and make its middle transparent.
7. Duplicate the object and apply a 10% of blur on it to achieve a glow effect.
8. Combine the glow and the shape into one object.

9. Export the final object to PNG format and test it with different sizes in Unity to find the one that fits best.



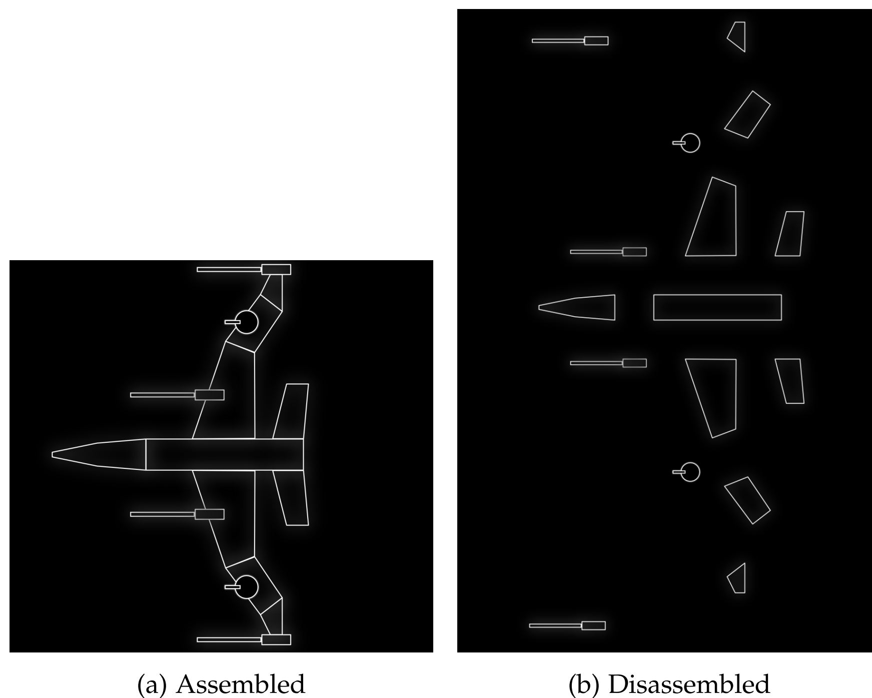
(a) Inside Inkscape (b) Final design

Figure 2.7: Design of the Enemy5

### 2.1.7 Boss

The next object designed was the Boss, which appears at the end of the level and needs to be destroyed in order to complete it. Figure 2.8 shows the Boss, designed according to a large list of steps. For the sake of simplicity, only the main steps are following detailed:

1. Draw a basic design on piece of paper to use it as model.
2. Create the sixteen different parts of the ship, including the body, the front, the four wings, the four laser canons, and two projectile canons.
3. Add a glow effect to each part.
4. Disable the transparency in some parts of two of the four laser canons and the two projectile canons, in order to make them appear to be mounted on the top of the wings.
5. Generate two images: one in assembled state (see Figure 2.8a), and the other one disassembled (see Figure 2.8b). The first one will serve as a model to assemble the second one part by part in Unity.



(a) Assembled (b) Disassembled

Figure 2.8: Final design of the Boss

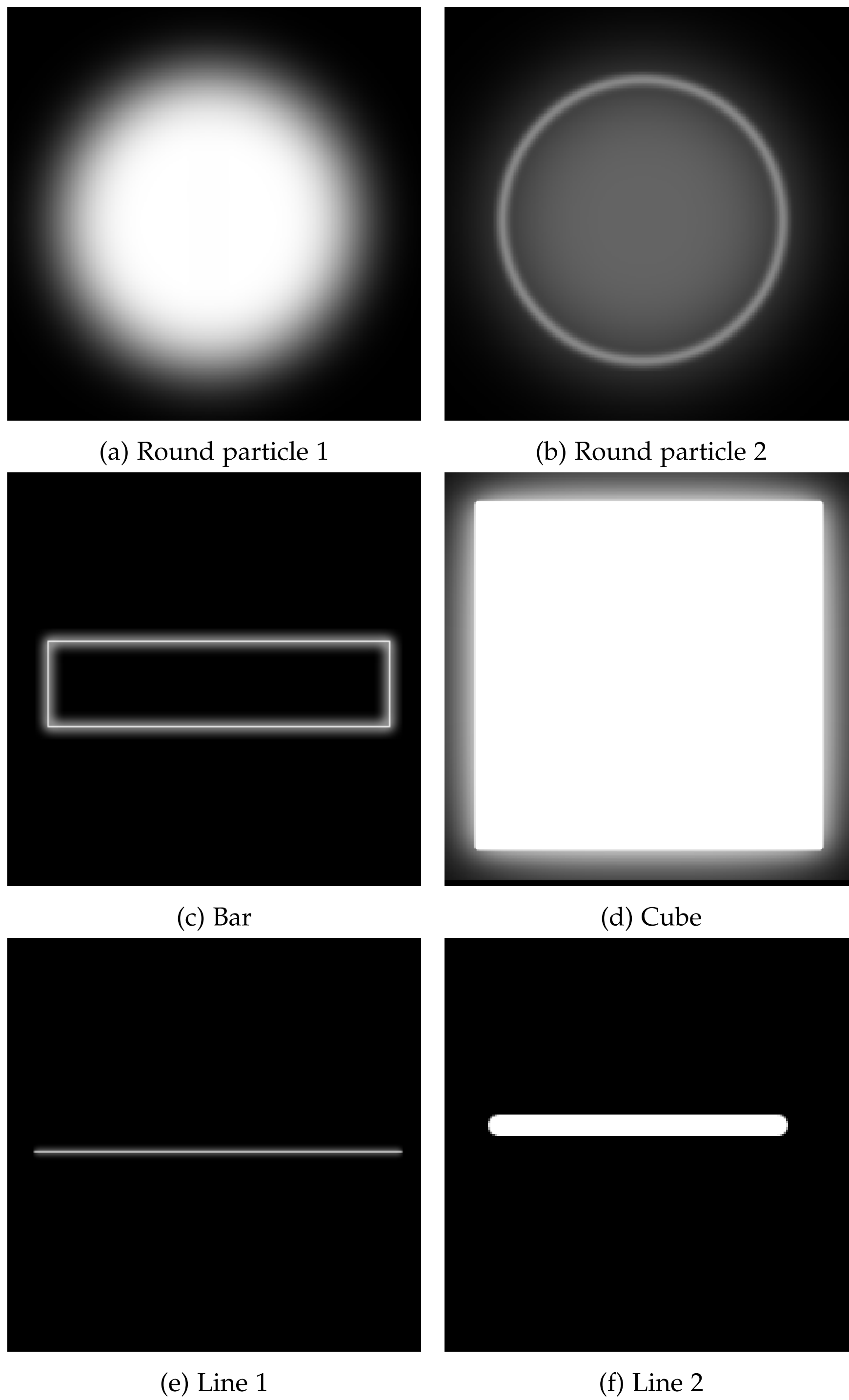
### 2.1.8 Miscellaneous objects

Besides the Player and the Enemies, I have also designed a few more simple objects that are used in different parts of the project. These objects are illustrated in Figure 2.9, and subsequently presented:

- Two round particles: used in projectiles and explosion particles.
- Bar: used in different menu elements.
- Cube: used in different menu elements.
- Two lines: used in lasers and user interface elements.

## 2.2 Sound design

As the game needs some sound effects, I have used two pieces of software to make them by myself: Bfxr and Audacity. In particular, I have created a variety of sound effects with them, as explained in the following sections.



(a) Round particle 1
(b) Round particle 2
(c) Bar
(d) Cube
(e) Line 1
(f) Line 2

Figure 2.9: Miscellaneous graphic objects used in this project

### 2.2.1 Bfxr

Bfxr [10] is a free-ware, open-source sound effects generator for game developers. It provides a set of templates to generate sounds for video games, and it also allows to modify them by playing around with different options. In addition, the sound effects can be exported to WAVE (Waveform Audio File Format). Figure 2.10 shows the interface of Bfxr, which is split in three parts. The left one has some buttons to generate random sounds with predefined effects, following presented. The center part has all the editing options for each sound. And the right part shows a graphical representation of the sound and includes the common options such as copy, paste, save and export.

Figure 2.10: Interface of the Bfxr software

### 2.2.2 Audacity

Audacity [2] is a free, open source, cross-platform audio software for multi-track recording and editing. It is available for Windows, Mac OS X and Linux. Figure 2.11 shows the interface of Audacity, which at the top shows the typical options menu, below it there are buttons to control the sound reproduction and recording, and a list of input and output sound devices. At the center, there is a workspace that allows simultaneous editing and mixing of various sound effects.

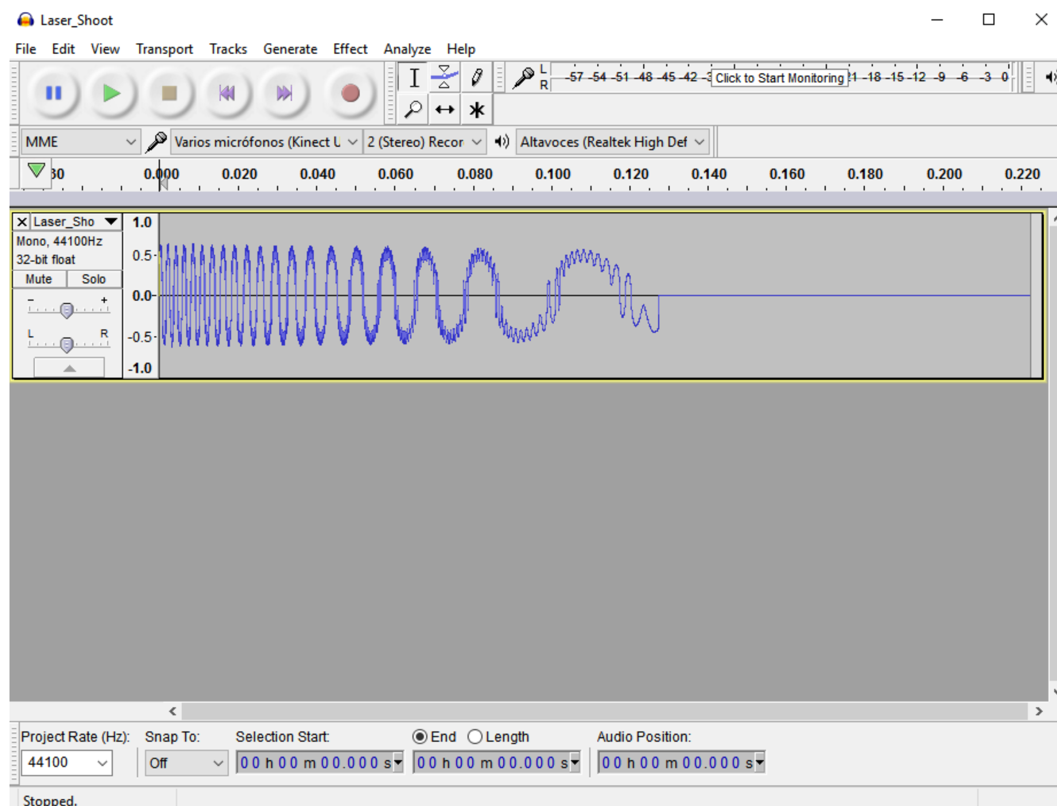



Figure 2.11: Interface of the Audacity software

Audacity has a generate tool to create different types of sounds, including the following ones:

- Chirp: a short, sharp, high-pitched sound.
- DTMF tones: like those produced by a keypad on a telephone.
- White, pink and brownish noises: three types of noise that allow to mask different sounds.

- Silence: a straight line wave.

These sounds can also have different wave types, such as sine, square or sawtooth. Audacity also provides a variety of different effects and filters that can be applied to a sound. The ones most used in this project are:

- Amplify a selected audio, i.e. increase or decrease its volume.
- Change the pitch, the speed and the tempo of a selected audio.
- Echo a selected audio, i.e. repeat it many times by gradually softening it.
- Fade in and Fade out a sound, i.e. gradually change its amplitude from the start to the end making it louder or softer.
- Repeat a sound a number of times specified.
- High and Low pass frequency filters, that allow to increase or decrease the amplitude of a sound below or above a cutoff frequency specified.

### 2.2.3 Sound effects development

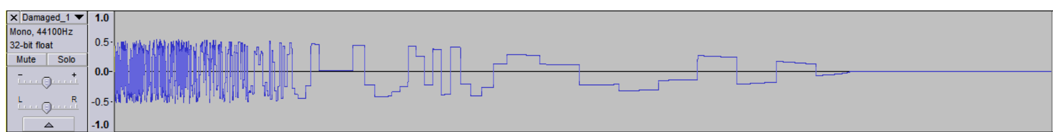
All the sound effects used in this game were made by combining Bfxr and Audacity. The process has been carried out according to one of the following manners, depending on the particular sound: (1) generate the sound, and then adjust it using only Bfxr; (2) create an empty sound, and then design it using a variety of different filters and tools provided by Audacity; (3) generate the sound using Bfxr, and then modify it using Audacity; and (4) record a real life sound, and then create a game sound from it using Audacity.

A total of six sounds were made with these tools (see Figure 2.12): Damaged_1, Damaged_2, Explosion_1, LaserBeamCharge_1, Powerup, ProjectileShoot_1. These sounds are used in some of the following places: Player shooting projectile or getting damaged by an enemy, or vice versa, Enemy exploding and Boss charging lasers.

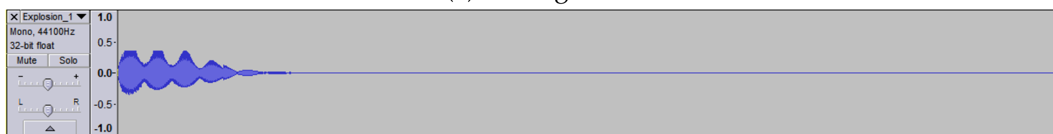
### 2.2.4 Music

I have a Space Music pack that I bought in 2015 from GameDev Market[1], a marketplace for high quality, affordable game assets handcrafted by talented creators around the world. Although I have included all the songs in the project, only two of them are used: one for the Main theme, which is played in the background during the normal level progression; and other for the Boss theme, which starts to be played when the Boss appears on the screen.

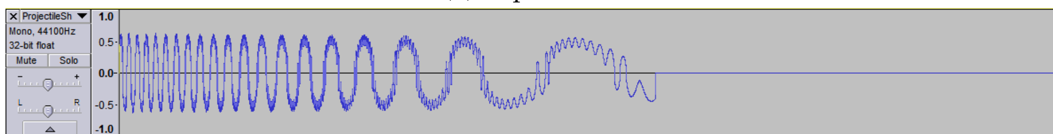
[1] www.gamedevmarket.net




(a) Damaged_1




(b) Explosion_1




(c) ProjectileShoot_1

Figure 2.12: Graphic representation of different sound effects

# Chapter 3

# Game design and development

This chapter explains how the game and its logic were designed and developed inside Unity, how each object was created and how they all interact with each other. It also includes a detailed explanation of how every tool provided by Unity was used. Appendix A includes some useful class and flowchart diagrams used for game development.

## 3.1 Unity

Unity [21] is a game engine (framework) used for cross-platform game development, created by Unity Technologies. It provides a set of advanced tools and API's to make game development easier for developers. The primary languages used in it are C# and JavaScript.

The Unity components used in this project are detailed as follows:

**GameObject:** It is a base class for all Unity entities in scene.

**Prefab:** It allows users to store GameObject with all attached components and properties, and it works like a template from which new object instances can be placed into a scene.

**Scene:** It contains objects of game, in such a way that each level inside a game is a scene.

**Camera:** It is used to show the contents of the scene, and it has two types of projections (see Figure 3.1): orthographic and perspective. In the first one, objects do not need to have depth and they seem flat. This projection is the most used in 2D games. Regarding the perspective projection, it shows the world in the same manner that humans see it. In this case, objects have

depth which gives us the ability to easily differentiate them and correctly judge their distances from the observer. This type of projection is the most commonly used in 3D games. As an example, given two objects of the same size but at different distances from the observer, in the perspective projection the closest object seems bigger than the other one, whilst in the orthographic projection they seem to have the same size no matter the distance.

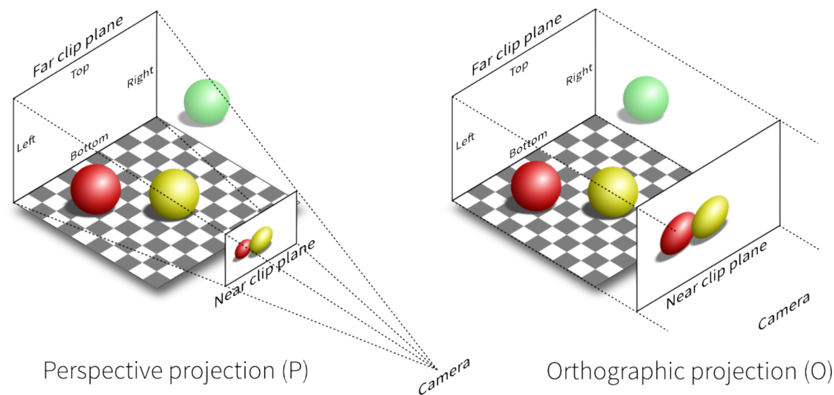



Figure 3.1: Difference between orthographic and perspective projections

**Transform:** It determines the position, the rotation, and the scale of each object in the scene. Note that every GameObject has a Transform component.

**RigidBody and RigidBody2D:** It is attached to a GameObject, and puts it to under the control of the Unity physics engine, making the object to be able to be affected by gravity and controlled using different forces like for example gravity. It is also used by Collider and Collider2D classes for detecting collisions between different objects.

**Collider and Collider2D:** They are used to detect collisions between different GameObjects, that have a RigidBody or Rigidbody2D component attached to them. Note that a GameObject can have multiple colliders attached to it for advanced functionalities. Figure 3.2 shows some examples.

**Script:** It is a component used to control the game logic by code. Scripts in Unity can be written using C# or JavaScript programming languages.

**Texture:** It is a bitmap image, that can be in different formats, such as PNG, BMP or JPG.

**Sprite:** It is 2D graphic object equivalent to a 3D model in 3D environment, but simpler since a Sprite is just a texture. A single Sprite can also contain

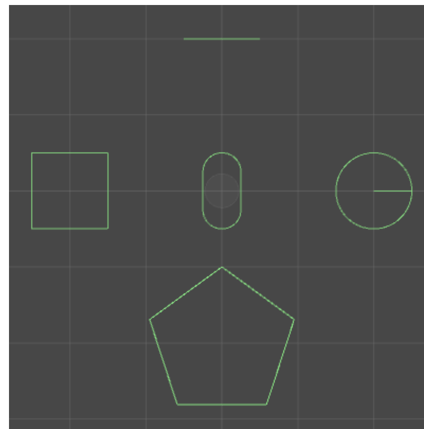


Figure 3.2: Different 2D colliders inside Unity editor

different parts of the same object that can be assembled in Unity scene, and used for animating it.

**SpriteRenderer:** Every Sprite needs a SpriteRenderer object to be able to render it into a scene. SpriteRenderer also provides a special API for Sprite control, allowing to change its color and split it in multiple parts.

**Canvas:** It is an object that represents the area in which all user-interface elements should be placed, such as game menus or relevant information for the player.

**LineRenderer:** It is used to create and draw a line between two or more points in 2D or 3D space.

**Particle:** It is a small object displayed and controlled by a ParticleSystem. It can be either a 2D Sprite or a 3D object.

**ParticleSystem:** It allows to simulate different graphic effects such as fire, liquids or explosions. By means of different APIs available, it is possible to create different effects by combining multitude of Particle objects and controlling them in different ways.

**Material:** It defines how the lighting interacts with the surface of an object. It can be used to simulate different types of surfaces, such as wood or plastic.

**Shader:** It is a script executed by a GPU that contains algorithms to calculate the color of each pixel of an object based on both lighting and material data. The most commonly used are Vertex and Fragment. They are usually stored in different files, although Unity combines them into a single one for simplicity reasons. The Vertex Shader receives vertex data as input, processes it, and

sends it to the graphic pipeline illustrated in Figure 3.3. After some steps, it sends the data to the Fragment Shader that processes it pixel by pixel, in order to compute the final color. The data can contain textures, normal maps, colors and other useful variables.

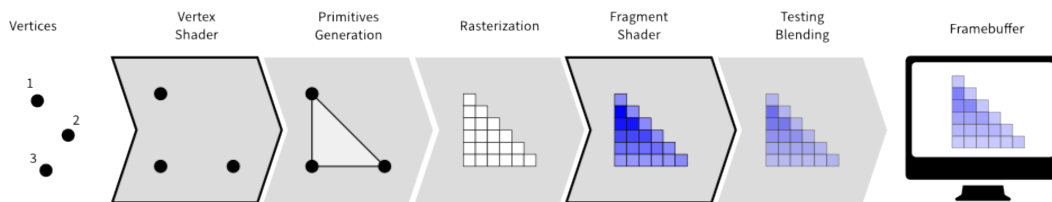



Figure 3.3: Common computer graphic pipeline

**AudioSource and AudioClip:** An AudioClip needs an AudioSource component to be played inside a scene. The AudioClip can be imported from many audio formats, such as MP3, OGG or WAW.

**AudioListener:** It is a virtual microphone inside a Scene that records sounds and plays them trough users' speakers. When a Camera is created, it has an AudioListener component attached to it. In order to play AudioClips, a Scene needs to have at least one AudioListener.

**Tag and Name:** In order to identify GameObjects in scripts, each object has a Tag or a Name.

**PlayerPrefs:** It is a static class that both stores and accesses players' preferences and data between game sessions. Their values can be saved into one of three types: integer, float and string. In code, they are retrieved by using dictionaries.

## 3.2 Project creation

Unity offers a very simple way of creating a new project (see Figure 3.4), and also allows to import standard assets included with it. As I have created all assets, except the music themes, I did not find the need to use them.

By default, Unity creates a directory known as Assets in which it stores all project development files. In order to be able to sort them, I have created a folder structure inside this folder, which can be seen in Table 3.1.

Additionally, Unity creates an empty scene with camera in it as illustrated in Figure 3.5. From this moment, the developing of a game can be started.

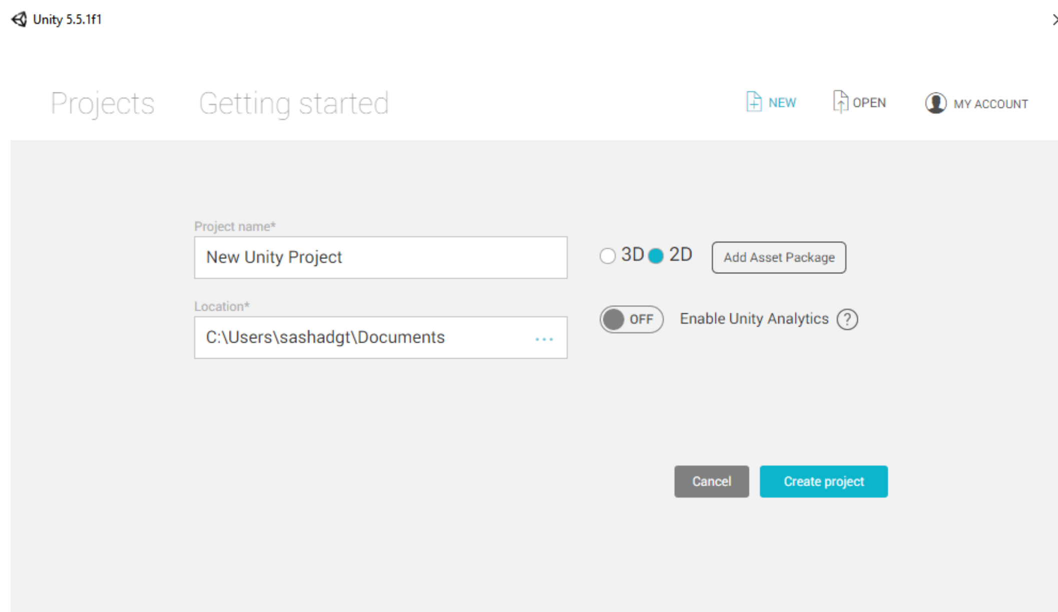



Figure 3.4: Unity default project creation dialog

Table 3.1: Folder structure inside the Assets folder created by Unity

| Folder name | Elements stored |
|---|---|
| Images | Image files that are not Sprites |
| Materials | Material data |
| Music | Music tracks |
| Prefabs | Prefab objects |
| Scenes | Scenes such as the Main menu and the Level1 |
| Scripts | Scripts used for game programming |
| Shaders | Shaders objects |
| Sounds | Sound effects |
| Sprites | Sprites creaded in Inkscape in PNG format |

## 3.3 Camera

As this is a full 2D game, I have setup the Camera to have an orthographic projection. After trying out different configurations, the Camera has a dimension of $28.6 \times 16$ Unity units and its background is solid black (see Figure 3.6) By default, the Camera includes the GUI Layer component to display the user interface, in addition to the AudioListener component to play sounds.

A Box Collider2D and a script DestroyByBoundary.cs were added to allow the

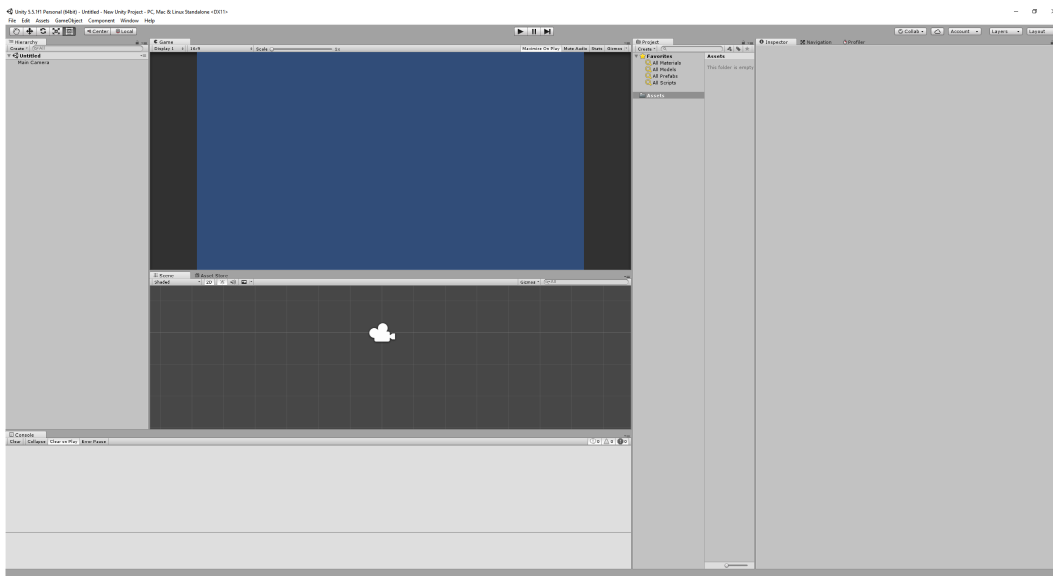


Figure 3.5: Unity editor with an empty scene

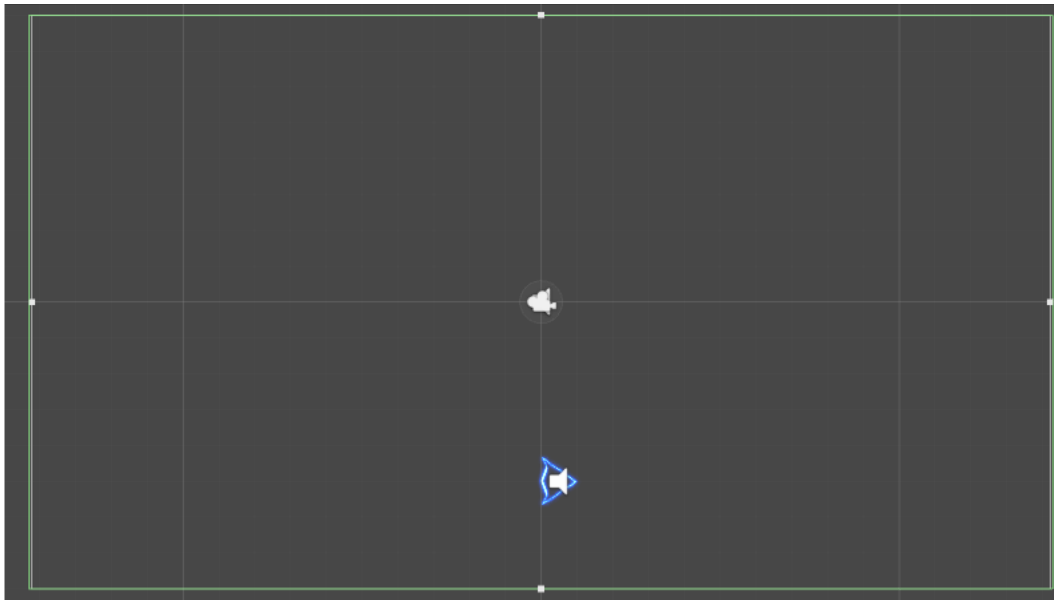


Figure 3.6: Camera setup

following functionality: Box Collider2D has a trigger option enabled which means that if something is in a contact with this collider, it produces different events. In this project, the event is OnTriggerExit2D (see Figure 3.7), used when an object exits the collider (i.e. camera space). Note also that the object gets destroyed when

quitting the camera space.

```
void OnTriggerExit2D(Collider2D collision)
{
        Destroy(collision.gameObject);
}
```

Figure 3.7: Code of the event for object destruction

## 3.4 Particle systems

In order to simulate various graphical effects, I have created several particle systems, each one with different functionalities and needs.

As a theme of the game is set up in the space, the first ParticleSystem created was StarField, that simulates many stars moving trough our level (see Figure 3.8). It has been done using the default Particle included in Unity. It has a white color and its duration is set to 10 seconds after which it simply loops itself from the start. The speed of the stars is set at 0.05 Unity units per second and there can be at most thousand of particles at the same time.

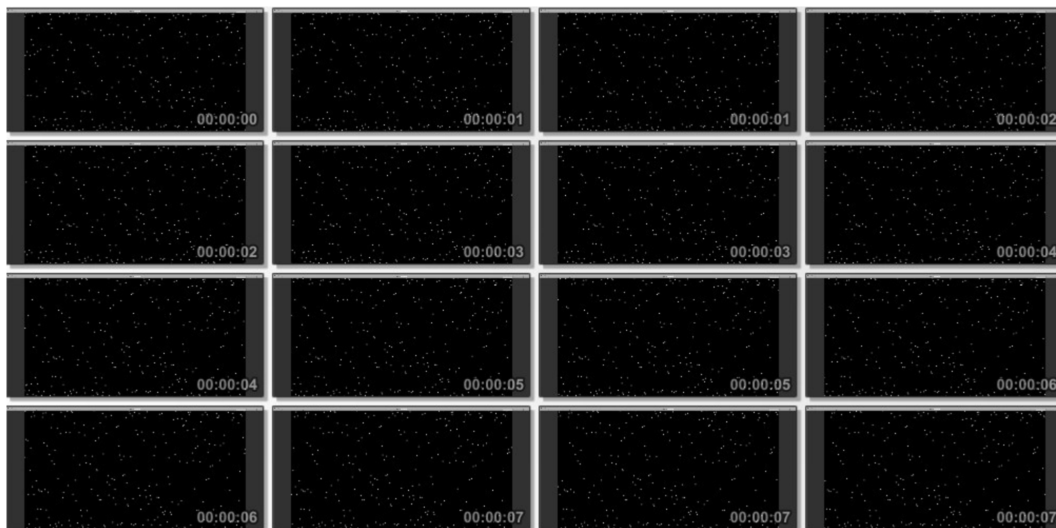



Figure 3.8: StarField movement

Both the Enemies and the Player produce an explosion when they are destroyed, and for that the Explosion ParticleSystem has been created. It creates a

random number of particles, each one having one of four different colors (see Figure 3.9): red, yellow, green and blue. All the particles shoot out outwards from a single center point, and can be bounced by camera borders. When the system is initialized, it plays an explosion sound effect with a duration of one second, which is auto-destroyed at the end.

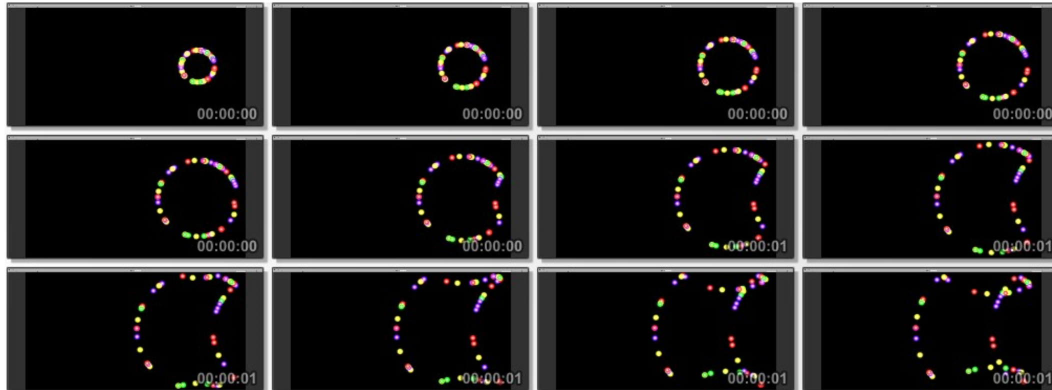



Figure 3.9: Zoomed in Explosion Particle at the span of one second

For the Boss element, I have created two different ParticleSystems. The first one is the spawn ParticleSystem (see Figure 3.10), and it is used when the Boss spawns. It has a duration of one second, and produces different cutting like effects of red color inside a circle. The other one is the Laser Particle System (see Figure 3.11), and it is used when the Boss shots its lasers. It has a four-second duration, and it simulates charging the laser canon with light particles, which go from white to red color.

## 3.5 Shaders and materials

From the beginning of this project, I wanted the objects in the game to have a neon glow effect. After exploring various possibilities, the process finally carried out to achieve this effect the following: when designing objects in Inkscape, I have blurred them to get a glow effect, and then I have developed a Shader in Unity that converts that glow effect into Neon lighting. For this task, I have downloaded the source code of the Unity standard Shader and made the following modifications to its Fragment shader:

1. Get the main Sprite.
2. Apply color to the main Sprite and get the colored one.
3. Average the RGB values of the colored Sprite to get the grayscale one.

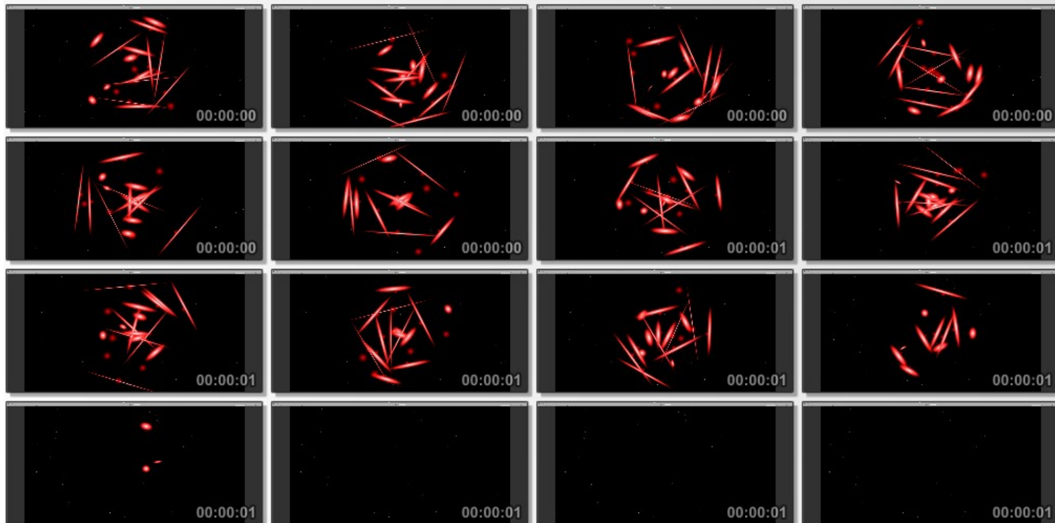



Figure 3.10: Boss spawn particle animation at the span of one second

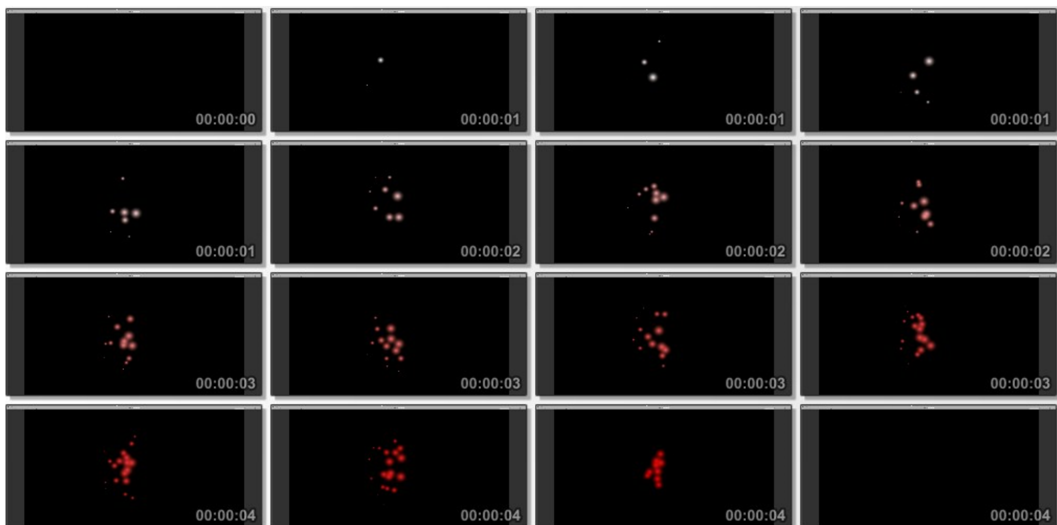



Figure 3.11: Boss laser charge animation time line

4. Compute the final Sprite as:

$$finalSprite = (colorSprite * 3 + graySprite * \alpha) * \alpha$$

where $\alpha$ is the alpha transparency value of the main Sprite.

I have made two versions of the Shader, which are Neon2D and Neon2DTexture. In order to apply a Shader to the object, a Material has to be created and then applied to the respective object. Notice that, in this project, the relationship between

Sprite Materials and Shaders is the following: a Sprite has one Material assigned to it, and this Material has one Shader assigned to it.

## 3.6 Game controller

GameController is an object that controls the general purpose of the game logic. In this project, I have created an object of this type using the GameController script. It has different functionalities, such as controlling game-states or updating user interface elements. All these functionalities are subsequently presented:

- Functions for controlling game-states:

  **Playing:** All necessary data is initialized. If this function is used after a pause, it continues from the point in which the game was paused.

  **Paused:** The execution of the game logic is paused and the Pause Menu is shown.

  **Lost:** The player is dead, and so the execution of the game logic is paused and the Lose Menu is shown.

  **Won:** The player has completed the level, and so the score is computed and the Win Menu is shown.

- Functions for controlling and updating user interface elements:

  **Score:** It shows the Player's score.

  **Player life:** It shows how much health the player has.

  **Game timer:** It shows the current game time.

- General game data and auxiliary functions:

  **Player:** A reference to the Player object, and its controlling script.

  **Music:** Both Main and Boss music themes, and the logic to control them.

  **Fixed enemy damage:** The damage done to Player by the Enemies. After a careful testing and some feedback received from different users, all the enemies and their projectiles (excluding lasers) have a fixed amount of damage, which is set at 35 hit points. The Player's health points are set to 100, so it takes three hits to be destroyed, which makes the game well-balanced in terms of difficulty.

  **Background speed:** Speed at which the Moving Background container object moves.

**Boss:** A reference to the Boss object in order to be able to spawn it at the end of the level.

**Highscore:** High-score logic using PlayerPrefs data.

**Audio volume:** Control of the general audio volume using PlayerPrefs data.

Most of the scripts need the GameController for different functionalities. For that reason, all of them have a function called InitGameController, in order to search the GameController inside the scene every time that an object is instanced. An error is produced if the GameController is not found.

## 3.7 Moving Background

Moving Background is an object that serves as a container for other objects inside the scene, as illustrated in Figure 3.12. Those objects include enemies, barriers, and different triggers. It is setup in a way that it is always moving from right to left at a specified speed, just like all the objects inside it.

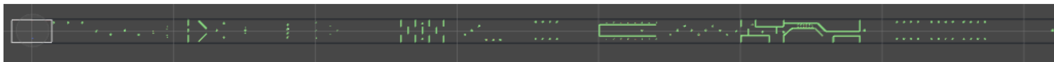


Figure 3.12: Zoomed out view of all the objects (highlighted in green) inside the Moving Background

The game is made in such a way that the camera is always still, and the things that are really moving are the background and the stars inside the StarField Particle System, thus creating an illusion of camera movement. When objects inside the Moving Background enter the camera space, they disconnect from it and begin to move by themselves using their unique logic. Figure 3.13 shows the code that controls Moving Background movement.

```
void Update () {
        Vector2 position = transform.position;
        speed = gameController.backgroundSpeed;
        position = new Vector2(position.x - speed
           * Time.deltaTime, position.y);
        transform.position = position;
}
```

Figure 3.13: Movement code for Moving Background

## 3.8 Base classes

Although Player and Enemies are different space ships, they share many properties. For this reason, I have created a parent class to use the inheritance provided by object oriented programming. Therefore, the BaseShip class was created, whose properties are shown in the code included in Figure 3.14.

```
public GameController gameController;
public GameEntityType type; //type of object
public float healthPoints; //current healthpoints
public float totalHealthPoints;
public float moveSpeed;
public float shootingRate;
public List<GameObject> canonList; //list of
   canons which shoot projectiles
public ParticleSystem explosion;  //particle that
   plays when object is destroyed
public bool blink = false;
public bool alive = true;
```

Figure 3.14: BaseShip properties

One of the particularities in this class is the Blink function (see Figure 3.15), which makes an object blink when its damaged in order to alert the user. This function is somewhat special because it uses a yield statement, which may make the code execute concurrently. Therefore, it is necessary to check if the object is alive before and inside of the function.

Similarly, another base class called BaseProjectile has been created. It is used as a parent class for different projectiles inside the game including both Player's and Enemies' projectiles.

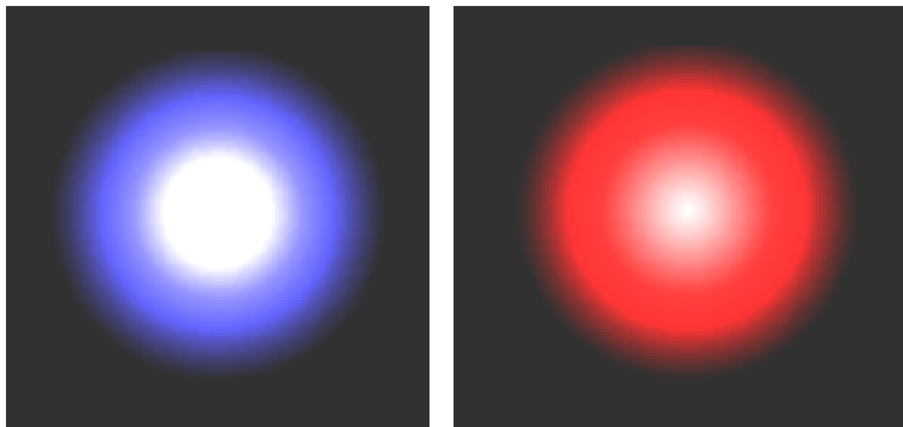
## 3.9 Projectiles

In this game, there are different types of projectiles, each one with its own unique logic and functionality, used by the Player or the Enemies. The Player and all its projectiles are blue (see Figure 3.16a), which is more user-friendly; whilst the Enemies and their projectiles are red (see Figure 3.16b), which seems more menacing and indicates danger. In the final version of the game, both Player's and Enemies' projectiles go straight. Regarding the Enemies' projectiles, they are used

```
if (alive && !blink){
        blink = true;
        Color oldColor =
           GetComponentInChildren<SpriteRenderer>().color;
        GetComponentInChildren<SpriteRenderer>().color =
           Color.black;
        yield return new WaitForSeconds(blinkTime);
        if (alive){
                GetComponentInChildren<SpriteRenderer>().color
                   = oldColor;
        }
        blink = false;
}
```

Figure 3.15: Shortened version of the code used for blinking objects

in different manners depending of the particular Enemy: some enemies shoot projectiles directly to the Player, whilst others are rotating and shoot a barrage of projectiles to many different directions. Figure 3.16 shows the different projectiles inside the game.



(a) Player projectile (b) Enemy projectile

Figure 3.16: Different types of projectiles in Unity

## 3.10 Lasers

Lasers are another way of destroying things inside the game. They are used by the Player as a secondary weapon, and by the Boss as one of the primary weapons

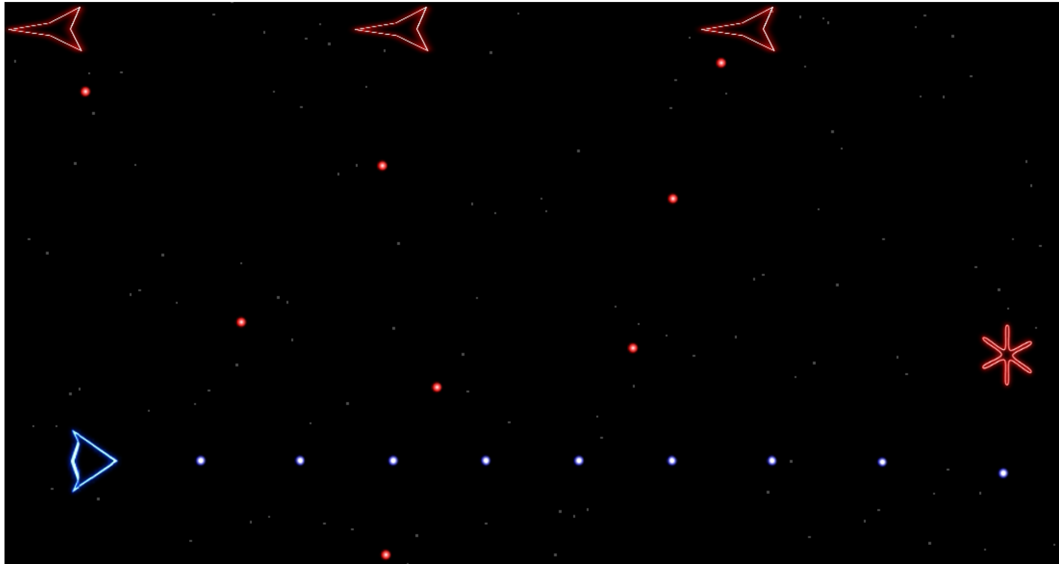
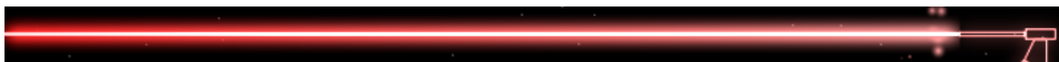


Figure 3.17: Game screen shot showing different projectiles

it has. They are made using the LineRenderer component combined with a script to control them. They have been created by means of a technique called Ray Casting, which casts a straight line (ray) at a desired direction and checks if it intersects with some objects. It has been programmed in such a way that, in case of intersection, it returns many useful values such as the object intersected or the position of intersection.

By default the ray is cast at an infinite distance and intersects with all objects in its path. In order to avoid unnecessary computations, I have programed it to only cast the ray within the camera limits. Once the ray is cast, a straight line is created with Line Renderer using both ray position and direction to draw it. As with other objects, the Player's laser is blue and the Boss' laser is red, as can be seen in Figure 3.18.



(a) Boss laser

(b) Player laser

Figure 3.18: Different types of lasers in the game

## 3.11 Player

The Player object (see Figure 3.19), which is the one controlled by the user when playing the game, can do very simple actions like shooting projectiles and lasers, as well as moving inside the camera space. Although it looks simple, it is composed of many different components which are listed bellow:

- A RigidBody2D of Kinematic type, which means that instead of using the Unity physics engine, it uses physics done by me. It only produces collisions if the Collider is set to have a Trigger enabled.
- A Circle Collider2D, with the same size that the Player Sprite.
- An AudioSource that contains the sound used when the Player is damaged.
- A SpriteRenderer component needed to display the object, with the following objects and properties:

  Sprite designed in Inkscape (see Section 2.1.2).

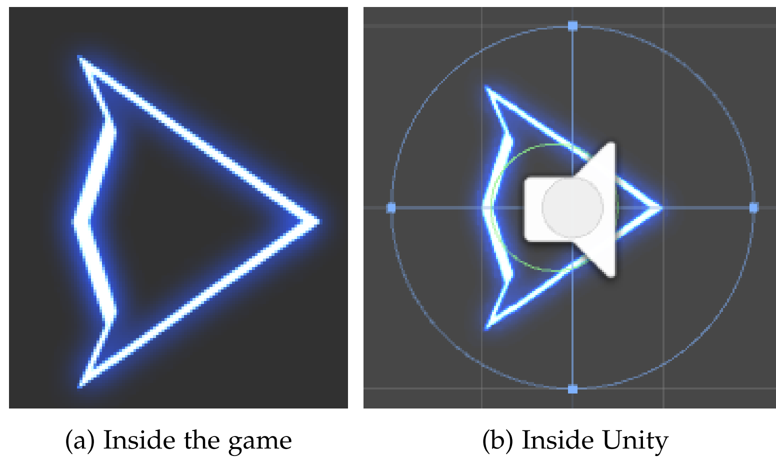
  Blue color specified in hexadecimal.

  Material with a Shader assigned to it.
- A central canon GameObject used for giving the starting position when spawning projectiles.
- A Laser canon used for shooting the laser described in Section 3.10.
- A Particle System used when the Player is destroyed to spawn an Explosion Particle System, described in Section 3.4.
- A PlayerController script that controls all the above logic (see Figure 3.20).

## 3.12 Enemies

All of the Enemies inside the game have the same components and share the same base logic, described as follows:

1. After starting the game, the Enemy object is placed inside the Moving Background object.
2. When the Enemy enters the camera space and collides with the Camera Collider, it disconnects from the Moving Background and starts using its own unique logic.



(a) Inside the game (b) Inside Unity

Figure 3.19: Player object controlled by the user when playing the game

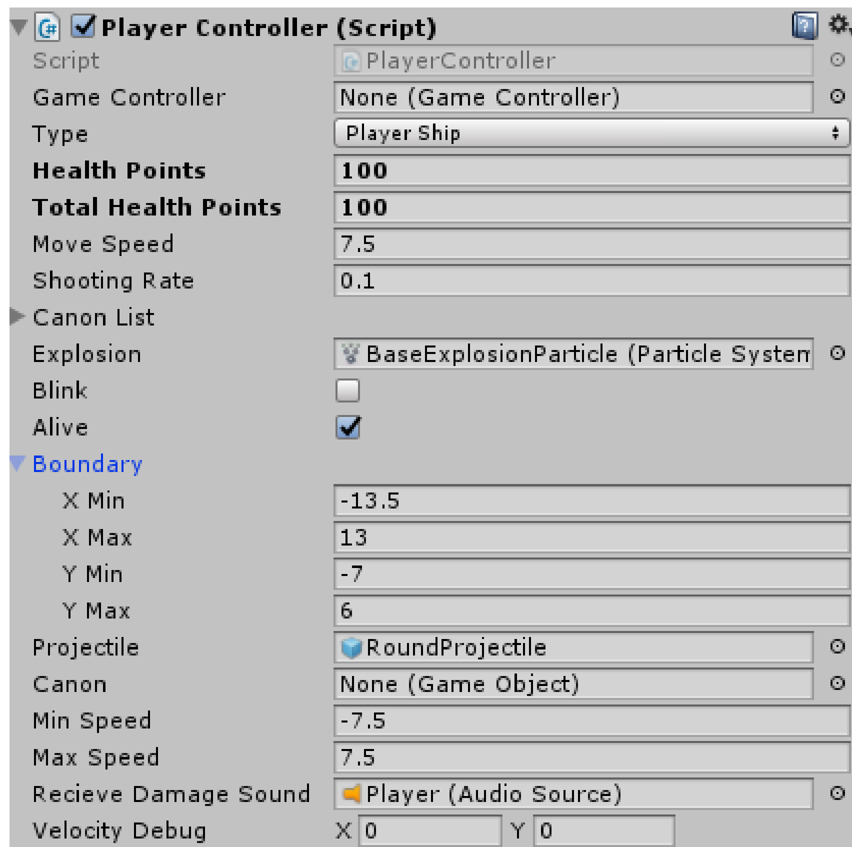



Figure 3.20: Player properties and values inside Unity editor

3. If the Enemy collides with the Player, it gets destroyed, damages the Player (35 hit points) and produces an explosion.
4. If the Enemy collides with the Player's projectile, it receives a damage which is defined inside the projectile.
5. If the Enemy's hit points reach zero, the Enemy adds its reward points to the GameController score value and destroys itself producing an explosion.
6. If the Enemy leaves the camera space, it gets destroyed without producing explosion since it will not be used anymore.

In addition to the base logic above described, each enemy has its own unique logic:

- Enemy1:

  It moves from right to left at a speed $s_1$.

  It shoots a projectile at the Player every $x$ seconds, which is done by finding the Player position and calculating the direction of the shoot.

- Enemy3:

  It moves from right to left at a speed $s_1$ while rotating around its center.

  It shoots four projectiles at different directions while rotating. In same parts of the level, I have grouped up to four of these enemies together to create a stressful environment for the Player, because of all the things it has to keep track off on screen.

- Enemy4:

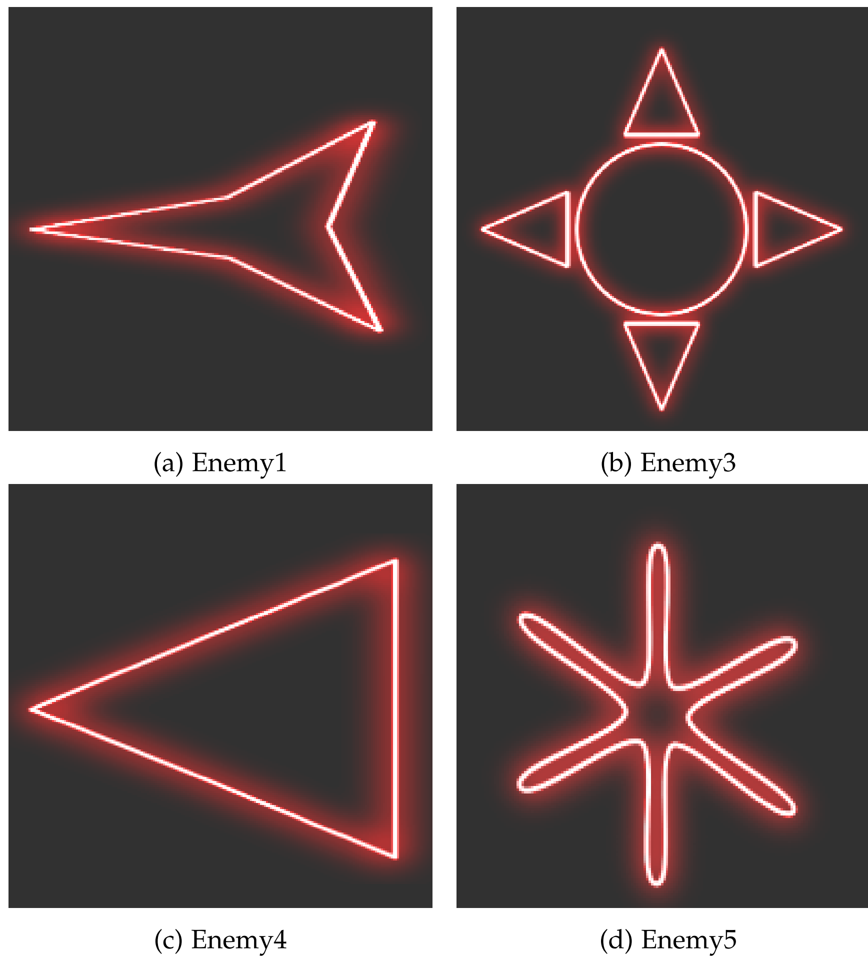
  It moves from right to left at a speed $s_2$, faster than the other enemies. Additionally, it is smaller and harder to shoot at.

- Enemy5:

  It chases the Player at a speed $s_1$, simulating a cat and mouse game.

  Many enemies of this type have been grouped together to make more difficult for the Player to destroy them.

Enemies share the Player's object structure and components, but they are red and have a different Sprite for each one. Additionally, some of them have one or more canons. The four enemies are depicted in Figure 3.21.



(a) Enemy1 (b) Enemy3

(c) Enemy4 (d) Enemy5

Figure 3.21: Enemies inside the game

## 3.13 Miscellaneous objects

Besides Player and Enemies, the scene also contains many visible and invisible objects, used for different purposes and listed below:

- The Laser Barriers (see Figure 3.22) are able to destroy the Player with only one touch, so it needs to evade them in order to survive. All of the Laser Barriers are inside the Moving Background object and move with it, and they get auto-destroyed when leaving the camera space.
- The Speed Triggers are invisible objects placed at various points of the level, that allows to change the speed of the Moving Background speed. In this manner, some parts of the level are faster than others.

- The Boss Trigger at the final of the level, that first creates a beautiful Particle System to simulate a cutting space, next changes the music theme to use the special Boss one, and finally spawns the Boss inside the level.

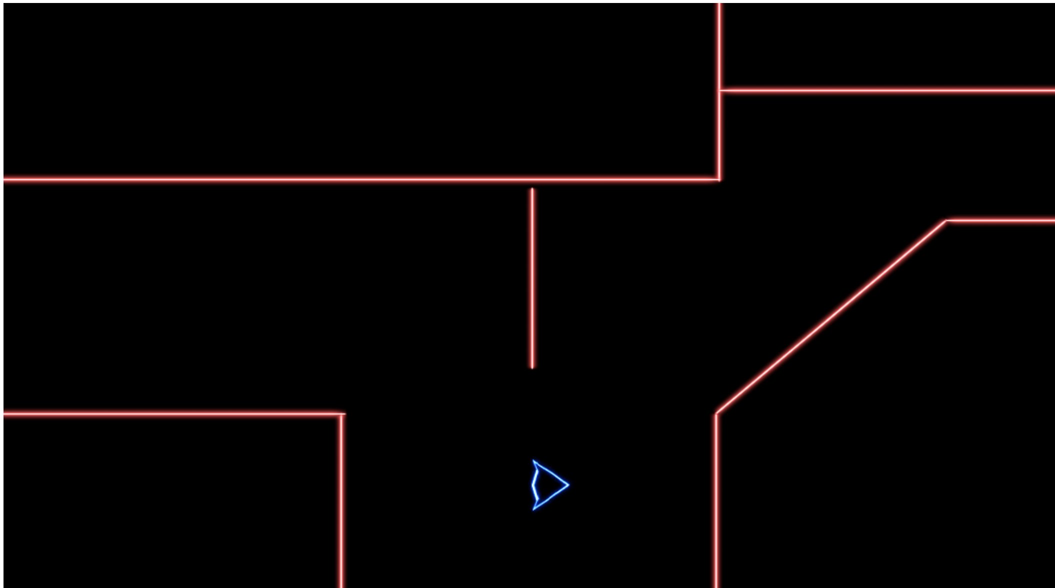


Figure 3.22: Player going around different Laser Barriers

## 3.14 Boss

Every game needs a boss character, and the one here presented has one. The Boss is the most complex ship inside the game, composed of sixteen different parts designed in Inkscape and reassembled in Unity. Its body is made of two main parts, its two main wings have six parts each one, and it also have two little wings at the back. The two Projectile canons located at the main wings are able to rotate, they are always directed at the Player and shoot to it in two-second intervals. Regarding the four Laser canons, they are controlled by a precise timing logic shooting at different intervals. In order to be able to destroy the Boss and complete the level, the Player needs to shoot down all six canons while evading their powerful shots. With the aim of seeing progress when the Boss spawns, a life bar appears for each part that needs to be destroyed.

The Lasers canons follow this logic:

- The inside lasers fire for four seconds, in 12-second intervals. Four seconds before firing, they show the Laser charging effect with a 4-second duration.

- The outside lasers fire for four seconds, in 16-second intervals. Four seconds before firing, they show the Laser charging effect with a 4-second duration.

The logic of combining lasers and projectiles produces a challenge in destroying the Boss. Although the lasers and the projectiles are parts of the Boss, they also act like standalone Enemy ships and give the Player a big amount of reward points for destroying them. Figure 3.23 illustrates the Boss in the middle of the game.

Figure 3.23: Game screen shot showing the Boss when charging its inner laser canons, with one of the outer ones damaged, and its projectile canons shooting at the Player

Regarding the balancing points of the Player's, Enemies' and Boss' weapons, they are summarized in Table 3.2. In addition to this information, it should be noticed that all the Enemies make the same amount of damage to the Player, which is set to 35 hit points as previously explained; and the Boss Laser canons instantly destroy it.

## 3.15 User interface

In order to play the game, a menu system with different options and user interface elements has been created.

Table 3.2: Balancing values of both Player and Enemies (u/s = units per second, p/s = projectiles per second)

| | Health Points | Reward points | Move speed | Canons | Shooting rate |
|---|---|---|---|---|---|
| Player | 100 | - | 7.5 u/s | 2 | 10 p/s |
| Enemy1 | 50 | 50 | 3 u/s | 1 | 1 p/s |
| Enemy3 | 100 | 50 | 2 u/s | 2 | 1 p/s |
| Enemy4 | 20 | 50 | 4 u/s | - | - |
| Enemy5 | 40 | 50 | 4 u/s | - | - |
| Boss Inner Laser | 500 | 2000 | - | 2 | 1/12 p/s |
| Boss Outer Laser | 500 | 2000 | - | 2 | 1/16 p/s |
| Boss Canon | 500 | 2000 | - | 2 | 1/2 p/s |

The first menu shown just after starting the game is the Main menu, from which the user can select one of the following options (see Figure 3.25a):

**New Game:** It starts a new game and, by default, it is set to load the Level1 scene.

**Options Menu:** Different configuration options were planned to be put here, but Unity already presents the user a menu with different options when starting the game. For this reason, it only includes two options (see Figure 3.25b):

- **Volume:** The Volume slider lets the user to change the game sound volume, using the PlayerPrefs.
- **Back:** It goes back to the Main menu, and saves the modified PlayerPrefs.

**Controls Menu:** It leads to the Controls menu, showing the user the Keyboard and Game Pad controls for the game (see Figure 3.25c).

**Exit:** It closes the game.

For each menu button, different options can be configured indicating what to do when the buttons are pressed. For example, the actions when the user presses the up or down buttons have been configured in simple manner, allowing the user to be able to go only up or down as illustrated in Figure 3.24. The same actions are also used for the other menus.

For the Level1 scene, I have created the following menus:

**Pause Menu:** It is shown when the user presses the pause button ("Esc" on the Keyboard or "Start" on the Gamepad). It contains the following options (see Figure 3.25d):

- **Resume:** It resumes the game.

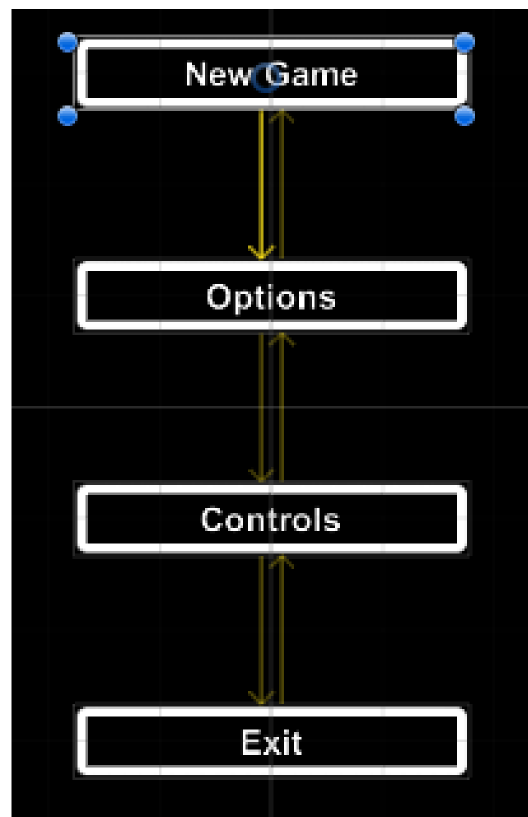



Figure 3.24: Navigation path of actions in the Main menu

**Restart:** It restarts the game.

**Main Menu:** It returns to the Main menu scene.

**Lose Menu:** It is shown when the Player is destroyed, with a big red "YOU DIED" text on the top. It contains the following options (see Figure 3.25e):

**Restart:** It restarts the game.

**Main Menu:** It returns to the Main menu scene.

**Win Menu:** It is shown when the Boss is defeated and the game is won. It shows the user's score and the high-score (see Figure 3.25f). If the score is bigger than the high-score, it also updates the high-score using the PlayerPrefs class.

**Main Menu:** It returns to the Main menu scene.

Besides the different menus, there is a simple interface (see Figure 3.26) that shows the current score, the current playtime and the Player's life. All this information is controlled and updated using the Game Controller.

(a) Main Menu (b) Options Menu

(c) Controls Menu (d) Pause Menu

(e) Lose Menu (f) Win Menu

Figure 3.25: Different game menus

Figure 3.26: Interface with the current user information

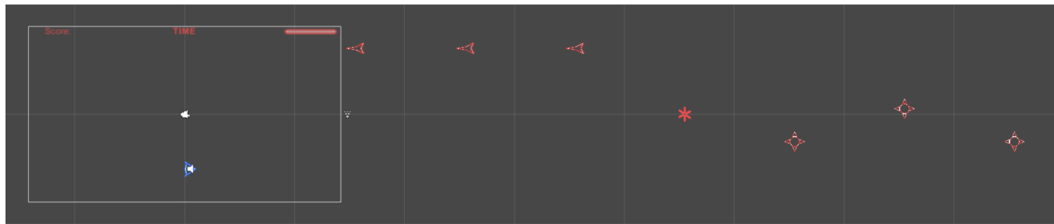
## 3.16 Level design

After creating every element as described in the previous sections, and in order to create a good and playable level, I started the level design by placing a couple of different enemies on the scene, and then I played it while trying to find out how it could be improved. The same process was repeated by placing barriers and other types of enemies, in such a way that each iteration was larger and more balanced

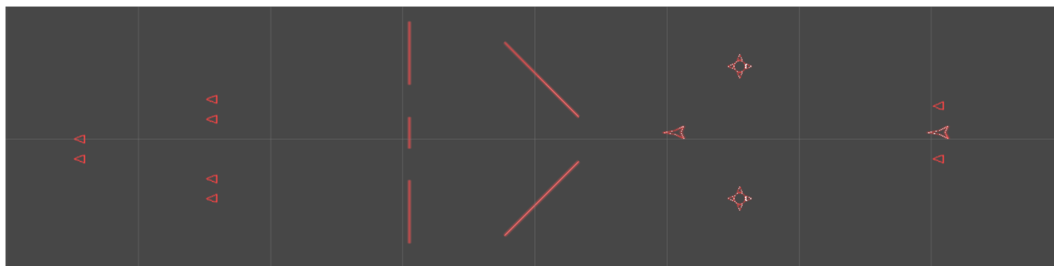
than the previous one. Additionally, I also sent the level to other people in order to obtain some useful feedback, which included comments like:

- Make the Player move slower because it was very difficult to control it when moving too fast.
- Redistribute the placement of both enemies and barriers.
- Make some feedback appear on the screen when someone gets damaged (the Blink function was made for this issue).
- Include a sound when the Player gets damaged.
- Make all the enemies do the same amount of damage since it was very confusing that each one was different.

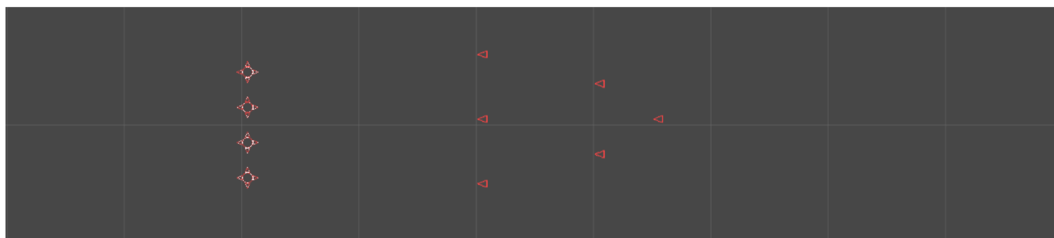
By the end of this project, I have played to the same level more than 250 times, with many variations of it. Now, I think that the final result is quite balanced and playable for many people. Since the entire Level1 is too big to be place in one single image, Figures 3.27 and 3.28 show it sliced into nine different pieces. Note that the first part corresponds to the start of the game, whilst the last one is the end.



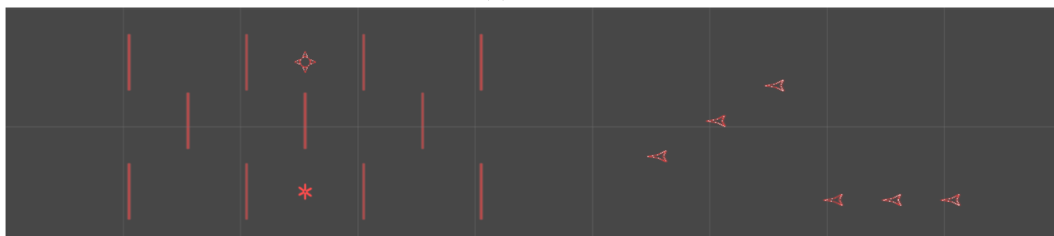

(a) Part 1 (Start)



(b) Part 2



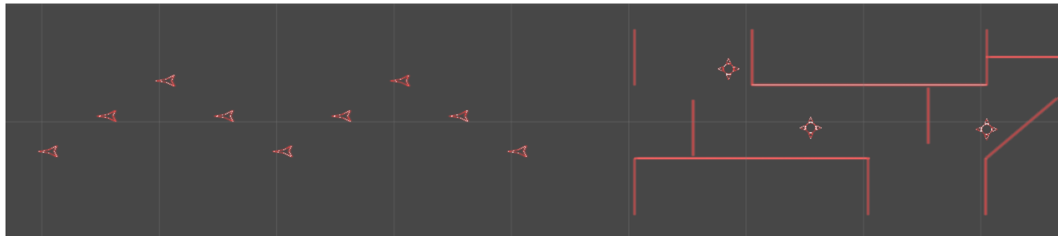
(c) Part 3



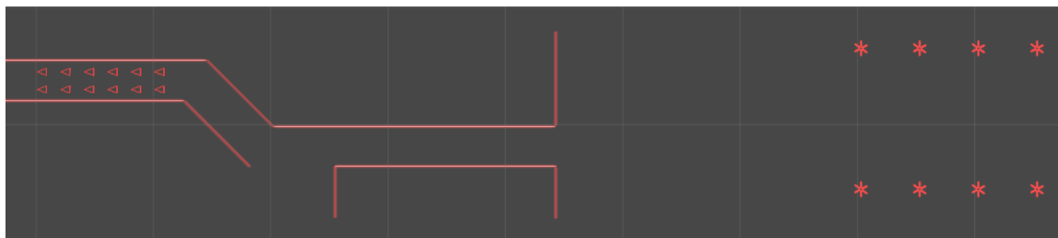
(d) Part 4

Figure 3.27: Level1 sliced, parts from 1 to 4

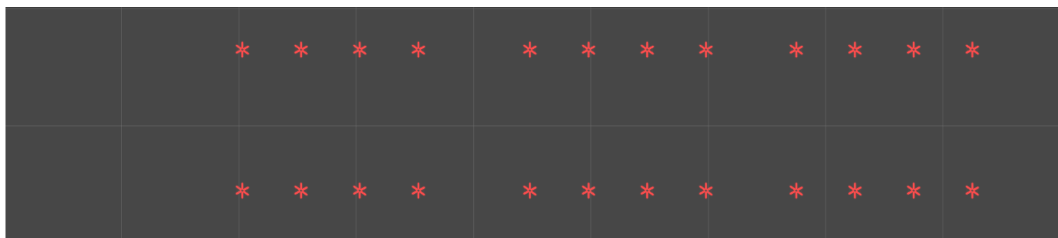
(a) Part 5



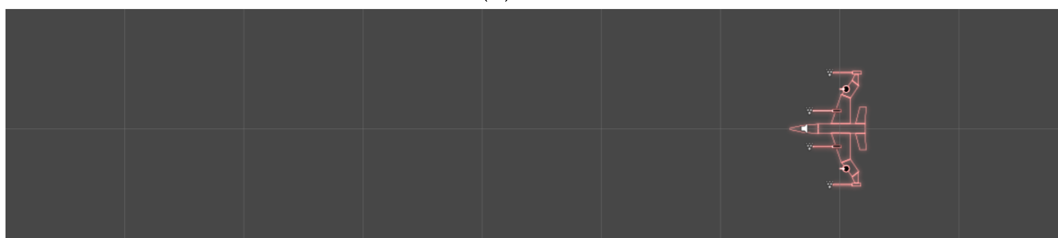
(b) Part 6



(c) Part 7



(d) Part 8



(e) Part 9 (End)

Figure 3.28: Level1 sliced, parts from 5 to 9

# Chapter 4

# Accessibility and colorblindness

Nowadays, when video-games are played by millions of people, there is a strong need for these games to be able to be played by people with different disabilities. For that reason, I have decided to include some accessibility options in this project and, in particular, I have focused on colorblindness.

Colorblindness, or color vision deficiency, is a disability that makes a person to be unable to perceive different colors or variations of the same color. It can cause trouble in person daily life such as inability to see the difference between traffic lights or to distinguish different types of fruits. Generally, people are able to adapt to it, but sometimes it is very hard for them to do it [5].

## 4.1 Types of colorblindness

Although there are many types of colorblindness, I have decided to center my work on the three most common types, which are subsequently described [5, 17, 12]:

**Protanopia:** reduced sensitivity to the red light, which causes that red is seen as yellow-greenish color. People with protanopia are most likely to confuse (or most likely see to A as B):

- Dark brown with dark green.
- Dark orange with dark red.
- Different shades of red with black.
- Different shades of blue with different shades of red.
- Different shades of purple with dark pink.
- Medium green with orange.

**Deuteranopia:** reduced sensitivity to the green light, which causes that green is seen as yellow-greenish color. People with deuteranopia are most likely to confuse:

- Blue-green with gray and medium pink.
- Bright green with yellow.
- Medium red with medium green.
- Medium red with medium brown.
- Pale pink with light gray.

**Tritanopia:** reduced sensitivity to the blue light, which causes that green is seen as blue, and red as pink. People with tritanopia are most likely to confuse:

- Dark purple with black.
- Light blue with gray.
- Medium green with blue.
- Orange with red.

## 4.2 Accessibility: implicit vs explicit

When designing and developing a video-game, the developer can choose to incorporate accessibility options explicitly or implicitly, depending on both game style and mechanics:

**Explicit:** It means that the selected accessibility option cannot be incorporated in the base game design and has to be done as a separate option. Thus, many times it is included as an afterthought and it does not work very well. For example, I could include the colorblind option and let the player choose its type of colorblindness, which will work but not always in a good way. In addition, it will also require the player to go through various options and test them.

**Implicit:** It means that the selected accessibility option is a core part of the base game design and seamlessly work in the game. This project is an example, in which I have decided to give both Player and Enemies different shapes to distinguish them, and created a single color-palette manually tested with the most common colorblindness types.

## 4.3 Color palette

In order to create a single working color palette that can be differentiated by people with different types of colorblindness, I have decided to use a free-ware software known as Color Oracle.

Color Oracle [6] is a colorblindness simulator for Windows, Mac and Linux operating systems, that applies a special color filter on the full screen in order to show the user how a colorblind person sees the screen. There are a total of three filters, one for each type of colorblindness previously mentioned: protanopia, deuteranopia, and tritanopia.

Using this software, I have executed my game and tested it to see if the objects with different colors could be distinguished (mainly the Player and its projectiles from the Enemies, the Enemies' projectiles and the obstacles). In case of having problems to distinguish them, I have tested it with other colors. The colors that can be perfectly distinguish for the three types of colorblindness are the ones used in the final version of this project. Figure 4.1 illustrates the final color palette and its equivalence to how people with different types of colorblindness see it.

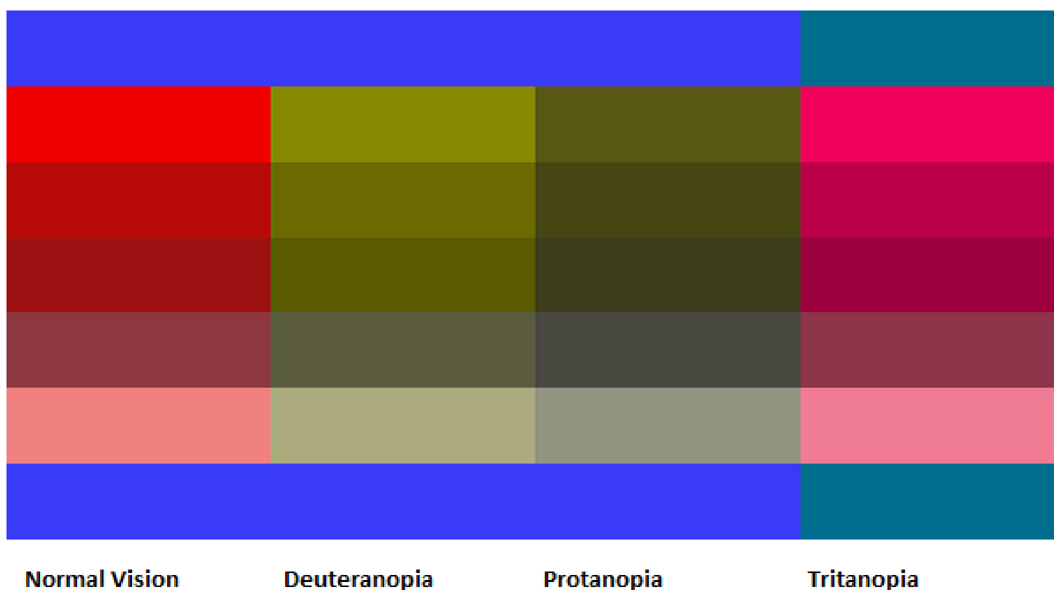



Figure 4.1: Color palette used in this project as seen by people with normal vision and the three different types of colorblindness considered

# Chapter 5

# Methodology

As in any project, some planning and methodologies have been considered to develop this video-game and, among all of them, I have used a combination of Kanban and Scrum. Using only the necessary parts from both of them, I have adapted them for one person environment. Both Kanban and Scrum are agile methodologies that focus at releasing working versions of software early and often [22], which was perfect for the project at hand since I needed to obtain feedback after each version. In this context, projects can be split into sprints, which can be defined as time boxes of one month or less during which a list of specified tasks can be considered as "DONE".

In the development of this project, I have used GitHub for version control and project hosting [8]. Github is a web based hosting for different projects that consist of code, and it provides a Git version control system as well as other tools such as a simple task management system based of Kanban [1, 14]. Additionally, it allows to create a wiki page for hosting documentation and it provides other tools for easier project management.

## 5.1 Development life cycle

The development life cycle here refers to the development of each unique object inside this project. In general terms, the process consists in designing graphical and sound objects, and using them to create Unity objects. After that, each object is tested in a standalone environment which is usually an empty Unity project. Then, the object logic is developed and both integrated and tested inside the full project. When those steps are finished, the project is sent to different people for testing and gaining feedback. The feedback obtained is used in the next cycle, which again starts from revising graphical and sound objects if needed. Figure 5.1 includes the procedures done when developing objects for this game.

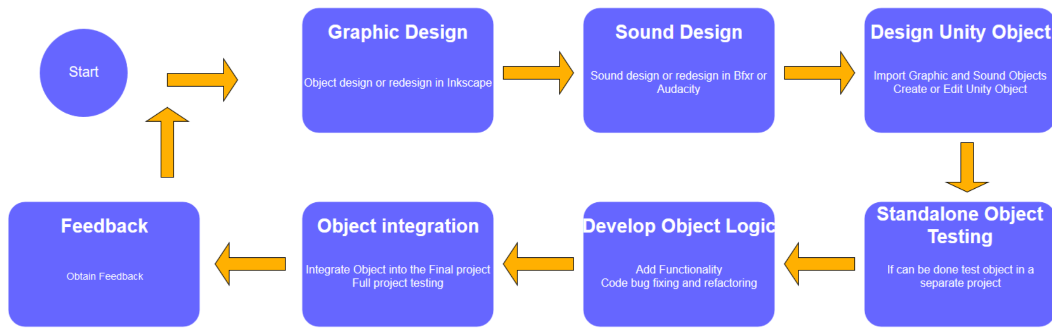



Figure 5.1: Development cycle of a game object

## 5.2 Task management

After the definition of the objectives, I have translated them into a list of simple tasks and added them to my Kanban task backlog which consisted of the following columns to efficiently manage the project:

- TODO: tasks that needed to be done.
- WIP: tasks that I was doing at that moment.
- FIX: things that needed to be fixed, re-factored or optimized.
- IDEAS: ideas that I had when developing the project.
- DONE: column with the start and the end date for each Sprint.

Then, I have created a schedule consisting of two-week Sprints as described in the Scrum methodology [19]. At each sprint, I did a limited amount of tasks from backlog, and did not have more than two tasks in progress, which allowed me to manage my time efficiently. If there were new tasks that needed to be done, I added them to the backlog and then took them to do at the start of each sprint. These tasks usually included bug-fixing, code refactoring, and redesigning some systems. After finishing each task, I replaced its basic description in Kanban backlog with detailed description of all the work that it required. Additionally, after finishing each sprint, I sent my game to some of my friends in order to obtain feedback from them and used it to improve my game.

During the first two months of the project, I spent the 60% of the time reading the documentation and watching video tutorials to learn every needed tool and to understand how everything works, the 20% making tests and prototypes of said functionality, and the remaining 20% integrating it into the project and making additional tests. After this initial two-month period, I have learned how to use

the software for developing each and every object. Because of that, my working speed accelerated and I had more time to really design, program and develop the functionalities needed. As an example, I found out that at the beginning it took me around three to four hours to make graphic design of a player or enemy, while towards the end I could do the same thing in around fifteen minutes. Regarding the last month, it was mainly spent writing the documentation.

As all the tasks that are inside the Kanban backlog could not be represented in one image or diagram, Figure 5.2 illustrates a live example. A summarized list of the tasks is included in Appendix B.

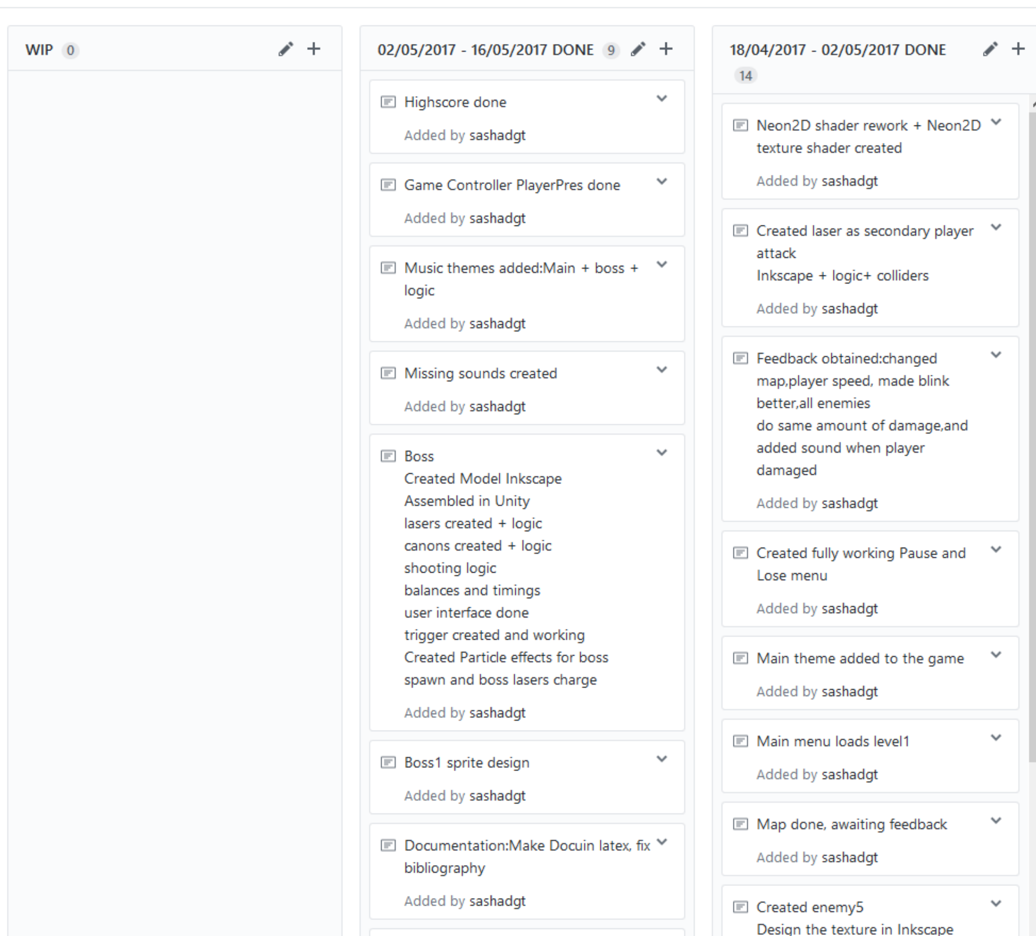



Figure 5.2: Screen capture showing part of my Kanban on GitHub website

# Chapter 6

# Results

This chapter presents the final results of different tasks made during the whole project. The results include tests made during the graphic design, testing colors for colorblindness disability, game performance and different control schemes.

## 6.1 Graphic design

When designing the graphic aspects of the project, one of the most important parts was to correctly export SVG images to PNG format, and next import them in Unity without losing any quality.

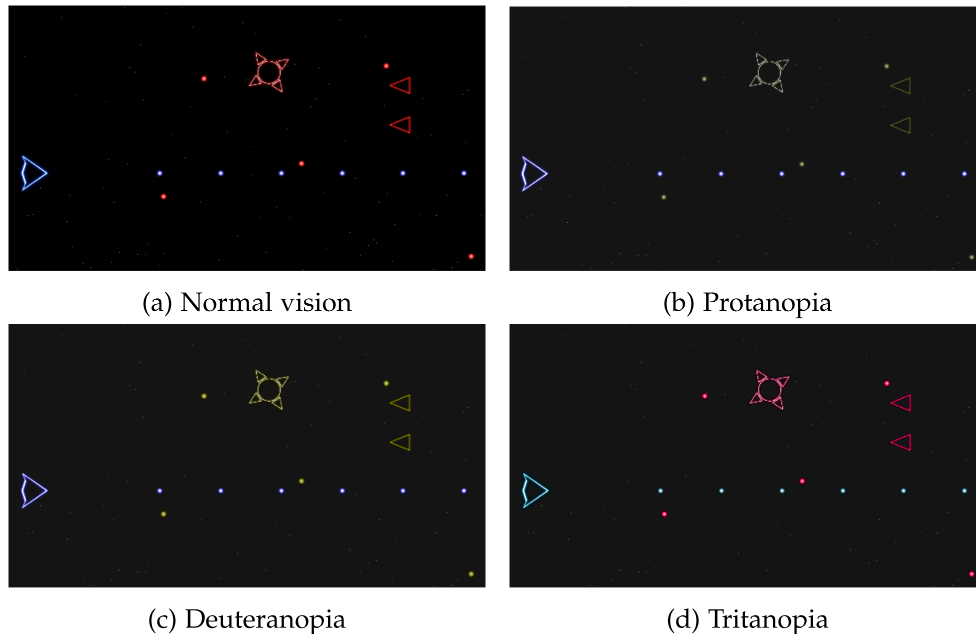
In Unity, if you import an image with very small size, for example $64 \times 64$ pixels, and then display it at an appropriate size, it will look pixelated. On the other hand, if you use an image with very big size, for example $4096 \times 4096$ pixels, Unity will produce artifacts such as aliasing when downsizing it for display, because the filters used in this process are made to be fast but not accurate. For that reason, I have made many tests with different images at several sizes, and I found out that the best looking images have an approximate size of $512 \times 512$ pixels.

## 6.2 Colorblindness testing

After making several tests using different colors and taking into account the three colorblindness filter types, I have chosen the best colors that allow the user distinguish between the Player and its projectiles, and the different objects that it has to destroy or evade in order to win. These objects include the Enemies and their projectiles, the Laser barriers, the Boss and its weapons, projectiles and lasers.

The final color configuration, obtained as a result of this testing stage, can be seen at Figure 6.1, which shows the differences between the Player and the

Enemies inside the game. It should be highlighted that the most important aspect here is that the user has to be able to differentiate between allied units and hostile units: allied units have blue color, whilst hostile units have other colors. Note that the color palette finally used in the video-game has been previously presented in Section 4.3.



(a) Normal vision (b) Protanopia

(c) Deuteranopia (d) Tritanopia

Figure 6.1: Examples showing how the different elements of the video-game are seen for the three types of colorblindness considered

## 6.3 Game performance

As this is a simple 2D game, a very important aspect is that it can be ran in a wide variety of hardware without major performance loss. Broadly speaking, the performance in games is measured in frames per second (fps), which is the frequency rate at which the game images are displayed on the screen. This measure is called frame-rate, and its ideal value in games is 60 fps. At that frame-rate, the game-play feels responsive and smooth.

Other important concept in terms of game performance is Vsync. It synchronizes the frames sent by the GPU with the refresh rate of the screen, and limits it at its maximum value which in most screens is 60Hz. The real performance in game is obtained with this option disabled, and the bigger the fps the better. However, users usually prefer to have it enabled, because otherwise some artifacts such as

screen-tearing may appear on the screen.

All the testing has been carried out by using MSI Afterburner [15], a software that provides detailed information and statistics of the game performance. The testing has been done on different operating systems, including different versions of Windows (7, 8, and 10), and Linux (Ubuntu and Debian). In this sense, it should be noticed that the performance across all of the operating systems considered was almost the same. Regarding the computers tested, all of them had Intel processors inside and gave an average of 60 fps with the Vsync option enabled, which is the best result possible for screens with refresh rate of 60 Hz. Table 6.1 presents additional data about the game performance testing and the resource usage on different hardware configurations. Judging by the frame rate with the Vsync option disabled, the performance obtained is more than enough to perfectly enjoy the game.

Table 6.1: Game performance on different hardware configurations

| | | | Memory used | | | |
|---|---|---|---|---|---|---|
| CPU | RAM | GPU | RAM | VRAM | Resolution | Vsync off |
| Core i7-5930K | 16GB | NVIDIA GTX 980 TI 6GB | 90MB | 444MB | 2560x1440 | 1300 fps |
| Core i7-2670QM | 8GB | NVIDIA GT 540M 2GB | 100MB | 167MB | 1366d768 | 500 fps |
| Core i5-M460 | 4GB | ATI Mob. Radeon HD5470 1GB | 108MB | 256MB | 1600x900 | 425 fps |
| Q6600 | 4GB | ATI Radeon HD7000 1GB | 105MB | 188MB | 1400x900 | 750 fps |
| Core i5-4570 | 3.5GB | Intel HD Graphics 4600 512 MB | 255MB | 240MB | 2048x1152 | 150 fps |

## 6.4 Game controls

The final controls for Xbox 360, Xbox One or any compatible XInput[1] game controller were setup in the following way: the left stick is used for Player movement and menu navigation; buttons A and B are set as Primary and Secondary shoot actions; menu options can be executed using button A; Start button is used for pausing and resuming the game. All the game pad controls are illustrated in Figure 6.2.

With respect to the keyboard controls, they are following detailed: Arrow keys are set for Player movement and menu navigation; Ctrl and Alt keys are set for Primary or Secondary shoot actions; and Esc key is used for pausing and resuming the game. All the keyboard default controls can be seen at Figure 6.3. Alternatively, the user can use (W, A, S, D) keys for Player movement and menu

[1]XInput is an API that allows applications to receive input from the Xbox 360 Controller for Windows.

navigation, and mouse left and right click for primary and secondary shooting actions.

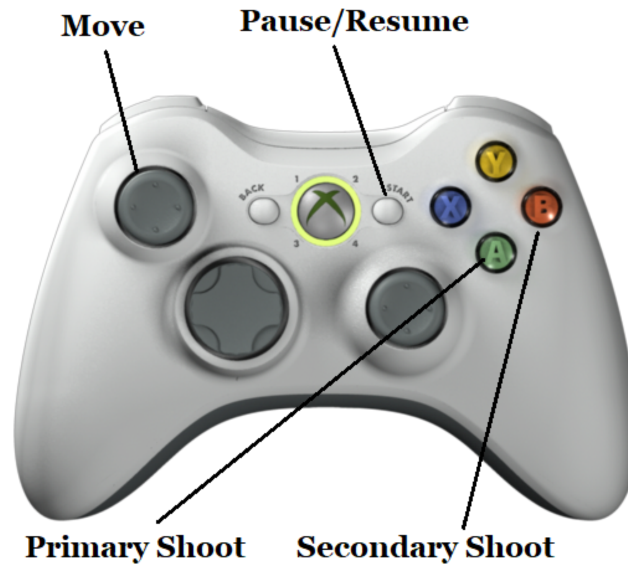



Figure 6.2: Default Xbox 360 game pad controls

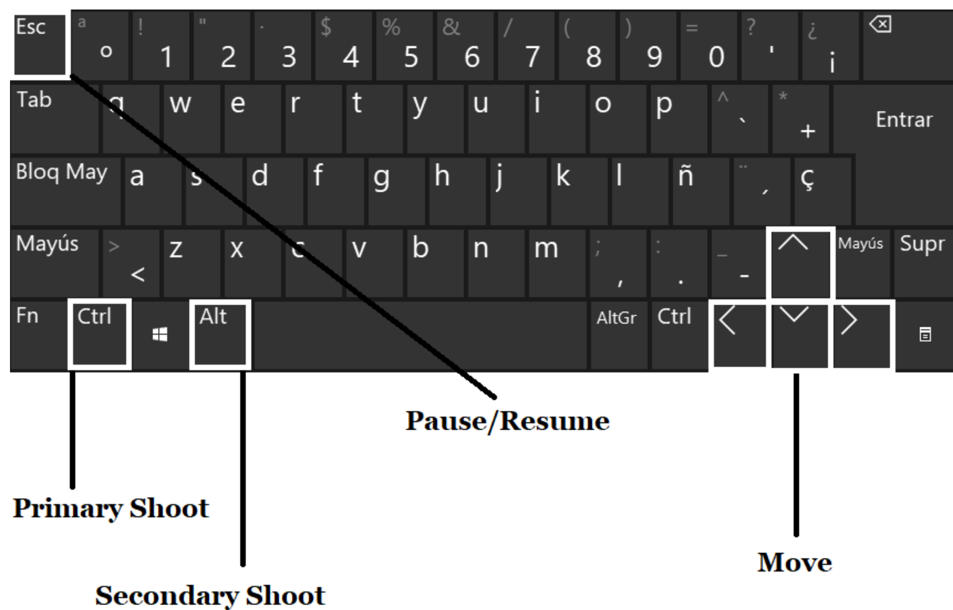



Figure 6.3: Default keyboard controls

If users do not like the default controls, they can always change them before starting the game in the options that Unity provides by default when starting the game. This is possible thanks to Unity having a special set of actions predefined, such as for example: Horizontal and Vertical axis give access to the navigation keys; while actions "Fire1" and "Fire2" let you access to other functions, which in this project are used for Primary and Secondary shooting.

# Chapter 7

# Conclusions

The main objective of this project was to design and develop a 2D shoot 'em up video-game and learn the maximum amount of things possible while doing it.

For this purpose, the different graphic elements of the video-game were designed, allowing me to learn how to efficiently use the Inkscape software in order to achieve the desired results. In this manner, each design has been done faster and better than the previous one. Similarly, I have learned how to design simple albeit necessary sound effects for the game by combining audio software, more specifically Audacity and Bfxr.

Regarding the video-game itself, all of the game logic and the objects were designed and subsequently implemented using Unity and its 2D toolset. As a result, the video-game is composed of one single perfectly playable level which includes many different objects and obstacles to overcome. This is the part that I have most enjoyed, because I really like programming and designing game logic. Additionally, coming up with different Enemies and Boss logic provided me a fun challenge, and gave me the opportunity to learn many different issues about game design and development.

Furthermore, I discovered many interesting things about developing a game for people with colorblindness disability, and found out a way for them to thoughtfully enjoy the game.

On the other hand, the obtained video-game can be played at different operating systems and it has a very good performance even on dated computers. Moreover, it can be easily ported to any of the current generation game consoles.

Finally, I would like to highlight that most of the things done throughout this project were learned while developing it, as I had very little experience in the whole process of video-game development. I had made very simple video games without using full fledged game engines, and never before had I designed any sound effect or graphic material for them.

## 7.1 Future work

In this project, I have done all the necessary to achieve the main targets by designing and developing my own video-game. However, I already have a few ideas to extend this work and improve the video-game:

- Create a complete projectile system with many types of projectiles and ways to shoot them, using different approaches.
- Create more levels with different game-play options, each of them with new enemies and bosses.
- Create a power-up system that allows players to get new powers, such as super-speed.
- Make a different kind of reward logic, like including a multiplier to calculate the final score.
- Make different types of ships that can be used as the Player.
- Create a module for Unity to accept graphics in SVG format.
- Create a similar video-game but using other game engines, such as Unreal Engine and 3D Graphics.

# Appendix A

# Diagrams

This appendix presents different diagrams used during the development of this project. Some of them show how different classes are related, what they are and their properties; while others show the use case flowcharts of interactions between different objects.

## A.1 Class diagrams

This section includes four class diagrams with all the classes used in this project, as well as their properties and methods. In some cases, the relationship between classes is represented as a hierarchy:

- Figure A.1 shows how inheritance is used between the Player, the Enemies, and the parent class BaseShip. Each class includes all its properties and functions.
- Figure A.2 shows how inheritance is used between the PlayerProjectile, the EnemyProjectile, and the parent class BaseProjectile. Each class includes all its properties and functions.
- Figures A.3 and A.4 show miscellaneous classes and structures used in this project. Each class includes all its properties and functions.

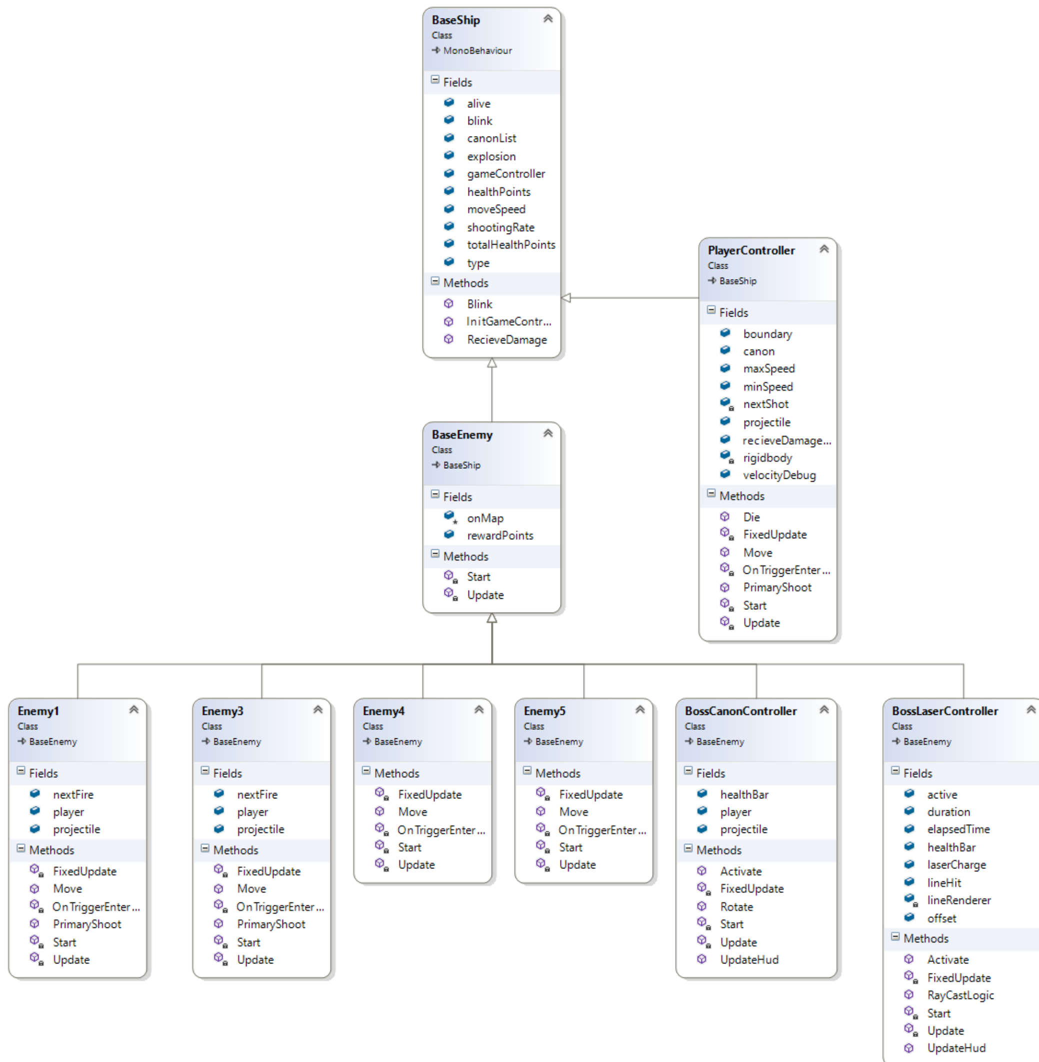



Figure A.1: Class diagram of the BaseShip as the parent class, and the Player and Enemies as the children classes.

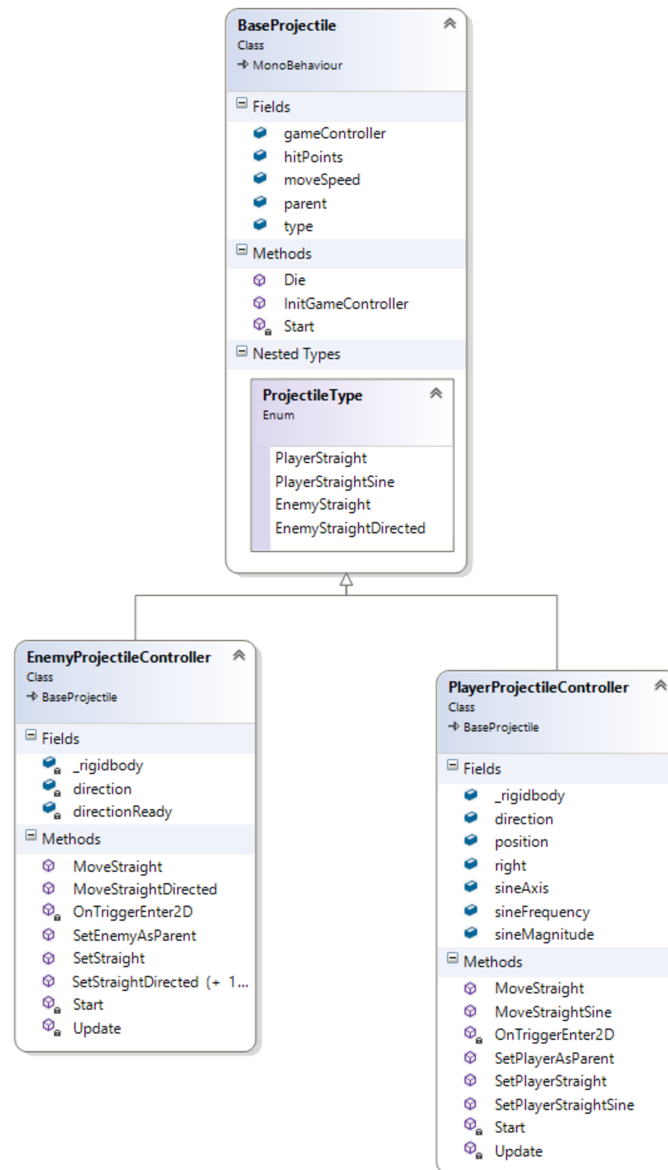



Figure A.2: Class diagram of the BaseProjectile class and its children classes
Class diagram of the BaseProjectile parent class and its children classes

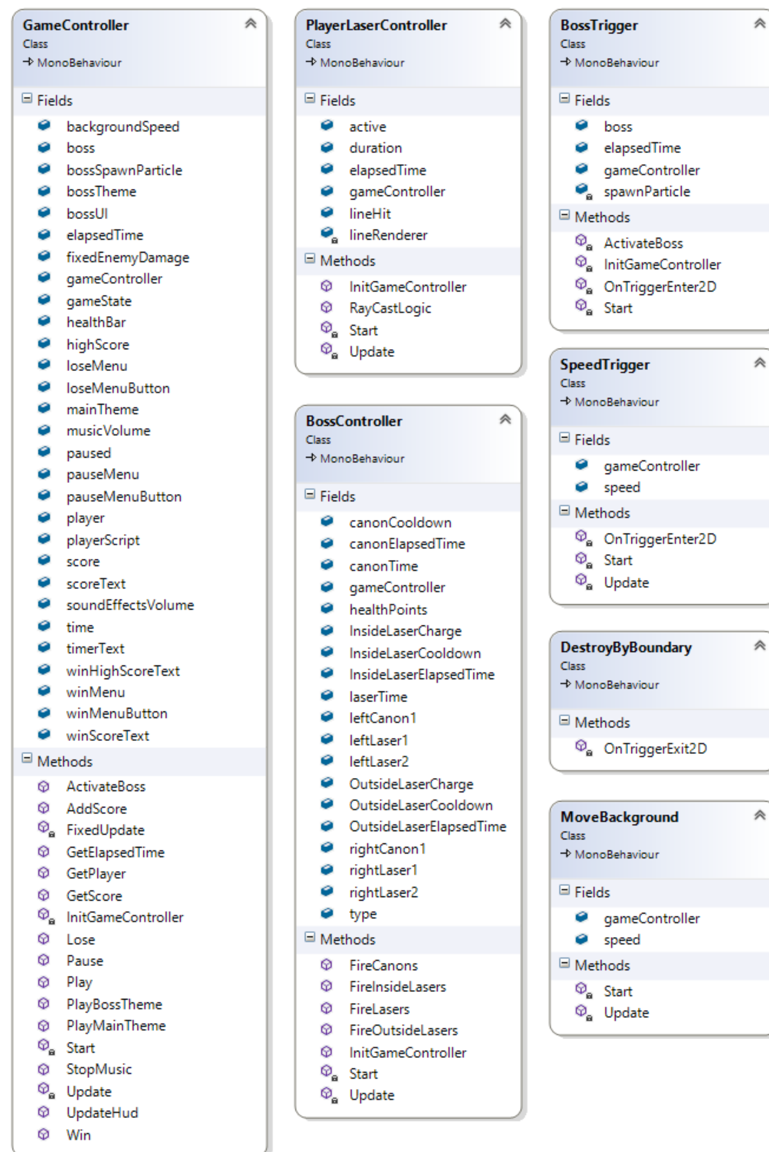



Figure A.3: Class diagram with the GameController, the BossController and other additional scripts

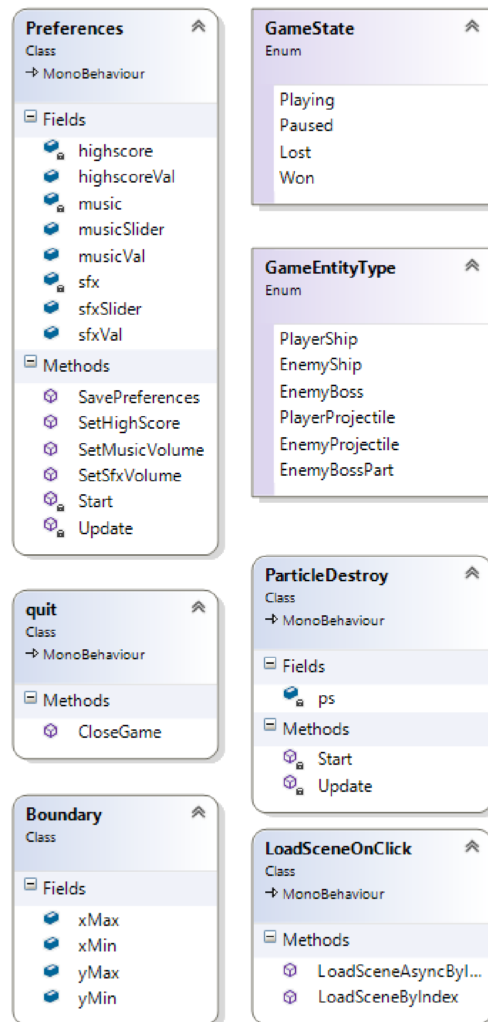



Figure A.4: Class diagram with the miscellaneous classes

## A.2 Flowchart diagrams

This section includes six flowchart diagrams with the interactions between the objects designed and developed in the project:

- Figure A.5 shows how the Main menu has been made with all its options: New Game loads the Level1 of the game, Options allows the user to change the volume and go back to the Main menu, Controls displays the default controls for the game and allows the user to go back to the Main menu, and Exit closes the game.
- Figures A.6 and A.7 show, respectively, how the Player and the Enemy projectiles work during their lifetime and how they interact with other objects.
- Figures A.8 and A.9 show, respectively, how the Player and the Enemy objects work, what actions they do and how they interact with other objects.
- Figures A.10 shows how the Game Controller object controls the game logic, the user interface elements, the player preferences, and the different gamestates.

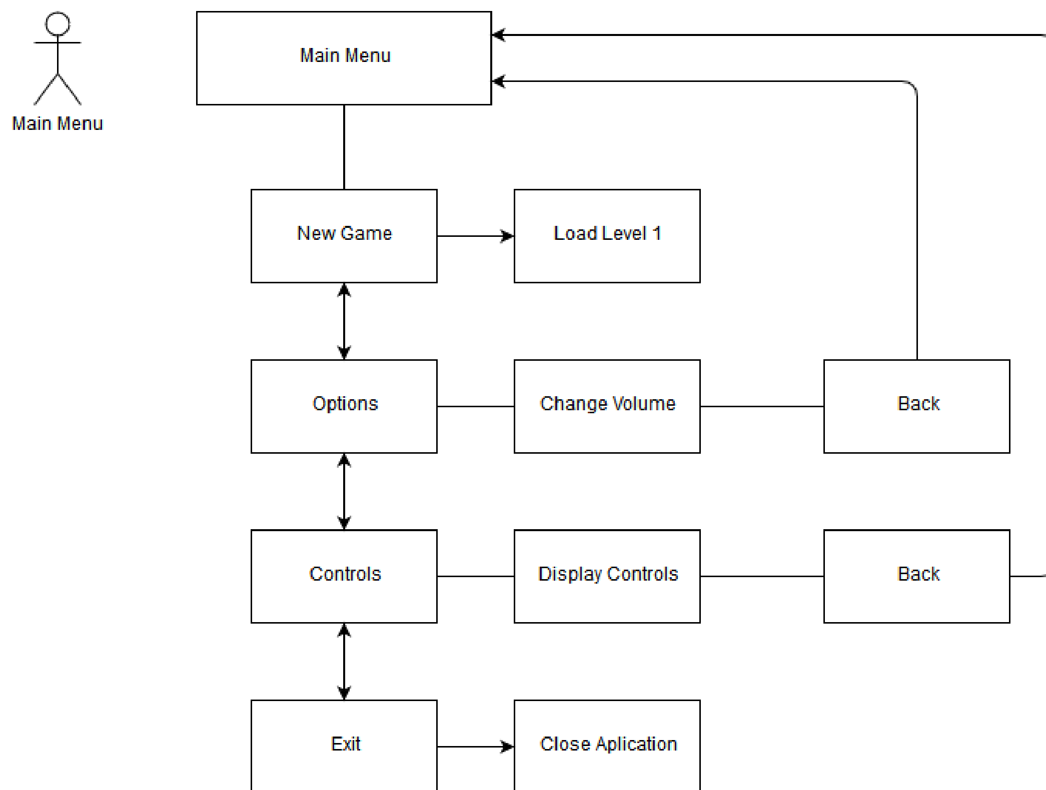



Figure A.5: Flowchart diagram of the Main menu

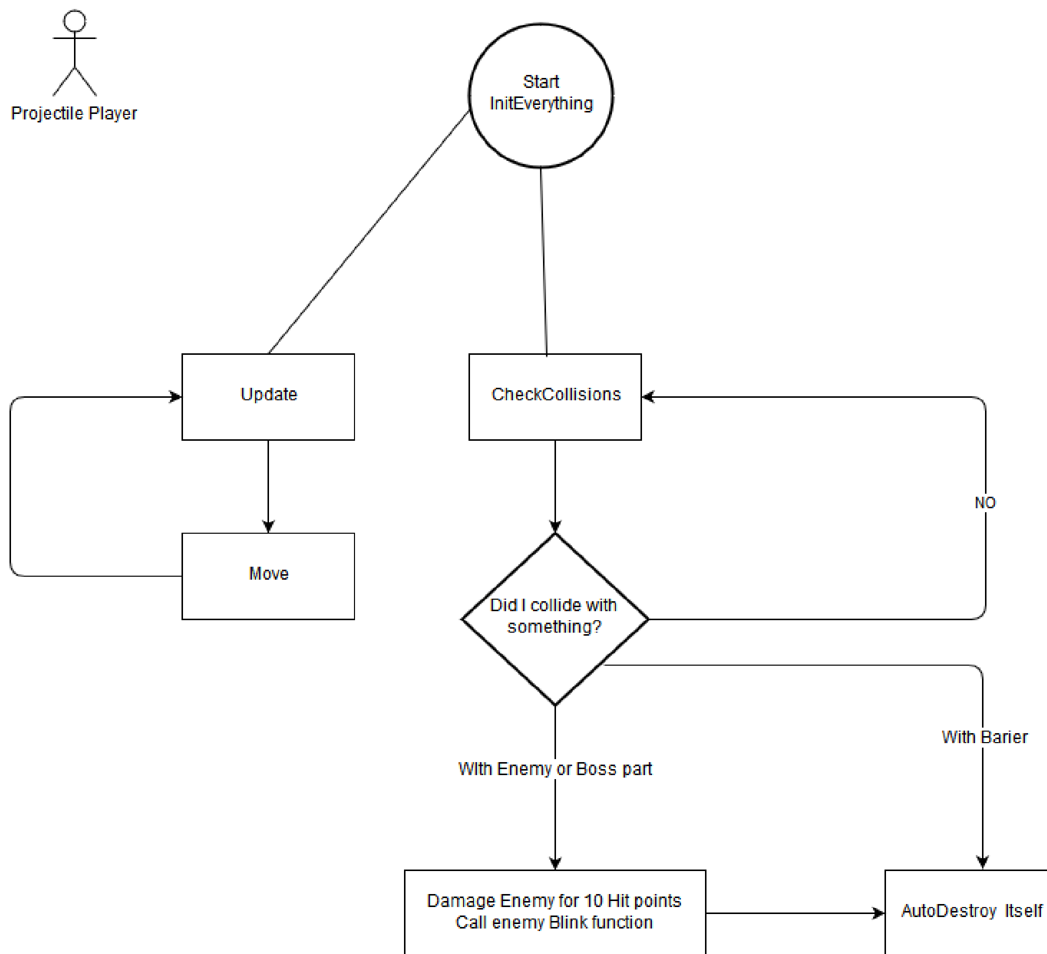



Figure A.6: Flowchart diagram of the Player Projectile

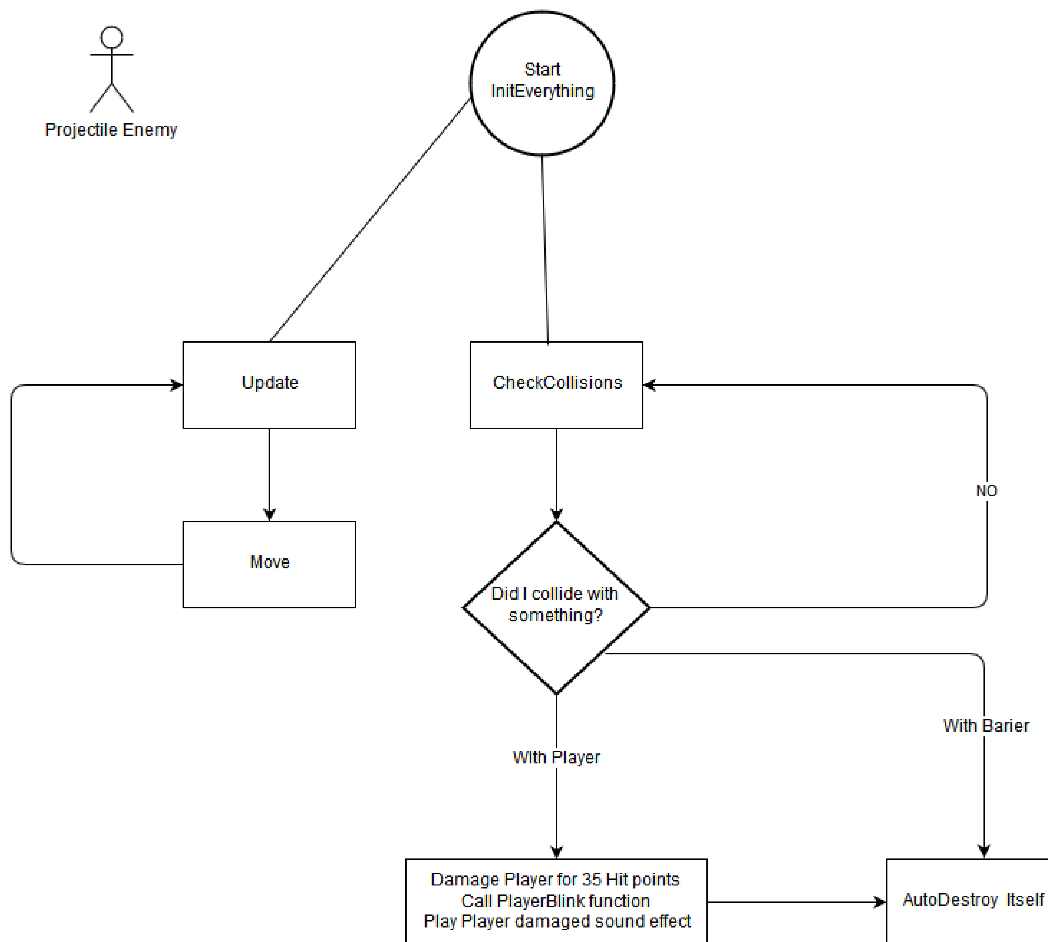



Figure A.7: Flowchart diagram of the Enemy Projectile

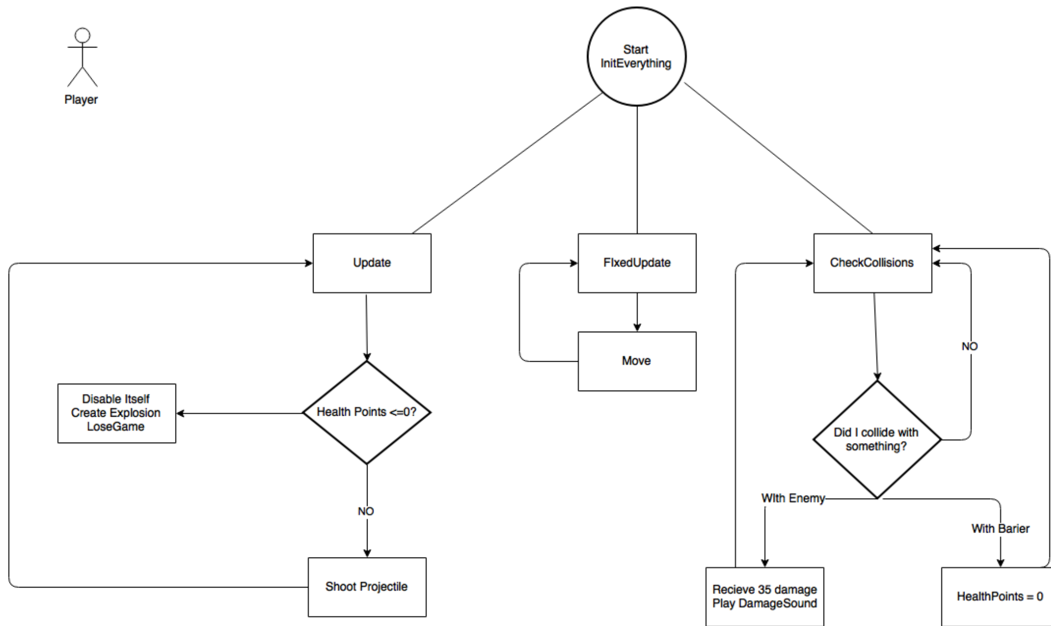



Figure A.8: Flowchart diagram of the Player

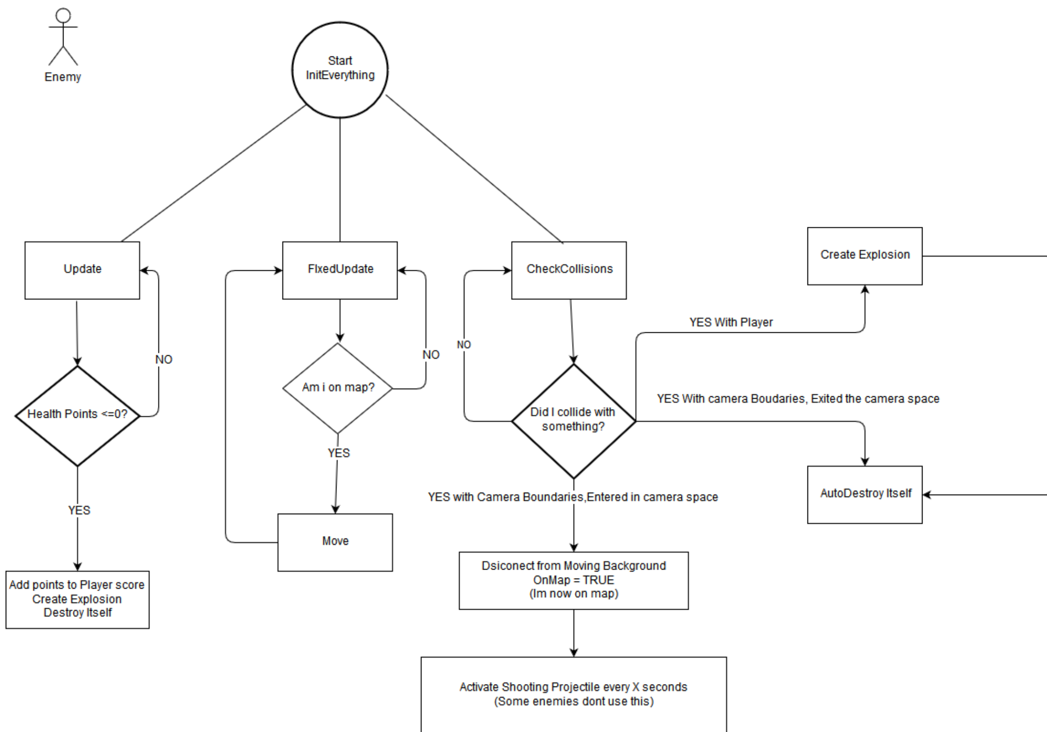



Figure A.9: Flowchart diagram of the Enemy

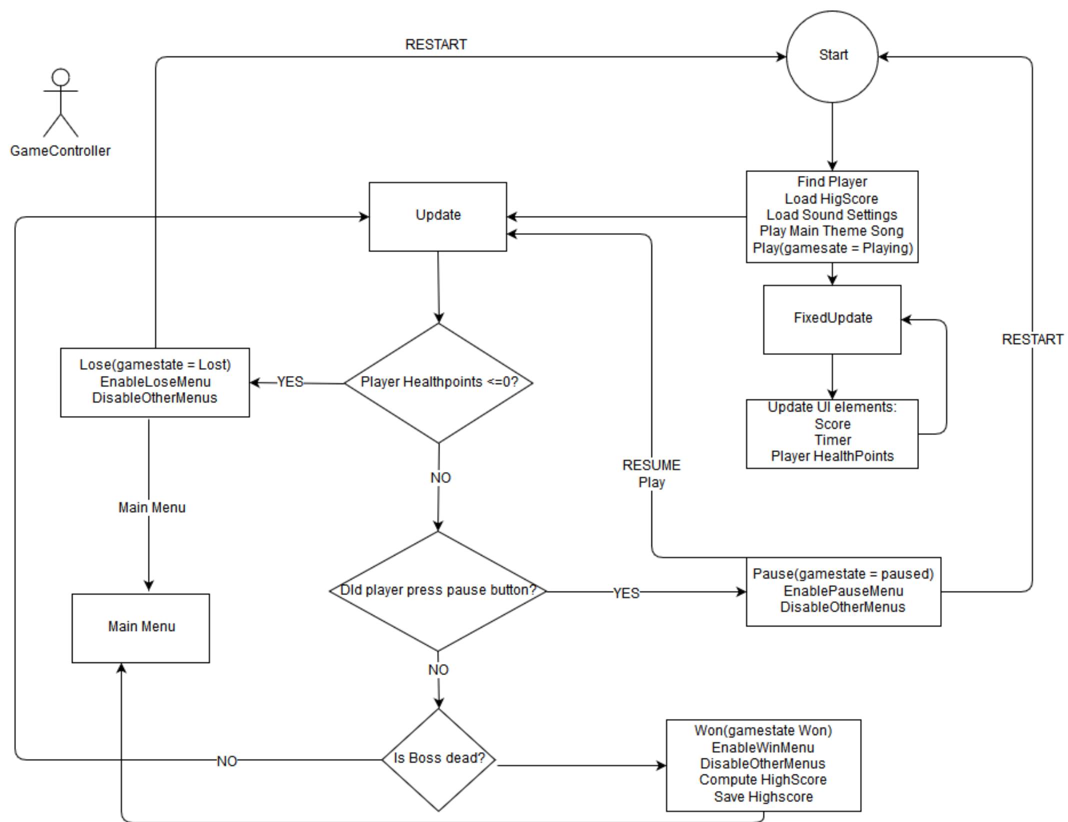



Figure A.10: Flowchart diagram of the Game Controller

# Appendix B

# Task calendar

This appendix includes a detailed list with the main tasks carried out during the project. They are separated in sprints, and their respective dates are specified.

## B.1 Sprint 1: 21/02/2017 – 07/03/2017

1. Create the project and configure the camera.
2. Create the Player.
    - Design the texture in Inkscape.
    - Create the object in Unity.
    - Program the moving logic.
3. Create the Enemy1.
    - Design the texture in Inkscape.
    - Create the object in Unity.
    - Program the moving logic.
4. Start the level design, and test it with the Player and the Enemy1.

## B.2 Sprint 2: 07/03/2017 – 21/03/2017

1. Make the Base class for Enemies and Player.
2. Make the Enemy1 shoot to the Player.
3. Make different projectiles.

- Inkscape design and movement logic.

4. Create the Explosion particle.
5. Make the collision logic between Enemy and Player, and their projectiles.
6. Test the level design.
7. Create the StarField particles.

## B.3 Sprint 3: 21/03/2017 – 04/04/2017

1. Create the Neon2D Shader.
2. Create the Materials from the Shader and apply them to the Player and the Enemy1.
3. Create the Materials for projectiles and particles.
4. Test the different texture sizes (in PNG and SVG formats).
5. Create the Enemy3.
    - Design the texture in Inkscape.
    - Create the object in Unity.
    - Program the moving logic.
    - Program the shooting logic.
6. Test the level design with additional enemies.

## B.4 Sprint 4: 04/04/2017 – 18/04/2017

1. Create the Enemy4.
    - Design the texture in Inkscape.
    - Create the object in Unity.
    - Program the moving logic.
    - Program the shooting logic.
2. Create the GameController with base logic.
3. Create the Barriers.

- Design the texture in Inkscape.
- Create the object in Unity.
- Program the moving logic.
- Program the collision.
- Place them on the map.

4. Make the Player and Enemies Blink when damaged.
5. Create the shooting sound effects.
   - Design in Audacity and Bfxr.
   - Add them to Unity.
6. Create the Main menu with options.
7. Create the User interface items and their logic.
   - Healthbar: logic, material, objects, and texture in Inkscape.
   - Timer and logic.
   - Score and logic.
8. Add reward for killing Enemies, and balance their health points and speed.
9. Add map limiters for easier game design.
10. Extended the map created and added.

## B.5 Sprint 5: 18/04/2017 – 02/05/2017

1. Created the Enemy5.
   - Design the texture in Inkscape.
   - Create the object in Unity.
   - Program the moving logic.
   - Program the shooting logic.
2. Map done, awaiting feedback.
3. Main menu loads level1.
4. Main theme added to the game.
5. Create the fully working Pause and Lose menus.

6. Feedback obtained.
   - Change the map.
   - Modify the Player speed.
   - Improve the Blink function.
   - Use the same amount of damage for all enemies.
   - Add sound when the Player is damaged.
7. Create the Laser as secondary player attack.
   - Inkscape design, logic and colliders.
8. Neon2D shader rework and Neon2D texture shader created.
9. Configure the Xbox360 gamepad controls.
10. Make the game always have a 16:9 aspect ratio.
11. Start to write the Documentation.

## B.6 Sprint 6: 02/05/2017 – 16/05/2017

1. Boss
   - Boss Texture created in Inkscape.
   - Boss assembled in Unity.
   - Lasers and their logic created.
   - Canons and their logic created.
   - Shooting logic created.
   - Balance and timings created.
   - User interface done.
   - Spawn trigger created and working.
   - Particle effects for Boss spawn and Boss Lasers charge created.
2. Missing sounds created.
3. Music themes added: Main and Boss, and music control logic.
4. GameController PlayerPrefs done.
5. High-score done.
6. Continue with the Documentation.

## B.7 Sprint 7: 16/05/2017 – 30/05/2017

1. Game done.
2. Code cleanup, refactoring, bug fixes.
3. Continue with the Documentation,

## B.8 Sprint 8: 30/05/2017 - 20/06/2017

1. Finish the Documentation.

# Bibliography

[1] David J Anderson, *Kanban: successful evolutionary change for your technology business*, Blue Hole Press, 2010.

[2] Audacity, *Audacity - Free, open source, cross-platform audio software for multi-track recording and editing*, [Online] Available: http://www.audacityteam.org/, last accessed: june 2017.

[3] Chirs Roper, *The Games of Atari Classics Evolved: Part 2*, [Online] Available: uk.ign.com/articles/2007/10/22/the-games-of-atari-classics-evolved-part-2, last accessed: may 2017.

[4] Chris Jordan Barrish, *A Detailed History of Shoot 'Em Up Arcade Games*, [Online] Available: www.libertygames.co.uk/blog/a-detailed-history-of-shoot-em-up-arcade-games/, last accessed: may 2017.

[5] Colour Blind Awareness CIC, *Types of colour blindness*, [Online] Available: www.colourblindawareness.org/colour-blindness/types-of-colour-blindness/, last accessed: april 2017.

[6] Color Oracle, *Design for the Color Impaired*, [Online] Available: colororacle.org, last accessed: may 2017.

[7] Entertainment Software Association (ESA), *ESA Annual Report for 2016*, [Online] Available: www.theesa.com/wp-content/uploads/2017/05/ESA-AnnualReport-Digital-5917.pdf, last accessed: june 2017.

[8] GitHub Inc., *Github*, [Online] Available: www.github.com, last accessed: june 2017.

[9] Ian Beck, *Jets'n'Guns*, [Online] Available: www.insidemacgames.com/reviews/view.php?ID=695, last accessed: may 2017.

[10] Increpare Games, *Bfxr - Make sound effects for your games*, [Online] Available: http://www.bfxr.net/, last accessed: june 2017.

[11] Inkscape, *Draw freely | inkscape*, [Online] Available: https://inkscape.org/es/, last accessed: june 2017.

[12] Bernhard Jenny and Nathaniel Vaughn Kelso, *Color design for the color vision impaired*, Cartographic Perspectives (2007), no. 58, 61–67, [Online] Available: colororacle.org/resources/2007_JennyKelso_ColorDesign_hires.pdf , last accessed: june 2017.

[13] Jim Whitehead, *Game Genres: Shmups*, [Online] Available: classes.soe.ucsc.edu/cmps080k/Winter07/lectures/shmups.pdf, last accessed: may 2017.

[14] Craig Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*, Pearson Education India, 2012.

[15] Micro-Star Internationall Co., Ltd. (MSI), *Afterburner*, [Online] Available: www.msi.com/page/afterburner, last accessed: june 2017.

[16] Moby Games, *Genre: Scrolling shoot 'em up*, [Online] Available: www.mobygames.com/game-group/genre-scrolling-shoot-em-up/, last accessed: may 2017.

[17] National Institutes of Health, U.S. Department of Health and Human Services, *Facts About Color Blindness*, [Online] Available: www.nei.nih.gov/health/color_blindness/facts_about, last accessed: april 2017.

[18] Newzoo, *Global Games Market Will Reach $102.9 Billion in 2017*, [Online] Available: newzoo.com/insights/articles/global-games-market-will-reach-102-9-billion-2017-2/, last accessed: june 2017.

[19] Ken Schwaber and Mike Beedle, *Agile software development with Scrum*, vol. 1, Prentice Hall Upper Saddle River, 2002.

[20] Shoot Em Up, Wikia, *Shmup genre history*, [Online] Available: shmup.wikia.com/wiki/Golden_Age, last accessed: may 2017.

[21] Unity Technologies, *Unity - Game Engine*, [Online] Available: https://unity3d.com/es, last accessed: june 2017.

[22] VersionOne, Inc, *What Is Kanban? An Introduction to Kanban Methodology*, [Online] Available: www.versionone.com/what-is-kanban/, last accessed: june 2017.