INTERACTIVE GRAPHICS – FINAL PROJECT

FUEL RACE

FLAVIA MONTI, 1632488

SEPTEMBER 2020

Introduction

Fuel Race is an endless runner game realized as final project for Interactive Graphics course. The main player of the game is a Transformer and its goal is to collect much fuel tanks as possible avoiding obstacles on the road.

Environments, Libraries and Tools

The project is realized with HTML, JavaScript, ThreeJS, GLTFLoader, jQuery, Bootstrap.
ThreeJS use WebGL to create 3D graphics, it is simpler than WebGL and offers an easy way to build a complete 3D environment. GLTFLoader is a specific library built on ThreeJS that can be used to load .gltf and .glb files.
jQuery library helps in the DOM manipulation, it simplifies some JavaScript's mechanisms.
Bootstrap framework is used for stylesheet, different objects in the project use Bootstrap's style. Different Bootstrap's component requires JavaScript function contained in other libraries, so also Popper.js is imported to help in positioning button or tooltip.
The models are realized using Blender tool. All the models created are saved as .gltf files and then imported into the project using GLTFLoader.

Elements of the Scene

Many objects of different types and geometries are added to the scene. Function *main()* in /js/main.js calls many functions to initialize all the elements and objects.
initAll() add the renderer, the camera and the light to the scene (the variables of these objects are contained in /js/utils.js).
The camera is a perspective camera with field of view as 75, positioned at (0,0,100) looking at (0,0,0). The project contains only one light and it is an ambient light, colored of beige.
setSky() set the sky of the world game and a Mesh object composed by a PlaneGeometry with a texture is added to the scene. In this function is also set the fog property of the scene.
setGround() create the ground composed by the grass and the road. To create this, two Mesh objects are created, both objects are planes with textures. One object represents the grass and the other represents the road.
The result after these functions are called can be seen in Figure 1.

Figure 1 - Game's World

Models

As said before, the models are created using Blender and imported into the project with GLTFLoader. In /js/uitls.js it is initialized the loader variable and the function *loadModels()* in /js/main.js load all the models using that variable.

For models like houses, trees, obstacles and fuel's tanks an important aspect need to be explained. Loading new objects and adding them to the scene requires lot of calculation and this can be very bad in performances of the game. In the project all the previous models are loaded and created only one time and they are reused every time they go outside the camera.

In Figure 2 it is possible to see all the models loaded and added to the scene. This is the game's page.

Figure 2 - The game

Houses

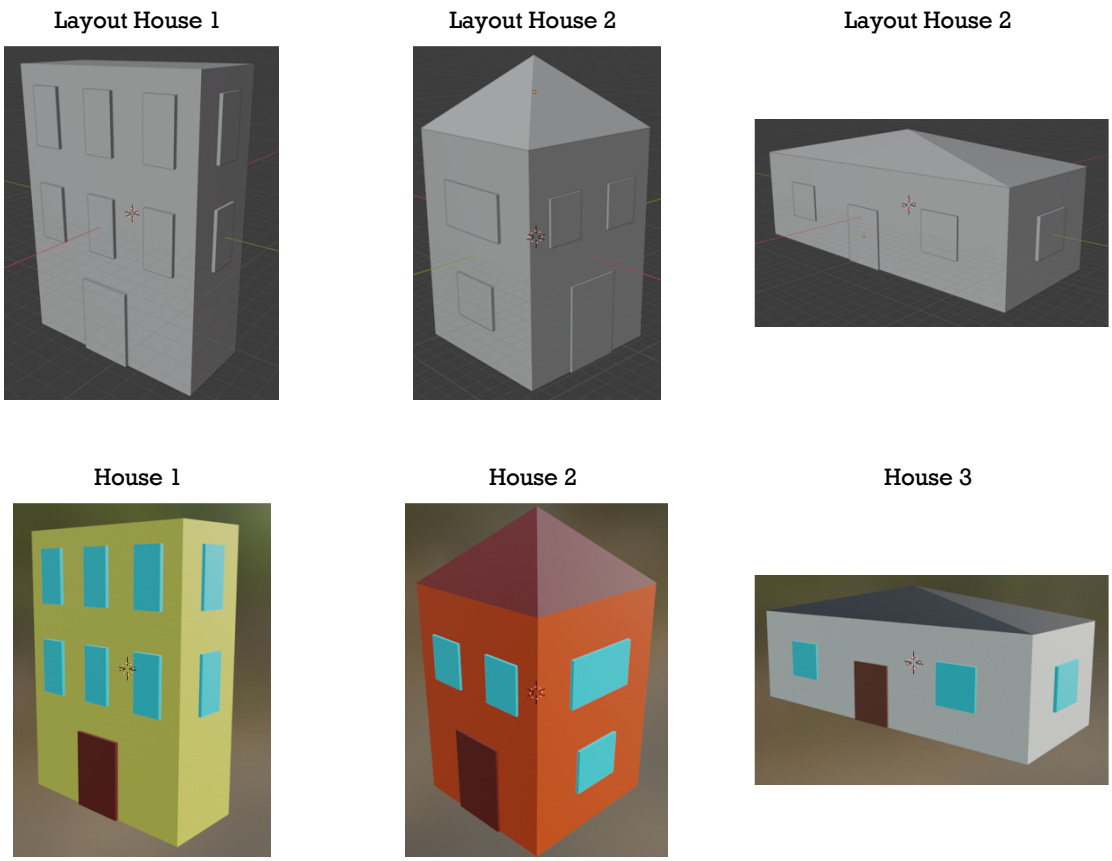
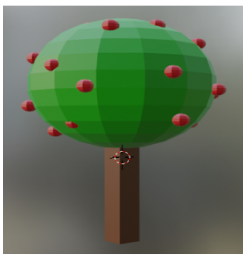


Figure 3 – Houses's models

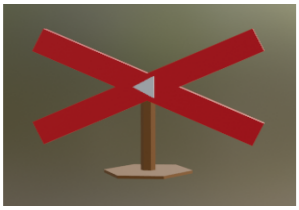
There are three different type of houses, in Figure 3 there are the images of them. Two houses per type are created, 3 on one side of the road and 3 on the other side. When one house reaches a position outside the camera view it is re-positioned back on the scene.

Trees

6 trees are created and repositioned during the execution of the game; their position is in between the houses. A tree is composed by a cube representing the trunk, a sphere representing the leaves and lot of smaller spheres representing apples. In Figure 4 it is possible to see the model of the tree.



Tree



Obstacle

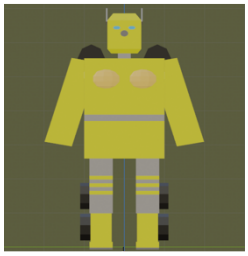
Fuel tank

Figure 4 - Tree's model, obstacle's model, fuel's model

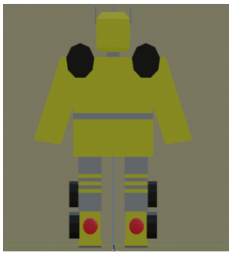
Obstacles and Fuel's tanks

The number of obstacles and fuel's tanks created during the game depends on the difficulty chosen by the user. These two objects are added to the scene depending on a delta clock, initialized in /js/main.js, then a random value decides whether to add to the scene a fuel's tank or an obstacle, and another random value decides where to put the object on the street. A function checks if an object is overlapped to another object and if this is the case move a bit the object.
The game is based on these objects that are positioned randomly, the parts that are constant and do not change during the game are total number of these two objects, delta clock and speed, that depends on the difficulty chosen.

Transformer



Front



Back

Side

Figure 5 - Transformer's model

The transformer is the hierarchical model and main character of the game. Its hierarchical structure is reported in Figure 6 and its model is reported in Figure 5.
It is loaded and added to the scene together with all the other objects. It is positioned on the center of the street and it's moving like walking and user can move it.

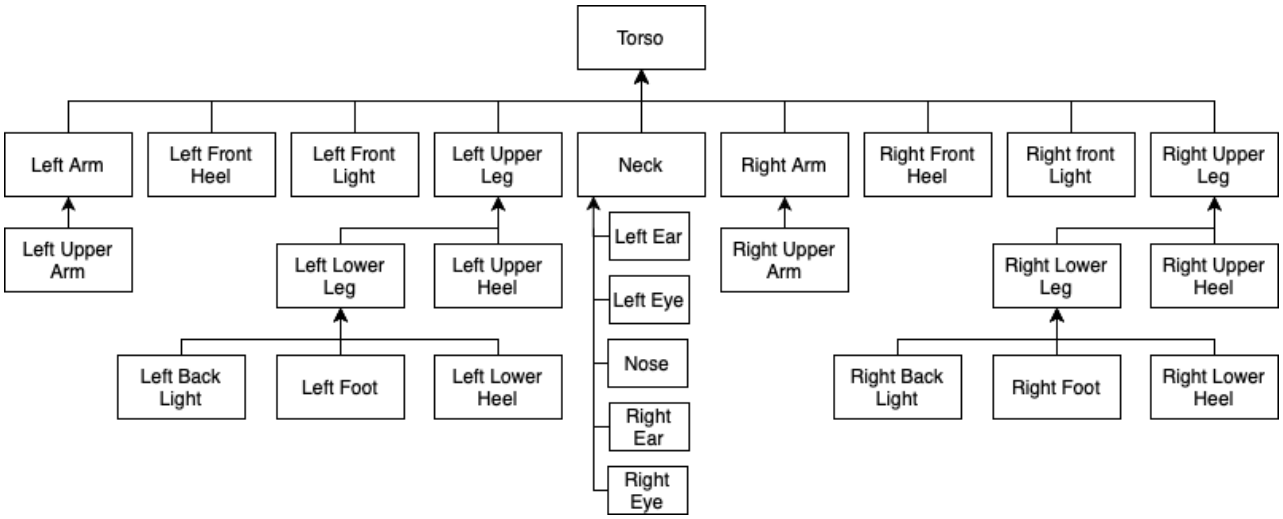


FIGURE 6 - HIERARCHICAL STRUCTURE OF TRANSFORMER

GAME AND INTERACTIONS

The project is composed by two html pages. First page contains instructions and a select menu to choose the difficulty of the game (easy, medium, hard). The second page is the real game page. This page contains some information about the game, like lifes of the Transformer and the score representing the number of fuel's tanks collected, and a button to exit the game.

As said before, depending on the difficulty chosen by the user there are different number of obstacles and fuel's tanks, also the speed is modified and delta time to let appear the objects. This is done by the function *setLevel()* in /js/main.js, where it searches the parameter speed in the URL.

User use left and right arrows to move the transformer on the street in order to collect fuel's tanks and to avoid obstacles. The street is divided in three lanes and Transformer moves in these lanes.

When Transformer hits three times an obstacle, loses all the lifes and a "Game Over" sign appears on the page together with an audio and a button to go back the game.

When Transformer hits a fuel's tank, the score is incremented and an audio is played.

Function *checkCollision()* checks if Transformer hit an obstacles or collect a fuel's tank. This control is done by checking the distance between the Transformer and the object. This function also updates the number of lifes and the score.

ANIMATIONS

Animations are done in the project and they are not imported from the outside.

Function *renderTransformer()* contains all the instructions to animate the Transformer. Transformer's model is traversed and depending on the part this is modified.

Transformer is walking, so its legs alternating bent and its foot lift one at a time. To simulate better the walking also its arms alternating move back and forth. All the heels of the Transfomer also rotate (those on its back and those on its legs).

When user interacts with the game, trunk of transformer is translated, in this way all the model change its position.

All the other objects move from the horizon through the transformer with a speed determined by the difficulty of the game.

Observations

If you want to download and play the game locally on your computer, you need to execute a local server with for example python and then open *index.html* page.
Not all the browser works well with the game. When using Chrome all works fine, when using Safari the Transformer blocks whenever hits an object and when using Firefox all works ok except for the audio that does not work.

Link to play the game

https://sapienzainteractivegraphicscourse.github.io/final-project-flavia-monti/

References

Blender documentation: https://docs.blender.org/manual/en/latest/

Bootstrap documentation: https://getbootstrap.com/docs/4.1/getting-started/introduction/

GLTFLoader library: https://threejs.org/docs/#examples/en/loaders/GLTFLoader

jQuery documentation: https://api.jquery.com

ThreeJS documentation: https://threejs.org/docs/