
SAPIENZA
UNIVERSITÀ DI ROMA

Interactive Graphics

# Final Project

*Presented by:*
Andrea Aurizi

*Submitted to:*
Marco Schaerf

Anno Accademico 2021/2022

# Contents

# Chapter 1

# Introduction

## 1.1 Basic idea

The idea of this project is to create a platform game where the player is able to move a man though a corridor full of dangers, survive until the end and beat the final boss with at least one life left.

Figure 1.1: Game screenshot

## 1.2 User Guide

W: Move forward

S: Move backward

A: Move left

D: Move right

Space bar: jump

B: change camera setting

K: change colour lights on the final stage.

R: restart after win or defeat.

At the beginning of the game there is a tutorial, that appears in the advanced gui, and it's developed to introduce the player to the principal commands that he needs to know (move and jump).
Clicking B is possible to switch to two different camera configurations:
- arcade mode (suggested). Very high, looking to the ground with distance visual limitations;
- immersive mode, that is the default one, that is directed to the back of the player and it allows to see until the end of the corridor.
Lives are displayed in the top right corner in game and they are 10.

# Chapter 2

# Environment

## 2.1 Libraries

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

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

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

cannon.js:
Library used for physic contents.

## 2.2 Scene

A scene is a container where meshes, lights and cameras are placed and, thanks to the engine, is made to work. The scene is a really minimalist environment (light, ground and skybox) where are located the imported main charter and the other imported meshes. This scene is analyzed in this section.

### 2.2.1 Medels

dude.babylon is an imported hierarchical model with a complex structure and texture used in the game as main character.
The function that permits the import is BABYLON.SceneLoader.ImportMesh(id name, path, filename, scene, function (Meshes, particleSystems, skeletons).
Meshes is an array of meshes that composes the model, particleSystem is a variable to handle particles and skeletons contains all the bones as shown in the following figure.
Inside ImportMesh function all the commands to handle Meshes are inserted like scaling, rotation, translation and physics.
The animations of the mesh imported with it have not been used, but i develop an animation by myself as asked in requirements.


Figure 2.1: Dude with relevant bones and the other imported mesh

**Hierarchical model - The Final Boss**

In order to satisfy the request for a more complex hierarchical model than the kangaroo of the second homework, i developed a final boss where over half of the component were animated.
The model is composed of cubes, cylinders, cones and sphere.
Each of them is linked to another one with a parent attribute as every mesh. The bigger cube (box upper) is the root parent of the model.

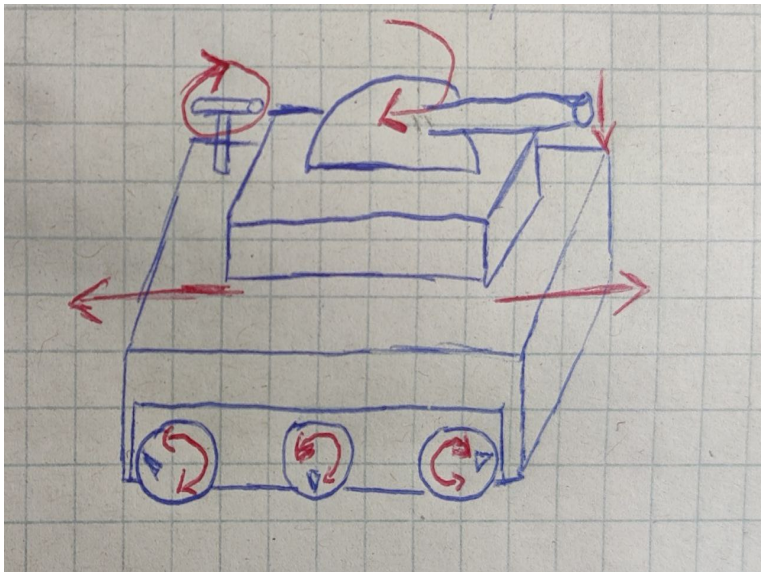

Figure 2.2: Dude with relevant bones and the other imported mesh

### 2.2.2 Camera

In the game scene there is a follow camera. ”Follow” indicates that there is the possibility to define a mesh as lockedTarget and the camera will center the vision to it.
The chosen mesh is the imported ”dude”, so every time the player moves it, camera will translate with it.

### 2.2.3 Lights

There is an hemispheric light and three pairs of spot lights.
A hemispheric light is an easy way to simulate an ambient environment light.
When the character arrives into the boss range, the hemispherical light turns off and the first pair of spot lights turn on.

Each pair of spot lights have a different colour: red, green and blue.

### 2.2.4 Texture

Every object, model or mesh in the application has textures. The used material is StandardMaterial and texture are images or colors applied to it.
The world is closed within a skybox to which sky images are applied as texture. The cube is bigger than the ground and it's possible to see it switching in immersive camera mode.

### 2.2.5 GUI

Babylon.GUI uses a DynamicTexture to generate a fully functional user interface.
Gui is composed of Rectangles and Textblocks that displays fps, life number and an hint section.
These components are above the screen and will follow you in the game.

# Chapter 3

# Physic

Collisions and gravity are very important part in a platform game.
To enable physic in Babylonjs it's only needed to apply to scene the following function:
scene.enablePhysics(gravityVector, physicsPlugin)
where gravityVector is a 3-dimensional vector that define the direction and the intensity of the simulated gravity acceleration and physicPlugin is Cannonjs.
To allow interaction between objects, the physics engine use the impostor, which represent also the complex object with simpler one.
The impostor can be assigned physical attributes as mass, friction and a coefficient of restitution.
Cannon makes available different shape for the impostors (box, sphere, ground, particle, plane, mesh), but for a simpler computations I decided to use only boxes and sphere.

## 3.1 Joint

The physic realization of the wrecking ball was a challenge, because the swing didn't allow the the wrecking ball to maintain a fixed distance from his 'origin'.
So I located another sphere over the first one and i applied a sphereImpostor without the gravity, by the setting of mass equal to zero. So it is like fixed.
To conclude I used the function addJoint to keep the two spheres at a fixed distance and I create a line which has them for extremes.

## 3.2 Hitbox

When we consider more complex meshes like dude I decided to cover him in a invisible box and apply the impostor to it. Cannon gives the possibility to apply MeshImpostor but it's heavier(in terms of memory) than a common boxImpostor. After that box is set as parent of the mesh so if there is an interaction or an animation, mesh and box move in the same way.
Along the entire path, the buildings and the crates are covered by invisible box of mass 0 that impose the character to follow the expected path.

## 3.3 Interactions

Interactions between impostors are handled with registerOnPhysicsCollide that add a callback function that will be called when an impostor collides with another impostor. Every time there is a collision between two meshes it triggers what is written in the function.
registerOnPhysicCollide is inserted in scene.registerBeforeRender(function() ...); that execute the function before the scene is rendered.
Since the rendering is started in a loop, collisions are monitored in every moment. Interaction are all between a mesh and dude hitbox.
Walls, wrecking balls, poisonous soil and ammos interaction with the player activates the callback function that removes a life and plays a scream sound.
Street and floating ground interaction with the player enables jump command.

# Chapter 4

# Animation

## 4.1 Walking walls

Player z positions trigger walls animation, that just translate the objects in Y axis. Each walking wall is composed by two walls, each one with its own hitbox, after they reach a specific location the wall is dispose. Position control and translation are develop in the RegisterBeforeRendering function. Dispose function delete the elements from the scene.

## 4.2 Wrecking ball

Knowing the structure explained above, the animation is done applying an impulse to the demoler1.sphereImpostor. physicsImpostor.applyImpulse(new BABYLON.Vector3(300000, 0, 0), demoler1.getAbsolutePosition());
The first vector define the impulse direction and intensity and the second one retrieve the position where apply the impulse.
It gives to the sphere a linear speed that decrease in time. The commuter of the sphere is given by the joint with the other sphere that, imposing a fixed distance between the two, does not make it move in a straight line.

## 4.3 Poisonous soil

There is a box over the ground that simulates a ground covered by poison. Above there are 13 static floating grounds where it is possible to jump to avoid damage.

The jump is realized with the same function used with the wrecking balls. When the spacebar is pressed an impulse is applied under the character hitbox that makes it jump. Given that the character can jump only when he is standing on a surface, I used a flag 'jump check' that is set false by the handler of spacebar and then reset to true when the character collides with a surface (ground, poisonous soil or floating ground).

## 4.4 Character and boss movement

Dude is composed of a skeleton with 57 bones that are editable (translation and rotation). In order to make a smooth animation, i included translations and rotations of arms and legs in registerBeforeRender function.
When the event w,a,s,d for the character movement are detected, the body (dude hitbox) is translated on Z axis with 'w' and 's' or X axis with 'a' or 'd', body rotates and animation starts in loop.
If keys are all up dude returns with his bones to the initial position and rotation.

The animation of the final boss is triggered by player position and it starts by rotating on Y axis the antenna and on Y axis the sphere to which the cannon is attached. When the cannon is parallel to the path, the sphere rotates on X axis aiming the cannon at the height of the character.
After five seconds timeout the root model is traslated on X axis, so the child components follow it. After a total of 45 seconds, animation stops.

## 4.5 Ammos

When the final boss starts moving on X axis, after 0.5 second start the animation of the tank attack, that fires sphere of cannon that translate on Y and Z axes. Ammo are physics object (sphere) that do damage to the character.
Every ammo is created after a time interval, that decrease twice before the end of ammos and the defeat of the boss.
After a specif position, the ammo is disposed.