Interactive Graphics

TLS - The Last Shot

Presented by:
Daniele Iacomini

Submitted to:
Marco Schaerf

Anno Accademico 2022/2023

Contents

1 Introduction **1**

2 Environment **3**

2.1 Libraries . . . 3

2.2 Scene . . . 4

2.2.1 Models . . . 5

2.2.2 Hierarchical model - The Robot . . . 6

2.2.3 Camera . . . 7

2.2.4 Lights . . . 7

2.2.5 Textures . . . 7

2.2.6 GUI . . . 7

3 Physics **8**

3.1 Hitbox and bullets . . . 8

3.1.1 Interactions . . . 9

4 Animation **10**

4.1 Robot . . . 10

4.1.1 Walk to the arena . . . 10

4.1.2 The Game . . . 10

4.1.3 Particle System . . . 11

4.2 Character movement . . . 11

Chapter 1

Introduction

The idea behind TLS - The Last Shot

This project was born from the idea of initially wanting to replicate a shooter with a historical setting, similar to Call Of Duty World at War. To make it more interactive, the actual game mode was then added, a timed game in which the player has to destroy the robot before time runs out. The game comes with 3 levels of difficulty, based on how much time the player has to destroy the Robot.

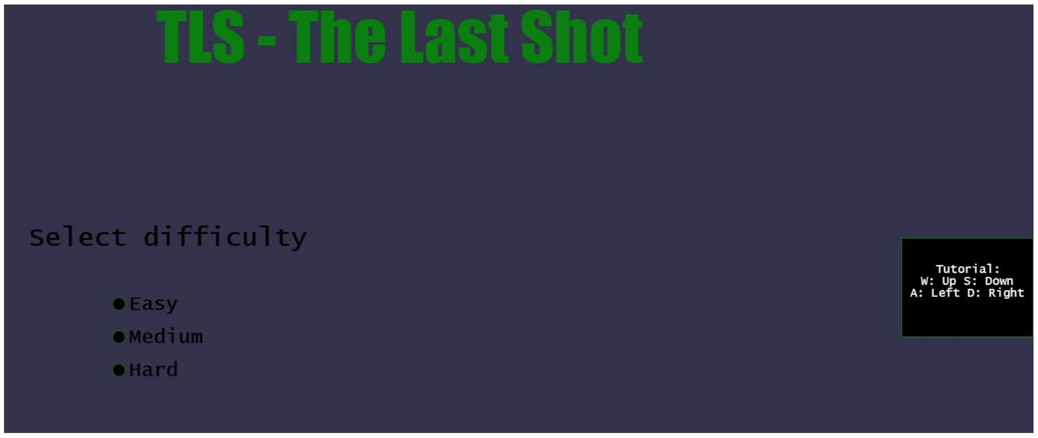

Figure 1.1: Home Page

User Guide

W : Move forward

S : Move backward

A : Move left

D : Move right

Space bar: jump

M : hallway lights

T : Arena lights

U : Start animation

P : Start game

Left click : Gunshot

R : Weapon reloading

All of the user interactions just described are present within a TextBlock throughout the game as reminders to the player of what they can do. The player can move the camera exactly like an FPS game, so rotating the mouse will also rotate his or her view.

Chapter 2

Environment

2.1 Libraries

babylon.gui.js:
Library used to develop and display advanced user interfaces(gui)

babylonjs.loaders.min.js:
Library used to import external objects and models.

cannon.js:
Library used for physic contents

pep.js:
Library to support the develop advanced user interface.

2.2 Scene

A scene in Babylon.js represents a virtual space in which 3D objects, lights, cameras and other entities can be created, positioned and manipulated.
Within a Babylon.js scene, you can define and manage the geometry of objects, apply materials and textures, add lighting effects, configure animations, and much more. It is the main context in which all rendering and interaction takes place. The figure below shows the start of the main scene in the project.

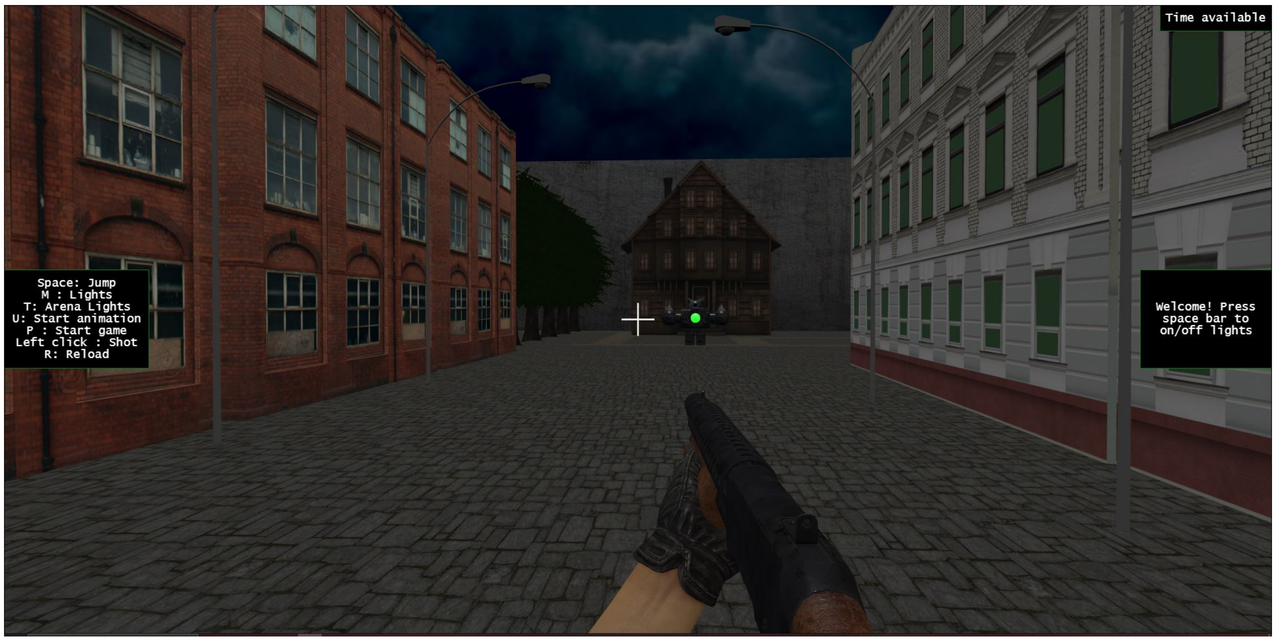

Figure 2.1: Main scene

2.2.1 Models

Since this is a first-person game, there is no clearly visible model, but as the term itself says, the in-game framing allows only arms and rifle to be seen. To do this, the ImportMesh function was used, which allows the import of a pre-existing model. Meshes is an array of meshes that makes up the model. Inside the ImportMesh function are all the commands for managing meshes, such as scaling, rotation, position, and camera parentage. The animations in this case represent the firing of the bullet and the reloading of the gun.
In addition to this, however, many other imported models were used to make the environment as realistic as possible.

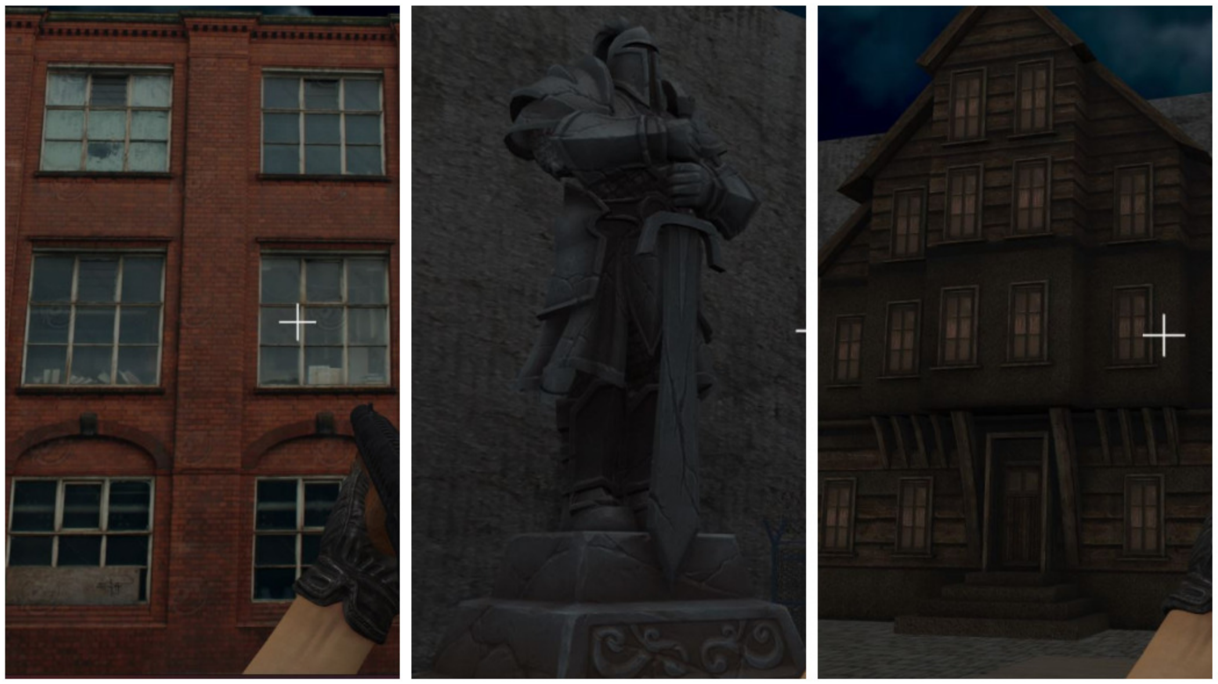

Figure 2.2: Models imported

2.2.2 Hierarchical model - The Robot

The request was to create a more complex hierarchical model than the one in the second homework. Thus was born the idea to create a robot that had different parts of its body animated by rotations and translations. The model consists of cubes, cylinders, cones, and spheres. Each is connected to another with a parent attribute as each mesh. The torso of the robot is actually an intersection between a box and a sphere and is the main parent of the model. Later we will see that in reality the torso will also be the child of an invisible box, called a hitbox, which was used for collision management in the game.

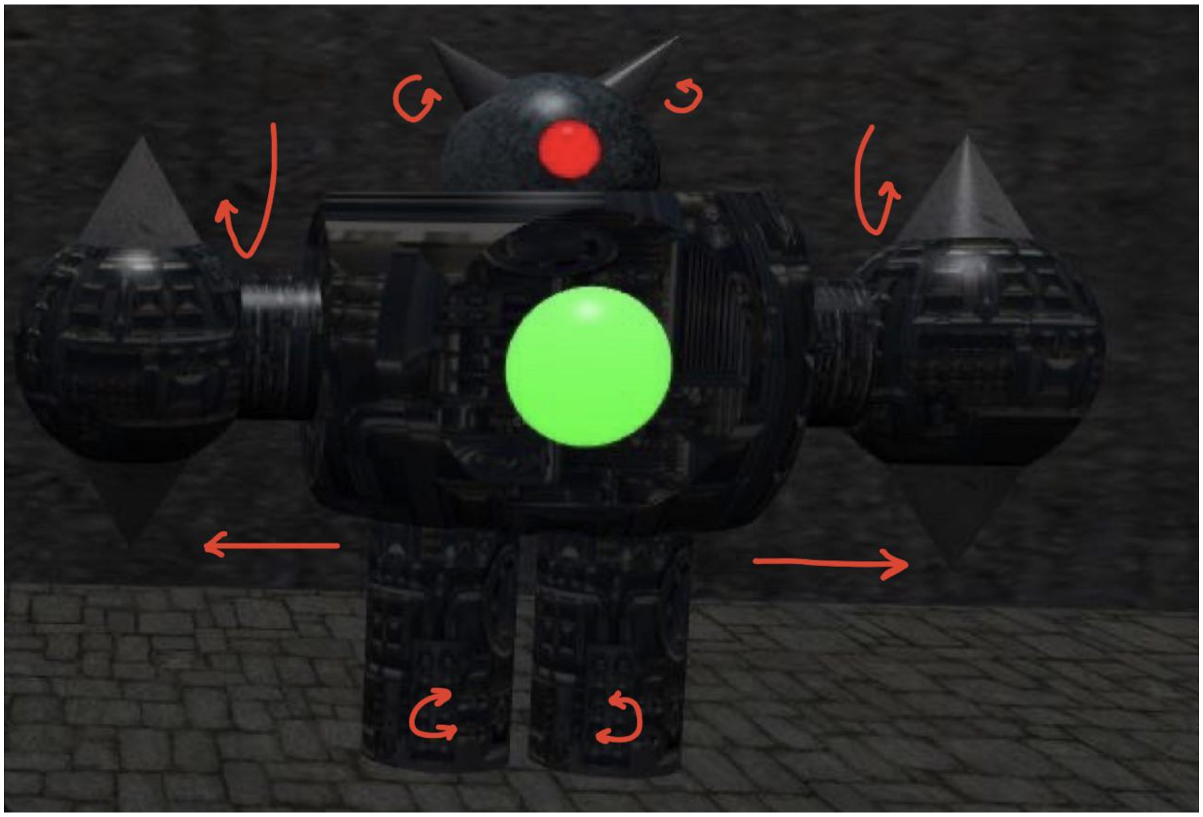

Figure 2.3: Hierarchical model

2.2.3 Camera

There are two different cameras in the game scene, a universal camera that is the one that regulates the player's movements, which then responds to movement commands and moves accordingly. The other camera, on the other hand, is a FreeCamera used to create the gun pointer. This camera is very useful because it is the one that is used in projectile management in that the projectile must go in the direction of the camera related to the pointer and not the universal motion camera.

2.2.4 Lights

There is one hemispheric light and several point lights in the scene. The hemispheric light is very useful for simulating ambient light. The point lights, on the other hand, were used and placed near each lamp in order to make the light coming from the lamp more realistic. The robot during its animation to reach the center of the arena, near the entrance, as it passes, the 3 point lights in the arena will light up, and then these lights as well as those in the hallway will be controllable by pressing a button.

2.2.5 Textures

Every object, model or mesh in the application has texture. The material used is StandardMaterial and the textures are images or colors applied to it. In the specific case of the sphere in the center of the robot's torso, the texture changes when a certain condition occurs. The entire scene is enclosed within a skybox to which images of the sky are applied as textures to make it more realistic.

2.2.6 GUI

Babylon.GUI uses a DynamicTexture to generate a fully functional user interface. Since there are a number of interactions that the user can perform, it is important to keep track of this, and so useful rectangles and text blocks with summaries of this important information have been included within the scene. The box at the top right contains the seconds available, relating to the difficulty that the user has chosen in the initial scene. These seconds will be dynamically decremented to the end. The box on the right, on the other hand, updates dynamically based on what the user can do, such as, for example, once a certain point on the map is reached, the user can press a key and start the animation. The box on the left instead is a summary of the user interactions available.

Chapter 3

Physics

3.1 Hitbox and bullets

Collisions and gravity are very important elements in an FPS game. To enable physics in Babylonjs, simply apply the following function to the scene:
scene.enablePhysics(gravityVector, physicsPlugin) where :

- **gravityVector** is a three-dimensional vector that defines the direction and intensity of the simulated gravity acceleration
- **physicPlugin** is Cannonjs.

To enable interaction between objects, the physics engine uses the **Impostor**. The impostor can be assigned physical attributes such as mass, friction, and coefficient of restitution. Cannon provides several forms for impostors (box, sphere, ground, particle, mesh), and in this case it was applied to box and sphere objects.
As mentioned earlier, the hierarchical model has torso like main parent, and all elements are connected between it, but the torso is the child of the **hitbox**, which is responsible for the actions expected from the collision.
The impostor is also applied to the bullet, which must also have the **ApplyImpulse** method necessary to generate the motion. The bullet has mass, which is precisely why its motion will not be precisely along the z-axis, but it will still fall downward, driven by the force of gravity.

3.1.1 Interactions

Interactions between impostors are handled with **registerOnPhysicsCollide**, which adds a call-back function that will be called when an impostor collides with another impostor. Whenever a collision occurs between two meshes, what is written in the function is triggered, and some action will start.
registerOnPhysicCollide is inserted into **scene.registerBeforeRender()**, this executes the function before the scene is rendered.
Since the rendering is started in a loop, collisions are monitored at all times, every frame in the game is constantly checked.Within the function designed for control, the robot's life decreases, so that subsequently the output conditions are carried out starting from that datum. Furthermore, the exit is foreseen even if the time runs out.
Although it is not exactly an effect given by physics, but for each model imported into the scene, a transparent box has also been created (isVisible(false)) which prevents the user from entering the models.

Chapter 4

Animation

Babylon.js has its own method to animate objects correctly. Use frames, keyframes and framerates to generate packed animations. I then created animations by translating and rotating the meshes in the registerBeforeRender function and using the physics features. For simplicity, I indicate that the movement from right to left is along the x axis, while that from forward to backward is the z axis.

4.1 Robot

4.1.1 Walk to the arena

The robot consists of two large different animations. In the opening scene we find the robot in the middle of the corridor waiting for the input to be given. At that point, after pressing a key, the robot will begin to move in the direction of the arena, and the movement is a combination of rotations and translations, and each part of its body moves according to a very specific logic. Once near the arena, the arena lights will automatically turn on and the robot will position itself, ready to start the game.

4.1.2 The Game

The second animation, on the other hand, is simpler than the first one as the robot will move along the x axis, always within the area. The robot's number of lives is 20, and as it loses lives, its speed will increase and the color of the sphere in the center of its torso will change.

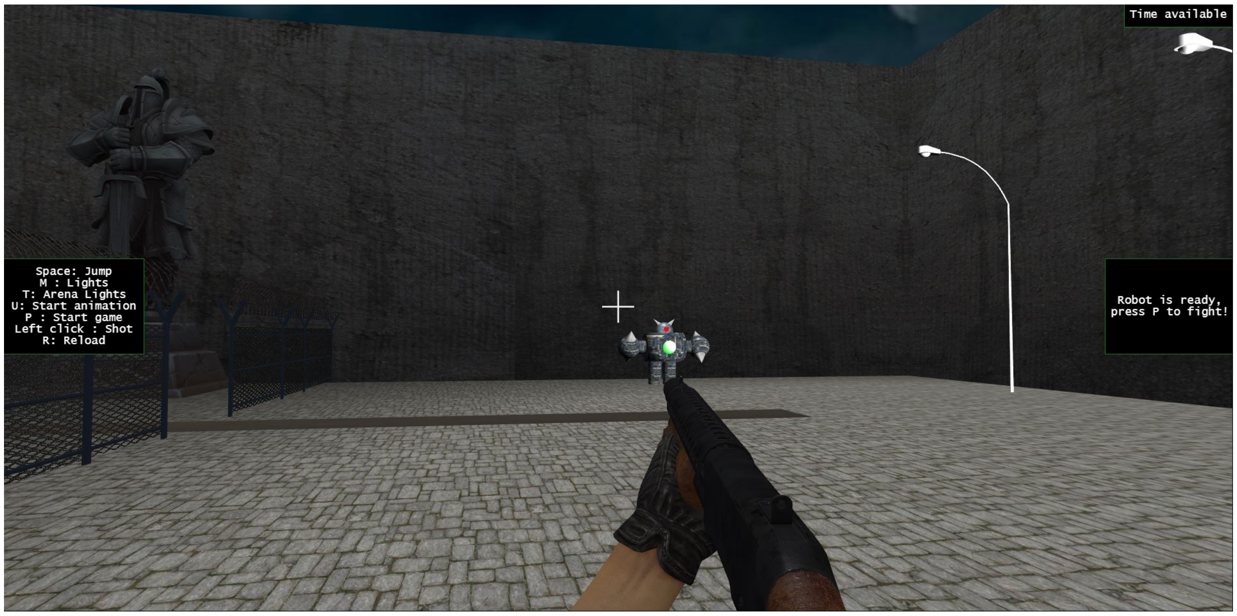

Figure 4.1: Game mode ready

4.1.3 Particle System

To make it more realistic that the robot is destroying itself I used a Babylon module called **ParticleSystem**. It is an integrated module that allows you to easily create and manage complex and realistic particle effects within your 3D scenes. It is a very powerful component that allows you to add immersive visual effects such as smoke, fire, sparks, snow and many more.
In this case, a ParticleSystem object is created for the explosion effect. Various properties are set to control the appearance and behavior of the particles. Specifically, the emitter is set to the robot's torso, indicating that particles are to be emitted from it. To start the explosion effect, call the start() method on the object.

4.2 Character movement

Being a fps game, there are no particular elements to animate, as there is no real model with components. In any case, however, with the model it was possible to import simple animations of shooting and reloading the rifle, but for project indications they were not included.