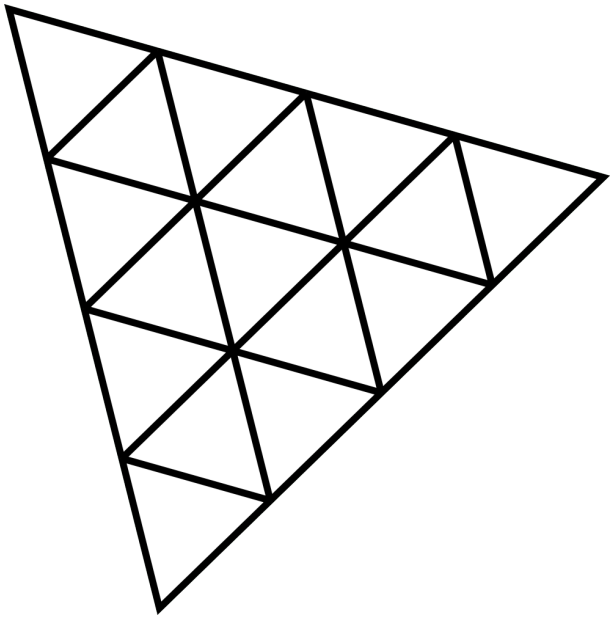
Interactive Graphics - Final Project

Surfer Odyssey

Davide Renzetti

1838417



Contents

1 Introduction **2**
1.1 Requirements 2

2 The Game **3**

3 Libraries and Code **5**
3.1 Main.js 5

4 Models and Textures **7**
4.1 Robot 7
4.2 Surf 7
4.3 Lanes 8
4.4 Obstacles and Points 8

5 Animations **9**
5.1 Robot 9
5.2 Cubes 9

6 Lights **10**

7 Music **10**

8 Browser Testing **10**

1 Introduction

Surfer Odyssey is a endless runner game. Some elements of the game are inspired by the movie "2001: A Space Odyssey".
The player controls a robot with a surf, and he has to avoid obstacles and collects red and yellow cubes. The game ends when the player hit an obstacle. It is written in javascript, using Three.js.
The game is playable here.

1.1 Requirements

- **Hierarchical model:** a model of a robot was imported; see the section "Models and Textures" for more informations;
- **Lights and Textures:** three different lights and some different textures have been used; see the sections "Models and Textures" and "Lights" ;
- **User interaction:** the player can control the robot, turn on/off the music and change the game speed; see the section "The Game";
- **Animations:** the robot has different animations; see the "Animations" section;

2 The Game

The first screen is a menu, where the player can disable/enable the music, learn the commands, change the speed of the game or start the game.

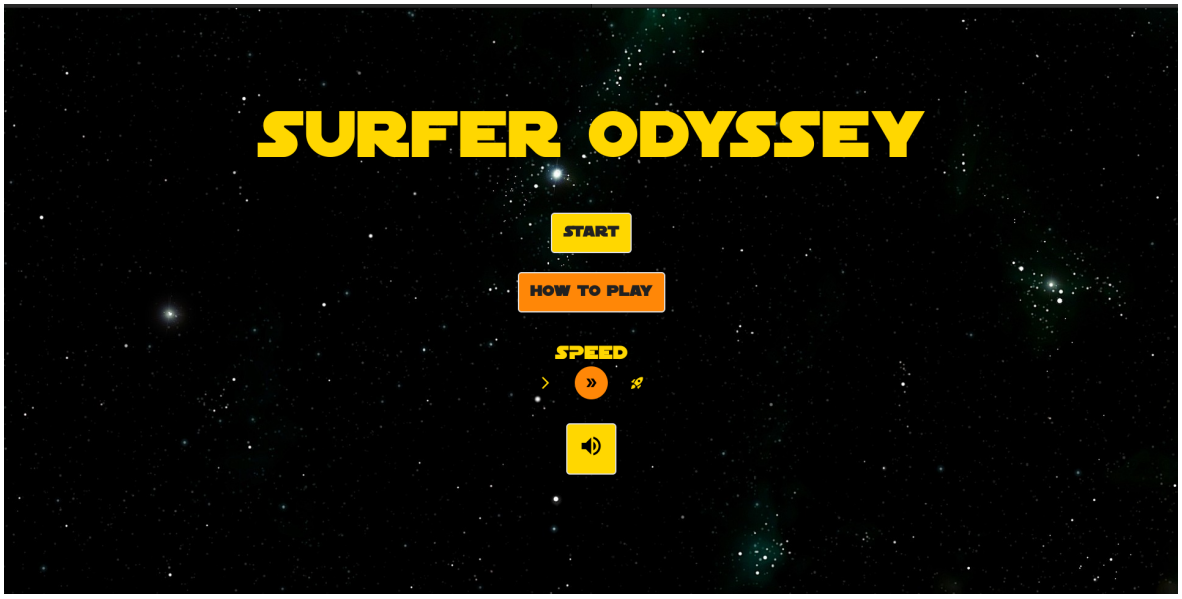


Figure 1: Menu

The button "HOW TO PLAY" leads to a page where the player can learn the commands.

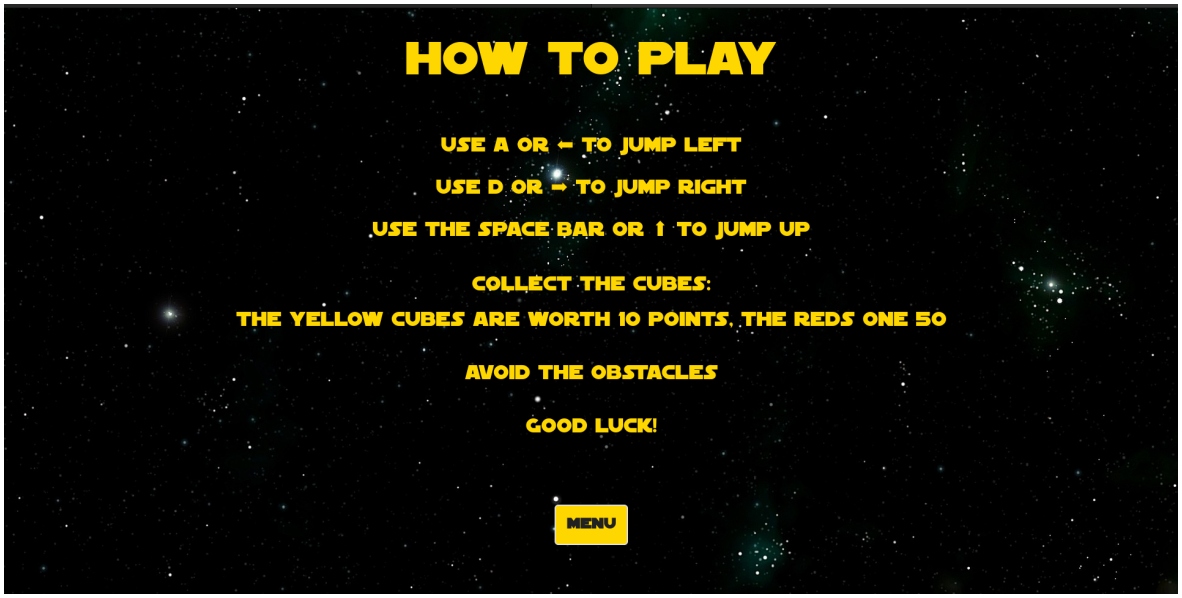


Figure 2: How to Play

When the player press start, the game starts; the player can control the robot with the keys shown in the "HOW TO PLAY PAGE". In the scene, we can see three lanes, each one with obstacles and yellow and red cubes. In the background there is the universe, with some planets that become closer and closer as the game go on. In the right corner of the screen, the score is present, divided in three subcategories: total score, distance and points.

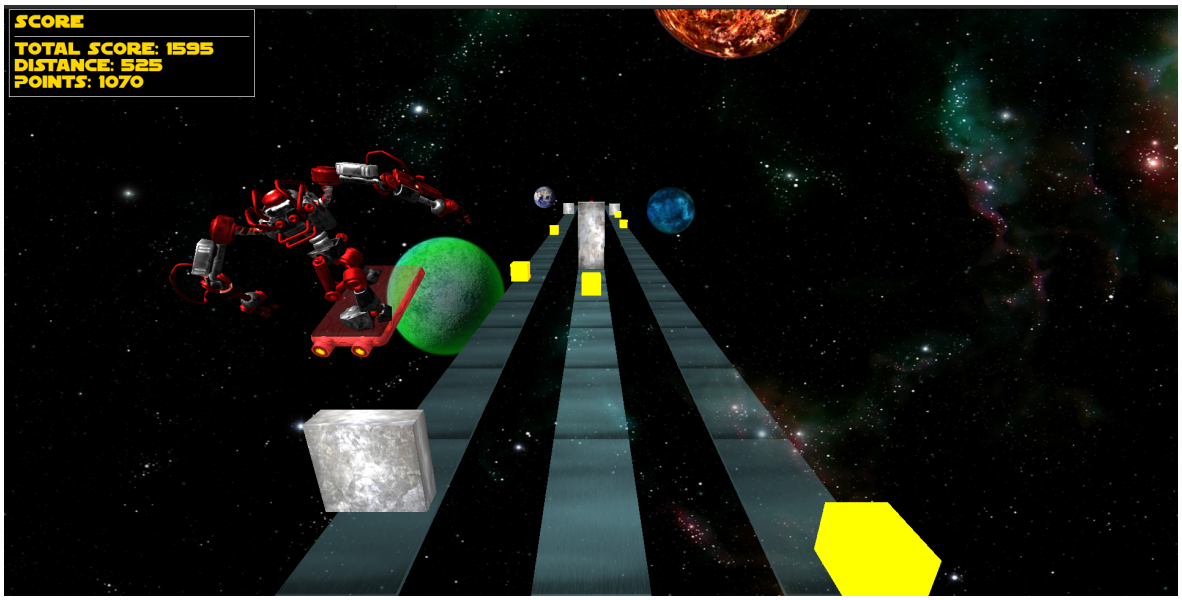


Figure 3: The Game

If the player hit an obstacle, the run ends, and a game over screen appears. Here, the player can see the score of the run, play another game or go back to the menu.

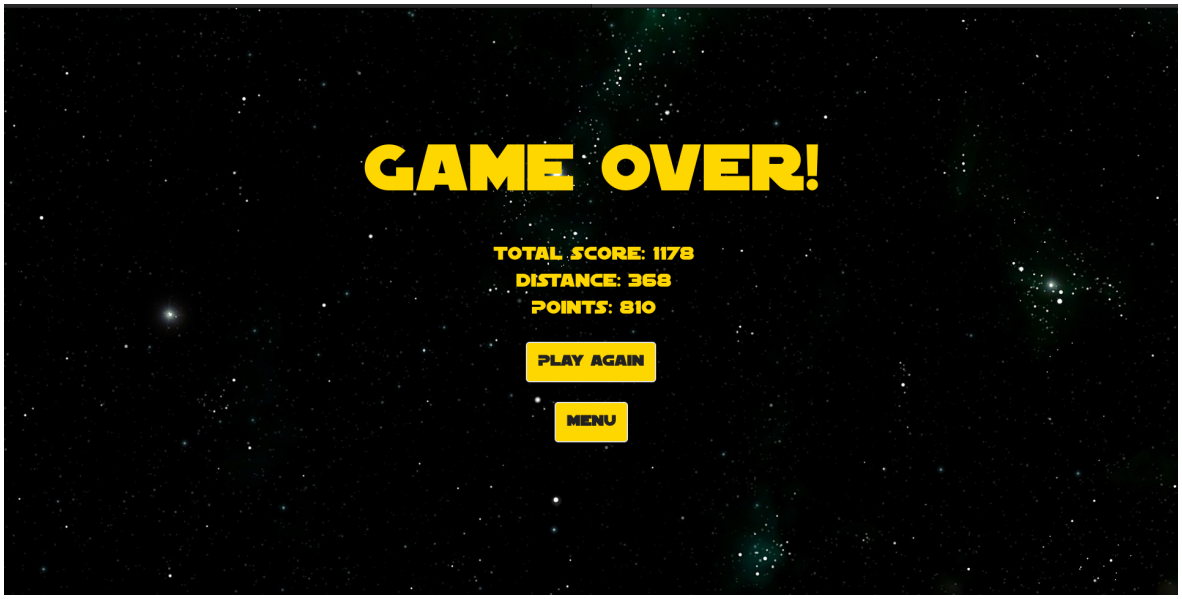


Figure 4: Game Over screen

3 Libraries and Code

The code of the application is organized in three main file:

- **index.html:** it contains the html code;
- **main.css:** in this file there is all the css code;
- **main.js:** the code that makes the game working;

The following external code has been used:

- **Three.js:** the main library;
- **GLTFLoader:** a module used to load external 3D models;
- **RoundedBoxGeometry:** used to create object with rounded edges, it was used to make the surf model;
- **Tween.js:** it permits to make smooth animations;

3.1 Main.js

In the main.js file,the function createScene() initialize the scene. This function is responsable to set the position of the camera, create the lights, load the audio and the background; it also create all the 3D objects of the scene. The objectCreator class was used to make the objects of the scene:

```
class ObjectCreator {

  static createSurf(){
      ...
  }

  static createLane(x){
      ...
  }

  static createPoint(){
      ...
  }

  static createObstacle(i){
      ...
  }

  static createLight(color, intensity, position){
      ...
  }
}
```

The animations are handled by the Animator class:

```
class Animator {

  static jumpUp(object, rotate){
    ...

  }
```

```

  static jumpLeft(object, rotate){
    ...
  }

  static jumpRight(object, rotate){
    ...
  }

  static idleAnimation(){
    ...
  }

  static legRotation(){
    ...
  }

  static elbowRotation(val,time){
    ...
  }

}
```

The illusion of an infinite run works in this way: the character and the lanes are stationary, while the obstacles and the points are moving towards the character; also the texture of the lanes is moving. The movement of the objects is performed by the updateObjectsPosition() function, while the texture movement is done by the render() function. The objects are not deleted and recreated constantly, because this would be computational costly, but the same objects are reused: in fact, when an object reach the end of the scene, that object is moved to the other side of the scene in a random way:

```
if(z_pos > 10){
        child.position.z = -objectsParent.position.z-85-Math.floor(Math.random()
    * 160);
        child.position.x = lanes[Math.floor(Math.random() * 3)];
}
```

The function checkCollision() is responsible for the collision between the character, the obstacles and the points. The infinite movement of the planets on the background is possible due to the functions setIntervals() and updatePlanets(). Every time that the player restart a new game, the function resetGame() reinitializes the variables and move all the objects to a coherent position.

4 Models and Textures

4.1 Robot

A model of a robot was imported from Sketchfab. Then, the color of its texture was modified with GIMP, to have a better contrast with the background. This model has a complex hierarchical structure.

Figure 5: The orignal model of the robot

4.2 Surf

A model of a surf has been created with Three.js; it is composed by 5 components: a rounded parallelepiped and 4 cilinders (2 for the reactors and 2 for the fire of the reactors). All the components have a Phong material composed by a basic color and a texture. The parallelepiped and the reactors have a red color and the same static texture. The fire of the reactors has dinamic texture created with a video:

```
var fireVideo = document.createElement("video");
fireVideo.src = 'images/fire.mp4';
fireVideo.loop = true;
fireVideo.muted = true;

    ...

var fireTexture = new THREE.VideoTexture(fireVideo);
fireTexture.format = THREE.RGBAFormat;
fireTexture.minFilter = THREE.NearestFilter;
fireTexture.maxFilter = THREE.NearestFilter;
fireTexture.generateMipmaps = false;

const reactor_light_material = new THREE.MeshPhongMaterial({
  color: 0x11ffEee,
  map: fireTexture,
});
```

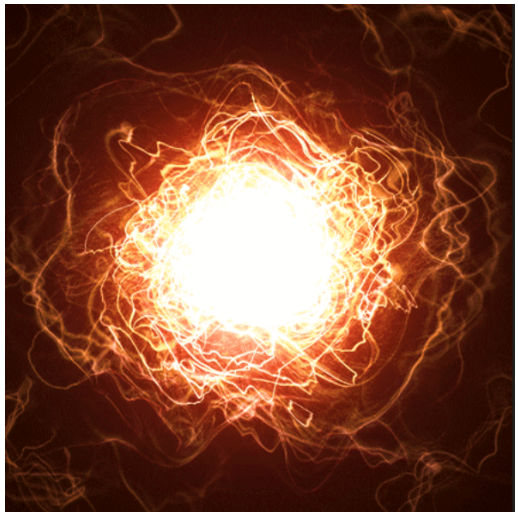


Figure 6: A frame of the dinamic fire texture

4.3 Lanes

The three lanes are parallelepipeds with the following material on them (the value of the variable "tras" changes between the middle and side lanes):

```
const material = new THREE.MeshBasicMaterial({
      color: 0xffffff,
      map: lanes_tex,
      transparent : true,
      opacity: tras,
});
```

Every minute, the lanes change color.

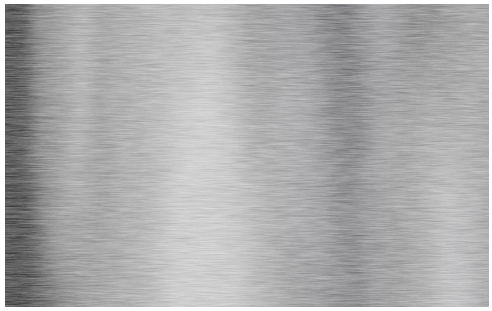


Figure 7: The texture of the lanes

4.4 Obstacles and Points

There are two types of obstacles: low and high; both of them are parallelepipeds with a metal texture:

```
const material = new THREE.MeshPhongMaterial({
      color: 0xffffff,
      map: tex,
});
```

Figure 8: High obstacles texture

Figure 9: Low obstacles texture

The points are yellow and red cubes:

```
const geometry = new THREE.BoxBufferGeometry( 1, 1, 1 );

    ...

const material = new THREE.MeshPhongMaterial({
      color: c,
});
```

5 Animations

All the animations are handled by the animator class. The animations are made with the help of Tween.js.

5.1 Robot

Constant Animation The robot has a constant animation: it move the body and the arms left and right.

Jump up When the the space bar or the up arrow key are pressed, the robot jump. During the jump, the arms and the elbows flex inward; also, the joints of the legs and the ankles perform a rotation upwards. Sometimes, in a random way, the robot twirls around.

Jump Left and Right When the right keys are pressed, the character jump on the adjacent lane; while doing this, the upper body slightly tilts. Also in this case, randomly, the character make a twirl.

Other Every 11 seconds, the robots rotates his elbow; also, every 9 seconds, it rotates his ankle.

5.2 Cubes

The cube have a simple animation: they float up and down.

```
static cubesMovement(obj){
    cubeCont += cubeMovement;
    if(cubeCont == 530 || cubeCont == -1) cubeMovement = -cubeMovement;
    obj.position.y += 0.015*cubeMovement;
}
```

6 Lights

Three different lights have been used: two directional lights and one ambient light. The ambient light give a constant illumination to all the scene, while the two directional light are needed for a better illumination near the character.
The two directional lights have the following parameters (color, intensity, position):

```
ObjectCreator.createLight(0xffffff, 0.95, [-3,7,5]);
ObjectCreator.createLight(0xffffff, 0.2, [-5,7,5]);
```

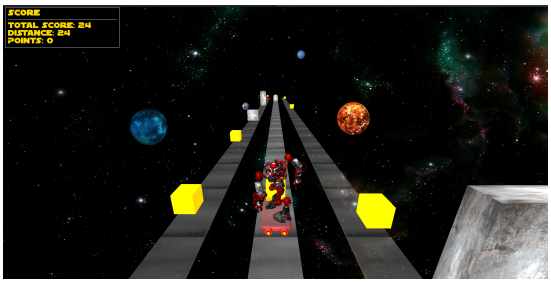


Figure 10: Ambient Light on

Figure 11: Ambient Light off

7 Music

The music used during the game is "The Blue Danube" by Johann Strauss II. The player can turn the music on/off, using the corresponding button in the menu.

```
camera.add( listener );
var audioLoader = new THREE.AudioLoader();
audioLoader.load( 'sounds/the_blue_danube.mp3', function( buffer ) {
    sound.setBuffer( buffer );
    sound.setLoop(true);
    sound.setVolume(0.75);
    loaded++;
},
          function ( x ) {},
          function ( err ) {
              console.log( 'Error' );
          }
);
```

8 Browser Testing

The game was tested on the following browsers, both locally and with github pages:

- Mozilla Firefox 102.0
- Google Chrome 103.0.5060.53
- Brave 1.40.109