Interactive Graphics

La Sapienza

A.Y. 2018/2019

The War of the Meteorites

Alessandro Giannetti 1592006
Edoardo Bini 1651353
Marco Ferraro 1854250

1. Introduction

“The war of the meteorites” is a game set in the Star Wars universe, in which the player impersonates a pilot struggling to survive a wave of asteroids flying on a spaceship through empty space.

The player controls the spaceship with the mouse and can left click to shoot and destroy incoming meteorites to avoid taking damage. If the player survives the wave of meteorites, a new threat is presented: an enemy spacecraft will appear on the screen attacking the player with lasers and the objective becomes to destroy the enemy.

Before going into the technical details of the project, we wanted to warn that it will not work correctly if loaded using GitHub pages on Google Chrome.

There seems to be a problem with loading the model from the folder "Spaceships". The returned error on the console is "infinity% loaded" and this seems to block the game.

Ultimately, this looks to be a bug of some sort, for which we have no control over. Therefore, we suggest **running the game on Mozilla Firefox.**

As a final remark, if the project is downloaded and run locally, it will work on both Chrome and Firefox.

1.1. Libraries and Tools:

- Three.js
- OrbitControls.js
- LegacyJSONLoader.js

1.2. Models:

- TIE Fighter
- ARC 170

2. Main Menu

The game grants the player multiple options and an “instructions” button (the question mark on the bottom right corner) on the main menu.

The player can choose the difficulty of the game and the model of the spacecraft.

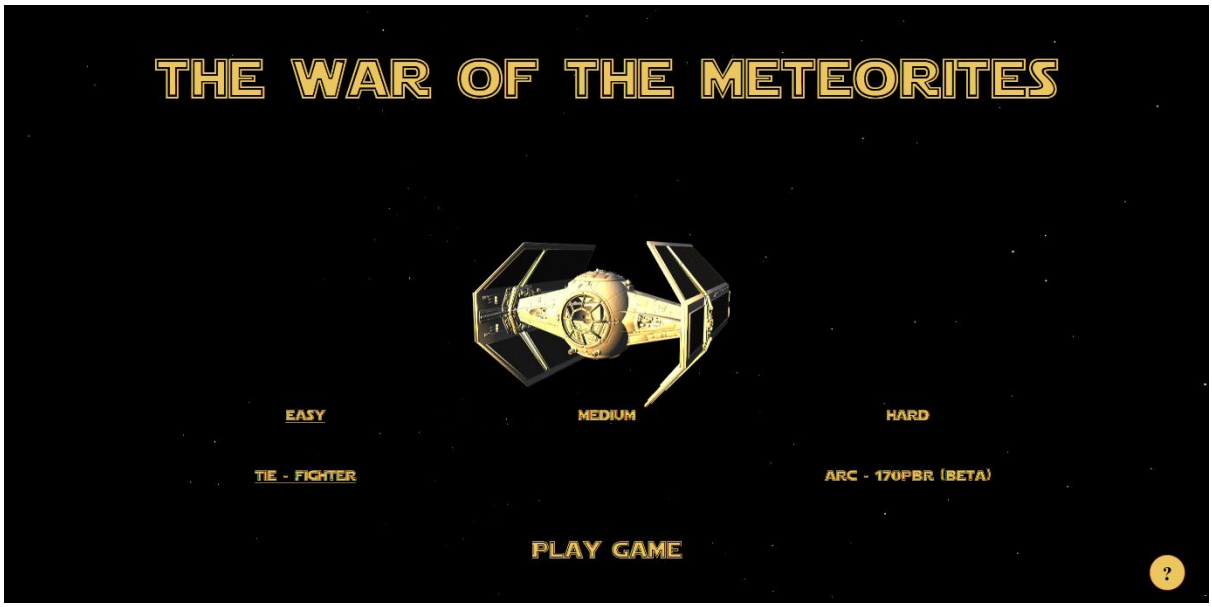

Figure 1: Main Menu

Changing the difficulty will spawn more meteorites and make them faster changing model will also change the position from which the players shoots his projectiles.

In addition, by increasing the difficulty level, the ship will suffer more damage and will require more points to reach the boss.

3. Scene

The scene is composed of many stars generated randomly in the background, the Sun and the Earth that serve as scenery and many asteroids generated randomly that move in the direction of the screen, coming towards the player.

3.1. Scenery generation

3.1.1. Stars

Figure 2 Stars

Stars are generated as spheres at random positions in a value range and the ones too close to the player are moved away with a simple control function.

```
var geometry = new THREE.SphereGeometry(0.5, 32, 32);
var material = new THREE.MeshBasicMaterial({color: 0xffffff, reflectivity: 0.1});
var sphere = new THREE.Mesh(geometry, material);
// This time we give the sphere random x and y positions between -500 and 500
sphere.position.x = Math.random() * 3000 - 1500;
sphere.position.y = Math.random() * 3000 - 1500;
sphere.position.z = Math.random() * 3000 - 1500;
```

We thought it appropriate to create about 1000 spheres to simulate a starry sky, and to make them look like objects with their own light source we used "**MeshBasicMaterial**" so that they are not affected by other light sources.

As for the repositioning of the stars if too close to the player we have simply made a translation on the x-axis and y-axis.

```
if (sphere.position.z < 0 && sphere.position.z > -200 && sphere.position.x > -80 && sphere.position.x < 80 ) {
   sphere.position.z = sphere.position.z + 210;
   sphere.position.x = sphere.position.z + 85 * Math.sign(sphere.position.x);
}
if (sphere.position.z >= 0 && sphere.position.z < 200 && sphere.position.x > -80 && sphere.position.x < 80) {
   sphere.position.z = sphere.position.z + 200;
}
```

3.1.2. Meteorites

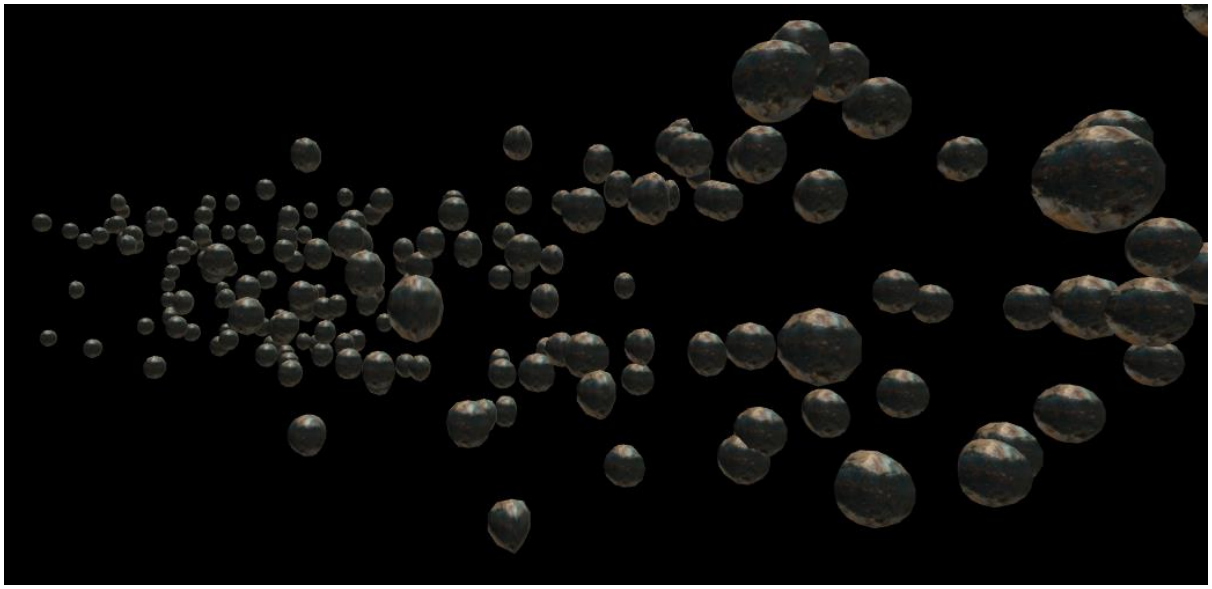

Figure 3 Asteroids - perspectives view

Meteorites are generated in a similar way, looping through the z coordinate, creating asteroids at each iteration in random x and y positions in a value range handled by the sum of random numbers (Bates' distribution), also randomizing the geometry of the sphere and adding the texture.

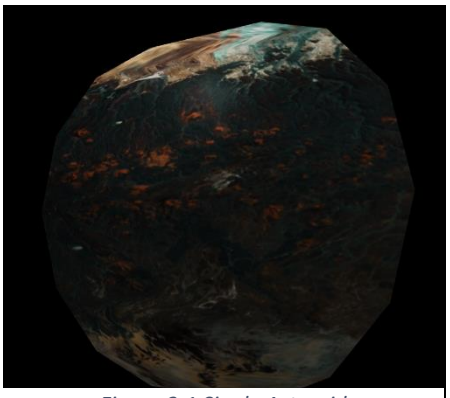

Figure 3.1 Single Asteroid

```
textureLoader.load('./texture/asteroid.png', function (texture) {
  var geometry = new THREE.SphereBufferGeometry(radiusAsteroids, Math.random() * 4 + 3, Math.random() * 6 + 5); // 3 32
  var material = new THREE.MeshPhongMaterial({map: texture, reflectivity: 0.1});
  var asteroid = new THREE.Mesh(geometry, material);

  // This time we give the sphere random x and y positions between -1000 and 1000
  asteroid.position.x = ((Math.random() + Math.random()) / 2) * 2000 - 1000;
  asteroid.position.y = ((Math.random() + Math.random()) / 2) * 1000 - 500;
  // Then set the z position to where it is in the loop (distance of camera)
  asteroid.position.z = zPosition;
```

3.1.3. Sun and Earth

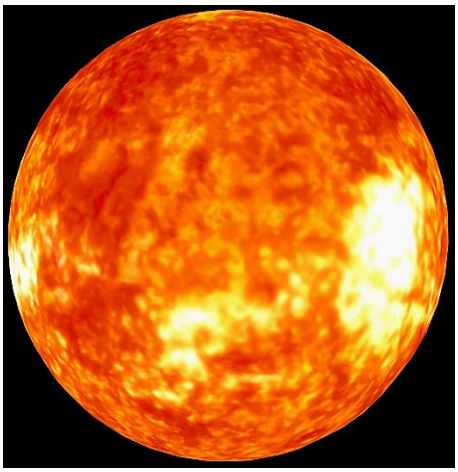

Figure 4 the Sun texture

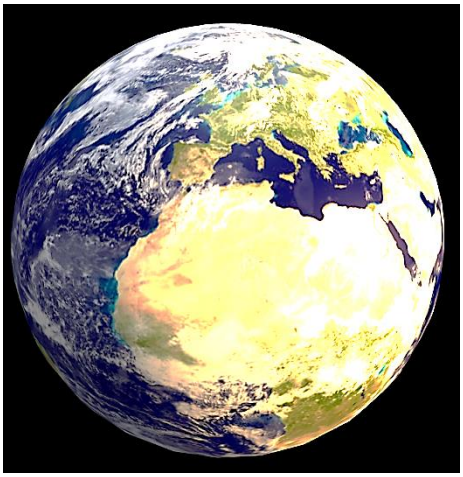

Figure 5 the Earth texture

The Sun and the Earth are simple texturized spheres.

```
textureLoader.load('./models/texture/sun.jpg', function (texture) {
    var geometry = new THREE.SphereBufferGeometry(340, 32, 32);
    var material = new THREE.MeshBasicMaterial({map: texture});
    var sphere = new THREE.Mesh(geometry, material);
```

3.2. Spaceship model generation

The model of the player-controlled spaceship is loaded into the scene as a JSON file (with ObjectLoader) depending on the selected ship by the player.

```
objectLoader.load("./models/" + name + ".json", function (obj) {
    scene.add(obj);
}
```

As mentioned earlier, we thought it would be nice to give the user the opportunity to choose their own spaceship. The models we loaded are as follows:

- TIE fighter
- ARC 170

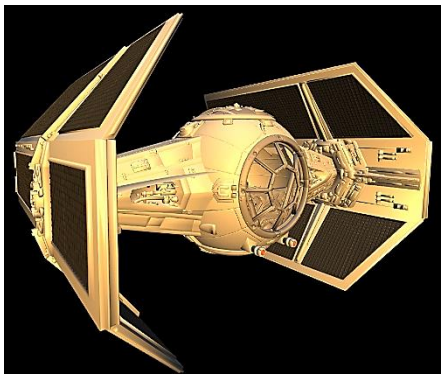

Figure 6 TIE FIGHTER

Figure 7 ARC 170

3.3. Boss Model

The boss spaceship is instead a *hierarchical model* stored in a separated JavaScript file (**spaceShipBossModel.js**), created with the use of the Three JS editor, which handles the nodes generation, and it is spawned on the scene whenever the player achieves enough points, depending on the level selected:

- EASY: 1500 points
- MEDIUM 3000 points
- HARD 4500 points

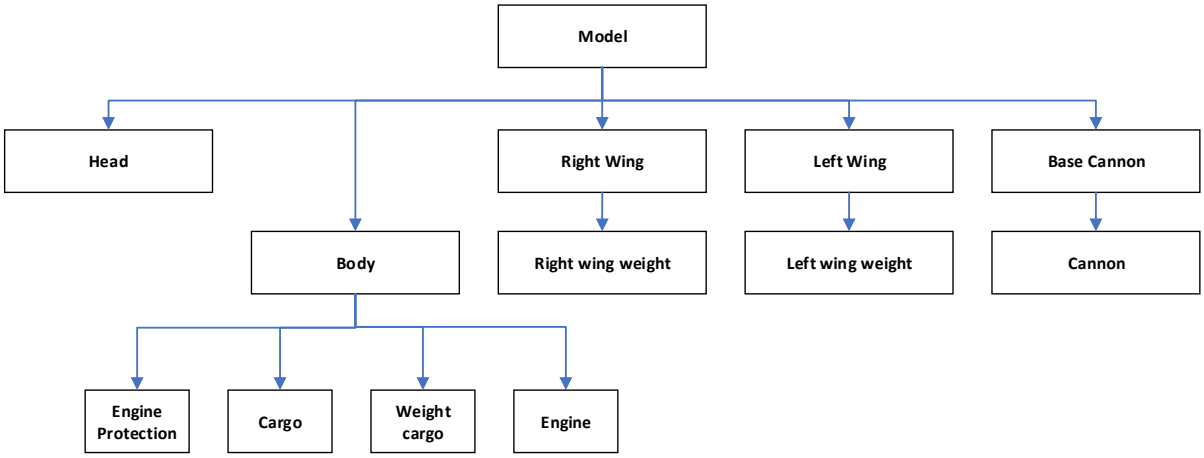

Figure 8 hierarchical model

Figure 9 front side view

Figure 10 right side view

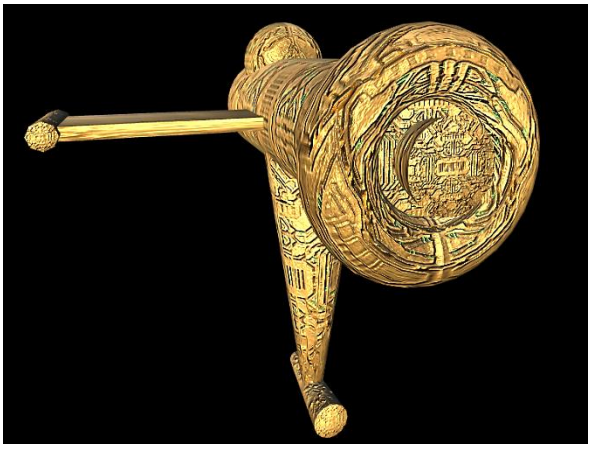

Figure 11 back side view

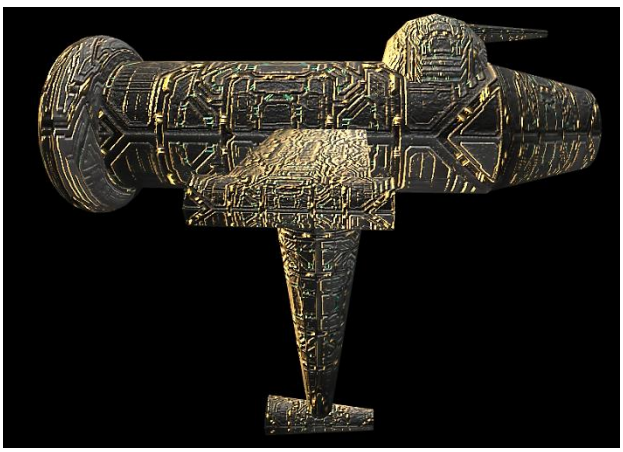

Figure 12 left side view

4. Lights

There are multiple lights in the scene generated using the appropriate ThreeJS methods:

- An ambient light, to give some uniform luminosity to the scene.

```
var light = new THREE.AmbientLight(0x404040); // soft white light
scene.add(light);
```

- A directional light placed inside the Sun model, to somewhat resemble a realistic Sun casting light onto the scene.

```
var sunLight = new THREE.PointLight(0xffb04a, 4, 0, 2);
sunLight.position.set(534, 200, -966);
sunLight.add(sun);
scene.add(sunLight);
```

- A point light just above the player's spaceship model, to give it better visibility.

```
var directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
scene.add(directionalLight);
```

5. Animations

Most of the objects in the project are animated to make the scene more believable, here are the main components:

5.1. Meteorites

The animation of the meteorites simply consists in a constant movement on the Z axis towards the player and rotation across the other axis depending on the meteorite id.

Whenever the meteorites surpass the player model, their z coordinate is reset with a new random X and Y position to ensure variety.

The game also checks if the meteorites have been hit by the player's lasers using collision detection based on Euclidean distance (taking into consideration the radius of the sphere). In such case, the meteorites are reset to the starting Z position.

5.2. Spaceship

The player's model is animated depending on the mouse position, which directly affects the position of the ship and its rotation along the Z axis, to give a better sense of a direction change.

If the player has turbo activated, the ship is also moved forward quickly and returned to the initial position when the turbo is released.

When the player shoots, green cylinders are generated and moved towards the meteorites at high speed starting from the player's position.

5.3. Boss

The boss spaceship is moved according to the player's position along the Y-axis, to give the impression that the enemy is reacting to the player's choices.

Moreover, its cannons are constantly moved to be directed towards the player, so that the bullets represent a real threat to be constantly avoided.

The wings of the spaceship also rotate depending on the Y direction it is going towards.

The bullets are constantly being generated from the cannons' position. As with the player's model, the bullets are high-speed cylinders (red this time) that move towards the player and have collision detection.

5.4. Death/Victory

Whenever the health bar of either the player or the boss is completely depleted, a short death/victory animation plays.

The models move in the space and rotate accordingly to the position they were in before the animation was triggered for 8 seconds, which is the time for the models to disappear from the screen and be fluid while doing it.

In the case of the boss model death animation, the wings and the turret are animated separately, detaching from the body.

6. Sound

Sound represents an important component of the scene, almost every action has a related sound:

- Background music is always active during the game to immerse the player into the Star Wars world.
- The player's spaceship produces a looped sound that is directional depending on the model's position.
- Shooting, pressing the instruction button, triggering the boss battle and destroying the boss/meteorites all provide sounds to go along with them.

to play an audio track we have implemented the following function:

```
function playSound(name, loop, positional) {
// create an AudioListener and add it to the camera
    var listener = new THREE.AudioListener();
    camera.add(listener);
// create the PositionalAudio object (passing in the listener)
    var sound = new THREE.PositionalAudio(listener);
// load a sound and set it as the PositionalAudio object's buffer
    var audioLoader = new THREE.AudioLoader();
    audioLoader.load('sounds/' + name + '.mp3', function (buffer) {
        sound.setBuffer(buffer);
        sound.setLoop(loop);
        sound.setRefDistance(20);
        if (name === "TIE-fighterExplode" || "TIE-fighterFire")
            sound.setVolume(0.4);
        sound.play();
    });
```

7. User Interface

The user interface is superimposed onto the scene using HTML and CSS.

It's mainly composed of text/buttons:

Title, difficulty options, spaceship model selection, instructions menu, victory screen and pause screen and finally health bar and boss health bar, which are dependent on values handled in the JavaScript with the collision detection system described above.

There is also the pause menu which is composed of the pause text and a main menu button.

8. User Interactions

In summary, the interactions that the user can make are many:

- **Mouse movement**: through the movement of the mouse the user can move the spaceship on the y-axis and x-axis.
- **Left mouse click**: by clicking the left mouse button the user can shoot lasers.
- **ESC button**: By clicking the ESC key the user can pause/resume the game or decide to go back to the main menu from the pause screen if necessary.
- **T button**: by clicking the T (Turbo) button, the user will have the opportunity to give a boost to their spaceship, making it accelerate.
- **G button**: by clicking the G (God mode) button, the spaceship will become immune to any damage. This feature has been implemented so that it's easy to test the entire game. When God mode is activated, the life bar of the spaceship will turn blue, to signal that it is now invulnerable.

9. Conclusion

“The War of the Meteorites” uses many different elements to realise a simple game that includes all the requirements of the assignment and more.

We tried to achieve the best possible result by combining the techniques learned during the course while also managing the limited resources available in a web environment.

The mechanics of the game are compelling enough to create an enjoyable experience overall, and building it provided us with precious insight to deal with interactive graphics projects for the future.

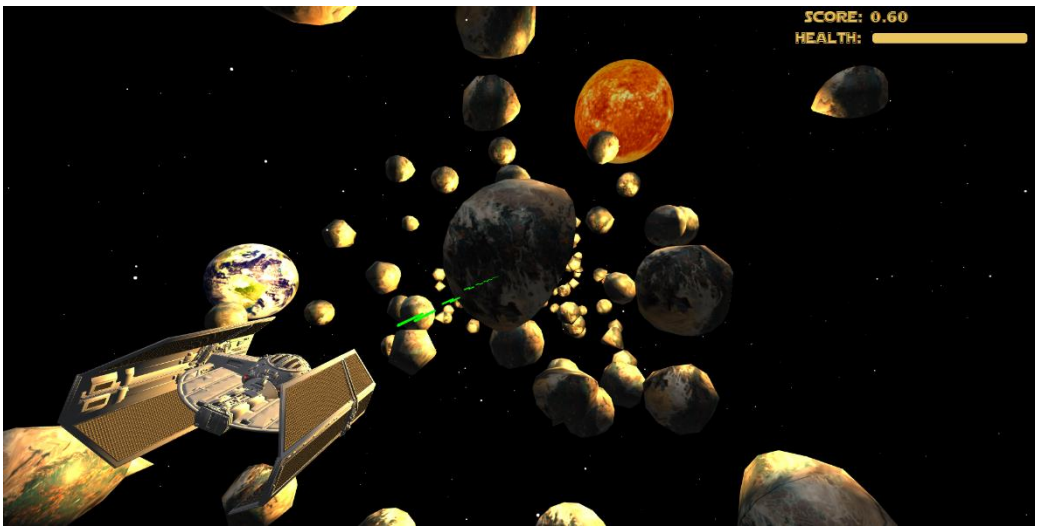

Figure 13 player vs asteroids