Sapienza Interactive Graphics Course
Final Project

Cabangcla.1806064 - Toccacieli.1799754

25 July 2021

1 Sapienza Athletics Game

Sapienza Athletics is a game set in a stadium , it rapresent the well-known sport of obstacle course. The main goal of this game is to avoid the obstacles , moving right or moving left, jumping over or slide below them in order to avoid to crash and continue the run. In the track we have 3 lines. On those lines , we will find different types of obstacles. Based on the type of obstacle that we will face, will be task of the user to choose the best action, in order to avoid them.

1.1 User Manual

Your primary game input device is the keyboard. You can controll the runner by using $WASD$ or the arrow keys input $\leftarrow \uparrow \rightarrow \downarrow$:

- W / $\uparrow$: **Jump**
- A / $\leftarrow$: Move to **left**
- S / $\rightarrow$: Move to **right**
- D / $\downarrow$: **Slide**

2 Environment, Libraries And Models Used

To develop the project we mainly used **Three.js** that is a JavaScript library and an application programming interface based on WebGL to create and display animated 3D computer graphics. Furthermore, two javascript libraries **Dat.gui** and **Stats.js** are used which respectively allows to create a user interface to modify the game settings and to measure the program performance by showing the frames per second during the run time. The project was implemented using the **Visual Studio Code** IDE with a sharing functionality that allowed the team to write simultaneously on the same source code. In addition, different color textures (images) and bumpMaps were created with **Adobe Photoshop**. All the 3d models used in the project have been implemented and designed by the team using a mesh of primitives of Three.js library.

3 Technical Aspects Of The Project

3.1 Lights

The lights used in the development of this game are divided into two main events : Day and Night. During the **day** we decided to implement an **AmbientLigt** to go with three **Directional Lights**, this decision have been made by us in order to give a full vision over the map , avoiding spots where light would not reach completely. The three directional lights , beeing emitted in a specific direction , are respectively emitted from the right , left and from behind the main character. During night we apply an **AmbientLight** too , but the amount of light is way less , giving a moonlight effect to the field. Being in a stadium involves the useof stadium lights ! That is why we decided to implement some lamps , that will be described later on, where we implemented **Spotlights** (one for lamp). Each stadium lamp in fact in provided with one **Spotlight** that from the lamp it self aim to the field , in order to give visibility, to the runner, of the trak and the eviroment that is around him (so to the User). The Spotlight properties have been modeled in order to give a specific angle , intensity , distance and decay to the light emitted by it. In order to give more realism to the lamps we decided to apply an **Emissive** property to the lamp bulbes , such that when Day mode is active , they are not affected by light , and when Night mode is activated , they are completely

bright, giving the effect to be turned on and off switching from Day to Night and vice versa. Lastly in order to apply effectively the change between night and day we have used the method *visible* that ,applied with a boolean value , let the lights visible ,or not ,such that we could easily switch between the two different modes (night and day mode), by switching this boolean value.

3.2 Color and Bump Textures

For the world objects of the game , we used different Texture types called into the phongMeshMaterial (by THREE.JS). The first type we used is **Color**; we have applied this to the most of our objects , such as the main character , spectators , roof of the stadium, and details such as the lamps bases and the olympic logo. The second type that we used is the maptexture (implemented using the method provided by $THREE.TextureLoader$) that let us implement **Texture Image** directly onto our objects and then we repeated the texture when needed thanks to $THREE.RepeatWrapping$ and the method $texture.repeat.set(value, value)$. With this ones we were able to load as much as many textures images we wanted and model them in order to respect the size of the object that was host of the texture withoput loosing any quality of the image. We have implemented this to the stadium (having spectators as a texture image), for the track, for the field , for the banners and bostacles , and lastly for the lamp bulbes. Seeing that just the texture on the field and track didnt give enough reality we have decided to implement the **Bump** on them , and this was possible thanks to Adobe Photoshop. In fact thanks to it , from the texture we were able to generate the **Bump** texture related to it , having the possibility to model the amount of shapes and depths of our texture model. Then we have implemented it in our code and here , down below, is shown the process :

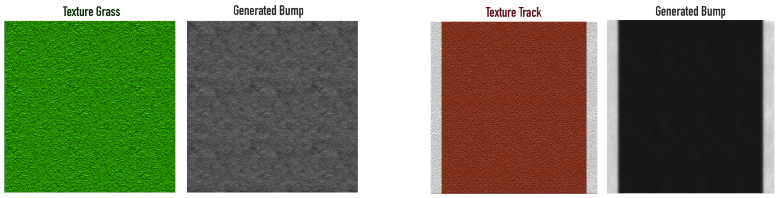

Figure 1: Textures and their Bump

3.3 Hierachical Models

3.3.1 Hierarchical model of the scene

In Figure 2 is represented the hierarchical model of the scene inside of which there are the two cameras, one for the third person view and the other for the first person view placed as a child of the head of the main character, moreover different lights are defined that are activated and deactivated according to the time of day.

3.3.2 Hierachical model of the human

Figure 3 shows the hierachical model of the human. The main components of the human are highlighted in red color and those are the main joints used to perform the animations of the runner and spectators as we will see in the next paragraphs.

4 Animations

All the objects in the scene are animated through the interpolation functions of the Tween.js library which allows to determine the value of the displacement, rotation and offset of the textures, Meshes and cameras to be applied in a given time interval.

4.1 Runner Animations

For this section we will refer to the table shown in the previous 3 , where we show clearly how the main character is defined. The animations of the Runner are nine in total , one for the starting position , one for loop running , two for the jump , two for the slide , one for move to the left and one for move to the right , and the last one that is when we crash into an obstacle. Then animations are implemented using TweenJs.

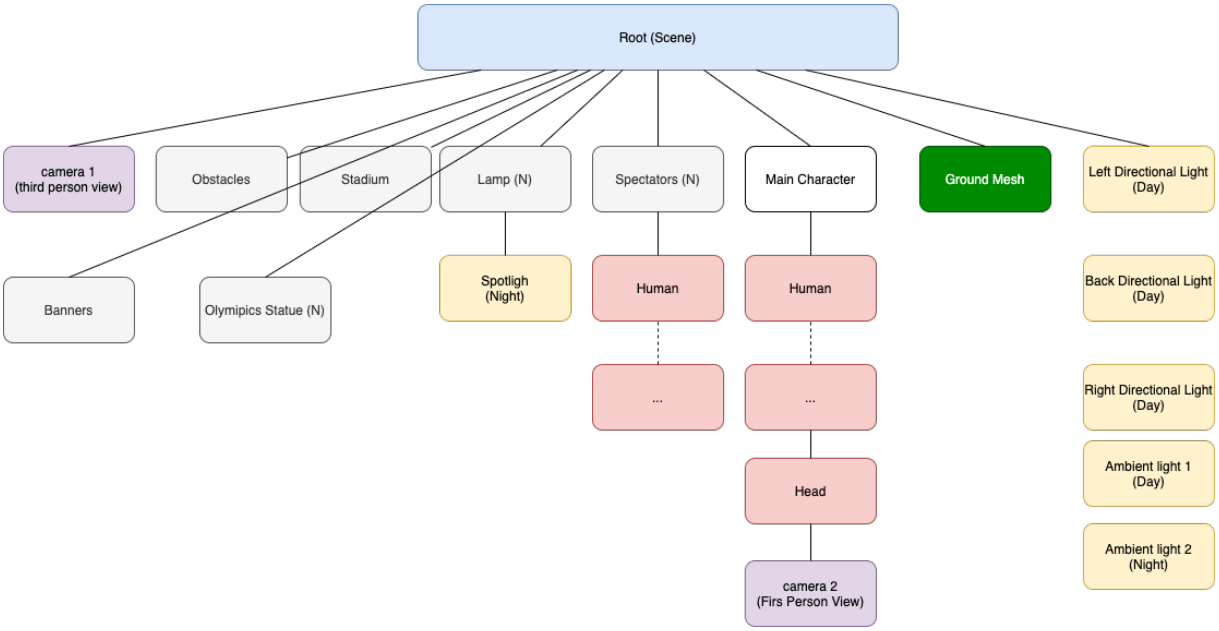

Figure 2: hierarchical model of the scene

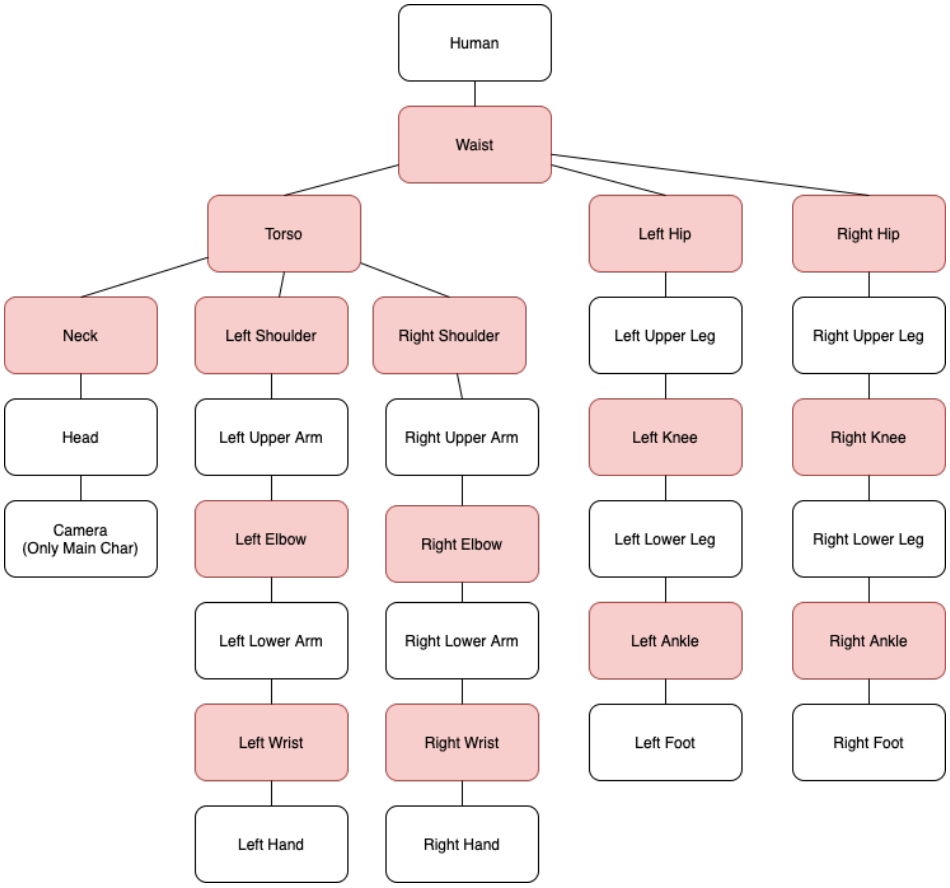

Figure 3: hierarchical model of the human

4.1.1 Running

For the animation **Running** we use the property *loop* (applied only to the main animation) that let the animation run infinetely (except when overrided or subject to another event). In order to tell to the particular joint involved where to move or where to rotate we use the method *.rotate/.position* with the method *.to*() in order to direct the rotation or position of the joint. The animation *Running* involves all the body modeling the different parts of it through the connectors of the different joints. We created a primitive *sphere* to every joint such that we could model this sphere and give a full movement to the different joints. This *spheres* are respectivly the elbow , knee , ankle , wrist and hips. To them we connected the different joints such as shoulder , lower arm , upper arm , foot , neck , waist , lower legs and upper legs. So for the **Running** animation we modeled the top starting from the **Torso** where we implemented a movement , that goes from left to right and returns, so that all the upper body such as Neck , shoulders and their children moves synchronously with it. Then going more deep we modeled the **Shoulders** such that they would rotate front and back asynchronously in order to give the effect of someone who's running. For the down side of the human we have modeled the **Hips** giving the same effect as the shoulders (just with a different value) of asynchronous rotation , back and front. The **Knees** have been rotated to from 0 to 90 degrees in order to give the bending effect to it , same thing applied to the ankle , to apply the bending of the **Foot**. As said to all of this , it have been applied the *loop* property such that it would repeat and so give an infinite animation back and forth and so what it result is a running animation.

4.1.2 Starting Sprint

Now we pass to the single animations that will not require a loop property. The first animation that we implemented is the starting position , here we positioned the runner into the usual starting pose , bending the two **knees** differently , one for the landing leg position and one for the sprint leg position, with shoulders that point to the ground as a support. When we press start the animation will be triggered , and from this position the human joints will rotate smootly to the running position explained before and the **Waist** position will smootly increase along the y axis in order to stand up.

4.1.3 Jump

Once our human is running we need to prepare him to the obstacles , so we implement a **Jump** animation that will let him avoid the low type of obstacles. The animation , starting from the running one , will simply increase the **Waist** position , always along the y axis in order to raise the body, and rotate the **Hips** in order to avoid touching the obstacle (Left and right respectively in opposite direction). All of this in a single animation without the use of *loop* such that when finished the body will smothly return to the running animation (veing infinite it always work). To the jump as the slide we added a secondary variant , that is the flip. The flip in fact as the slide one simply rotate the **Waist** around 360 degrees in order to do a full spin around hiumself. For the jump as for the slide we had to bend the **Knees** and the **Hips** in order to not touching the obstacle.

4.1.4 Slide

As said before we implemented a **Slide** animation for the second type of obstacle that require to slide instead of jump. Here we simply use again the Tween animation with the respective *.to*() method that will position the **Waist** down to the ground and extend all the joints in order to fit under the obstacle. As said before even here we implemented a variant that is the flip , that in contrast to the jump animation this one is implemented on the ground so changing the value of the **Waist** position.

4.1.5 Move Right and Left

To move right and left we have modified the values of the position and rotation regarding the leg involved in the changing (so left leg if we are going left and right if we are going right). In addition to it we have added a light oppositite position to the torso in order to give the acceleration effect. All of this always in concomitance with the running animation.

4.1.6 Crash

Here we simply activate an animation when the collision is true , and stop the running animation using the property explained before , that is the *Override*. So we simply position all the joints to their default position and move the a **Waist** down bending the knees to 90 degrees on the back, such that gives the effect of a human that is in terrible disappointment for losing. Here we stop all the possibilities to jump or slide too.

4.2 Animating the cameras

As we said earlier there are two main cameras. These are also animated in the sense that they follow the runner as he moves between lanes. In addition, to create the visual effect of running and jumping, the positions of the two cameras are also animated with tween, moving slightly up and down continuously and higher when making the jump.

4.3 Animation of the secondary objects

The secondary objects (stadium, lamps, obstacles, Olymics statues, Spectators) of the scene are represented in the hierarchical model of the scene (figure 2) colored in grey . All objects to create the illusion of moving forward moves towards the negative direction along the z axis. As for the ground, which is always stationary, instead, its textures are animated by tweening the offset value. As for the spectators, they have the same hierarchical model as the runner and are three types of them, who perform different animations, the first is the spectator who applauds, the second is the spectator who jumps, and the third is the spectator who jumps with the sign in its hands.

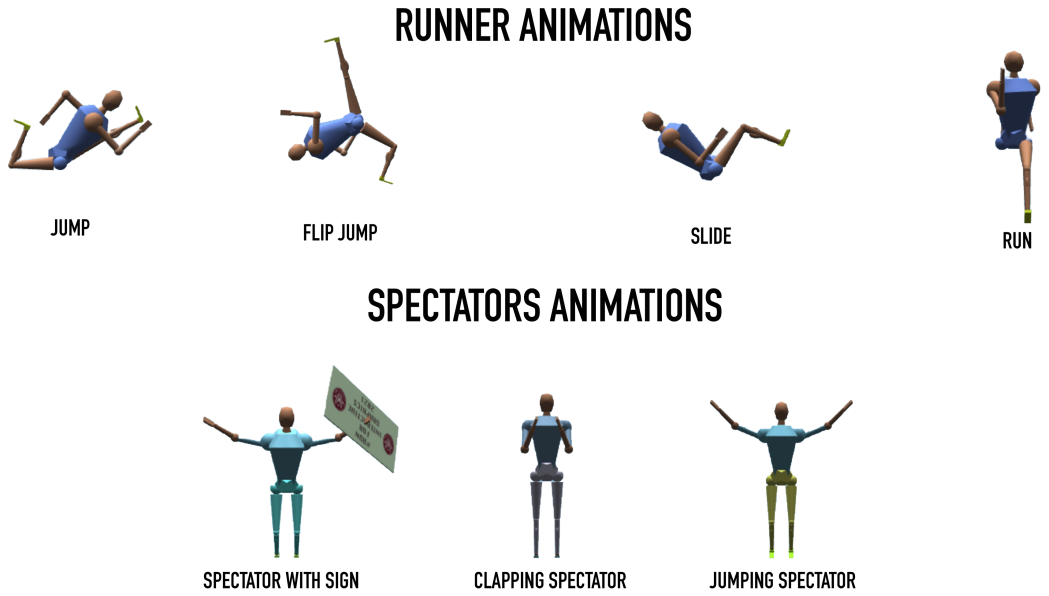

Figure 4: Animations of the runner and the spectators

5 User Interactions

User interactions are implemented through the standard **Event listeners** and the **Dat.gui** library that allows us to modify the parameters of the game via a small menu on the top right of the screen. The user can interact with the game in the following ways:

1. Dat.gui interactions
    - Change day time to night/day (switching all the lights in the scene)
    - Switch between first person view and third person view
    - Customize colors of the runner outfit using a color palette
2. Input game commands (left,right,up,down) using Event Listeners