# GalacticRace

Ivan Nocenti
1309557

September 21, 2019

## Contents

# 1 Introduction

GalacticRace is a racing video game, the player will have to control a car and make a complete lap of the track in the shortest possible time. Leaving the track will result in defeat.
From the initial menu the player can start a new game (Start button), read the instruction of the game (Instructions button) and read credits (Credits button).
Once you click on the start button, the game will ask you to set the difficulty the player wants face the track. A greater difficulty will reduce the width of the track making it more difficult to keep the car on it.
A stopwatch will show the player how long it takes to make the race and it will be shown at the end of it.

## 1.1 Instructions

Now let's see the game instructions: The game is very simple and easy to play, just use the classic keyboard keys w, a, s, d to accelerate, turn left, brake and turn right, respectively. Mostly, the button w is held down and the machine will go faster, but its control becomes more difficult.

# 2 Implementation

## 2.1 Enviroment

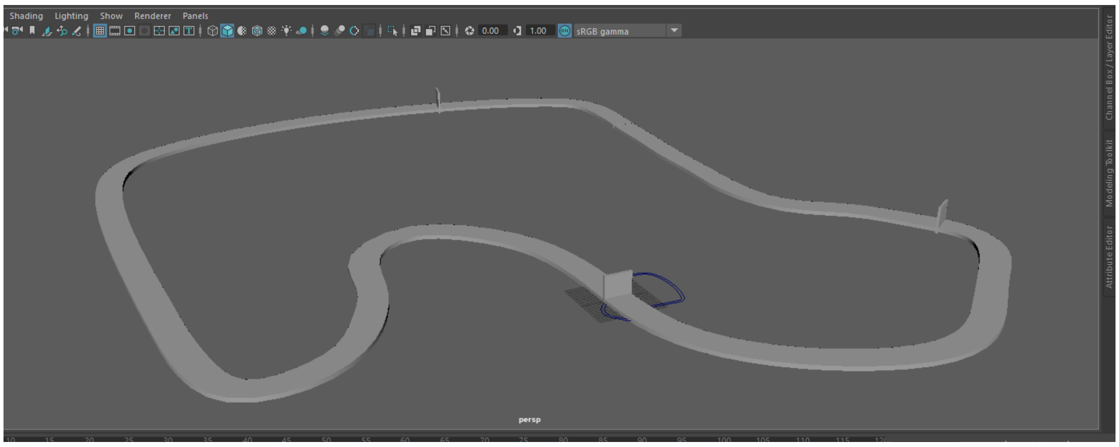
I have used Babylon.js that is a real time 3D engine using a JavaScript library for displaying 3D graphics in a web browser via HTML5. The source code is written in TypeScript and then compiled into a JavaScript version and it is available on github and distributed under the Apache License 2.0. The Babylon.js 3D engine and user code is natively interpreted by all the web browser supporting the HTML5 standard and WebGL to undertake the 3D rendering.

## 2.2 Library

- babylon.max.js
- babylonjs.loaders.min.js
- babylon.gui.min.js
- numeral.min.js
- ammo.js

## 3 Scenary

The scenario consists of a track created using the Maya 2018 software following a tutorial on youtube: `https://www.youtube.com/watch?v=p_xLMc67ABI`.



Once the track was created it was exported via the MayaToBabylon plugin. While the machine's mesh was taken from the following link: `https://www.turbosquid.com/3d-models/racing-car-sport-max-free/1136183`

Through the babylon functions I also created a skybox where a texture was applied to simulate a galactic environment.

```
    skybox = BABYLON.MeshBuilder
      .CreateBox("skyBox", { size: 1000.0 }, scene);
```

Listing 1: SkyBox

## 4 Light and Texture

### 4.1 Light

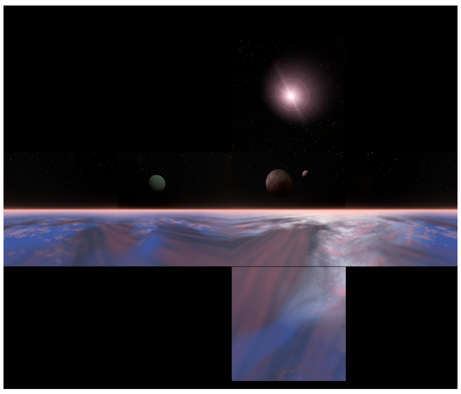
To simulate a light coming from a distant star, I used a hemisferic light in such a way as to completely illuminate the whole play environment.

```
    var light = new BABYLON.HemisphericLight("light1"
      , new BABYLON.Vector3(0, 1, 0), scene);
    light.intensity = 0.7;
```

Listing 2: HemisphericLight

### 4.2 Texture

I applied a texture to the skybox to simulate a space environment, the texture contains a huge planet and several stars around it.



```
    skyboxMaterial.backFaceCulling = false;
    skyboxMaterial.reflectionTexture =
       new BABYLON.CubeTexture("./texture/" + texture + "/5dim",
           scene);
    skyboxMaterial.reflectionTexture.coordinatesMode = BABYLON.
       Texture.SKYBOX_MODE;
    skyboxMaterial.diffuseColor = new BABYLON.Color3(0, 0, 0);
    skyboxMaterial.specularColor = new BABYLON.Color3(0, 0, 0);
    skybox.material = skyboxMaterial;
```

Listing 3: SkyBox Texture

## 5 Physics

In this project I have used the ammo.js library for enable the physics in babylon. Ammo.js is a direct port of the Bullet physics engine to JavaScript, using Emscripten. The source code is translated directly to JavaScript, without human rewriting, so functionality should be identical to the original Bullet.
With this code the physics library of ammo.js is enable:

```
scene.enablePhysics(new BABYLON.Vector3(0, -10, 0), new BABYLON.
   AmmoJSPlugin());
```

Listing 4: Ammo.js Plugin

Physics was used to move the wheels of the machine and to simulate its mass. It has also been used to make the track "solid" so that the machine does not fall into a vacuum.

```
        var physicsWorld = scene.getPhysicsEngine().
           getPhysicsPlugin().world;
        let mesh = task.loadedMeshes[0];
        track = mesh;
        mesh.physicsImpostor = new BABYLON.PhysicsImpostor(mesh,
           BABYLON.PhysicsImpostor.MeshImpostor , { mass: 0,
           restitution: 0.9 }, scene);

        for (var i = 1; i < 4; i++) {
            checkpointMesh.push(task.loadedMeshes[i]);
            task.loadedMeshes[i].visibility = 0;
        }
```

Listing 5: Ammo.js Plugin

I used three invisible cubes arranged along the route of the track to check if the player was actually performing a complete lap or tried to cheat. Each time the player crosses a "checkpoit" the next one is enabled and if he goes back, the one he has already passed is disabled. In this way I was able to control the normal crossing of the track.

```
for (var i = 0; i < 3; i++) {
```

```
            if (checkpointMesh[i].intersectsMesh(chassisMesh, false)) {
                switch (i) {
                    case 0:
                        if (checkpoint[1] == false) {
                            checkpoint[0] = true;
                        } else {
                            checkpoint[1] = false;
                        }
                        break;
                    case 1:
                        if (checkpoint[0] == true) {
                            checkpoint[1] = true;
                        } else {
                            checkpoint[1] = false;
                        }
                        break;
                    case 2:
                        if (checkpoint[1] == true) {
                            checkpoint[2] = true;
                            var now = new Date();
                            var time = numeral(now - date).format("0,0[.]00");
                            EndGame("You Win! Time: " + time);
                            engine.stopRenderLoop();
                        } else {
                            checkpoint[2] = false;
                        }
                }
            }
        }
```
Listing 6: Ammo.js Plugin