SAPIENZA
UNIVERSITÀ DI ROMA

Interactive Graphics
Final Project
2020 Spring

Dipartimento di Ingegneria informatica, automatica e gestionale

Nijat Mursali | 1919669

Yunus Emre Darici | 1900161

Alper Calisir | 1888955

Contents

Introduction

The fundamental idea of this project is to create a game where the player is able to run a car through a city and survive through collecting the rewards which is fuel in our case. The main character in our game is a car driving through a city that contains lots of elements such as buildings, cars, streetlights, traffic lights, parking lots, trash cans and other kinds of elements. All the elements inside the game are created by our team.

In this game, we have used several external libraries (such as three.js), tools such as audio effects which were free to use, but in any case, this project would not have plans for commercial use as we don't own the rights for those sound effects. In this report we will discuss the reasons and techniques we have used throughout the final project implementation and discuss the advantages/disadvantages of the methods chosen. This report is divided into 3 parts in which we start by explaining the development process and then we move to discuss the conclusions/results for our project.

Development

In this part of our report we will discuss the development process of our implementation. First, we will detail our environment then talk about technical aspects of our project and then last components that we added and their interaction with each other and environment will be discussed.

Technology Stack

To develop the application, we have used **three.js** as requested. We have just used this library and all other kinds of codes such as animations, sounds were written by us. The following list shows the list of libraries we have used in this project:

1. **three.js -** a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser.
2. **stats.js -** used to check the frames per second in the project

The structure of the project, the files and the directories contained in the main folder are structured in the following way:

HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. Our project is represented as a single page web application with *index.html* as the main entrance point and consists of two parts:

1. head - which includes data, like name of the web page, author and imports
2. body - which describes structural hierarchy of UI elements on the web page

We have used HTML for the front-end development where we needed to store our JavaScript libraries and our HTML elements for showing the elements in the page.

CSS

The *css* folder contains one *css file* where we have used CSS (which stands for Cascading Style Sheets) that was needed to implement to make the front-end development look good by defining the aspects of UI elements and fonts for our project.

JavaScript

The *js* folder is the fundamental component of our project, where we have putted all the code we needed to write, in order to implement our project. Other JavaScript files that we have used for this project were *three.min.js, OrbitControls.js* and *stats.js.* As we have already got familiarized by the previous homework's, *three.min.js* is used to create the scene, lights, models and show the loop of the game. The fundamental point in our project is probably the JavaScript file where we wrote all our codes for our project. Our JavaScript code divided into 6 files

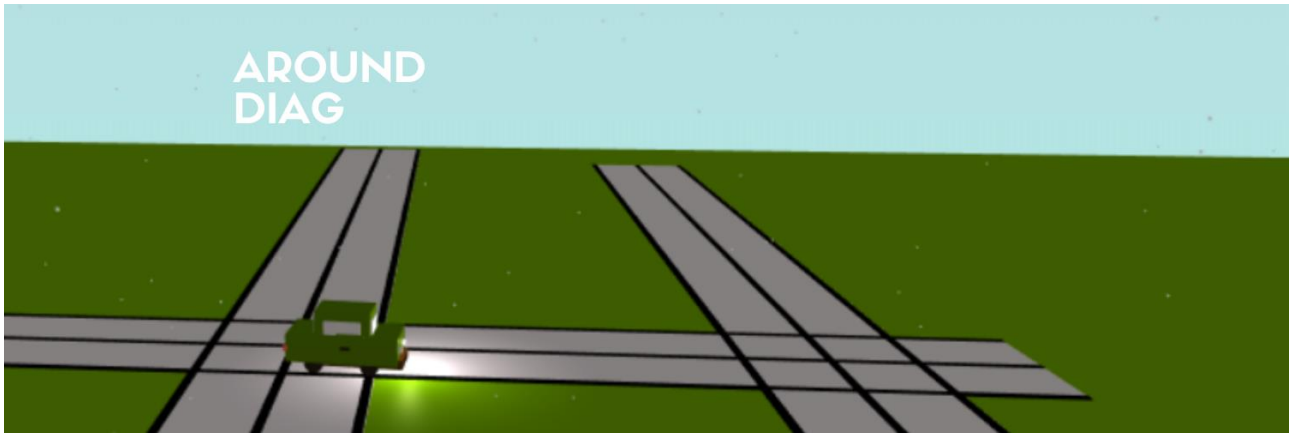
- **models.js** - where we created several models such as buildings, different types of cars, trash bins, street and traffic lights and other important objects for our game. We also added the placement for each object in this file as we needed to place objects in our scene
- **scene.js** - where we created the scene itself which was probably the most important step. In this step, we created all the important cases such as day/night cycle, rain effect, lights, sounds and other elements.
- **main.js** - where we dealt with the loop of the function by checking the car update, rain rendering and stats for the game. We also included the controls for the game in this file where it took the key-codes from keyboard and called the functions for specific cases
- **levels**.js - where we created/added the functions that contain the elements such as fuel, road and other kinds of objects we have created. In this file, we simply managed to add the logic to start and end the levels, resetting game and other elements for the game
- **loaders.js** - where we initialized several functions such as scene, lights, sounds, controls, levels and other kind of important functions
- **collisions.js** - where we simply implemented the collision detection where we simply added the collidable parts of the objects into array for each object and using the functions

Sounds

For this project, we have used several sound effects such as ambient sound, car horn, drive and start sound which we have got online. As we have mentioned in previous paragraphs, we will not use this project as commercial use, so we didn't need to care a lot for the copyright.

Components

Figure 1: Around DIAG Scene



Scene

Our scene contains several elements such as buildings, different types of cars, streetlights, trash cans, trees and other kinds of elements that were created by us. However, the scene without those elements is just a finite grass field and blue sky around as shown above.

Pole/Street Light

This was the first basic component that we have added when we try to configure our environment. It is just a basic pole which is used for streetlight purpose. We use *createPole()* function to create our poles. It takes x, z and rotation arguments and then sets the position of the pole using these x and z coordinates. Rotation is used to configure y coordinate of our pole. These poles are used for the purpose of streetlight.

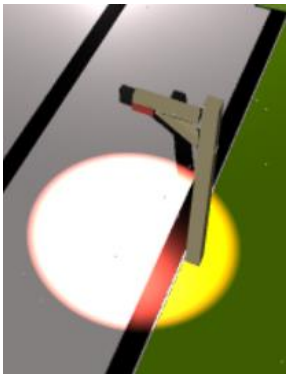


Figure 2: Street Light

Buildings

Buildings are one of the important elements in our game which is a *Mesh* element we have created in our project. We have also added collision into that using *Ray* that comes with *three.js.* There are 4 types of buildings: 2 of them are commercial buildings that are positioned northwest in our map. This part is a financial district of our world. The other one is a hospital building. There are 2 hospital buildings on our map but only one of them is used. The other hospital building is closed. Lastly, there are personal homes that are sparsely distributed in our map. The decision not to place so many buildings in our map comes from the fact that using a lot of buildings creates a lot of stack for our fragment shader which will slow our game if populated. We use *createBuildings()* function to create our buildings. Inside the function we call 4 different methods all assigned with the separate form of a building complex. They are *createBuilding(), createBuilding4(), createBuildingHospital()* and lastly *createHouse()* respectively. They all take x, z and rotation arguments and then sets the position of the building using these x and z coordinates. Rotation is used to configure y coordinate of our building or house. They are collidable so this means that you cannot cross them with the car.

Figure 4: Financial District

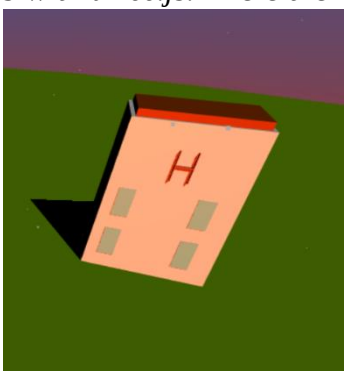


Figure 3: Hospital

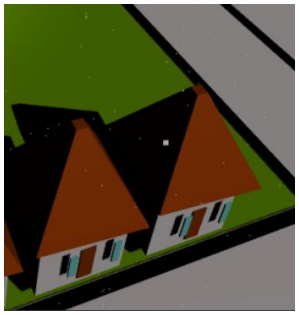


Figure 5:Suburbs

Traffic Lights

If we are building an environment where we drive car, it is a necessity to put traffic lights as well. In our environment we put traffic lights to some places. Though we can put traffic lights to every corner we choose not to implement that due to lack of computation power that fragment shader generates. When we create our level we create our traffic lights by calling *createTrafficLights()*. In this function we have designated some coordinates to erect so we use these specific coordinates to erect our traffic lights. We use createTrafficLight() function to create our traffic lights. It takes x, z coordinates and rotation arguments and then sets the position of the building using these x and z coordinates. Rotation is used to configure y coordinate of our traffic light.

Figure 6: Traffic Lights

Trash Bins

In our environment we also putted the trash bins close to the suburbs of our environment. We care about the environment. Hence, they are all different colours in the sense that they receive different kind of trashes.

Figure 7: Trash Bins

Trees

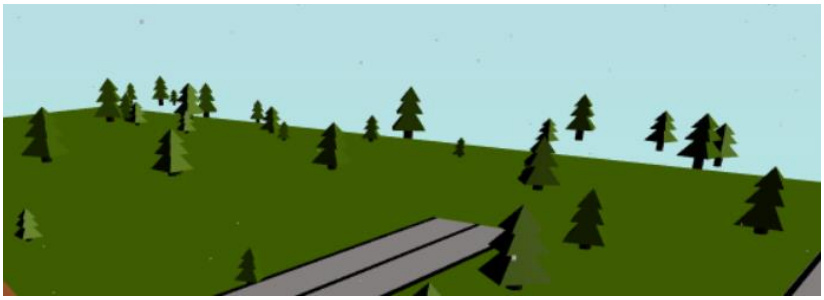


Figure 8: Forest in our environment

A world without trees is a world that is not worth to live and not possible one too. Thus, we have randomly putted trees in our environment. These are also obstacles(collidable by technical usage) if faced by player, player needs to diverge or else get stuck. We use *createTrees()* function to generate these random trees. Inside the function we for loop 150 trees and create them. While we create, we also created a condition where the program check if the tree is in between some objects. If so, the tree is not generated on that region. We randomly take *scale* to make our trees more realistic by not making all of them in the same size. Same goes for *rotate* as well. While we spawn our trees, we use delay to not erect all of them at the same time. This improves our efficiency and speed.

Parking Lot

Every crowded city needs a parking lot. Thus, we have putted a parking lot in our environment. This is created by calling *createParking()* function which receives *x, z, scalex, scaley, scalez* and *rotation* arguments. It sets the position of our parking Mesh with *(x,0,z)* and then scale it using the given arguments. Lastly y coordinate is rotated according to the given *rotate* argument.

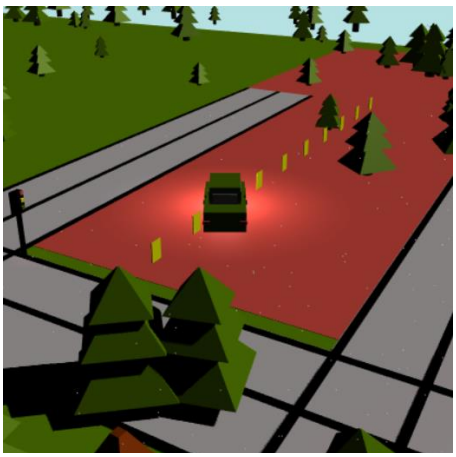


Figure 9: Parking Lot

Cars

Car objects are one of the important parts in our game and a must one. There are 4 types of car in our environment. They are the green personal car that we use to drive around, police cars, taxis and other personal cars. We create the models shown in the below figures by using *createCarVersion2(), createCarVersion3(), createPoliceCar(), createJeep().*

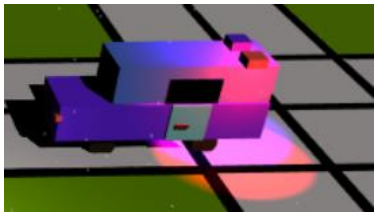


Figure 13: Police Car

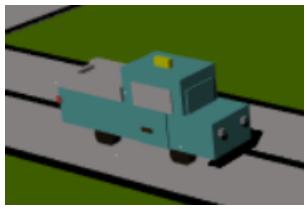


Figure 12: Taxi

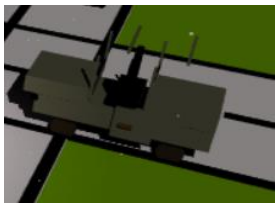


Figure 11: Jeep

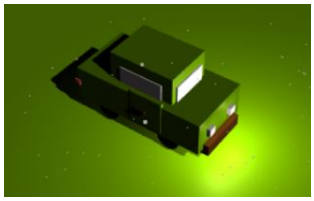


Figure 10: Our Car

Car Interaction

The main character in our game is the car that is driving around the city. The user has control of the car during the whole game: the interaction of the car is implemented by throwing a listener over the keyboard that lets the user execute several actions. In the following paragraphs, we have described them one by one.

1. Move forward:
   In order to implement this case, we have got the key-codes of the keyboard for both *Arrow Up* or *W* keys where it was needed to move the car forward with the help of the listener. For this case we defined an *if statement* which checks for those key-codes and if those key-codes are pressed the current speed is summed up with acceleration.
2. Move backward:
   In order to implement this case, we have used the same structure as in move forward where we got the key-codes of the keyboard for both *Arrow Down* or *S* keys where it was needed to move the car backwards with the help of the listener. For this case, we defined an *if statement* which checks for those key-codes and if those key-codes are pressed the current speed is subtracted with acceleration. We have also implemented the key-up case, so if the user releases the keyboard buttons it stops going backwards.
3. Move left:
   To implement this case, we have used almost the same step, but instead of going forwards or backwards we needed to move the car to the left. To do this, we have checked for the angle of the car, so if the user presses the *Arrow Left* or *A*, it checks for angle and assigns the rotation of the mesh into that angle.
4. Move right:
   The same thing applies for this case where it checks if the user presses *Arrow Right* or *D* buttons and checks if it's not left one, it goes for the right movement.
5. Car horn sound:
   In order to implement this step, we have again got the key-code for *H* button in the keyboard and using the *if statement* we have managed to create a new function which by using the *Three* it received the audio from the path.

```
function createCarHornSound() {
    var listener = new THREE.AudioListener();
    camera.add(listener);

    var sound = new THREE.Audio(listener);

    inGameSound = new THREE.AudioLoader();
    inGameSound.load('sounds/car-horn.wav', function (buffer) {
        sound.setBuffer(buffer);
        sound.setLoop(false);
        sound.setVolume(0.2);
        sound.play();
    });
}
```

6. Car engine start sound:
   The same implementation appears in this case as well where we got the key-codes for the *E* button on the keyboard and called the function we created for engine start.
7. Police Car engine light:
   It is possible to alter the signal lights of police car. By pressing *P* button, it is possible to open and close these lights. To implement this functionality, we have used *THREE.PointLight* in three.js library.

Road

Like the saying “All roads lead to Rome” every city requires a Road system. Thus, we have added a several roads and cross sections. User is not punished nor required to use the roads precisely to reach its objective but using it makes game more fun in a player perspective. We create the whole roads by calling *createRoads()* function in our level design. Inside of that function we create these single roads by calling *createRoad()* which takes *x, z, scalex, scaley, scalez*, rotation as an argument.

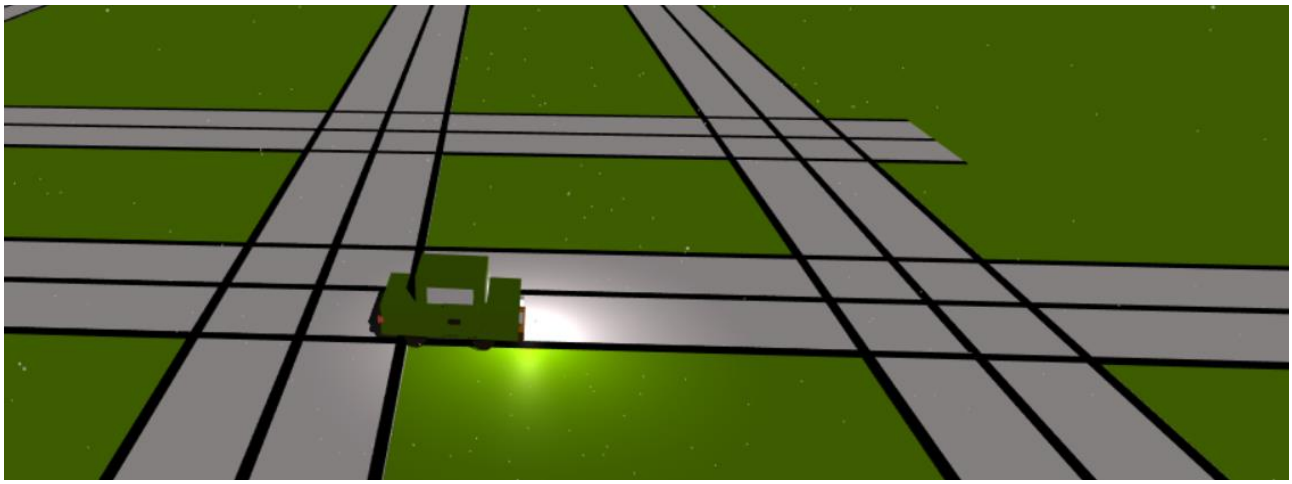


Figure 14: Roads

Fuel

Figure 15: Fuel

This is a crucial component in our game. Without this car cannot continue its survival. This should be generated throughout the map as the player try to reach it without colliding with the obstacles and within time. Player earns 1 point for each fuel and the goal is to maximize this points without being overrun by the time. We create fuel in our level design by calling *createFuels()*. It generates *x* and *y* coordinates by using *Math.random()*. Then by acquiring these x and y coordinates we send them by *createFuel()* which sets the position of the Fuel in our map.

Day/Night Cycle

To make our environment more realistic we have added a Day/Night cycle. This is achieved by using *DayNightCycle()* function when we create our scene. Inside that function we calculate the *sunAngle.* We decided our *dayDuration* to be 30 seconds which means a whole day takes 30 seconds to complete and the *sunAngle* changes accordingly. We created *sunSphere* using *THREEx.DayNight.SunSphere(),* sunlight using THREEx.DayNight.SunLight() and skydom using *THREEx.DayNight.Skydom()* and then add them to our scene.

Snow/Rain

This is a feature that makes our environment more realistic. Throughout the time cycles rain and snow starts. This is achieved by using *RainSnowCycle()* function when we create our scene. We first set our *rainCount* as 5000. We add randomly calculated position to our *rainDrop* and then push these ones to our *rainGeo*. We define the material of rain by using *THREE.PointsMaterial*. Lastly, we create rain by using both *rainGeo* and *rainMaterial* and then add it to our scene.

Code Parts

Fundamental Functions

1. createScene();

This function sets up the width and height of the screen to use them in order to set up the camera ratio and the size of the camera. Moreover we created it for adding fog effect ,setting the position of camera, creating renderer, filling entire screen, shadow rendering and also updating camera and renderer size.

2. createLights();

By createLights() function we created hemisphere light as a gradient colored light function which has 3 parameters as sky color ,ground color and intensity of light. Also shallow function is used in this function which depends on sun rays basically.

3. loop();

This function is for checking the car movement and collision. Rendering the scene and handling the growth of animation are put in loop in here ,in case of global collision it starts the loop again.

4. init();

This function is helpful for us to store all the important elements that we needed to initialize for the game. In this function we called the functions for scene, lights, sound and other important elements.

5. createGround();

This function simply creates a ground element that takes the box geometry with MeshPhongMaterial and applies a material into that. As our world is big enough, we made the ground element 5000x5000.

6. createCar();

This function creates the new car object which is the main character for our game and adds it into the scene. Car() function was made to combine all the elements of a car inside the mesh element and add the controls, speed, acceleration and other elements for moving the car around.

7. createLevel();

This function was made to handle all the elements inside the environment such as fuel, road, parking lots, buildings and other elements. *endLevel()* function is the opposite of *createLevel()* function which simply removes the elements after the level finishes. *createLevel()* function uses *startGrowth()* and endLevel() uses *startShrink()* functions.

8. createControls();

Basically we used this function in order to control horn ,light and movement of car by hand on the keyboard .As aim of project user will use the car by hand ,when he presses the “A” or “<--” then car move left or when player wants pause the game it will be possible by clicking “ESC” .

9. resetGame();

This function was made to handle the resetting of several elements such as car, timer and fuel when the time or fuel finishes in each level. car.reset is made to reset the position of the car into (-300, 25, -150) when time or fuel finishes.

Other Functions

1. DayNightCycle();

This function was made to handle the day/night cycle where we simply took the several elements such as sun, light and skydom and updated them in order to make the day/night cycle effect. After every second the sun object is simply going from one direction to another and during this time the light intensity changes.

2. RainSnowCycle();

This function was made to illustrate the rain which is a simple geometry (or simply lots of simple geometries) that we needed to update in loop() function. What it simply does is that it gives a velocity to the rain drops and decreases the coordinate in y direction to make the effect of rain falling down from the sky.

3. createForestSound();

This function is made in order to handle the environment sound that we took from the path and add it to the loader for sound effect.

4. handleWindowResize();
5. createEngineStartSound();

Simply it is the sound which is being heard from the engine working. During the loop when you click “W” or front arrow ,It will start car engine so at the same time engine will give working sound.

6. createCarHornSound();

The same idea applies for this case as well where we created the function for horn sound that takes the sound effect from the loader and we call that function when the user presses H button.

7. createCarEngineSound();

When the car is going on the road ,this function run the sound on the road until collision.

8. carBackLightsOn(ifPressed);

Create Level Functions

1. createFuels();
2. createRoads();
3. createParkings();
4. createBuildings();
5. createPoles();
6. createTrafficLights();
7. createBins();
8. createCarV2();
9. createTrees();
10. startTimer();

Conclusion

This project was aimed to understand and use in a real environment the WebGL technologies to create a graphic that could be interactive. From our point of view, we have learned a lot on the technology creating a project that can be useful for something in the future. At the moment, the project is hosted in GitHub Pages which has the link of https://sapienzainteractivegraphicscourse.github.io/final-project-successors/. For the future development, we will be adding other kinds of elements into our scene and make levels to make the game more interesting.

References

1. three.js – JavaScript 3D library
   https://threejs.org/docs/index.html
2. JavaScript Performance Monitor
   https://github.com/mrdoob/stats.js/