Report Interactive Graphics
'Jenzo'
Teacher: Marco Schaerf

Leschiera Lorenzo, ID: 1325697
Lucia Jessica, ID: 1789469

a.y. 2017-2018

Contents

1 Jenzo Game

1.1 Overview

The game is characterized by two main screens: the first one is contains the menu and permits to start the game clicking on the Start button, whereas the second one is that in which the game takes place.

1.2 Introduction

`Jenzo` is a shooter game, in which the player is the shooter and the victims are the other characters who are inside the scene. The `aim` of the game is to shoot all victims.
In the scene you find both moving characters and still characters, to which the player can shoot.
From the initial menu the player can start a new game (`Start` button), read the instruction of the game (`Instructions` button) and read credits (`Credits` button).
At the `Start` clicking, the player is called to set the number of characters to shoot. Then he can start the game clicking on `Play!` button.
As soon as the game page is loaded, the shooter will be on the rooftop due to default parameters' definition. From its overview, the shooter can change the direction pointed by the viewfinder by simply move the mouse.
After the victim has been selected and the viewfinder has been positioned, clicking with the mouse fires the bullet.

There are only `9` bullets available for the shooter and the remaining of them are indicated on right bottom of the screen in the bullet icon.
At the shot of the bullet, the time will be slowed down in order to follow the bullet's trajectory and see if the target will actually be hit.

If the shooter has been precise, the hitted body part of the target will turn red.
According to the hitted body part, points are awarded. Moreover, for every bullet unfired, `1000` points more are earned.

From the main screen it is possible to click on the :

-  icon to be able to set the different game parameters, or
-  icon to return on the main menu.

Among different settings, is possible to set the daytime, the gravity and the position in which the shooter will be positioned i.e. therefore the prospective from which he will be able to aims the victims.
Finally, is also possible to set the default values (by clicking on the `Default` button), to cancel the changes made or to save them.(*Figure 2*)

2 Implementation

Let's see how we have employed this game.

2.1 Enviroment

We have used a `Babylon.js` that is a real time 3D engine using a `JavaScript` library for displaying 3D graphics in a web browser via HTML5. The source code is written in TypeScript and then compiled into a JavaScript version and it is available on github and distributed under the Apache License 2.0. The Babylon.js 3D engine and user code is natively interpreted by all the web browser supporting the HTML5 standard and WebGL to undertake the 3D rendering.

2.2 Library

The library used are:

- **babylon.custom.js**: that is the base library necessary to build 3D application using `Babylon.js`.
- **babylon.inspector.bundle.js**: that is the library through it is possible to debug the babylon objects.
- **Oimo.js**: that is the library containig the physics.
- **jquery.js**: jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.
- **jquery-ui.min.js**: jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.

3 Technical aspects

The project is composed by the following `js` files:

- **Animation.js**: it contains the functions needed to animate the character.
  In particular in this file are defined the rotation of the character's body parties and two functions that permit it to translate and simulate the walk.
  There are two functions that simulate the walk, one for each of the two characters who move in the scene. In particular one walk function is defined along z direction, and the other one along x direction.

- **Character.js**: it contains all detail regarding character like its hierarchical structure and the behaviour of its body when it is hit. In this file there is also the function that enables the physics for the character.

- **JavaScript.js**: it is the main file in which the scene is created, the render function is defined and are defined all other functions that allow the interactions with the scene.

- **Settings.js**: it contains all the functions necessary for the game settings.

and by the following two `html` files:

- **Index.html**: it is the start menu of the game,

- **StartPage.html**: it contains the scene of the game.

4 User interaction: game settings

It is possible to set the parameters of the game from the main screen of the game by clicking on the top right on the *gear icon*.
Different settings available to the player are:

Daytime: possibility to set three different moments of the day: Day, Sunset and Night. (*Figure 4*)

Number of characters: possibility to set number of adversary to shoot, with a minimum of three characters to a maximum of five. (Settable only at the beginning of the game, when it is loaded after the start or restart command).

Gravity: through a bar is possible to vary the value of the gravity. This leads a variation in the projectile trajectory and in the behavior of the characters when hit.

Placement: possibility to choice the position in which the shooter (player) will be positioned, and in particular on the *rooftop*, or on the *street* or on the *balcony*.

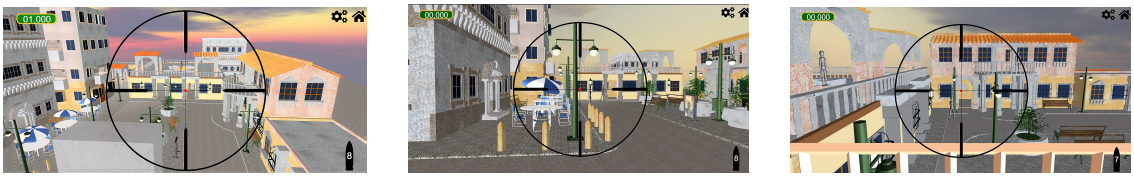


Figure 1: Different placement settable.

However, it is possible to start playing directly because there are default parameters already set:

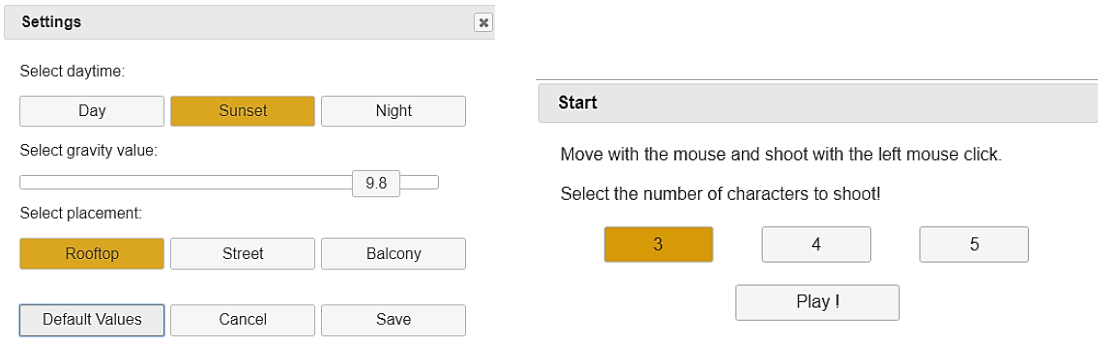


Figure 2: Default values

5 Scenary

Scenary is defined in `JavaScript.js` and it is imported from an existing scene created for **Autocad 3D max** software. The scene is modified and adjusted for our purpose, using **Maya** software. In particular we have used `MayaToBabylon` library to export the file in babylon format. We have often used *Maya* also when we needed to easily know the exact position of the objects, like when we was interested in to set the light for the night mode taking as reference the street lamps position.

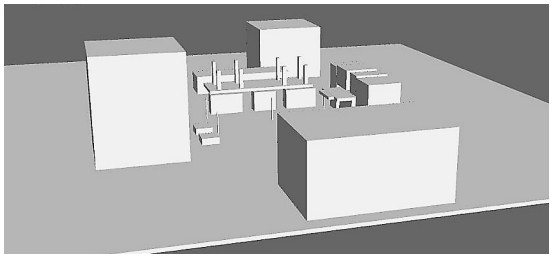


Figure 3: Collision blocks

In order to maintain a lightweight model we choose to implement the collision relatively to the scenary througth simple boxes defined in the `collision.babylon` file generated througth *Maya* software.

We have also defined a box (`skybox`) where we have applied the texture of three possible type of sky that the player can be choose.

6 Lights and Textures

6.1 Lights

The Lights are defined in the `Settings.js` file in the function `getLight()`. Depending on the chosen `daytime` the function returns the expected light setting. In fact, we have set the light for the three different moment of the day changing its positions and its intensity. In particular, when the night mode is setted, the `getLight()` function returns a vector of four values representing the light that comes from four of the street lamps.
For the other two moments of the day, we have setted only one value representing the light comes from the sun.
We have used:

- **PointLight** for the **street lamps** and for the **sunset**.
  A PointLight is a light defined by an unique point in world space. The light is emitted in every direction from this point.

- **HemisphericLight** for the **day**.
  A HemisphericLight is an easy way to simulate an ambient environment light. A hemispheric light is defined by a direction, usually 'up' towards the sky.

6.2 Textures and Materials

Among different textures applied, we have used a pre-created and imported texture in order to represents the entire world of the **city**.
On this texture we have positioned another one: a fixed **viewfinder** through which the shooter can targets the victims.
At last, we have also used three different textures for the **sky**, one for each of the possible choice that can be make by the player selecting one of the specific daytime (Day, Sunset, Night).

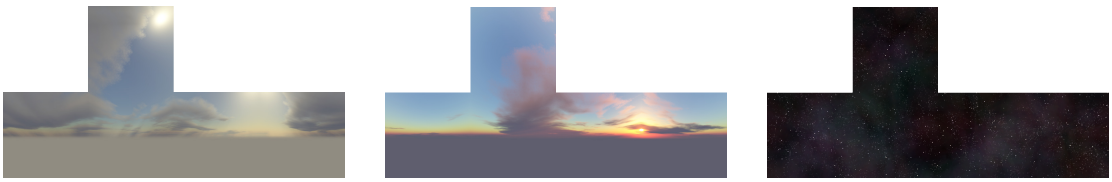


Figure 4: Textures used for the sky. Rispectively: Day, Sunset, Night.

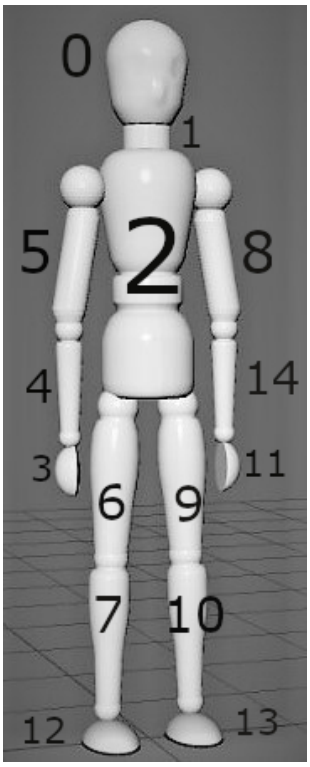
Figure 5: Day, Sunset and Night Textures applied to the worl game.

For **bullets** and **characters** we have used the Materials.
In particular, we have applied a **Cropper** material for the first one, and an imported material for the second one. Moreover, for the character, we have set a color change of the material when a particular part of the body is hit.

Figure 6: Textures used for the character. Rispectively before and after to be hit.

7 Chararcters

7.1 Hierarchical structure



The structure of our character is based on a hierarchical model.
It was initially imported as a pre created and unique mesh and thanks to `Maya` software we had divided it in more parties.
In particular we have divided it in 15 meshes: head, neck, body and all parts of its limbs.
Using the babylon function `addChild()` is possible to create a hierarchical structure where, in our case, the body is the chosen root.

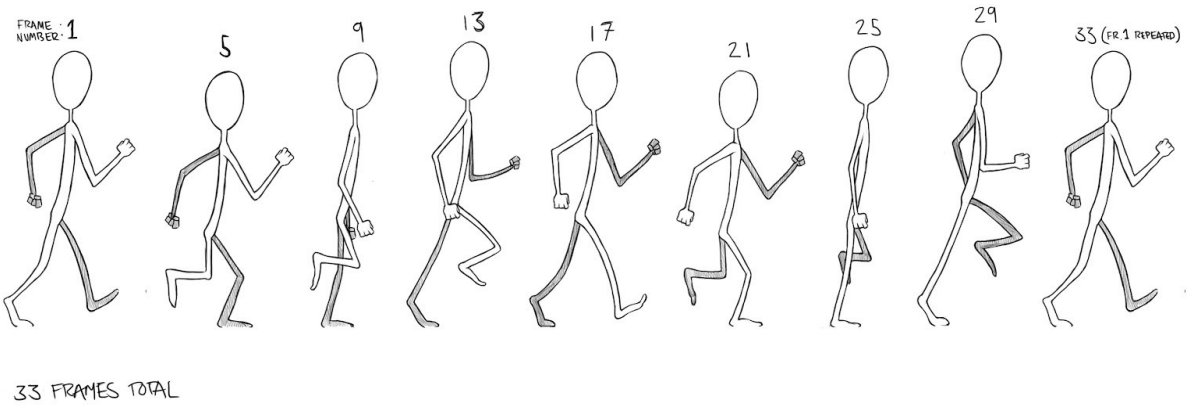
Specifically, we have realized this structure:

- **body**^{2} $\rightarrow$
  - $\rightarrow$ **neck**^{1} $\rightarrow$ **head**^{0}
  - $\rightarrow$ **forearm**^{8,5} $\rightarrow$ **arm**^{14,4} $\rightarrow$ **hand**^{11,3} (respectively for r and l)
  - $\rightarrow$ **upperleg**^{9,6} $\rightarrow$ **downleg**^{10,7} $\rightarrow$ **foot**^{13,12} (respectively for r and l)

In order to assign a score when a character is hit, for each mesh composing its body, we have used the `state` property of the babylon mesh, writing inside it a string containing a specific score relative to each part of the body.
One score is assigned for each hit character. The total score is obtained by adding up all the points gained killing the victims plus a bonus linked to the number of un-fired bullets. Obviously, to do this, every time the string representing the score has to be added to other scores, it must be parsed to an int number and then added.

7.2 Animations

In order to realize the walk animation of characters, we took as references the following image. In particular, we took the angles of all character parts of body and we used these values for our purpose.



The `Animation.js` contains the functions that allow the character to walk. In order to generate a fluid animation through the value obtained from the fixed frames, we used a linear interpolation between the various frames using the methods provided by babylon. To coordinate the movements of the different parts of the body we used the `AnimationGroup` function that groups the individual animations to start them all together.

8 Physics

`Babylon.js` has a plugin system for physics engines that enables the user to add physics interactions to the scene's objects. The physics engine calculates objects' body dynamics and emulates "real-life" interactions between them. Among the plugins available for the physical engines we have chosen the `Oimo.js` because it is more lightweight.

8.1 Characters Physics

Our game design choice to animate the character dynamically (the part of the body responds in a realistic way to the hit and not in a predetermined manner) has led some difficulties.
If the character has a hierarchical structure, as in our case, when physics is set for every part of the body, babylon assigns a single collider (`phisicsImpostors`) for all the parts, making the collision appears to the root of the structure (that in our case is the body).
This setting of babylon causes any impulse applied to a part of the body to be automatically perceived only on the body, "blocking" the independent movement of the other parts. If you leave this type of setting, what happens is that when the character is hit, it will fall stiffly and then in an unnatural way.
To remedy this we decided to "disable" the hierarchical structure at the time of collision

with the bullet.
For each part of the body we have assigned a `physicsImpostor` with its own mass and elasticity value. In this way every part of the body can respond independently from the others.
What remains to be done is to tie all the parts of the body into one and we have done this using the `physicsJoint` that binds two impostors at a certain distance by simulating a real joint.

8.2 Bullet Physics

To hit a character the bullet must be fired from the "camera". Like for the body parts of the character, also for the bullet we have assigned a `physicsImpostor`.
At the click of the mouse, the bullet is fired in the direction of the camera target. We have used a babylon `applyImpulse()` function that, through an impulse vector and a point of application, fires the bullet.
In order to calculate the direction of the vector we have used the `Ray.CreateNewFromTo()` function (an invisible ray) that takes two points of space as input and returns the direction vector. The two points used as input are the position of the camera and its target.

9 References

https://jquery.com/
https://jqueryui.com/
https://www.babylonjs.com/
https://lo-th.github.io/Oimo.js/
https://www.autodesk.it/products/maya/
https://www.autodesk.it/products/3ds-max/
https://free3d.com/3d-model/wood-dummy-79325.html
https://www.turbosquid.com/3d-models/free-building-3d-model/569935