Interactive Graphics
Project – July 25th
Tania Sari Bonaventura – 1916415

1. Introduction

The following technical report presents the different steps of implementation of the project.

The report is organized as follows. Section 2 provides a description of the environment and tools used to develop the project; Section 3 describes the scene and the models implemented in the project; Section 4 and Section 5 describes the lights and textures applied in the scene; Section 6 describes the user interactions; and finally, section 7 describes the animations implemented in the scene.

2. Environment and tools

To develop the project, I mainly used the Three.js library directly uploaded from the web (https://cdn.skypack.dev/three@0.129.0).

In addition, I used the following plugins and libraries:

- Orbit controls (found in the '*controls*' folder)
- GLTF loader (found in the '*loaders*' folder)
- Tween (imported from the web)
- dat.gui (imported from the web)

Note: when working on local, to make the dat.gui work on Google Chrome, I had to disable the "Enable Javascript source map" and the "Enable CSS source map" in the Settings of the Console.

3. Scene

The idea of the project is a simple scene made by an open meadow where two robots are playing a football pass on a sunny day and dancing. Above them, some clouds are passing by.

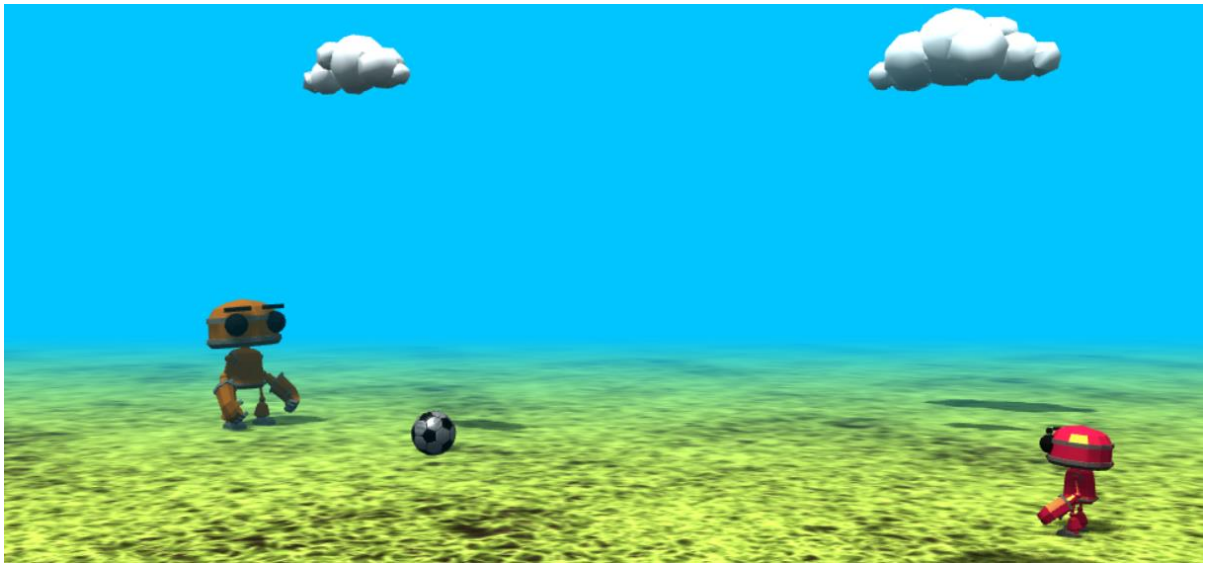
First of all, I created a Three.js scene and added a light blue background to it to represent the sky and some fog to make the impression that the meadow was infinite.

```
var scene = new THREE.Scene(); // root of the scene graph
scene.background = new THREE.Color( 0x00c5ff ); // light blue
scene.fog = new THREE.Fog( 0x00c5ff, 20, 100 ); // light blue
```

In the scene I then added several models that can be found in the '*models*' folder:

- Two robots
- A soccer ball

- Several clouds



The model of the robot is an example taken from the https://threejs.org/examples/ and in particular is the RobotExpressive one downloaded from https://github.com/mrdoob/three.js/tree/master/examples/models/gltf and loaded using the GLTF loader mentioned in Section 2. With the same model, a second smaller robot is placed in the scene, by scaling the model and by changing the color of the mesh.

On the other hand, the models of the soccer ball and the one of the clouds are downloaded from https://sketchfab.com/feed and are again loaded with the GLTF loader mentioned before.

The model of the soccer ball is scaled as to be of a reasonable size compared to the two robots.

Both the robots and the soccer ball are places in the field of grass, simply created with a THREE.PlaneGeometry of size 2000x2000.

As for the clouds, the model is loaded several times with a different scaling factor and two of them are rotated along the y axis, so they do not look all the same. The clouds are finally positioned in different places to give a realistic look.

4. Lights

Two lights are implemented in the scene:

- A hemisphere light
- A directional light

The hemisphere light is a light source positioned directly above the scene. The sky color of the hemisphere light is set to *0xffffff*, i.e., white, while the ground color is set to *0x444444*, i.e., a dark gray color.

On the other hand, the directional light is used to represent the sun and it is also set to *0xffffff*. The directional light is set above the scene, and it points toward the center of it. The light is enabled to cast shadows and the exact position of the light is defined in the following:

```
dirLight.position.set( 0, 20, 10 );
dirLight.castShadow = true; // enable light to cast a shadow
```

Since the light in enabled to cast shadows, I also enabled all the models in the scene to cast and receive a shadow, except for the mesh of the ground that only receive shadows since it would not be able to cast a shadow to other objects.

5. Textures

The textures are applied to the ground mesh using the THREE.TextureLoader and can be found in the '*grass*' folder. In particular, I used two images to define the material of the ground:

- A grass image
- A normal map texture

The grass image is used to define the base color of the ground, while the normal map is added to give it a more realistic look. Both the textures are repeated along the surface for 100 times to cover all the ground. To appreciate the effects given by the normal map texture, you need to play around with the directional light position.

Note: to use a texture saved in local, I used the python server.

As for the other objects in the scene, the textures were already applied in the imported models. I just played around with the color of the smaller robot by changing it to the one I liked the most. I therefore defined its material as a Phong material, defining the specular color so that it will change its color based on the light, giving it a metal effect.

6. User interaction

The user can interact with the scene in several ways. The first one is by changing the viewpoint. To do this, I first defined the camera as a **perspective camera** that is determined by four parameters:

1. **Field of view**: determines the angle of opening along the y axis
2. **Aspect ratio**: determines the ratio of the width and the height
3. **Near**: determines the closest plane of the viewing volume
4. **Far**: determines the furthest plane of the viewing volume

I set these 4 parameters and the camera position so that the objects are inside the view volume.

```
const fov = 45;
const aspect = window.innerWidth / window.innerHeight;
const near = 0.01;
const far = 1000;
camera = new THREE.PerspectiveCamera( fov, aspect, near, far );
camera.position.set(0, 3, 25);
camera.lookAt(0, 0, 0);
```

Then, to let the user interact with the camera by changing the viewpoint, I used the OrbitControls plugin. Therefore, from a computer, the user will be able to:

- Zoom in/out with the middle mouse or with spreading or squishing two fingers
- Allow the camera to orbit around the target with the left mouse

- Pan with the right mouse

Otherwise, if the user accessed the scene with a mobile device, he will be able to:

- Zoom in/out by spreading or squishing two fingers
- Allow the camera to orbit around the target touching with one finger
- Pan by touching and moving two fingers simultaneously

Another thing that the user can do is to change the color of the directional light and its intensity with the corresponding buttons and sliders created with the dat.gui library. Furthermore, with the corresponding sliders, the user can decide the position of the directional light and of its target, and it can observe how the shadows move accordingly, the effects on the normal map of the meadow, the effects on the shininess of the ball and of the small robot.

Finally, there are three buttons designed with CSS:

- "Start football pass"
- "Ball back to position"
- "Start dancing"

The user can press the red button "Start football pass" that will start the football pass animation of the two robots. If the user presses the red button but the ball is not in position, nothing will happen. On the other hand, the blue button "Ball back to position", will position the ball back to its initial coordinates so that the football animation can start again. If the blue one is pressed when the ball is already at position, nothing will happen. Finally, the green "Start dancing" will start the dancing animation of the two robots.

7. Animations

To implement the animations, I used the Tween.js library imported from the web (https://cdn.jsdelivr.net/npm/@tweenjs/tween.js@18.5.0/dist/tween.esm.js).

The simplest animations are the one of the clouds. They start moving as soon as the program is opened, and they are implemented with a simple tween. For example, for the first cloud the settings are the following:

```
var target = {x:30};
var tween = new TWEEN.Tween(cloud1.position).to(target,12000);
tween.repeat(Infinity)
tween.yoyo(true)
tween.start();
```

The duration of the animation and the x position of the clouds varies from cloud to cloud, but the basics are always the same: they all reach a certain position x, they go back to the initial position, and they repeat the same movements infinite times.

As for the animations of the robots, the football kick is activated when pressing the button "Start football pass" designed with CSS as mentioned in Section 6. The movements are achieved by working on the coordinates of the models and on the angles of the bones of the hierarchical structure of the robots' skeleton. The animation of the football pass can be seen as divided into 5 steps:

1. The big robot brings its arms close to the body, lowers its head, and runs while alternating the movements of the arms, the legs, and the feet.
2. After the robot reached the correct x coordinate, it kicks the ball stretching the right leg while bringing the right arm back.
3. The ball starts rolling and moving towards the small robot while the first robot returns to its initial rotation parameters of all its instances.
4. The small robot stops the ball with its right leg stretched.
5. After stopping the ball, the small robot rejoices stretching its arms up and jumping.

All the animation of the football pass mentioned above are implemented with tween.js. I had to play with the delays and the chain functions to make the animation smooth and coordinated.

I also applied an easing function to the ball, so that it would start changing quickly towards the value, but then slows down as it approaches the final value, as a ball would normally do.

If the user presses the “Ball back to position” button, the ball will go back to its initial position, and the football animation can be executed again by pressing the “Start football pass”.

Finally, there is the dancing animation of the robots activated with the “Start dancing” button and it follows the same mechanism of the previous animations; thus, it is achieved by working on the coordinates and on the angles of the bones of the robot. In this case, we can see the dancing as divided into 3 steps:

1. The robots come closer to each other
2. The robots start dancing
3. The robots go back to their initial position

Note: The code runs correctly on Google Chrome, I did not try it on Mozilla Firefox. It also runs correctly on a mobile device – better to orient it horizontally.