Interactive Graphics

Final Project

2019/2020

Mauro Ficorella
Valentina Sisti
Martina Turbessi

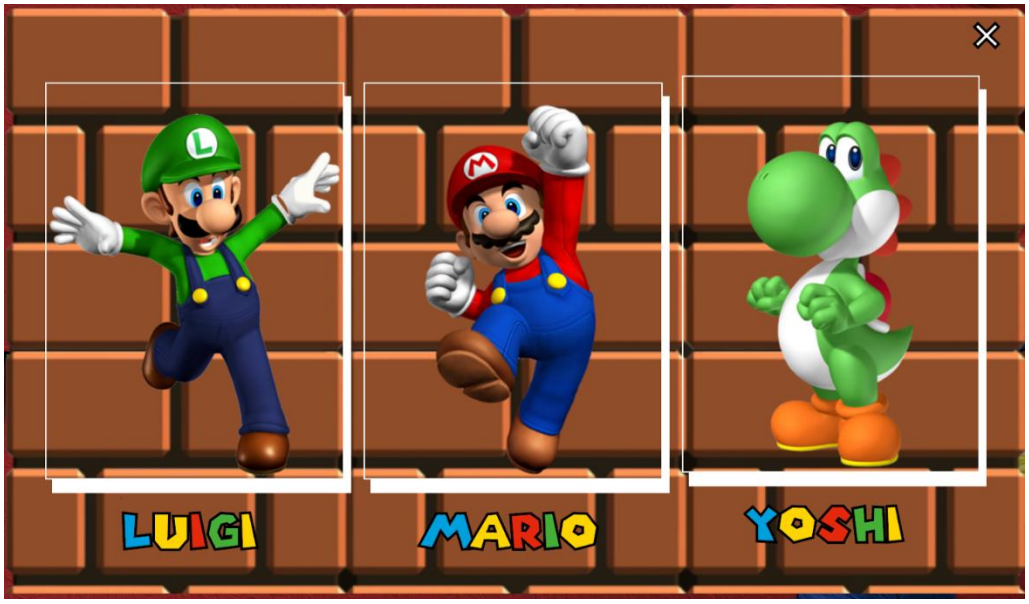
Summary

Introduction

We have chosen the famous Super Mario platform game as the main theme of interactive graphics' final project. More 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.



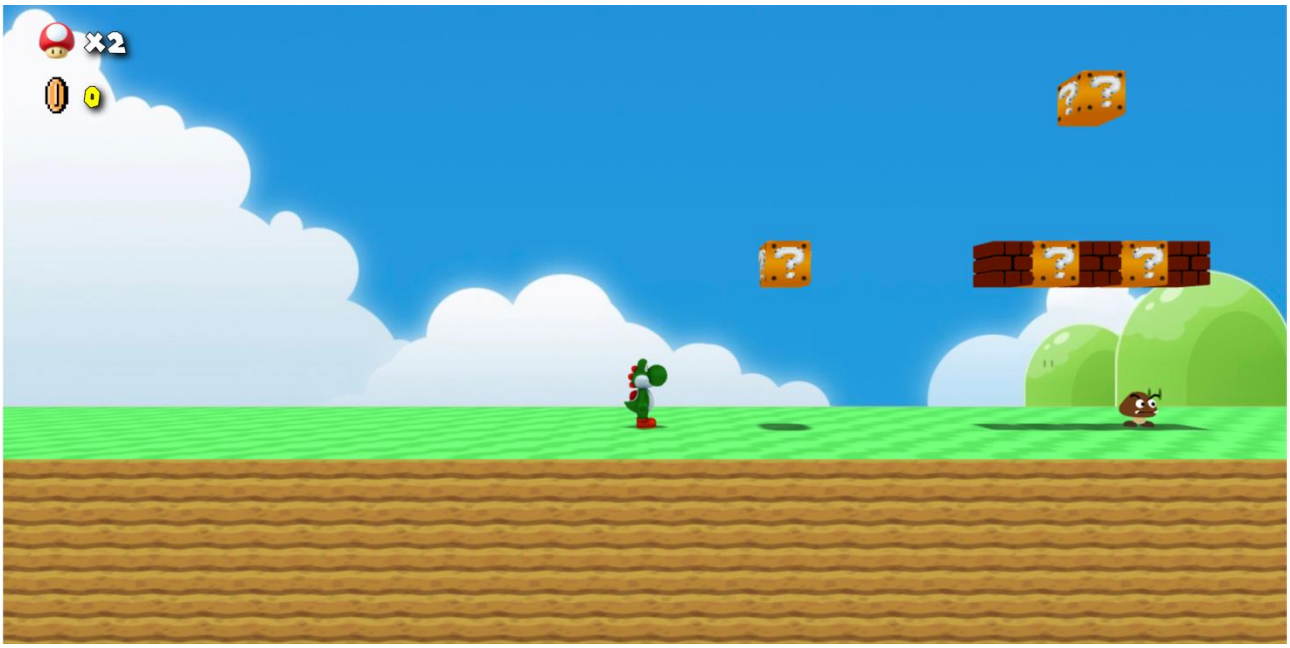
During the level, the player can interact with some rewards, like coins or power ups (power ups give the character one life); more deeply, each question box contains those items and character can earn them by jumping against them (hitting with head) from below.

In the level are also shown enemies that are the well-known goombas, taken from the original game: if the character collides with them, it loses life and respawns from the start of the level, and, if the character lose all his lives, the game is over; otherwise, if the character has earned more than one life, if it collides with goombas, it 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.

Scene



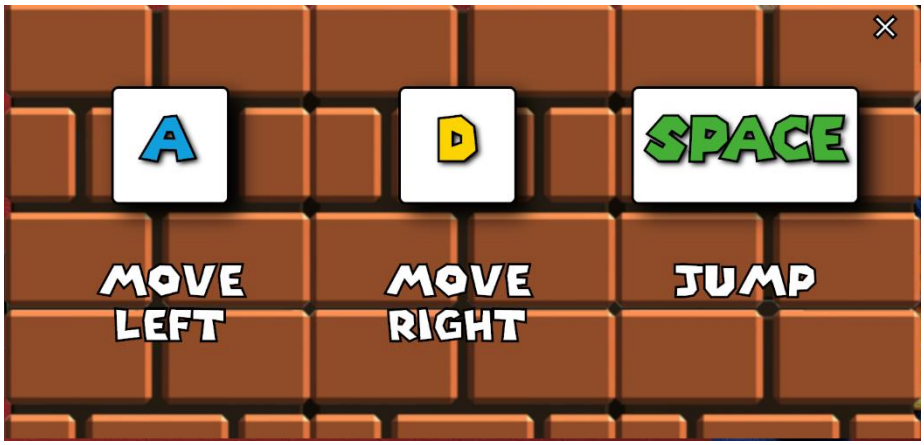
The environment of the game includes a main platform on which the character walks and a background image that repeats itself during the whole level; we used a perspective camera placed in a way that shows character in a profile view; then, in order to follow character's movements, we used the "lookAt" 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. We have also let directional light cast shadows. 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 (a Three.js plugin), 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 (and then we have made them invisible); 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 boxes also for goombas, bricks, pipes, stairs and ground. More precisely, we have used two different boxes for goombas, one for the head and one for the body: in this way we can discover if the character hit goomba on the head (killing it) or on its side (losing its life); regarding bricks, we used three different boxes: one for the top face, one for the side face and one for the bottom face; using these boxes allowed us to know if character collided from the bottom (to earn rewards), from the side (to stop it) and from the top (falling on them); regarding pipes we used two boxes, one on the side to stop the character if he collided with

them in this way and one on the top in order to allow the character so stop when it falls on them; regarding stairs we used the same approach used for the pipes with two boxes; finally, regarding power ups and coins, we used one box to know when the character collides with them and let it collect those rewards.

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 again a specific object or not. This method was useful, for example, to make the character fall from a brick when it did not touch it anymore.

Animations

Regarding animations, we used Tween.js library; this library allowed us to make smooth animations. We developed the following animations regarding the character: walk animation (moving arms and legs), jump preparation animation (bending the knees), jump animation (translating up and down during the jump), arm animation during jump (raising up and lowering arm), rotation animation (rotating torso), fall animation (translating down from object or when colliding with other objects), game over animation and win animation; then we also developed animation functions regarding goombas' walk and objects' movement (coins, power ups).

Inside these functions we modified characters body parts' angles and values in order to make them move in the proper way.

More specifically we defined, for all these tweens, starting values and values to be achieved at the end of the animation, as it follows in this example taken from torso's rotation tween:

```
var tweenLeft = new TWEEN.Tween(tweenStartLeft, groupRotate)
  .to(tweenGoalLeft, 400)
  .easing(TWEEN.Easing.Linear.None)
  .onUpdate(function () {
     torso.rotation.y = tweenStartLeft.y_leftRotation;
  })
  .start();
```

This code snippet shows how we used some tween functions into the animations: "to" function defines time duration until animation reaches goal (target) value, "easing" function defines how quickly the target value is reached, "onUpdate" function defines what should be done during the tween, "start" function started the tween. Into other tweens we used also other functions: "onStop" function defines what should be done when the tween is stopped; "onComplete" function defines what should be done when the tween is completed; "repeat(Infinity)" function make the tween repeat forever; "yoyo" function make the tween bouncing to and from the start and end values, in a smooth way.

Models

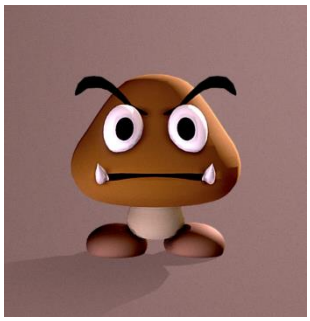
All 3D models used inside our game have been downloaded from the open source web site called "Sketchfab" as a gltf file. After that, we imported this gltf file into the code using GLTFLoader provided by Three.js.

Characters:

Regarding characters we import the hierarchical structure of each character's model without importing the existing animations; indeed, we realized animation manually using Tween.js.

Mario, Luigi and Yoshi have their own hierarchical structure. Due to this structure, we could move their body's parts independently from each other.

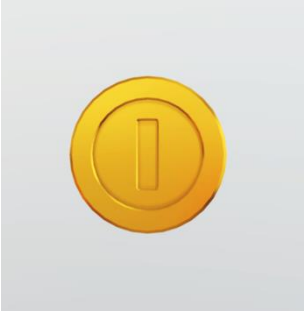
Enemies:



Goomba: Like the main characters, also goombas have their own hierarchical structure; this allowed us to let it move during the gameplay. We cloned those goombas in order to place them in multiple positions. Goombas were imported without importing the existing animation.

Items:

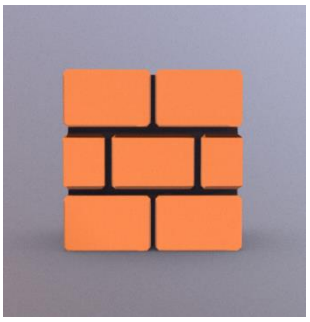
Power-Up: This model does not have a hierarchical structure and it was cloned in order to put multiple power-ups into the question boxes.



Coin: This model does not have a hierarchical structure and it was cloned in order to put multiple coins into the question boxes.

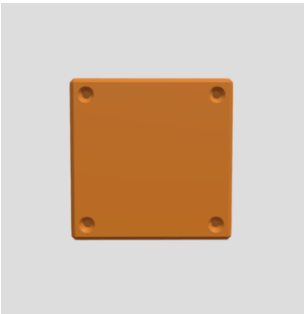
Coins' and power ups' animation was realized by Tween.js using the function objectAnimation(object, i). This function is called every time that the character hit the bottom of the question box: this behaviour allows the object to came out from the bricks and start rotating themselves.

Level objects:



Brick: This model does not have a hierarchical structure or animation and it was cloned in order to put multiple bricks along the level.

Question Box: This model does not have a hierarchical structure or animation and it was cloned in order to put multiple question boxes along the level.



Empty Brick: This model does not have a hierarchical structure or animation and it was cloned in order to put multiple empty bricks along the level. We have used those cloned empty bricks to create the stairs grouping them in the proper way.

Pipe: This model does not have a hierarchical structure or animation and it was cloned in order to put multiple pipes along the level.

Castle: This model does not have a hierarchical structure and it was positioned only once at the end of the level.

We have created the whole level's geometry into "level_build.js" file by cloning and placing all those objects in the correct position.

Audio

Starting from the initial window, we have implemented audio tracks downloaded from the internet.

When the main menu starts, we chose to set audio to muted; for this reason, we inserted an audio button in order to allow the player to set this to on/off state.

Then, once the game is loaded, the main audio track starts, played in loop so that the player can play as long as he wants by listening to this track without interruption. We also implemented a sound for character's jump, a sound when the character collides on the bricks' bottom in order to obtain power ups or coins, a sound when the character collects coins, a sound when the character kills goombas colliding with them from above, a sound when the character loses a life, a sound when the game is over and a sound when the player completes the level winning the game.

HTML

We have created different html pages, depending on which scenario the player is.

When the game loads, there is the main menu; inside this window the player can click on play button, controls button or unmute/mute audio button: if he clicks on play button, a "modal" shows up offering character choice between Yoshi, Luigi and Mario (the player can click on their name to choose it); if he clicks on controls button, another "modal" pops up showing how game controls are set; finally, if he clicks on unmute/mute audio button, he can enable/disable game music on this starting screen.

Talking about aesthetics of this page, we used a font (downloaded from the internet) inspired to the official Super Mario Bros' font; regarding buttons, instead, we implemented "hover effect" to highlight buttons when player moves the mouse cursor over them. When the level screen is loaded, we added, on the top left part of the screen, coins and lives indicators in order to keep the player updated, during the whole level, regarding those items.

If the player has no lives remaining and collides with a goomba, the character dies and the "game over" screen loads offering the player the possibility to know how many coins he has collected during the level and to return to the start page in order to retry starting another game or change character. A similar behaviour is shown when the player completes the level and wins the game: the "win" screen loads, then there are shown how many coins he has collected during the whole level and there is a button offering the possibility to return to the start page in order to restart another game or change character.

References

- https://threejsfundamentals.org/
- https://threejs.org/
- https://chandlerprall.github.io/Physijs/
- https://github.com/tweenjs/tween.js
- https://github.com/tweenjs/tween.js/blob/master/docs/user_guide.md
- Mario model: https://sketchfab.com/3d-models/mario-0254649f5b0047f5875749b7c6fd65f5
- Luigi model: https://sketchfab.com/models/ea261541e8b2447ab4bf07ac5ddb7d6b
- Yoshi model: https://sketchfab.com/models/eda1f06b7ce647f6b2fb9da44216de32
- Goomba model: https://sketchfab.com/models/3c91cc4ac8144d82a05309679a3dbbd5
- Brick model: https://sketchfab.com/models/c12a2715135b4f6a9816efdf90180a8c
- Question box model: https://sketchfab.com/3d-models/question-box-bbf45f32a35b471cac14043f5da484e8
- Empty block model: https://sketchfab.com/3d-models/super-mario-empty-block-1b65a6aa885448e0bf1b548409bea6ce
- Pipe model: https://sketchfab.com/models/3eedbc370ad946b9a76ab2bcd949c469
- Coin model: https://sketchfab.com/models/2d814553c1f64fbd9550d4f7af25dd32
- Power up model: https://sketchfab.com/models/1bd1720613e44292a5a70119d4c1a254
- Castle model: https://sketchfab.com/models/f4efbf6382dd403f92babbcfa85421a7
- Music downloaded from the internet