
SAPIENZA
UNIVERSITÀ DI ROMA

MASTER OF COMPUTER SCIENCE IN ARTIFICIAL INTELLIGENCE AND ROBOTICS

INTERACTIVE GRAPHICS

# "THE LITTLE KNIGHT"

# GAME PROJECT BUILT WITH THREE.JS LIBRARIES

FRANCESCA PALERMO

A.A. 2015/2016

# 1. Introduction

The project developed is a simple third person game in which the player impersonates a character named Knight.
The goal of the game is to reach the dragon on the other side of the cave, avoiding the lava's river.
In case of victory, the game stops and it appears a final screen which represent a dead dragon.
Otherwise, if the character falls in the lava, a losing screen will appear.

To move the knight in the environment I have set the controls in this way:

- Press W to move the character forward;
- Press S to move the character backward;
- Press A to move the character left;
- Press D to move the character right;
- Press Spacebar to make the character jump;
- Press C to look backward (like in a rear-view mirror)

The game is based on WebGL so it works with all browsers that support this graphic library but it's recommended to use Mozilla Firefox since Google Chrome and Safari have some problems loading the textures.

# 2.Code Analysis

The project is based on WebGL library that provides a graphic 3D API to browsers, allowing the creation of 3D scenes through javascript, working with the Canvas element of HTML5.

I have used two more libraries, beyond this one:

- Three.js, open source library based on WebGL, that provides a number of functions and structures that simplify the creation of complex objects and scenes, altogether leaving the programmer free to personalize the scene at his own.

- Physi.js (website), another open source library that handles Physic simulation through Three.js function.

The project is composed by:

- The Little Knight.html : the main file that you have to open to play the game and the one that recalls all the other .js files;
- main.js : the file that renders and initializes the scene;
- character.js: the file that actually creates the character and provides its animation;
- floor.js: this one creates the floor in the scene.

## *The Little Knight.html* script

As written above, this script is used to recalls all the other .js files and to start the game. It is also used to write the personalized shader (see also main.js) using the object THREE.ShaderMaterial. This object allows three.js to interface with vertex and fragment shaders defined by the user, transferring uniforms and attributes and obtaining position and colors of vertices and colors of fragments.

## *main.js* script

After having initialized all the variables, you can find in the script the initScene function which is the function that initializes all the objects, lights, camera and render's parameters. Regarding the render's parameters I enabled a soft shadow mapping to make the shadows more realistic. To be able to adapt better the canvas to the window's size, I have setup a listener on window's resizing.

All the objects in the scene are subjected to physics' law thanks to Physi.js.

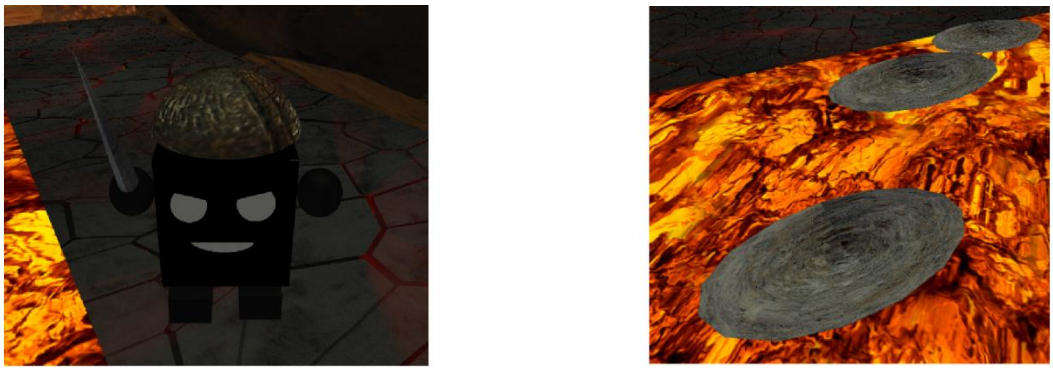

*Figure 1. The knight, group of rocks above the lava rendered in the scene*

The camera is a perspective one; it is placed behind the character and it is attached to it in order to follow him.

After the creation of the camera, I have designed the cave in which all the objects are positioned. First i tried to build a skybox, but it was impossible to apply to it Physi.js, so I changed and decided to use four plane for the walls and one hidden ground below the lava to give a sort of sinking effect when the knight falls down the river.
The real floor is made up in the section "Rocks Floor Spawn" which recalls the script floor.js.

*Figure 2. The images used as texture for the walls*

For what regards the lightning section there is a white soft ambient light spreading in the whole environment, an hemisphere light to lighten the all scene and a spotlight which represents the sun, that rotates around the cave. The rotation occurs along the yz plane and is characterized by the SunPeriod and sunRadius parameters. The intensity of the spotlight varies during the rotation and it simulates the different time of the day.

The last section of this function manages the complex 3D model that I added as the goal of the game, the Dragon. This model is imported as an *.obj file so, in order to deal with such file, I had to load a utility script in the html code, called OBJLoader.js, that is also provided by Three.js. This function allows  to manage the object's properties like its size, material and texture. There is a problem with the .mtl file which isn't loaded at all, so I used a personalized texture for the dragon.

*Figure 3. The Dragon.obj, goal of the game*

As written before in the previous script, I have created a personalized shader to better represent the lava and then build a river with this shader. I had to construct the river in four part because otherwise the texture would have been stretched.
Then I have written the code regarding the creation of the floor which recalls another script in the project called *floor*.js.

The *render* function is a simple function that demands to the browser a 60-fps-rendering of the scene (the API used to do this is *requestAnimationFrame*).

The *animate* function is similar to the previous one, but I use it to update some variables during the time, such as the directional light of the sun and its luminosity. Moreover, I added in this function a continuous check on the terminal condition of the game:this is achieved when the knight reaches the target or when the knight falls in the river; so once one of these things  happen, the character is not able to move anymore and a text appears on the screen through a *textured sprite plane*.

# *Character.js* script

This script is a function that builds and animates the character in the scene. It has two methods: *character.prototype.init* and *character.prototype.animate*.

The first one *physically* builds each basic objects used for the character and then merges them together to assemble a complex and hierarchical object and add it to the scene. After the geometric section, I had to provide some listeners on the key button, in order to create some Boolean variable needed in the motion part.

It is good to notice that to apply the texture on the character I had to adopt a *UV mapping* since different part of the texture are mapped onto different faces of the character's body.

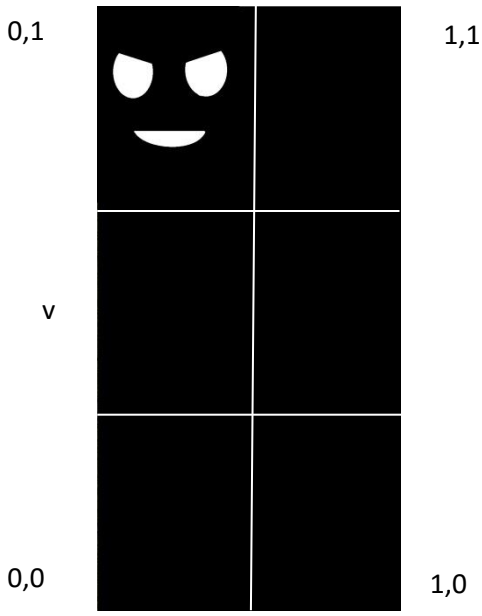


*Figure 4. UV Mapping of a generic texture*

The second method of the character's function provides the angular and linear velocity applied to *Knight* when the user presses a control key. It also provides the animations of character's feet, hands and the sword, in order to simulate a human-like walk.
All the parts of the body are enabled to cast shadow.

# *Floor.js* script

This is a script based on single function that is recalled in the *main*.js file to automatically build the floor in the scene.
It has a single method, *floor.prototype.init*, which provides the geometry, material, and texture of the object. It has six arguments to set up the position in the scene (3 of 6 arguments), the *x* and *z* dimensions and the mass of each part of the floor.