SpaceRun
Final Project

Riccardo Pattuglia

July 2020

1 Introduction

What I'm going to describe is the realization of the final project for the Interactive Graphics course. It is about a WebGL based game, for which I've used the JavaScript programming language to build the logic of the game, with the ThreeJS library, which simplifies hugely the writing of webGL based applications, since it provides a layer of abstraction over the bare webGL implementation and TweenJS to achieve the animations. Obviously, being a web based game it also requires some html, with some css for styling, but this is not the main focus of the report.

2 The Game

The game realized is an endless run game set into the space. A robot must run through a spaceship corridor avoiding the laser beams placed into the ship. The objective is to make the best score you can by making the longest run. The speed of the game increases while the robot advances through the spaceship and in case of hit with a laser beam he explodes.

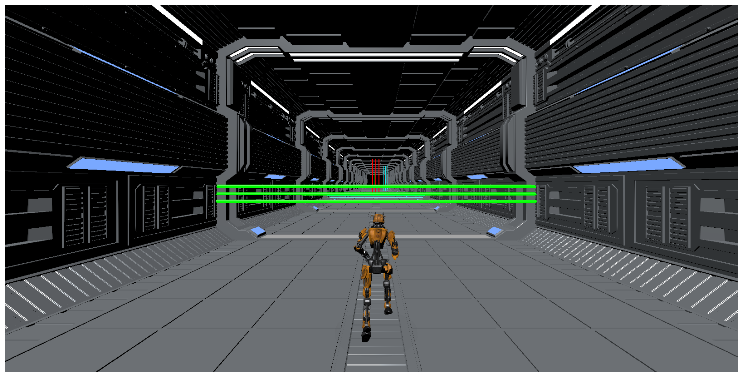

Figure 1: Gameplay

3 Development

The explanation of the development of the project can be divided into different parts, so that in each one we can focus on a specific part of the game,

3.1 Main Character: The Robot

The robot is a hierarchical 3d model made of many different parts (bones and meshes) that can be manipulated from the javascript code with ThreeJS, and has been imported from Sketchfab [1].

The robot is composed of almost 50 different parts that we can move (including fingers) but I used only some of them to achieve the desired animations.

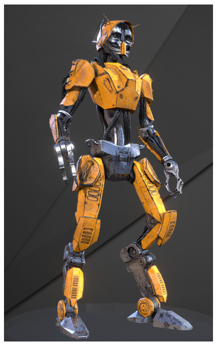

Figure 2: Main Character

3.1.1 Run Animation

The first thing I animated was the robot run, in which both legs and both arms must move accordingly to emulate the run, and also the robot trunk oscillates to make the animation more realistic.

The part involved into the animation are 14:

- 4 for the arms: left and right upper and lower arm.
- 8 for the legs: each leg is composed by 3 different parts plus the foot, there is the thigh, the calf and the so called horse link since it is similar to the horse leg as it rotates forward.
- 2 for the spine: it is divided into upper and lower part.

The animation is obtained by rotating the body parts with respect to the x,y,z axis and to have a smooth animation, independent from the FPS the TweenJS library has been used, since it gives the ability to decide how much time should require the animation to complete, and it also provides a way to chain the animations and loop them. In this way at the beginning of the run animation the body assumes the position of the first step, and then loops by rotating legs, arms and trunk to keep running. Also the spine the robot trunk is rotated towards to give the character the typical inclination of the run.

3.1.2 Jump Animation

Since the robot must avoid the obstacles he finds in his way he has the ability to jump over them. The procedure used to animate the jump is similar to the running animation: tweenJS helps with the continuous rotations updates, according to the time specified in input, and the parts involved into the animation are the same used for the running, obviously with different rotation angles. Both legs will fold forward by taking his knees towards the chest and both arms move over the head to give the impression of a real dynamic impulse. Moreover the robot is moved up with respect to the ground, this is the really necessary part of the animation otherwise it cannot avoid the lasers.

After touching the ground at the end of the jump the robot starts again to run.

3.1.3 Slide Animation

The other move given to the robot to avoid the laser beams is the ability to slide down them. Also here the sliding animation has been achieved by mean of tweenJS and always the same body parts are involved into the animation. Here there is a rotation of the body of the robot to go with his back to the ground and a vertical movement with respect to the y axis to have his back to actually touch the ground. Then, with one leg folded a bit and the other more extended he slides, while the arms will rotate at the height of the head opening his arms to expose the chest.

The robot will keep sliding for a limited period of time otherwise it would be too easy for the player to slide down the obstacles, and the sliding duration will change with the speed of the entire game, otherwise it would be too quick or too long to avoid the obstacles.

3.1.4 Horizontal Movement

The last movement the main character can take is the horizontal displacement on the x axis, this is achieved by simply making a fast transition from the actual point of the robot to the left or to the right with TweenJS, since in this type of games it is usually needed a fast lateral movement without too much attention to the animation of the body parts in this movement, that will not even be visible.

3.2 Scenario

For the scenario of the game it has been used a 3d model of a corridor [2], here we don't need to modify it but only to continuously cloning it while the robot is advancing through it, to give the impression that the corridor is infinite. For this reason there is a specific distance defined from the robot to the end of the corridor that whenever is covered leads the the creation of a new pieces of corridor that is attached to the end of the actual one.

Moreover there are some obstacles that are placed through the running path, the obstacles are laser beams and are of three different types:

- two horizontal lasers to jump over;
- three horizontal laser beams to slide down;

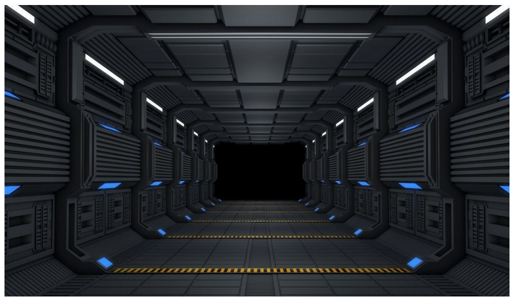

Figure 3: Corridor of the spaceship

- three vertical laser beams that must be avoided by moving horizontally;

The obstacles are placed at a fixed distance between them but the type of obstacle is different since it is randomly picked among the three possibilities. They have been realized by simple cylindrical geometries with an effusive color property to make them resemble actual lasers.

In the game there is also a directional light, color white, that allows the player to see the robot and also the scenario which otherwise would be not possible to see.

3.3 Explosion

The last important animation realized is the explosion generated whenever the robot hits one of the lasers. To create the explosion object into the scene I used an Icosahedron geometry while for the material I have used a ShaderMaterial with custom vertex and fragment shader and also a texture image. To create the explosion starting from an Icosaherdon geometry I've used the code for the generation of the Perlin Noise to modify the position of the vertices by moving the vertex along its normal by the displacement factor computed, while the fragment uses the noise value computed in the vertex shader to take the right color from the texture. Obviously altering the normals is not good for the computation of the lightning model but this is not needed since the object is completely emissive (producing light itself).

So the explosion is placed into the scene in the same position of the robot when the collision with one of the lasers is detected, and a tween makes it become bigger to give it an explosive effect.

3.4 Collision Detection

To implement the collision detection in a webgl environment there are many solutions, one of the most used is to use a physics engine which usually implements by default the collision between objects, but in this case, since the physics is not necessary for the purposes of the game, using a whole physics engine would have been overkilling. Moreover the collision detection system needed for the game

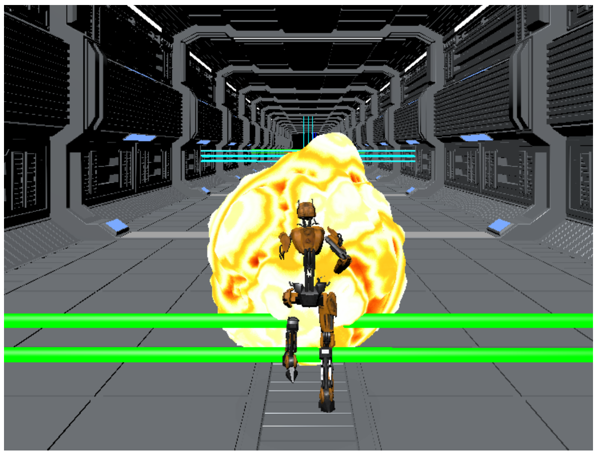

Figure 4: Game Over

purposes is very simple since only the interactions between the main character and the obstacles must be taken into account, for this reason I implemented them by simply relying on threeJS, but without using raycasting since it is slow and sometimes even not too effective.

Every time a new obstacle is created the respective box containing it is saved into an array, and while the robot advances it is computed the box containing the robot. A collision is detected whenever the box containing the robot and the one containing one of the obstacles collide.

This is a simple solution but it is also very effective and completely fulfill the necessity of a real collision detection system avoiding many computations.

4 Game Play

During the game the robot is continuously moving forward and also is the camera, which follows the robot at a fixed distance. To avoid the obstacles the player is able to use the robot animations, obviously the player must pick the right one at the right time to not explode.

The speed of the game increases every time the corridor is extended, while the points increase continuously with the progress of the robot.

4.1 Commands

There is a listener associated with the button arrows so that whenever one of the arrow is pressed the main character executes the related movement:

- up arrow: JUMP;
- down arrow: SLIDE;

- left and right arrows: LATERAL MOVEMENT;
- P to pause the game;

Whenever one action is started it cannot be changed until completion, so that if the robot is jumping is impossible to start a slide before touching the ground, also the lateral movement can be performed only while running and not while jumping or sliding, this is a personal choice since there many games of this kind allowing a lateral movement during a jump, but it seemed more realistic to me to block such possibility.

At the beginning of the game, and also in the pause menu, it is possible to see the list of commands available to the player, just to make it clear to him without make him guess which they are.

4.2 Audio

Like every game a good sound is important, as many games are played only for this aspect. For our game there are three different audio files:

- the soundtrack: played throughout the entire game;
- laser beam effect: it is played whenever the robot passes an obstacle;
- explosion effect: it's played when the player fails and makes the robot explode.

Since for some policies there are browser preventing the audio playback when there is no interaction with the web page, the audio is started only when the game starts and not in the initial menu. The soundtrack is played during the game in loop while the laser sound is played only when an obstacle is passed. The explosion will play only at the end of the run, when the robot explodes and the game ends showing the final score reached.

5 Conclusion

It is a funny game, simple to play, there are many possibilities to expand it in the near future, making it more complex, but now we can already see the power of webGL to build interactive games for the web and also the progress made by libraries such ThreeJS which make the development easier since they allow to focus more on the logic of the interaction than on the lower aspects of the bare webGL development. In conclusion it has been a pleasant experience.

References

[1] *https://sketchfab.com/3d-models/biped-robot-801d2a245e4a4405a0c2152b35b5e486.*

[2] *https://www.cgtrader.com/free-3d-models/interior/other/sci-fi-corridor-128d13da-7cf9-4cb0-a6e7-d2026b0817bd.*