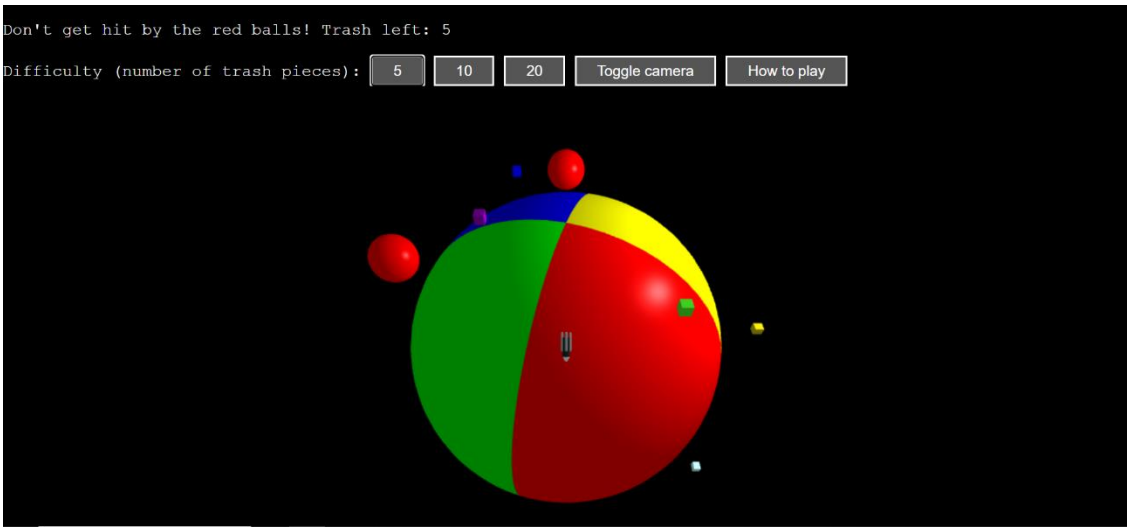
Final project

SpaceTrash

Interactive Graphics 2022/2023

Gonçalo Castelo Branco, Matricola 2069847



Introduction

My project consists of a simple single user game in which the user controls a rocket around a planet and aims to collect all of the “space trash” orbiting the planet as fast as possible, all while avoiding being hit by the giant red balls or the robot spider that are also orbiting the planet.

This project was developed using exclusively the Three.js JavaScript library.

User guide

When the page loads, the user finds a four-colored planet with a clean atmosphere and is invited to choose a difficulty level to start playing. In this game, the difficulty level corresponds to the number of trash pieces that will be scattered around the planet’s atmosphere to be collected by the player – in the current version the only options are 5, 10 or 20 pieces.

As the user, using the cursor, chooses the desired difficulty, the trash pieces will be generated, the planet will start spinning, the red asteroids will begin moving around the planet and the spider will start jumping around. In this moment, the user starts being able to control the movement of the rocket around the planet using the arrows, however the user must pay attention – the arrows increase and decrease the rocket’s spherical coordinates’ angles, which means that what is fixed is the rocket’s angular velocity, which makes moving sideways near the poles of the planet very hard. This way the user must plan well how they’ll get the trash near the poles in order to not get stuck and get hit by an asteroid.

Although the asteroids move in fixed orbits, the rotation of the planet makes it confusing for the user to keep track of them, which also helps in making the game more challenging and fun.

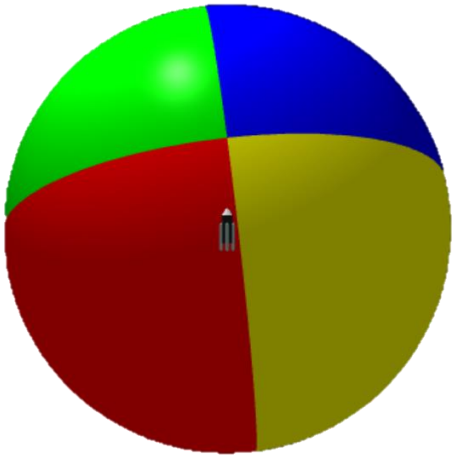
The button **Toggle camera** switches the user's point of view from outside, looking at the rocket and the planet, to a first-person perspective, where the camera follows the rocket around. This feature makes the game practically impossible to play, but it gives the user the satisfying experience of flying around the planet.

Also, the button **How to play** generates an alert informing the user of the basic rules of the game.

The model

We can divide the objects in our game in four categories:

- The planet
- The rocket
- Trash piece
- Asteroid
- The "spider"



The planet:

The planet consists of the four quarter-spheres "glued" together, to give it the four colored appearance it has. Each of these quarter-spheres is created using three.js's sphere geometry class and each of them is then rotated to become a whole sphere. As the game starts, the sphere starts rotating 0.01 radians per frame.

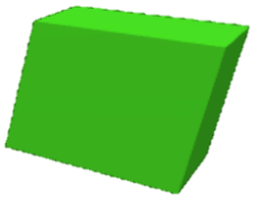
The rocket:

The rocket consists of 6 parts – a cylindrical body, an almost conical head and four capsule-shaped propulsors. Both the body and the head of the rocket are created using Three.js CylinderGeometry class, whereas the propulsors use the CapsuleGeometry class.

The rocket uses a hierarchical model where the propulsors and head are all attached to the body, which makes it easier to move and rotate the rocket as a whole.

When the game starts, the arrows control the increase and decrease of the rockets position. The velocity vector (the vector from the old position to the new position, every frame) is used then to adjust the direction to which the rocket's head is pointing.

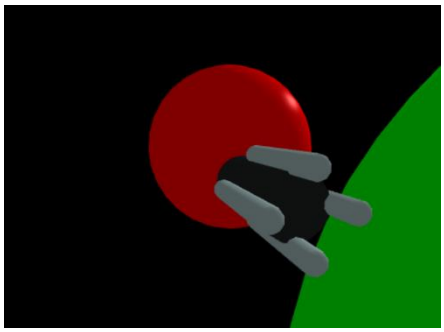
The trash pieces:



There can be 5, 10 or 20 trash pieces orbiting the planet, depending on what difficulty was chosen by the user. At its most complex, there are 9 cubes, 5 cones, 2 dodecahedrons, 3 tetrahedrons and 1 octahedron. These are all created and added to the scene every time the user chooses a level of difficulty and are all created using basic

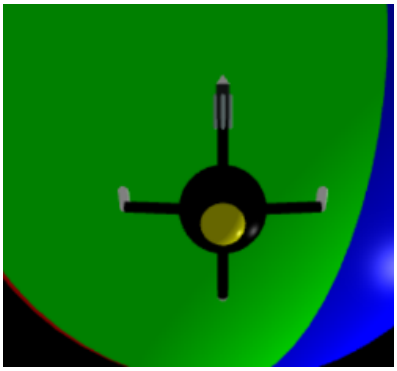
Three.js geometry classes. Every time they are created they are assigned a random position around the planet, making each game different from the last.

The asteroids:



There are two large red spheres that orbit the planet at a fixed velocity around the X and Y axes. These, too, were created using the SphereGeometry class from Three.js. Their velocity is fixed at 0.03 radians per frame and they always have the same latitude between themselves.

The robot "spider":



The spider is the most complex object in the project; it consists of a main body with a "decorative" yellow ball, connected to its four legs through 4 joints and each leg is composed by two parts connected to each other by a knee. The spider, similarly to the asteroids, walks around the planet and ends the game if the rocket gets too close. It has a fixed "bouncing" animation as it walks, which was made using simple keyframe animation. I registered the angles between the spider's body parts in 11 moments and interpolated the angles linearly to give it a smooth transition.

To make the game a bit more challenging, the spider's speed changes every few seconds, which is a factor the player must take into account.

Cameras

The scene can be see using to cameras: the main camera, that looks at the center of the planet and is aligned with the rocket's current position and the first-person point of view camera that follows the rocket around. Both cameras use perspective projection.

The main camera's position is updated with the rocket's position having the same angles in their spherical coordinates. Since the lookAt() function is used to keep the camera looking at the center of the planet, it is necessary to adjust the camera's "up" vector in some situations when the rocket crosses the poles in order to avoid the camera flipping upside down.

The "stalker" camera's position is updated using the rocket's position and the velocity vector, placing it always 3 units behind the rocket and always looking at it.

Both cameras adjust their aspect ratio if the window gets resized by the user.

Lights

The scene is illuminated by a basic weak light grey ambient light and a strong white point light that hits the scene sideways simulating a sun illuminating the planet.

Collision control

The radii of the trash pieces, the asteroids and the spider is calculated using Three.js's computeBoundingSphere() function and this value is stored in each object's userData property. At each frame iteration our program checks if our rocket has collided with either a trash piece,

an asteroid or the spider. This check is made using the objects' positions and their respective bounding spheres' radii.

If a collision is detected, in the case of a trash piece, the trash piece is removed and the trash piece counter is decremented; if the trash piece counter reaches zero, then the game ends with a win and the user is shown the time it took him to win. If, otherwise, the rocket collides with an asteroid, the game ends with a loss and the user is shown the time the game took and how many trash pieces he had left to catch.