Universita` La Sapienza

Interactive Graphics

Final project :

Car Racing Game

1**Author :**

Afrah Salwa

Matricola number :

1837991

Contents

1 Introduction :

This is a final project for interactive graphic course the project is a simulation for a simple car game with some obstacle to be avoided by the user namely the player .

2 development Environment :

Few tool has been used to create this car game , the main libraries were

1-ThreeJS: Three.js is a cross-browser JavaScript library and Application Programming Interface used to create and display animated 3D computer graphics in a web browser. Three.js uses WebGL

And the source code is hosted in a repository on GitHub

2- CannonJS : Cannon.js is an open source JavaScript 3D physics engine it includes the rigid body physics engine also simple collision detection, various body shapes, contacts, friction and constraints.

2.1 External tools :

Other external tools has been used for this project :

1- Autodesk Maya : is a 3D computer graphics application used in this project to model the car and the game environment , in addition to that create the objects with the texture and export the scent and the objects as .fbx file which is a format for 3D object All of these files are in the folder assets .(fig 1.0)

2- FBXLoader : is an external library used to load all the .fbx files the scenes , objects and the texture ,the authors and the source code are mentioned the in code

3- inflate.min.js and three.min.js :is library helper to load the .fbx and the second library from threeJS but the only different that it has a minimum functions .

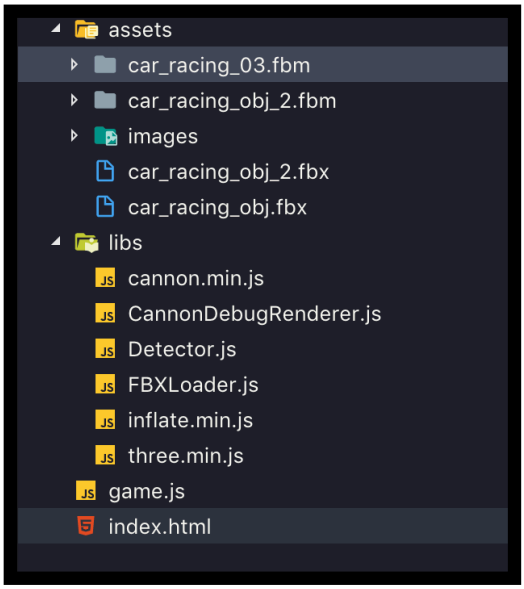

Fig 1.0

3 Technical Aspects :

3.1. Game structure :

At first I sketched the car game as concept starting with the Car and game environment such as the obstacle to be avoided and for this I used Autodesk Maya with the help of some tutorials on how to create the car and environment with collider as seen all this white objects are colliders it defines the shape of an object for the purposes of physical collisions. A collider, which is invisible, need not be the exact same shape as the object's mesh Fig 2.0 and then exporting the scene as .fbx file for later use in the game.js file to load the scene of the game

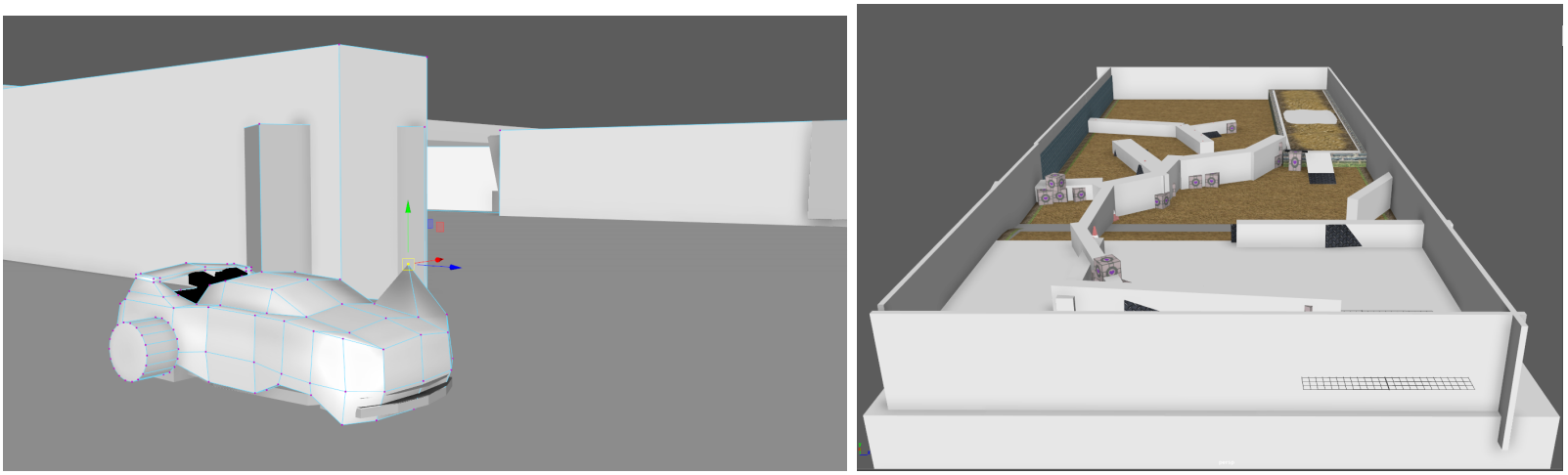

Fig 2.0

4 Implementation :

In the second step I download the threeJS and cannon.min.js libraries form Github and then In the game.js file which contains the main code for the game for to operate. First created class Game which contains a declaration for all variables for threeJS instances needed to display the game scene in the construct :

```
this.controls; //camera controls
        this.camera; //threeJs camera
        this.scene;  //threeJs scene
        this.renderer; //threeJs render
        this.fixedTimeStep = 1.0/60.0;
        this.assetsPath = "assets/"; //path to fbx file and textures
```

The second function inside the classes startGame that initiate the game when the user clicks the button .

```
 startGame(){
        document.getElementById('play-btn').style.display = 'none';
        this.initThreejs();
        this.animate();
    }
```

In the startGame() function there are two other function which need to initialize:

1- initThreejs : Inside this function I created a perspective camera and species the position in x,y,z .

```
this.camera = new THREE.PerspectiveCamera( 45, window.innerWidth / window.innerHeight, 1, 500 );
        this.camera.position.set( 0, 6, -15 );
```

Also initiated the scene and the **lights** , I chose the light type to be ambient which globally illuminates all objects in the scene equally, almost similar to sun light.

```
const ambient = new THREE.AmbientLight(0xaaaaaa);
    this.scene.add(ambient);
const light = new THREE.DirectionalLight(0xaaaaaa);
    light.position.set(30, 100, 40); //position in x, y, z space
    light.castShadow = true;  this.sun = light; this.scene.add(light);
```

And finally the render to display the scene with properties defining the renders's behavior which are the ration , the canvas width and heigh given the ration

```
this.renderer
 this.renderer.setPixelRatio(window.devicePixelRatio);
```

After initiating the initThreejs function now we will load the asset using the FBXLoader which loads the car_racing_obj.fbx file. It contains all 3d objects: scene and the car's parts.

Here I started creating the **hierarchical models** using the traverse function which is like a callback It is basically the iterator through the loaded object. the traverse() function which will be called for every child of the object being traversed, for example If I call traverse() on scene. I will traverse through the complete scene graph.

```
object.traverse(function(child) {
          let receiveShadow = true;
         if (child.isMesh) {
            if (child.name.includes("SkyBox")) {
            } else if (child.name == "Carbody") {
              //create car's parts hierarchy
              game.car = {
                chassis: child,
                bonnet: [],
                engine: [],
                wheel: [],
                seat: [],
                selected: {}
              }; //game.car object
```

In addition to the above traverse function I also created the camera which follows the car position .object3D is a a base class for most object and provide set of properties and method to manipulate object in 3D space

```
game.followCam = new THREE.Object3D();
```

Also loaded the cubes which are the obstacles for the game environment with their **texture**

```
tloader.setPath("assets/images/");
        var textureCube = tloader.load([
          "px.jpg",
          "nx.jpg",
          "py.jpg",
          "ny.jpg",
          "pz.jpg",
          "nz.jpg"
        ]);
```

4.1 initeCannon: After the above steps here I initiate the CannonJS the game physic engine and added the objects to be interacted with in the game which are the car and the obstacles colliders I used this demo project https://schteppe.github.io/cannon.js/demos/raycastVehicle.html

I defined the cannon world which is the physics hub that manages objects and simulation in addition to that I also defined the interacting materials (ground and wheels) and what happens between both material when they interact which are some simple physics such as :

```
const wheelGroundContactMaterial = new CANNON.ContactMaterial(
      wheelMaterial,
      groundMaterial,
      {
        friction: 0.3,
        restitution: 0,.
        contactEquationStiffness: 1000
      } );
```

Also in the initCannon created the bodies and shapes of car and wheels. In file **car_racing_obj_2.fbx** there is only one wheel and we need to create 3 wheels more. And set them in proper position, then rig the wheels then update the wheels. Which will explained in the animations section.
At the end of initCannon I used the createColliders() which defines it as the shapes of the objects for the purposes of physical collision , the collider which is invisible is the same shape shape as the object's mesh was created in Autodesk Maya as mentioned in the game structure.

3.3 animation :

Updating the wheels happens In the end of each step (60steps per seconds) initialized in the construct function at the beginning

```
this.fixedTimeStep = 1.0/60.0;/
```

And then the new position of the wheel are copied to wheelBodies[index].threemesh and quaternions

```
world.addEventListener("postStep", function() {
      let index = 0;
      game.vehicle.wheelInfos.forEach(function(wheel) {
        game.vehicle.updateWheelTransform(index);.
        const t = wheel.worldTransform;
        wheelBodies[index].threemesh.position.copy(t.position);
wheelBodies[index].threemesh.quaternion.copy(t.quaternion);
index++;})
```

Threemesh: which sets new positions for the wheels

Quaternions : A Quaternion describes a rotation in 3D space. It is mathematically defined as Q = xi + yj + z*k + w, where (i,j,k) are basis vectors. (x,y,z) can be seen as a vector related to the axis of rotation.

But first we must call the requestAnimationFrame which is a method tells the browser that we want to perform an animation and requests that the browser call a specified function to update an animation before the next repaint. Also specify the time between the calls .

```
const now = Date.now();
    if (this.lastTime === undefined) this.lastTime = now;
    const dt = (Date.now() - this.lastTime) / 1000.0;
```

```
   this.lastTime = now
```

after all initialization we call animate(). In this method we update the objects with every cycle of simulation.

```
if (this.world!==undefined){
 this.world.step(this.fixedTimeStep, dt, 10);
this.world.bodies.forEach( function(body){
 if ( body.threemesh != undefined){
 body.threemesh.position.copy(body.position);
 body.threemesh.quaternion.copy(body.quaternion)})}
```

simulation is basically a Cannon.js uses a computational algorithm called an integrator. Integrators simulate the physics equations at discrete points of time.This goes along with the traditional game loop where we essentially have a flip book of movement on the screen. So we need to pick a time step for Cannon.js. Generally physics engines for games like a time step at least as fast as 60Hz or 1/60 seconds
and then updateCamera(). The camera follows the car, and gets car position with specific height defined and finally render scene.

```
 this.renderer.render( this.scene, this.camera );
```

4 User interaction :

The user can move the car using the keyboard up ,down , left and right keys , all the event listeners for the user interaction were added in the initCannon() function and all the event listeners are handles with method in racing(event) code part .which contains at first the wheel force and the brake force and then initiated the brake force to zero so it moves after the braking and then used the switch statements to with the code between the keys the user uses, here is a snippet from the code :

```
switch (event.keyCode) {
      case 38:
        this.vehicle.applyEngineForce(up ? 0 : -wheelForce, 0);
        this.vehicle.applyEngineForce(up ? 0 : -wheelForce, 1);
        break;
      case 40: // backward
        this.vehicle.applyEngineForce(up ? 0 : wheelForce, 2);
        this.vehicle.applyEngineForce(up ? 0 : wheelForce, 3);
```

5 References :

threeJS documentation

CannonJS documentation

Udemy threeJS courses

Colan of Caerleon youtube channel