# **Interactive Graphics**

# **Final Project**

Immagine che contiene disegnando, cibo

Descrizione generata automaticamente

2019/2020

Mauro Ficorella

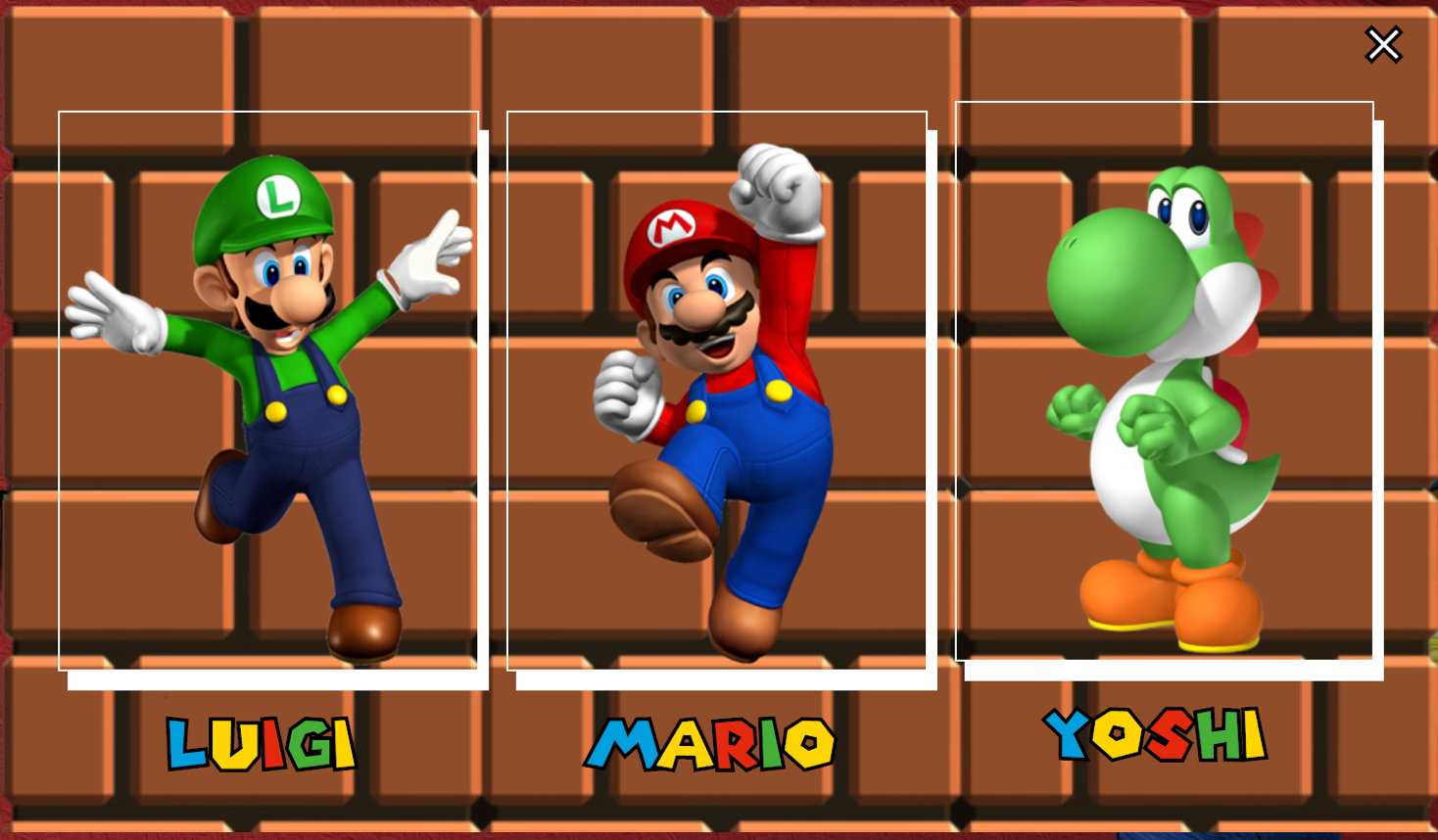
Valentina Sisti

Martina Turbessi

**Introduction**

We have chosen the famous Super Mario platform game as the main theme of the final project. Precisely, we recreated Super Mario Bros’ first level.

The main goal of this game is to finish the level collecting as many coins as possible without losing life against enemies (goombas).

At the beginning of the game, the player can choose between three characters: Mario, Luigi and Yoshi.

During the level, the player can interact with some rewards, like coins or power ups; more deeply, each question block contains those items and character can earn them by jumping against them (hitting with head) from below.

In the level are also shown enemies that are the well known goombas, taken from the original game: if the character collides with them, he loses life, and, if the character lose all his lives, the game is over; otherwise, if the character has earned more than one life, if he collides with goombas, he restart from the beginning of the level without losing coins. Another possibility is that the character collides with goombas jumping on them: in this case the goomba will die.

When the character reaches the end of the level, more precisely in front of castle’s door, the player wins the game and is shown a window containing how many coins he has collected during the level.

The project has been developed using Three.js as the main library, Tween.js for the animations and Physijs for collision detection.

**Immagine che contiene erba, tavolo, verde, sedendo

Descrizione generata automaticamenteScene**

The environment of the game includes a main platform on which the character walks and a background image that repeats itself during the whole level; we used a perspective camera placed in a way that shows character in a profile view; then, in order to follow character’s movements, we used the “lookAt” function inside the animate function to update its parameters at runtime with character’s position.

The entire scene is illuminated by an ambient light and a directional light that follows character’s movements. All these lights are implemented using Three.js library, more precisely in the following way:

const d = 100;

const color = 0xffffff;

const intensity = 1;

dirLight = new THREE.DirectionalLight(color, intensity, 100);

dirLight.position.set(0, 100, -620);

dirLight.castShadow = true;

dirLight.shadow.mapSize.width = 512;

dirLight.shadow.mapSize.height = 512;

dirLight.shadow.camera.near = 0.5;

dirLight.shadow.camera.far = 500;

dirLight.shadow.camera.fov = 50;

dirLight.shadow.bias = 0.0039;

dirLight.shadow.camera.left = -d;

dirLight.shadow.camera.right = d;

dirLight.shadow.camera.top = d;

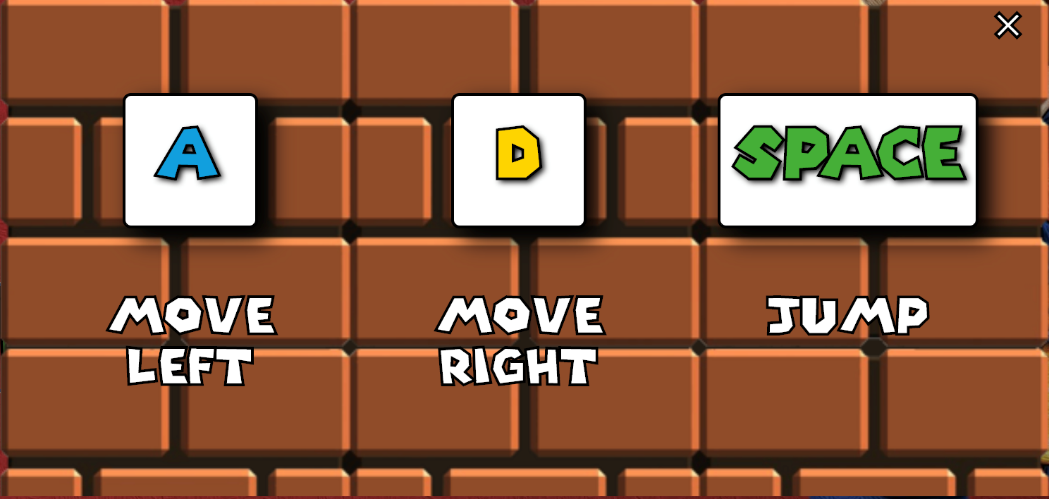
dirLight.shadow.camera.bottom = -d;

scene.add(dirLight);

ambientLight = new THREE.AmbientLight(color, intensity);

scene.add(ambientLight);

**Movements**

The movement was implemented through keyboard’s buttons; in this way we have computed the walking direction or jumps depending on which keyboard’s key was pressed.

More precisely, we used a listener for two type of events, “keydown” and “keyup”: keydown listener deals with keys pressed and keyup listener, otherwise, deals with keys no longer pressed. When the player presses “d” key, the character advances in the right direction; if the player presses “a” key, the character goes back in the left direction; finally, if the player presses “space” key, the character jumps. We also allowed the possibility to move in a specific direction while jumping: in fact, if the player presses “d” or “a” key during the jump, the character jumps but also translates in the related direction.

**Collisions**

In order to implement collisions in a way as more accurate as possible, we used the library called Physijs, imported in the following way:

Physijs.scripts.worker = "physijs\_worker.js";

Physijs.scripts.ammo = "ammo.js";

We used this library to make the main scene; then we made boxes (using Physijs’ “boxMesh”) around all level’s objects; those boxes were intended to detect the various collisions that happens during the whole gameplay. In order to do this collision detection, we added a listener to every box, aimed to detect “collision” events.

More deeply, in order to detect collisions on the right part of characters’ body, we used different boxes for characters’ feet, torso and head; in this way, we can understand if the character jumps on something or against something.

We also used different flags in order to understand who collides with whom; moreover, basing on these flags’ value we controlled animations’ behaviour in order to make them as coherent as possible: for example, if character collides with a block during walk phase, he stops walking immediately.

Indeed, we added also a listener aimed to detect if the character stops colliding with another object; more precisely, if the “contact\_normal” variable reached a precise value, we used this listener to stay updated if the character touches anymore a specific object or not. This method was useful, for example, to make the character fall from a brick when he didn’t touch it anymore.

**Animations**

**Models**

3D models are imported from open source web sites “Sketchfab” as a gltf format(??).

**Characters:**

**Immagine che contiene giocattolo, piccolo, plastica, colorato

Descrizione generata automaticamenteMario:** we import the hierarchical without importing the existing animations. Than we realired animation manually using tweenjs.

Immagine che contiene giocattolo, bambola

Descrizione generata automaticamente

**Immagine che contiene giocattolo, orologio

Descrizione generata automaticamenteLuigi:** we import the hierarchical without importing the existing animations. Than we realired animation manually using tweenjs.

**Yoshi:** we import the hierarchical without importing the existing animations. Than we realired animation manually using tweenjs.

**Items:**

**Immagine che contiene computer, sedendo, scrivania, topo

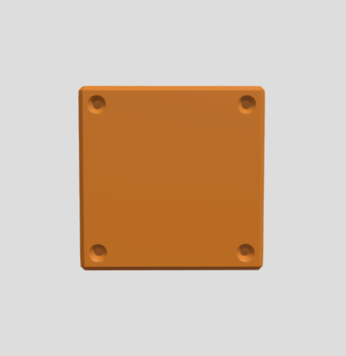
Descrizione generata automaticamente**

**Power-Up:** This model doesn’t have an hierarchical structure and it was cloned few times in order to put power-ups into the question box.

**Coin:** This model doesn’t have an hierarchical structure and it was cloned few times in order to put coins into the question box.

The animation of the coins and the power-up was realized by tweenjs with the function objectAnimation(object, i). This function was called every time that the character hit with the bottom of the question box and (permette agli oggetti di uscire e ruotare).

**Level block (?):**

**Immagine che contiene segnale

Descrizione generata automaticamenteImmagine che contiene arancia, giocattolo, uomo, mattone

Descrizione generata automaticamenteBrick:**

**Question Box:**

**Empty Brick:**

**Immagine che contiene sedendo, tazza, computer, tavolo

Descrizione generata automaticamente**

**Pipe:**

**Immagine che contiene giocattolo, orologio, sedendo, piccolo

Descrizione generata automaticamenteCastle:**