SAPIENZA
UNIVERSITÀ DI ROMA

INTERACTIVE GRAPHICS

Course from

Artificial Intelligence and Robotics

FINAL PROJECT presentation

A.A. 2020/2021

GUARDIANS OF THE UNIVERSE

Submitted by

Lorenzo Zuccari

1611757

Index

0. User manual 2
1. Introduction 3
2. Scene 4
3. User Interaction 5
4. Models 7
5. Lights 8
6. Animations 9
00. Technical chart 10

ENVIRONMENT 10
LIBRARIES 10
MODELS 11
TEXTURES 11
OTHERS 11

User manual

A simple user manual illustrating the commands of the game.

At the lunch of the game, is strongly recommended to set the zoom of the page according to the screen width and resolution, in order to improve the overall experience.

The program was tested on all the main web browsers. To better enjoy the game is suggested to use the **Microsoft Edge** (or Chrome) browser.

1. Introduction

The project aims to reproduce a simple endless running game. The main character, controlled by the player, needs to move and jump in order to avoid the obstacles encountered during its run inside each level of the game, with the goal of maximizing the number of points obtained during the match.

The theme of the game is inspired by the classic adventure TV series and games from the '80s, characterized by a mixture of fantastic medieval elements and stimulating electronic music and in particular Chiptunes.

The work was completed using WebGL, HTML and JavaScript languages. In particular, the THREE.js library was used as the foundation of the project. This is a JavaScript library used to manage the creation and manipulation of the 3D objects in the game and allowed to create and display 2D and 3D computer graphics through web browser in a more performing, easy and fast way, with respect to classic WebGL.

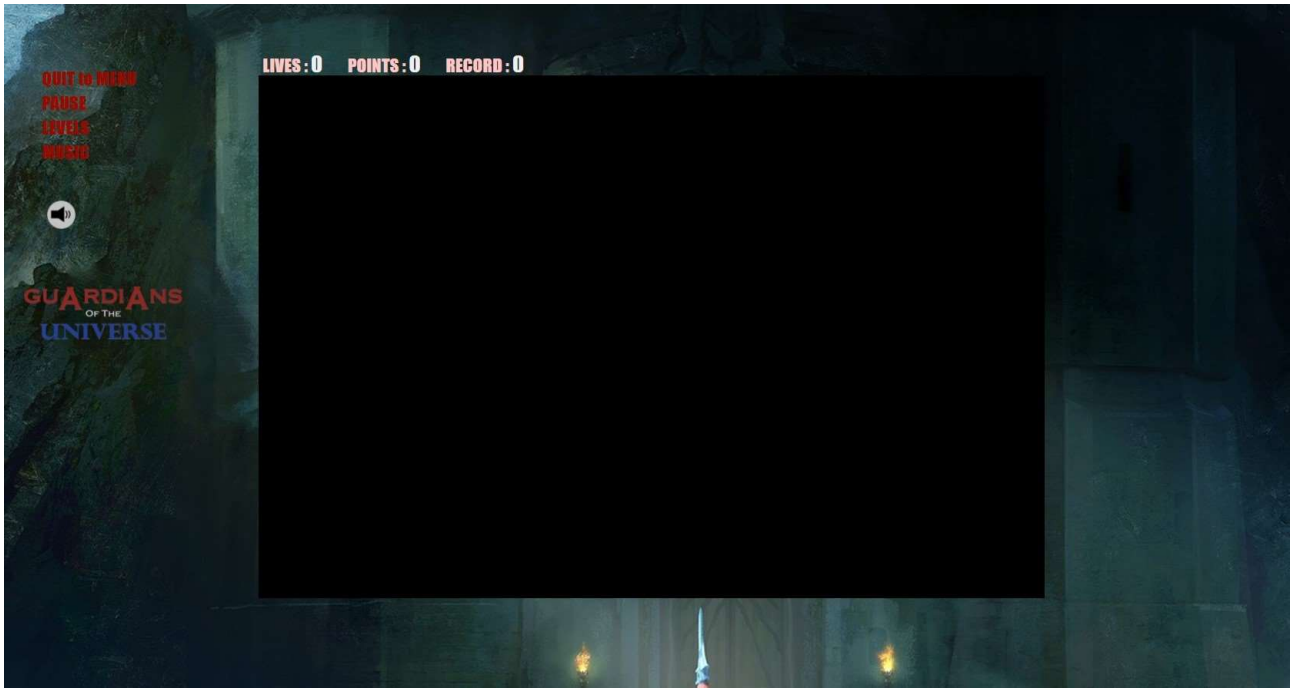
Moreover, in order to import external models downloaded from the web and create a more various and enjoyable game design, the GLTF library was imported and used, together with several models found online.

Finally, a CSS file was created to define and modify the presentation of the HTML documents used to define the structure of the pages of the levels. Particular focus was put on the adjustment of elements such as fonts, colors, disposition of elements (etc.) with the

style chosen for the game. The formatting of the main menu was directly included inside its HTML page.

2. Scene

The web page characterizing each level (*level 1 in the example above*) is defined by two columns:

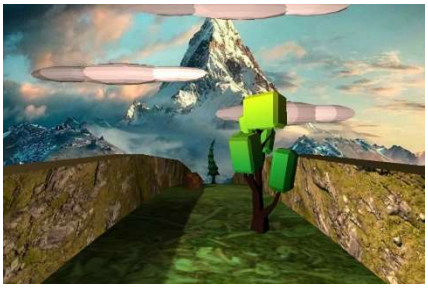


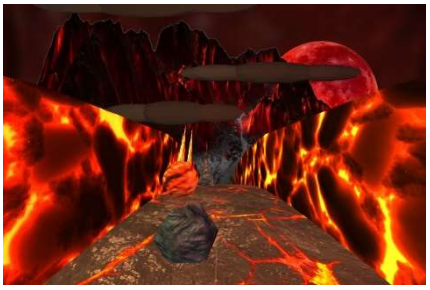
- The left one, ordering a series of buttons which allows the user to interact with the available options of the game;

- The right one, presenting a label indicating (in order) the number of lives left to the player, the current score of the run and the record achieved during the current game session, updated whenever the score achieved by the player at the end of a match overcomes the previous record.

At the centre of the page is located the canvas where the scene will be rendered at the start of the game.

Diving into the analysis of the structure of the game, we can see how each level is defined by its own theme and particular set of objects, created or imported in order to better characterize the world design.





The first level is structured to represent a mountain scenario, at day time, where the player must avoid the obstacles present on its path, like threes and rocks. This is the easiest level for difficulty, considering the type of obstacles and the velocity at which the player moves.

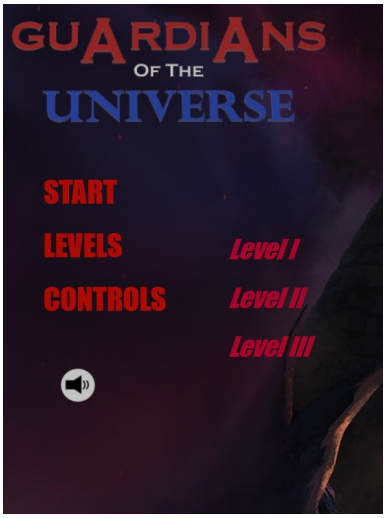
In the second level the player must travel the road of an old creepy castle, at night, where it will have to face a series of deadly traps, like rotating saws and pikes. This level can be considered of medium difficulty, as the obstacles will now be disposed on five columns instead of three.

In the third and last level, a more catastrophic scenario is depicted, as the player must escape the fury of an erupting volcano, with falling lapilli and burnt trees. This is the hardest level of the game, with an increased movement velocity and a less forgiven collision detection system.

3. User Interaction

Menu

In the main menu, a column with a series of buttons is positioned on the left side of the screen, to allow the player to start the game or personalize its experience. Initially, like in the other pages of the program, the music is disabled, leaving to the user the choice to play with or without it.



START – to start the game from the first level.

LEVELS – to open a submenu that allows to select the level to play.

CONTROLS – open a popup with the commands of the game.

Audio controls – Mute / Unmute music

Levels

Each level has the same button controls, divided in:

QUIT – to leave the game and return to the main menu.

PAUSE – to stop / start the game (same as P or Esc).

LEVELS – to open a submenu that allows to select the level to play.

MUSIC – to open a submenu that allows to select the music to play.

Audio controls – Mute / Unmute music

Each level has its own selected music, but the user can choose to change it according to its preferences.

Game Mechanics

Beyond button controls on the web page, the player can use keyboard to activate the main option controls, in particular as seen in the user manual:

- **Pause / Start** the game (stop/start the animations and the movement of the objects in the scene).
- **Restart** the match after the player loses all its lives.

the player must also use keyboard to control the movement during the game:

- **Move left**
- **Move right**
- **Jump**

Each action can be performed while the game is not paused or game over. Also the jump action temporally disables the left or right controls (as the player couldn't move while in air).

4. Models

Obstacles models

Models used in the game were imported or created in order to blend the style of each level with the objects inside it.

```
const gltfLoader = new THREE.GLTFLoader();
const obj_url = '../assets/scene/models/model1.glb';

var glb_LoadModel = function (obj, _ulr){
    gltfLoader.load(_ulr, (glb) => {
    model = glb.scene;
    model.castShadow = true; model.receiveShadow = true;
    model.position.y = -1.1; model.rotation.y = 0;
    console.log(model.position)

    obj.add(model);

    anime_model(model);

    });
}
```

External models (like threes and warriors) are imported using the GLTF loader with Three.js, as it revealed to be a really powerful tool to manage one of the simpler and most common format for 3D objects, the .glb file format.

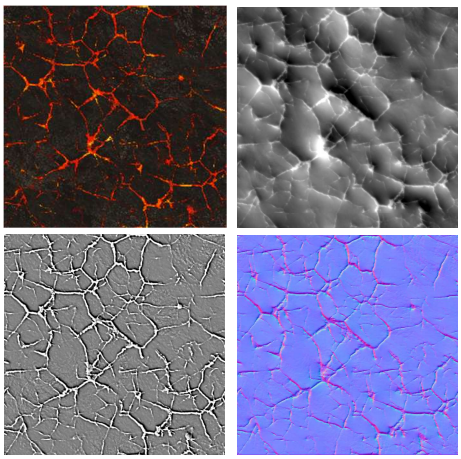
Blender was also used to convert in a clean and efficient way the objects coming in a different format, in order to maintain the same file format for each object loaded.

Textures

Beside the imported models, that kept their own textures, all the created objects were defined using materials characterized by sets of textures to better blend them in the scenario. Different types of maps were used for each object (color maps, normal maps, roughness maps, displacement maps, etc.).

```
const fence_diff = txloader.load("../assets/scene/textures/
const fence_norm = txloader.load("../assets/scene/textures/
const fence_roug = txloader.load("../assets/scene/textures/
fence_diff.wrapS = THREE.RepeatWrapping; fence_diff.wrapT =
fence_norm.wrapS = THREE.RepeatWrapping; fence_norm.wrapT =
fence_roug.wrapS = THREE.RepeatWrapping; fence_roug.wrapT =
```

```
var worldMaterial = new THREE.MeshStandardMaterial( {
    color: "#34353a",
    map: world_diff,
    normalMap: world_norm,
    roughnessMap: world_roug,
    flatShading:THREE.FlatShading,
} )
```



5. Lights

Different types of lights were used to better characterize each ambient of the game, and for each level they were differently resettled in order to optimize the overall yield of the level itself (day time, night time, dynamic lights, etc.).

In particular, in each level is possible to find:

- 2 point lights (one of which able to cast shadows), coming in different color in the various levels to suit the style of the ambient;
- 1 directional light;
- 1 ambient light, to keep a homogeneous illumination of the world.

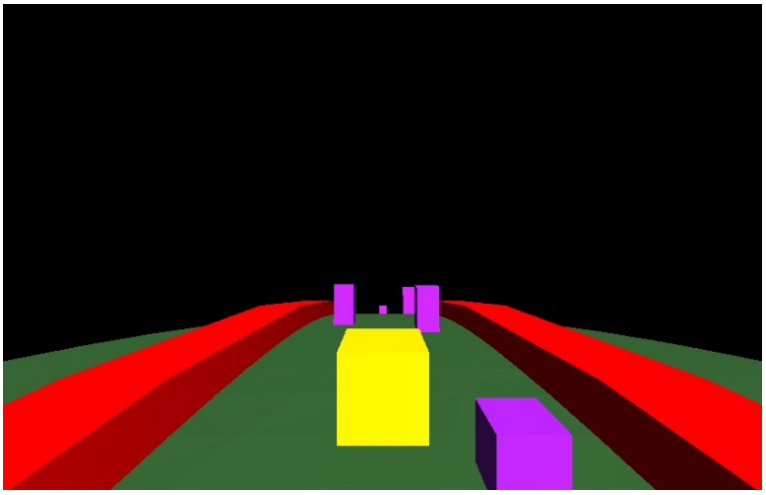
A particular effect was given in the second and third level to the 2 point lights, set one to a reddish color and the other to a yellow one, as their intensity oscillates alternatively to simulate the effect of a dynamic light.

```
//-----------
//  * Lights - level 1

var light1 = new THREE.PointLight(0xFFFFF, 0.75, 250); // (color; intensity; distance; decay)
light1.position.set( 20, 15, 5 );
light1.castShadow = true;

var light2 = new THREE.PointLight(0xFFFFF, 0.75, 250); // (color; intensity; distance; decay)
light1.position.set(-20, 15, 5 );
//light2.castShadow = true;

var dirLight1 = new THREE.DirectionalLight(0xffffff, 0); // (color; intensity)
dirLight1.position.set(-1, 50, 5);
//dirLight1.castShadow = true;
dirLight1.shadow.mapSize.width = 2048; dirLight1.shadow.mapSize.height = 2048;
dirLight1.shadow.camera.left = -70; dirLight1.shadow.camera.right = 70;
dirLight1.shadow.camera.top = 70; dirLight1.shadow.camera.bottom = -70;

var sunlight = new THREE.AmbientLight(0xfff2cc, 0.25); // (color; intensity)
sunlight.position.set ( 1, 50, -5);
//sunlight.castShadow = true;

var lightTorch = new THREE.PointLight(0xffffff, 0.0, 5); // (color; intensity; distance; decay)
```

6. Animations

Animations for the majority of the objects present in scene were created in order to make the world look more alive. The most important animations, common to all the objects and levels, are declared in the `Main.js` file, while the ones related to the specific elements of the singular levels are defined in the corresponding file.

Collision Detection



In order to avoid loading to much weight on the system and compromise the performances of the game, an *ad hoc* collision detection system was implemented instead of importing a physics engine in the program.

The player (yellow box in the picture) moves across the level while the obstacles travels in the opposite direction. At each iteration of the rendering, collisions between the player box and the obstacles are checked by the system, and if a collision is detected, the player obtains a temporary invincibility, indicated by rapidly flashing the model mesh, and loses a life, and when the last life is lost the game is over.

00. Technical chart

ENVIRONMENT

HTML:

`GOTU.html` - main menu page of the game
`level1.html` - web page dedicated to the first level of the game
`level2.html` - web page dedicated to the second level of the game
`level3.html` - web page dedicated to the third level of the game

JAVASCRIPT:

`MAIN.js` - main JS file of the project, common to each level page, contains the main functions of the program.
`level1.js` - JS file used to define the elements characterizing the first level of the game (mountain level).
`level2.js` - JS file used to define the elements characterizing the second level of the game (castle level).
`level3.js` - JS file used to define the elements characterizing the third level of the game (volcano level).

CSS:

`styles.css` - Cascading Style Sheets used to define the characterization of the html files dedicated to the levels.

LIBRARIES

Three.min.js library:
src="https://cdn.rawgit.com/mrdoob/three.js/master/build/three.min.js"

GLTF Loader library:
src="https://cdn.rawgit.com/mrdoob/three.js/master/examples/js/loaders/GLTFLoader.js"

FBX Loader library:
src="https://cdn.rawgit.com/mrdoob/three.js/master/examples/js/loaders/FBXLoader.js"

TWEEN.js Smooth Animation library:

downloaded and included in the *Utils* Directory
src=" https://github.com/tweenjs/tween.js/"

MODELS

Most of the external models used in this work were obtained searching on specialized web pages like:

SketchFab, at https://sketchfab.com/

TEXTURES

Most of the textures used were obtained through specialized web pages like:

Poly Haven, at https://polyhaven.com/textures

3D TEXTURES, at https://3dtextures.me/

OTHERS

Music inserted in the game is not owned by me, the playlist includes:

Holding Out for A Hero - Bonnie Tyler, Masters of the Universe: Revelations

Mind Games - Dread X Collection soundtrack album

Underworld - Dread X Collection soundtrack album

Patriots - Cyberpunk 2077 OST