Interative Graphics - Final Project
Space Defenders

Riccardo Vella - 1806809

September 2021

1 Introduction

Space Defenders is a game heavily inspired by the famous arcade game Space Invaders. The game, which takes place in space, consists in eliminating every enemy's spaceship before they reach the player or destroy its spaceship.
The game features Voxel based 3d models (made of cubes) which make the graphics feel modern while also resembling a classic, and older, style. An original soundtrack and sound effects were created to emphasize this nostalgic feeling in a modern and complex era.
In the next chapters we will go through all the technical aspects of the project, such as the external libraries and assets which were used, as well as the designing of the gameplay and as a description of all the interesting components of the game.

2 Environment

The game is built upon the Three.js library, which handles the majority of the graphics and some of the game logic, like the the game loop. The project also features Tween.js, responsible of the functioning of the majority of the animations in the game.

3 Assets

The game graphics is voxel based (everything is composed by small cubes) and many of the game objects are generated at runtime. A total of seven different models (with textures) of spaceships from the Itch.io website were used to design the player with its allies and the enemies in the game.
The soundtrack, which consists of one track, and the sound effects, a shooting effect and an explosion effect, were created from scratch.
Other assets include fonts, included the font-awesome library, and a logo, which is just a picture of the main spaceship.

4 Gameplay

When the game starts, an horde of enemies is generated. Each enemy moves towards the player while also steering and shooting at the same time. in addition to the enemies and the player, a group of allied ships is spawned in front of the player so that he can use these allies to protect himself from enemy attacks .The goal is to destroy every one of the enemy's spaceships before they reach or destroy the player. The player can move left and

right, or shoot.
If all enemies are destroyed the level ends and a new horde, more powerful than the last one, is generated. Following in the footsteps of many other arcade games, such as tetris, space invaders or pacman, this game has no end. The player tries to survive as much as possible and to obtain a better score.

4.1 The Spaceships

There are 7 types of different spaceships and each one has its peculiarities.

4.1.1 The Player

The player spaceship is obviously controlled by the player. It's able to shoot blue bullets and can move horizontally.

4.1.2 The Interceptor

The interceptor can absorb lots of damage before exploding, but it cannot move in any direction (an interceptor aircraft is in reality the opposite thing and the game counterpart as not to be considered related). It can be damaged by both the player and the enemies, so it is to be considered as a neutral spaceship. The player can use it as a barrier and get cover from the enemies bullets. At the beginning of the game 4 of them at spawned in front of the player and no other interceptor is spawned throughout the whole game.

4.1.3 The Basic Enemy

The basic enemies has low health and makes low damage, so it is not to be considered really menacing. It can be killed with one player's bullet in the first level.

4.1.4 The Power Enemy

The power enemy makes more damage and has more health than the basic enemy, but it's shooting rate is reduced.

4.1.5 The Rotating Enemy

The rotating enemy can rotate its body to be in two states: in the first state the rotating enemy will not be able to shoot, but will be more difficult to shoot at; in the second state the rotating enemy will be an easier target, but it will start shooting powerful bullets with high rate.

4.1.6 The Mothership

This enemy occupies the last row. Its bullets are very powerful and it has lots of health, but the really low shooting rate makes this spaceship less difficult to fight. Only three of them are spawned in their row.

4.1.7 The Bonus Spaceship

This spaceship will spawn randomly and will start moving horizontally behind the last row. It cannot shoot, and can be killed with one shot. The player will obtain bonus points and health if it kills the bonus spaceship before it gets out of reach.

4.2 How to play

The player interaction is minimal, here follows the list of the commands:

- **A** - Move left;
- **D** - Move right;
- **Arrow Up / Left Click** - Shoot;
- **Esc** - Pause the game.

The commands for the mobile version of the game are discussed in Section 9.

5 Entities, Objects and Collidables

In order to build a modular and reusable project, three important concepts have been created that make interactions and management of most of the game components possible: *Entity*, *Object* and *Collidable*. Almost every component of the game is considered an *Entity*, which is a component that can be updated and is identified with a unique id. But while entities are not very complex, an *Object* is an *Entity* which implements physics, from scratch, and that can be shown on the screen. Collision detection was also implemented from scratch, introducing the *Collidable*, which is an *Object* that has an $HitBox$ and is able to collide and be collided. It has also a *durability* and can die.
Here follows a list of the most important components in the game.

5.1 Spaceship

A *Spaceship* is a *Collidable* and the whole includes the player, enemies and allies. Each one of these types has different meshes, which are loaded from the assests. It can be in any of these 4 states exclusively, which will decide the current animation (see Section 7): *idle*, $movingleft$, $moving_right$, *stabilizing*. It can shoot with a maximum rate that depends on a certain $shooting_delay$ and when it does it spawns a *Bullet* with a certain force, that depends on its $cannon_power$. Lastly, it generates an *Explosion* before dying.

5.2 Bullets

It is a simple *Collidable*, which only has a *color*. It dies when it hits something or when it reaches a certain position on the z axis (when it does not hit nothing and goes really far). Its mesh is just a box generated at runtime.

5.3 Explosions

An *Explosion* is generated every time a *Spaceship* dies. It itself spawns a set of *Debris* objects, which are just boxes, and imprints them a force with random direction and variable magnitude. Each *Debris* fade out and dies after a certain amount of time, defined as $life_time$.

5.4 Sky

The *Sky* exists purely for an aesthetic reasons. It generates *Star* objects in a random position that lies on the circumference with a certain radius and z-axis position, and imprints on them a variable force. The stars will travel towards the player with different speeds, giving the feeling that they are at different distances from the player. Once they reach a certain position on the z-axis, which is out of the player's field of view, they die and new stars are generated by the *Sky*. Every *Star* has the same mesh, which is a white box, generated at runtime once.

6 Physics and Collisions

The physics consists in the application of simple concepts, that have been implemented from scratch, such as the laws of motion and the force of friction. As seen in Section 5, the type *Object* implements all of the physics.
An *Object* can be moved in space by applying a force, for this reason each *Object* has a *mass*, a *position* in space, a *velocity* and an *acceleration*. When a force is applied, the *acceleration* changes accordingly to the equation $a = \frac{F}{m}$. On each update, the new *velocity* is computed as $v' = v + a\Delta t$, where Δt is the amount of time passed since the last update, and the next *position* is computed as $p' = p + a\Delta t$.
Using this equations, an object will never stop its motion unless other forces are applied, which is what the first law of motion says. This is also what should actually happen in space, but, to make the game more enjoyable, the force of friction has been added too and it's computed using an approximation of the drag equation, as $F_a = v^2\mu$, where v is the velocity of the *Object* and μ is its friction coefficient. So, each *Object* has its friction coefficient, which defines the aerodynamics of it.
As seen in Section 5, a *Collidable* implements an interface for the collision detection, using an $HitBox$. An $HitBox$ h is a set of boxes and checking for its intersection with another $Hitbox$ h' means checking for the intersection of every box of h with any of the boxes of h'. A *Collidable* has an interface that allows to generate a custom $HitBox$ or modify an existing one, but is also able to generate a default one that fits the mesh in the best way with a single box.
To actually detect collisions with other objects, a *Collidable* has to subscribe to the $CollisionManager$ in one of the three possible groups:

- *player*, can collide with enemies and neutrals;
- *enemy*, can collide with players and neutrals;
- *neutral*, can collide with players and enemies.

This classification was done in order to check as few intersections as possible.

7 Animations

Each *Object* has a set of animations that can run at the same time, the interface allows to add animations and to run the animations an *animate* method has to be called in the *update* function. Here follows a description of all the implemented animations:

7.1 Spaceships Animations

Each *Spaceship* will be animated when moving or when stabilizing after a movement. These animations simply consist in tilting the spaceship to the left or to the right and on updating the $HitBox$, for this reason each spaceship defines a *tiltangle* and a *tiltspeed*, on which the final animation depends.

7.2 Other Animations

There are two other animations:

- The player's ship will have turbulence for the duration of the game. This animation consists in imprinting random forces to the spaceship.
- A special type of enemy (*RotatingEnemy*) will keep rotating by 90 degrees and back every 6 seconds. On its initial position the spaceship cannot shoot, but it is also difficult to shoot at it. While, when rotated, the spaceship can shoot (and it is really powerful) but it's easier to shoot at it.
- The *Camera* is animated when transitioned (see Section 8).

8 Camera

The *Camera* is an extension of the class *PerspectiveCamera* from three.js, but it can be transitioned from one position to the other and can also follow an object (only on the x-axis). The transitioning is used while switching from menu to game and vice-versa and it can not only transition the position of the camera in space, but also the point to which the camera is looking at. The $follow$ function is used to make the camera follow the player on the x-axis. This will not cause the camera to always share the same x position as the player, but it will gently transition the camera moving with a speed which is proportional to the distance of the player from the camera. In practice, the camera will allow the player to be off center by a bit.

9 Mobile Support

Space Defenders was not originally intended as a mobile game, but the simplicity in user interaction made it adaptable for mobile support. The menus and the user interface are responsive to all sizes and adapt really well to the mobile screen size, so the only real problem was in designing new commands for the game.

9.1 Commands

In particular, the player must be able to move left and right, he must be able to shoot and also pause. Here follows a list of the commands for the mobile version:

- **Swipe Right** - Move right;
- **Swipe Left** - Move left;
- **Touch** - Shoot.

The pause command is replaced by a pause button on the top-left corner of the screen, which is visible whenever the game is playing.
This system has some obvious problems: you can't move without shooting. I found this limitation to not have much influence on the gameplay, but for this reason the mobile version is still to be considered as an incomplete adaptation of the original game.

Figure 1: The two versions of the game. In the mobile version the menu adapts to the size and the commands for the games change.

Figure 2: In the mobile version, when the game is playing, the interface adapts to the size and the pause button is displayed in the left-top corner of the screen.