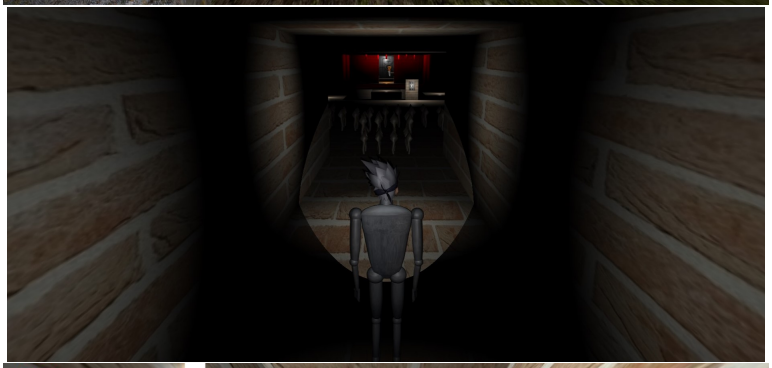
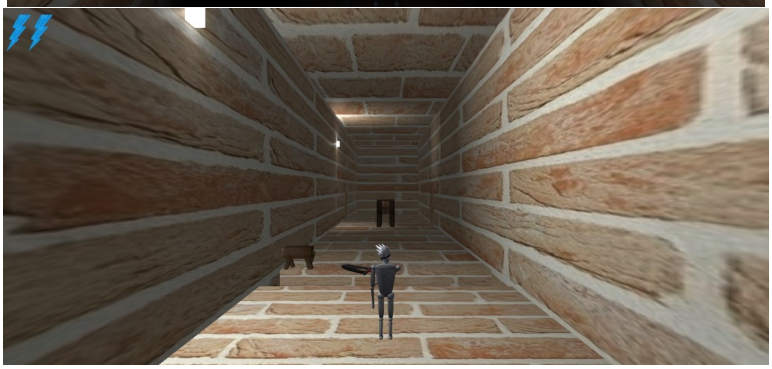
SAPIENZA
UNIVERSITÀ DI ROMA

Interactive Graphics Final Project

Ketbjano Vocaj - 1652134
Paolo Tarantino - 1666228
Matteo Russo - 1664715

July 2020





Contents

1 Introduction **3**

2 Levels Design **3**

3 Libraries **3**
3.1 Three.js 3
3.2 Physi.js 4
3.3 Tween.js 5

4 Imported Models and Hierarchies **5**

5 Lights **5**

6 Animations **6**

7 Interactions **6**

8 References **8**

1 Introduction

This project is the final step of the **Interactive Graphics** course held by professor Schaerf Marco in the 2019/2020 academic year. The main idea was to create an application using the tools approached during the lectures such as: graphic libraries and algorithms, animations theory and hierarchical models.
In order to fulfill the tasks we designed a platform game in which the user has to complete many challenges by interacting with the proposed environment.
The game is divided into three different levels, each one permits a different interaction to the user depending on the particular level design. The **Index** and **Outro** documents are purely related to the animation and require no user interaction; these two represent respectively the game's prologue and epilogue.
All of them are connected by a very simple horizontal storyline which acts as a glue for the presentation of the various levels: we have a main character which has to find a secret treasure located in a mysterious place and, in order to obtain it, he must face many obstacles along the road and defeat a strong antagonist in the last level.

2 Levels Design

After the initial prologue, the main character finds himself in a jungle looking for the precious treasure. All of a sudden, he triggers unintentionally the movement of a giant boulder that starts rolling towards him. He now has to run away from it in order not to be flattened. Along the flee, he will have to dodge the several obstacles like rocks, wood trunks and trees that will try to stop him. At the end he will be able to jump over a cliff and enter in a dark cave.
Now he has to find a way to get out from the cavern. Only a very feeble light illuminates the environment, however using the torch it's possible to light up the surroundings more. Passing through many traps, he will have to position two platforms by using an electronic controller ensuring to be very accurate otherwise he'll have to restart all over again. After dodging the dangerous lasers coming from the ceiling, the character has to solve a riddle in which he has to find a hidden key in a specific statue. To do so, a bigger torch is provided to him and after the solving he will be able to reach the final part of the tower.
The character has now found a sword that will be very useful to face the remaining threats. He'll have to dodge a hidden gorge and take down a wild grizzly which bars the way. Passing on a platform and tearing down the rotating dangerous totems will bring him to the final fight. A strong opponent will separate him from the final treasure. By dodging its shurikens and attacking him at the right moment will grant the character the desired win. The character has now access to the treasure: "The Master Degree in Engineering in Computer Science"

3 Libraries

Three libraries were employed for making the project: THREE.js, Physi.js and Tween.js.

3.1 Three.js

Every level environment was created using the suggested Three.js library (**v.117**), which was used to build up the scene and to import the several 3D models.
First of all, inside the `init()` function, we created and set the renderer at the right size, and we also created the scene where all the objects lie and where animations take place. A perspective camera was used in order to give more depth to the scene and to simulate the visual dynamic effects of human movement (as shown in **Figure 1**).

```
window.onload = function init() {

        renderer = new THREE.WebGLRenderer({ antialias: true });
        renderer.setSize( canvasW, canvasH );
        document.getElementById( 'viewport' ).appendChild( renderer.domElement );

        scene = new THREE.Scene;

        camera = new THREE.PerspectiveCamera(45,canvasW/canvasH,0.1,1000);
        camera.position.set( 0, 2, 3.9 );
        camera.rotation.x = -0.283;
        camera.add( listener );
        scene.add( camera );
```

Figure 1: Init function

Now, using the basic geometric shapes provided by this library, we created the level's surroundings and arranged

them in the scene in order to make a homogeneous room structure, with many peculiarities. For example, the walls, floor and ceiling were made up in this way by meshing the geometry with a particular material which simulates shiny surfaces (*Phong Material*); furthermore we added a texture for each different type of object.
For more complex geometries we created objects by importing 3D models, but this will be explained better in **section 4**.

```
texture2 = new THREE.TextureLoader().load('images/matt.jpg');
var floor4 = new THREE.Mesh(
    new THREE.CubeGeometry( 10, 5, 18 ),
    new THREE.MeshPhongMaterial({map:texture2})
);
scene.add(floor4);
```

Figure 2: Room's floor

3.2 Physi.js

Along the Three.js one we also employed Physi.js which we found very useful in managing collisions and physical based settings. As an example we can take the (**levelM.html**) game: in the final part, the character has to dodge wooden trunks and rocks right after the final cliff. Every object in that particular place, as well as the character, was provided of an hitbox that determined the range at which a collision should be triggered with the controllable character. The following images show how the level looks with the enabling/disabling of those hitboxes. Moreover, in the same level, Physi.js was used for implementing gravity in order to make possible to have the character fall if it slips off the edge of a platform. To do so, the last platforms were created as Physi.js Objects; meaning that the character was able to walk over them as long as it does not fall. This could be done thanks to the fact that the character's hitbox made collisions with every platform he works on.
Same thing in **levelK.html** where, in the last room, the collisions with the statues were implemented through different hitboxes as shown in the figure below.

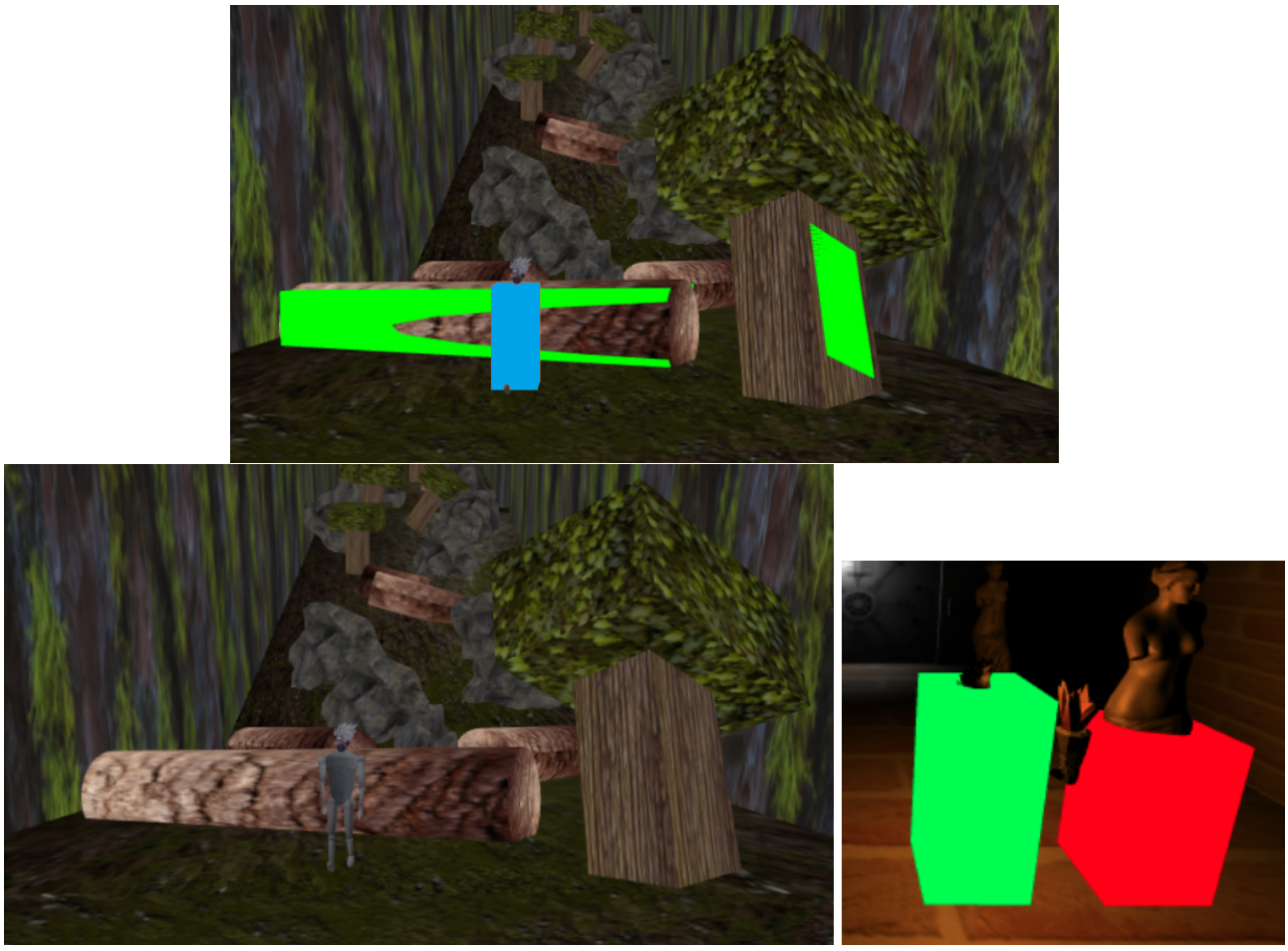


Figure 3: Different level's look by enabling the hitboxes

3.3 Tween.js

For making smooth animations Tween.js was the right library to be implemented, because it's very light in terms of computational efforts and also has many features regarding interpolation functions; it was mainly used in **levelK.html** for moving the mobile platforms back and forth, along two fixed positions through the X axis: the animation starts when the character reaches a certain point, then, when the two platforms reach their limit position, they return in the initial state and then loop until an event stops the animation.
For example, `floor5` has an initial position to start and a target position to end its movement (the two animations are chained in order to continuously repeat the movement); in order to make the level more unpredictable, the time intervals for the interpolation were chosen by calling a function that generates random numbers (**Figure 4**).

```
function tweenAnimation(){      //Lasers and platforms movements (IMPLEMENTED WITH TWEEN.js library)
    floor5Tween = new TWEEN.Tween(floor5.position).to({x:5, y:floor5.position.y, z:floor5.position.z}, Math.floor(Math.random() * 1000) + 2500);
    floor5Tween2 = new TWEEN.Tween(floor5.position).to({x:floor5.position.x, y:floor5.position.y, z:floor5.position.z}, Math.floor(Math.random() * 1000) + 2500);

    floor5Tween.chain(floor5Tween2);
    floor5Tween2.chain(floor5Tween);
```

Figure 4: Tween implementation

4 Imported Models and Hierarchies

As introduced in **section 3.1** we imported complex geometry through the `OBJLoader` and `FBXLoader` primitives, which take a 3D model (respectively in format **.obj** and **.fbx**) and load them in the application.
Some models were taken from the web, while others were created by scratch with 3d modelling software such as ***Maya*** and ***Blender***. For example rocks, torches and character's faces were downloaded from specific websites (see **References** section, chapter **8**); instead the complete body was made by connecting the limbs, which were modelled individually in ***Maya***; some other objects, as the sword, were obtained by modifying pre-existent models.

```
var textureFace = new THREE.TextureLoader().load('images/Face.bmp');
testa = new THREE.Object3D;
loader.load('mod/testac.obj',function(model){

    model.children[0].material = new THREE.MeshPhongMaterial({map:textureFace});
    testa.add(model);
    testa.position.set(0, 7, 0.2);

});
personaggio.add(testa);
```

Figure 5: 3D Object Loader

As we can see in **Figure 5**, the variable `testa` loads the head of the character and it is added to the `personaggio` variable, which represents the full body group.
For making a complete body hierarchy, the limbs were added to the body following this procedure and were set into specific position in order to have the so called "joint points"; in this way all the lower limbs are dependent to the respective upper part and all body parts are dependent to the main root.
In fact lower legs and arms (respectively `stinco` and `avambraccio`) were added, together to their upper part `gamba` and `braccio`, to the group that manages that particular limb (respectively `gambe` and `braccia`); finally all them are joined to `personaggio` for completing the hierarchical structure.
In the same way we build up the antagonist's and bears' body, with the difference that the first one was made by imported models and the second one was created by using the geometric primitives of Three.js. For example, thanks to this type of build, by calling `bear.children[2]` we can get access to the front right paw of the big bear.
Because of all the objects are added to the scene, we can see it as a giant hierarchy.

5 Lights

Three type of lights were used in this project: ambient light, point light and spotlight. The first one illuminates globally all objects in the scene equally, the point light emits light from a single point to all direction (like a lightbulb) and then the spotlight acts like a torch that emits light within a cone.
To better see the effects of the spotlight it's better linger on **levelK.html**: in this level, the crucial point is the play of light and shadows through the room. As shown in the **Figure 6**, the spotlight needs two more

parameters compared to the other two type of lights: angle of light dispersion and a target.

```
ambientLight = new THREE.AmbientLight(0xaaaaaa, 1, 50);

spotLight = new THREE.SpotLight( 0xcccccc, 1.0, 20 , Math.PI/9);
spotLight.position.set( personaggio.position.x , personaggio.position.y , personaggio.position.z );
spotLight.target.position.y = -2.5;
spotLight.target.position.z = -5;

light1 = new THREE.PointLight( 0xffffff, 1, 3 );
```

Figure 6: Lights

Since the spotlight has to move with the character, we set its position at the same coordinates of the body, so it can be updated each time in the render function; finally the target's distance and height are set very low in order to not make a light that illuminates all the scene like the ambient one.

6 Animations

A lot of animations are implemented, a part of them are recurrent in all the levels, while others are specific in the levels in which they are used.
For example the walk of the character is common throughout the whole game; it is achieved by rotating each limb alternately back and forth around its "joint point" of a increasing/decreasing angle, and at the same time by moving the entire body towards the wished direction.

```
function moveLimbs(){

    personaggio.children[2].rotation.x -= rotationLimbs;
    personaggio.children[3].rotation.x += rotationLimbs;
    personaggio.children[4].rotation.x += rotationLimbs;
    personaggio.children[5].rotation.x -= rotationLimbs;

    if(personaggio.children[4].rotation.x > 0.5 || personaggio.children[5].rotation.x > 0.5) {rotationLimbs = -rotationLimbs;}
    soundFootstep.play();
}
```

Figure 7: Walk

In the first level we can find the menacing rolling of the giant rock obtained with a rotation on its x-axis and with a translation toward the character.
In the dark cave we have the animations of the platforms and the lasers that exploit the useful tween library.
In the final level the most remarkable animations are the one of the walk of the bears, the blow of the sword and the enemy's attack with the dangerous shurikens. The first one is very similar to the character's walk, but in this case also the head has a peculiar rotation.
The cutting sword is a combination of several movements: specific translations and rotations occurred in the proper order and at the right moment, allow the character to draw the blade and to hit a blow, once or twice, accompanied with the characteristic sound of a cut.
The shurikens' behavior is managed by some methods called in the `render()` function: one of them load the 3D models and makes the object as we see; an other one handles the translation and the rotation of each object, by adding a random element in order to render the movement unpredictable; two implementations operate with time interval, and decide when stop or restart the motion, for allowing the character to hit the enemy.
All the animations are doable by the support of several Boolean variables, that handle collisions too. The jump and falling actions, that are correlated, exploit this particularity concerning the position of the character on the floor and along the y-axis
At the last, in the epilogue there is a long animation, made in a similar way of the previous ones, but without interactions by the user.

7 Interactions

The user interacts with the environment through the pressing of the keyboard buttons and are managed by the javascript event listeners. In fact the character's movements are triggered by the arrow keys and an event occurs each time a key is pushed; if the upper arrow is on then the character moves forward, if left arrow is on then it moves to the left and so on...
In **levelK.html** the user stops the moving platforms by pressing **1** and **2** on the keyboard and, once this is done, many things are triggered: the left hand animates, some sounds play in background, the platform stops

from moving and they also change colors depending on certain constrains satisfaction.
Other interactions are implemented in **levelP.html** where initially the user has to jump over a cliff by using **A** button, then it has to fight a bear by hitting it with a cutting blow with **Z** (this key, in addition to **X** key, will be also used to destroy the totems and kill the antagonist in the last room).
In every level there is a **Skip button** which helps the user to switch from a level to an other in order to see all the game without the need of completing every required challenge.

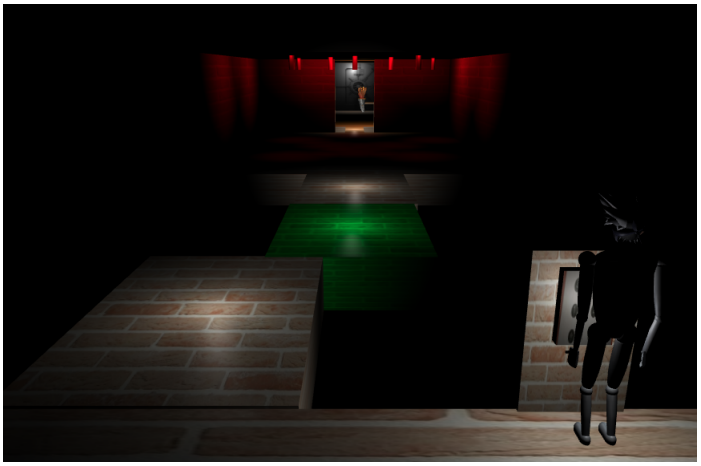


Figure 8: Pressing right key at the right time

8 References

https://threejs.org
https://github.com/tweenjs/tween.js/
https://github.com/chandlerprall/Physijs
https://threejsfundamentals.org/
https://free3d.com/it/
https://www.turbosquid.com/it/Search/3D-Models/free
https://sketchfab.com/3d-models/popular
https://freesound.org/