Summary

1 Introduction

"Save The Earth" is a game develops in Threejs. The scene represents the space, with moving planets and stars. In addition, the scene consists of several objects: a robot, a spaceship, the meteorites and other objects that the user must obtain to protect the earth. The user is represented by a robot, it's a hierarchical model, it can moves in the scene and it have to take the resources.
The aim of the user is to save himself from the spaceship and save the earth from the meteorites. The meteorites come from the back of the scene with a differents speed, when these pass near the earth, the earth's life is decremented, so the user have to destroy it before it approaches. When the earth is struck many times, it explodes and the user has lost. In addition, there is a spaceship that shoots towards the robot; the robot has a life and every time the spaceship strikes the robot, it's life is decremented until it dies and explodes, so the user has lost. If the user destroys the spaceship, a timer starts, so if he can keep the earth alive until the timer ends, the user has win and he is redirected to the victory page, otherwise, he loses and he is redirected to the loss page.

2 Environment

The environment represents the space. It's composed by 8 planets, 50 stars scattered in the scene and a texture to give greater verity to the scene.

Figura 1: Environment

In particular way, both the stars and the planets are spheres. All stars share the same texture, while each planets has its own texture.
In addition, a texture fills the scene:

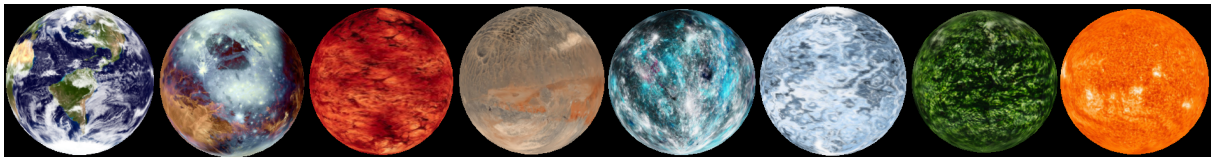
```
var scene = new THREE.Scene();
var background = THREE.ImageUtils.loadTexture('textures/cc.png')
scene.background = background;
```

2.1 Planets

See the code (PLANETS):

```
loader.load('textures/THE_APPROPRIATE_TEXTURE', function (
    texture ) {
  var geometry = new THREE.SphereGeometry( 1, 20, 20 );
  var material = new THREE.MeshBasicMaterial( {map: texture} );
  mesh = new THREE.Mesh( geometry, material );
  mesh.material.side = THREE.BackSide;
  mesh.position.z = -10.005;
  mesh.position.x = 14.505;
  mesh.position.y = -6.005;
  scene.add( mesh );
});
```

As we can see from the code each planet has its texture and its indipendent position in the scene.



(a) P1 (b) P2 (c) P3 (d) P4 (e) P5 (f) P6 (g) P7 (h) P8

2.2 Stars

See the code (STARS):

```
var i;
for(i=0;i<50; i++){
        loader.load('textures/THE_APPROPRIATE_TEXTURE', function
            ( texture ) {
        var geometry = new THREE.SphereGeometry( 0.02, 20, 20 );
        var material = new THREE.MeshBasicMaterial( {map:
           texture} );
        star = new THREE.Mesh( geometry, material );
        star.position.x = getRandomInt(-15,15);
        star.position.y = getRandomInt(-15,15);
        star.position.z = getRandomInt(-30,1);
        scene.add( star );
    });
}
```

As we can see from the code all stars has the same texture and its position is determined randomly in a defined range.

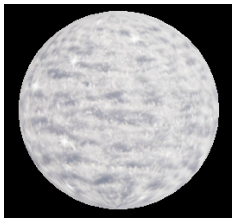


Figura 3: Star

All texture are simple JPG or PNG image downloaded from the web.

2.3 Light

The light in the scene is represented by the moon, so the moon rotating around the planets implies a circular movement of the light.
The moon is a simple sphere that revolves around the earth, in its orbit.

```
//MOON
var pivotPoint = new THREE.Object3D();
earth.add(pivotPoint);

moon = new THREE.Mesh( geometry, material );
moon.position.set(200, 4, 6);
scene.add( moon );
pivotPoint.add(moon);

var ambientLight = new THREE.AmbientLight( 0xcccccc, 0.4 );
scene.add( ambientLight );

var pointLight = new THREE.PointLight( 0xf0f0f0, 1 );
camera.add( pointLight );
scene.add( camera );

//In the render function
var vector = moon.getWorldPosition();
pointLight.position.set(vector.x,vector.y,vector.z);
pivotPoint.rotation.y = 0.4;
```

So the light moves to the instantaneous position of the rotating moon.

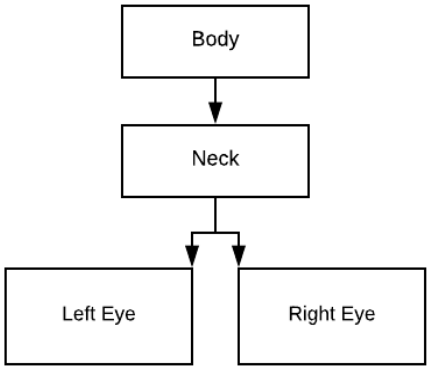
3 Robot

The robot is a hierarchical model composed by simple objects, in particular way, the robot has the **head**, the **body**, the **arms** and the **legs**, each of these contains other objects.

3.1 Head

The head of the robot is composed by a cylinder that represents the body of the head, two other cylinders for the eyes and a last cylinder for the neck. The principal function that creates the head is *createHead* (row 10 - robot.js).

```
var head  = this.createHead();
head.position.set(0, -0.6, 0);
this.model.add(head);
```



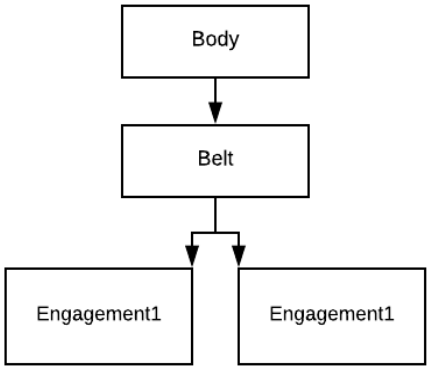
(a) Head diagram

(b) Head figure

3.2 Body

The body of robot is composed by a parallelepiped that represents the body of the robot, another parallelepiped represents the bust and two cylinder, the first on the bust, the second on the top of the body represents, the engagement, respectively, for the arms and the legs. The principal function that creates the body is *createBody* (row 6 - robot.js).

```
var body = this.createBody();
body.position.set(0, -80, 0);
this.model.add(body);
```



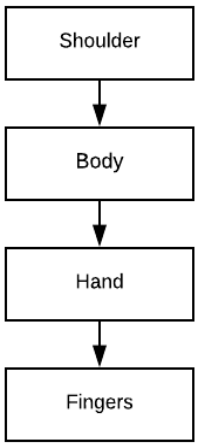
(a) Body diagram

(b) Body figure

3.3 Arms

The arms of the robot are composed by a cylinder that represents the shoulder, another cylinder represents the body of arm, while a set of parallelepipeds form the hand. The principal function that creates the arm is *createArm* (row 22 - robot.js).

```
var leftarm = this.createArm("leftarm");
leftarm.position.set(-1.25, 1, 0);
this.model.add(leftarm);
```



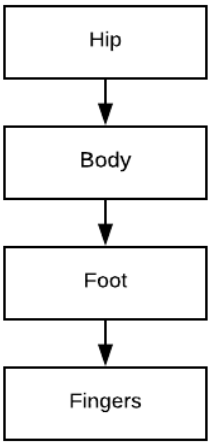
(a) Arm diagram

(b) Arm figure

3.4 Legs

The legs of the robot are composed by a cylinder that represents the hip, another cylinder represents the body of the leg, while a set of parallelepipeds form the foot. The principal function that creates the leg is *createLeg* (row 14 - robot.js).

```
var leftleg = this.createLeg("leftleg");
leftleg.position.set(-0.7, -54.5, 0);
this.model.add(leftleg);
```



(a) Leg diagram

(b) Leg figure

3.5 Complete

The robot has more parts and each of this has specific textures and colors (row 41 - robot.js). All animations, changes of appearance and movements are discussed in this section.

(a) 0 degrees

(b) 90 degrees

(c) 120 degrees

(d) 180 degrees

4 Meteorites

The meteorites are very important objects in the scene, which collide with the earth until it is destroyed. There are two type of model for the meteorities, downloaded from Free3d.

```
material_met = new THREE.MeshPhongMaterial({color: 0xC0C0C0 });
loader1 = new THREE.OBJLoader();
for(k=0;k<80; k++){
    loader1.load('model/Asteroid.obj', function ( object ) {
            object.traverse( function ( child ) {
```

```
            if ( child instanceof THREE.Mesh ) {
                child.material = material_met;
            }
        });
        obj1=object;
        obj1.position.z=getRandomInt(-1500,-1950);;
        obj1.rotation.y=getRandomInt(-60,60);
        obj1.position.x=getRandomInt(-1200,1200);
        obj1.position.y=getRandomInt(-800,800);
        obj1.scale.set( 0.8, 0.8, 0.8);
        arr.push(obj1);
        scene.add( obj1 );
  });
}
```

(a) Meteorites 1

(b) Meteorites 2

5 Spaceship

The spaceship is a model downloaded from Free3d.

```
texture_nav = new THREE.MeshPhongMaterial( {color: 0xC0C0C0 } );
loader6 = new THREE.OBJLoader();
loader6.load( 'model/SpaceShipDetailed.obj', function ( object )
    {
  object.traverse( function ( child ) {
    if ( child instanceof THREE.Mesh ) {
            child.material = texture_nav;
    }
  });

  navicella = object;
  navicella.position.z=-100;
  navicella.rotation.y=-7.9;
  navicella.scale.set(0.08,0.08,0.08);
  scene.add( navicella );
});
```

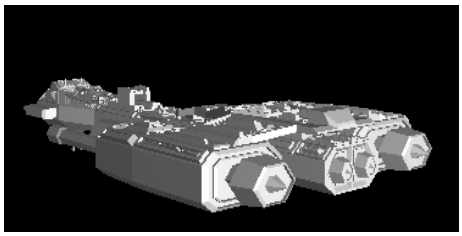
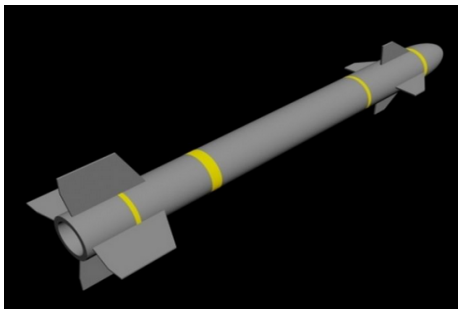


Figura 10: Spaceship

6 Resources

In the scene there are two different resources to save the earth: the cylinders and the rifle. Both are two models downloaded from Free3d.

```
loader4 = new THREE.OBJLoader();
loader4.load('model/OBJ_PATH', function ( object1 ) {
        obj = object1;
        obj.position.z =-22.04;
        obj.position.x = -24.505;
    obj.position.y = -0.505;
        obj.scale.set( 0.5, 0.5, 0.5);
    scene.add( obj );
});
```



(a) Missile

(b) cylinder

7 User interaction

The user, with the principal buttons and the mouse, can manage the movements of the robot and much more.

7.1 Change position of the robot and perspective of the scene

The user can change the position of the robot with different buttons of the keyboard:
A allows you to rotate the robot of 360 degrees.
C allows you to change the perspective (robot or scene perspective).
L allows you to move the robot forward in the scene.
M allows you to move the robot back into the scene.
P allows you to return to the default position regardless of the current position.
⇓ allows you to move the robot upward in the scene.
⇑ allows you to move the robot downward in the scene.
⇐ allows you to move the robot to the left in the scene.
⇒ allows you to move the robot to the right in the scene.

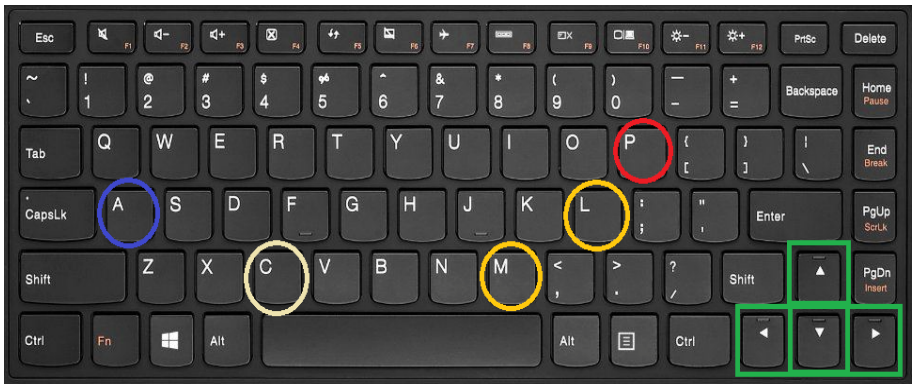


Figura 12: Keyboard command

7.2 Stop/Play animations

There are two buttons that allow to stop/play animations, in particular way, at the beginning all object are animated with rotations and translations, but the spaceship don't shoot and the meteorities don't arrive, so when the user click on "PLAY" the game starts.
Obviously, at any moment the user can stop the animations. In this case, this approach is used to better explain the game in the presentation.

Figura 13: Play / Stop command

7.3 Selection of resources

As mentioned in the introduction and how it will be discussed in detail in the next sections, the user can take two type of objects: the cylinders and the rifles. So, when the user gets these objects, a button is added at the top of the scene.

Figura 14: Select a resource

At the beginning, the robot has the hands (the default button is "H") , but when the user click on button "C", the cylinders are added to the robot, while, if the user click on "R", the rifles are added to the robot.
More clarifications are discussed in the next section.

7.4 Shoot

The robot has two rifles to destroy the meteorities and the spaceship, so when the button "R" is pressed, the user can shoot with the mouse click aiming at the lens. The default cursor when the robot is ready to shoot is the hand.

8 Animations

The most important part of the project are the animations, from the simplest to the most complex.

8.1 Planets

The most basic animation in the scene are the movements of the planets, indeed, everytime all planets are animated with a rotation on the y axis, but its remains in the same position.

```
mesh.rotation.y += 0.005;
...
mesh2.rotation.y += 0.005;
mesh3.rotation.y += 0.005;
```

8.2 Robot

8.2.1 Movement

The robot, as mentioned above, it can move in the space in all directions. The size of the space is limited for usability reasons, because beyond those distances the robot can't perform any action, so to facilitate the user, the robot's range of action is limited, limiting it to the usefulness of the game.
The user can shift the robot along the y axis with $\Uparrow$ and $\Downarrow$ buttons with a specific speed, in addition, it can shift in the x axis with $\Leftarrow$ and $\Rightarrow$ buttons, when the user press one of these button and shift the robot, it does a rotation in the y axis, when the user unpress the button the robot reset the rotation to return in the default rotation (0 degrees). When

the user press “L” or “M” buttons the robot shift along the z axis, with the same speed, when the robot shifts forward it has a rotation in the x axis to simulate the advance.

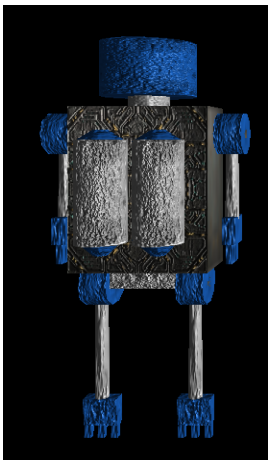
8.2.2 Take the resources

The robot has to take the cylinders to moves more fast in the space and it has to take the rifles to destroy the meteorites and the spaceship. So, when the robot gets close enough it takes the resource. To calculate the distance d between the robot and the resource, the formula of the distance between two points is used:

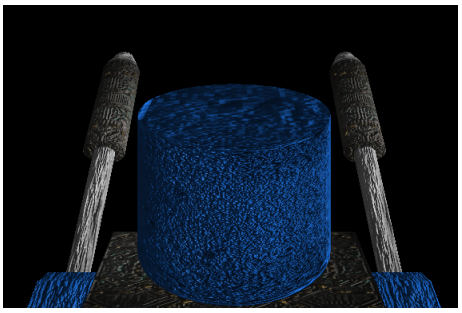
$$d = \sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2 + (Z_a - Z_b)^2} \tag{1}$$

```
var dist = Math.sqrt( Math.pow((bob.model.position.x-bomb.
    position.x), 2) + Math.pow((bob.model.position.y-bomb.
    position.y), 2) + Math.pow((bob.model.position.z-bomb.
    position.z), 2) );
if(dist<4){
    document.getElementById("bb2").style.display = "block";
    scene.remove(bomb);
    document.getElementById('jolly').play();
    get_bom=true;
}
```

So, when the robot gets the cylinders in the scene and the user click in the appropiate button, the cylinders are added to the robot (Figure a), if the robot gets the rifles and the user click in the appropiate button, two rifles are added to the robot, replacing the hands (Figure b).



(a) Cylinders



(b) Rifle

8.2.3 Shoot

When the robot takes the rifles, it can shoot with a simple click of the mouse in all direction of the scene. To simply the action of robot when the rifles are active, the rifles moves with the movement of the mouse.
As first thing, when the user click, the system gets the position of the rifles, when it has been taken, the shoot animations are added to the specific positions, one for rifle. The function *getWorldPosition* is very important, it allows to get the global position of the elements.

```
//GET POSITION AND ADD ANIMATION
var toRotate = bob.model;
toRotate = toRotate.getObjectByName("rightarm");
toRotate = toRotate.getObjectByName("bocs");
var vector = toRotate.getWorldPosition();
shot.position=vector.position;
scene.add(shot);

//ANIMATION
var shot_geometry = new THREE.SphereGeometry(1, 128, 128);
var material2 = new THREE.MeshLambertMaterial( { map: texture2 }
    );
shot = new THREE.Mesh(shot_geometry, material2);

var k = 3;
for (var i = 0; i < shot.geometry.vertices.length; i++) {
    var p = shot.geometry.vertices[i];
    p.normalize().multiplyScalar(1 + 6* noise.perlin3(p.x * k, p
        .y * k, p.z * k));
}

shot.geometry.verticesNeedUpdate = true;
shot.geometry.computeVertexNormals();
shot.scale.set(0.3,0.3,0.3);
shot.rotation.y=3;
```

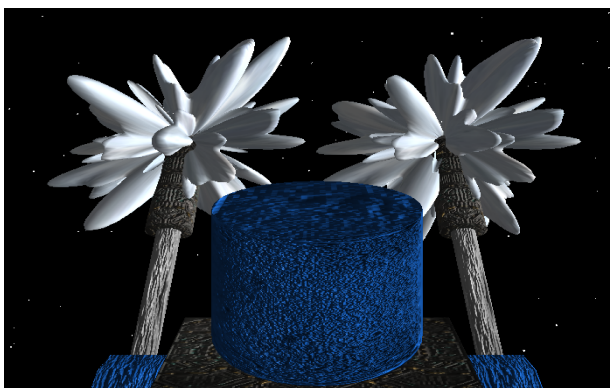


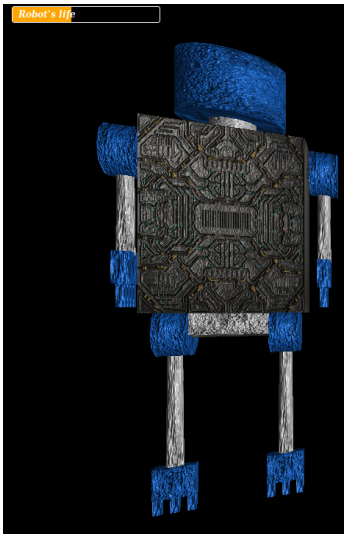
Figura 16: Shoot

8.2.4 Collision

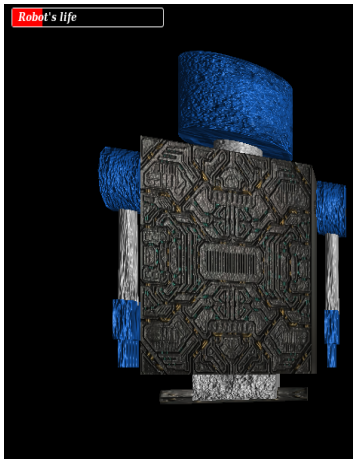
As mentioned above, the spaceship fires to the robot to destroy it, so when the spaceship hits the robot for the second time, the robot loses the legs that rotate away and loses the speed of the cylinders, at the third stroke the robot is destroyed.
The life of the robot is reduced with each collision. To calculate the distance d between the robot and the missile, the formula of the distance between two points is used:

$$d = \sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2 + (Z_a - Z_b)^2} \tag{2}$$

So when the missile goes from the spaceship, it save the position of the robot, so if the robot changes position, the missile don't affect it, otherwise yes.



(a) One shot



(b) Two shot

8.3 Meteorites

8.3.1 Movement

The types of meteorites, as mentioned in this section, come from the back of the scene with differents speed, each one turns on itself and moves towards the spectator. When the meteorite's position exceeds a certain position, it returns back into the space by resetting the animation so as to give a feeling of infinity.

```
if(arr[s].position.z<-40){
    if(s%2==0){
        arr[s].position.z+=0.91;
        arr[s].rotation.x+=0.031;
    }else{
        arr[s].position.z+=1.1;
        arr[s].rotation.x+=0.031;
    }
}
```

8.3.2 Collision

The meteorites continue their path or until they pass a certain position or until they collide with the earth. To intercept the collision the formula of the distance between two points is used:

$$d = \sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2 + (Z_a - Z_b)^2} \tag{3}$$

So when the meteorites collide with the earth, it is remove from the scene and it restart its animation from the back of the scene.

```
else{
      if(wins){
          scene.remove(arr[s]);
      }else{
          ind = crash.indexOf(s);
          if (ind >= 0) {
              crash.splice( ind, 1 );
          } arr[s].position.z = -900;
       }
}
```

At the same time, the life of the earth is decremented.

```
if(dist<200){
     if(!crash.includes(s)){
          crash.push(s);
          document.getElementById('error').play();
          ...
          life -=5;
    }
}
```

As mentioned above, the robot can shoot at the meteorites, so when the user pass on the meteorite with the mouse and click, this explodes. In particular way, when this is hit, it is removed and it replaced with another animation. This explosion is an randomic movement of many small elements to simulate the dismemberment of the meteorite.

```
function esplodi(x,y,z){
   parts.push(new ExplodeAnimation(x, y, z));
}
```

In this function, the coordinates of the collision point are passed, from which the animation must starts.

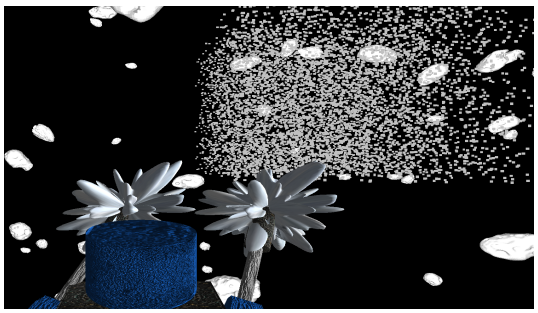


Figura 18: Explosion of meteorites

8.4 Spaceship

At the beginning, the spaseship moves in the x axis in a specific range.

```
if(navicella.position.x>=60){verso=false;}else if(navicella.
    position.x<=-60){verso=true;}else{}
     if(verso){
          navicella.position.x+=0.2;
          navicella.rotation.y-=0.002;
     }else{
          navicella.position.x-=0.2;
          navicella.rotation.y+=0.002;
     }
}
```

8.4.1 Shoot

When the button "Play" is pressed, the spaceship starts firing to the robot, so it fires a missile at the robot's position at the instant that the missile starts. The missile starts from the spaceship's position to the robot's position:

```
function sp(){
     a = {x: navicella.position.x, y: navicella.position.y, z:
         navicella.position.z},
     b = {x: bob.model.position.x, y: bob.model.position.y, z:
         bob.model.position.z};
     spar=true;
}

function lerp(a, b, t) {
     return a + (b - a) * t;
}

var newX = lerp(a.x, b.x, ease(t));
var newY = lerp(a.y, b.y, ease(t));
var newZ = lerp(a.z, b.z, ease(t));
meshsx.position.set(newX, newY, newZ);
```

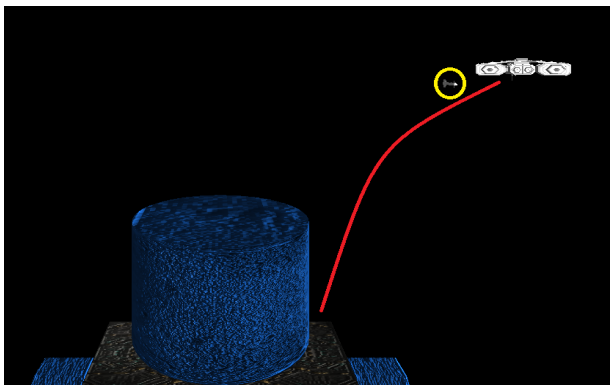


Figura 19: Shoot

8.4.2 Collision

As mentioned above, the robot can shoot at the spaceship moving the mouse in the appropriate position and clicking. This is possible with the raycaster library, see an example:

```
...
raycaster.setFromCamera( mouse, camera );
intersects1=raycaster.intersectObjects( navicella.children, true
    );
if ( intersects1.length > 0 ) {
    if ( INTERSECTED != intersects1[ 0 ].object ) {
        if(ex){
            navicella_colpita+=1;
            texture_nav.color.setHex(0xFF1F10);
            texture_nav.needsUpdate = true;
            ex=false;
            document.getElementById('bbs').innerHTML = '<b>'+
                navicella_colpita+'</b>';
        }
    }
}
```

So when the spaceship is hit, a variable is incremented and the spaceship changes color momentarily to help the user to understand when he hit it. When the spaceship is hit for 10 times, it explodes.

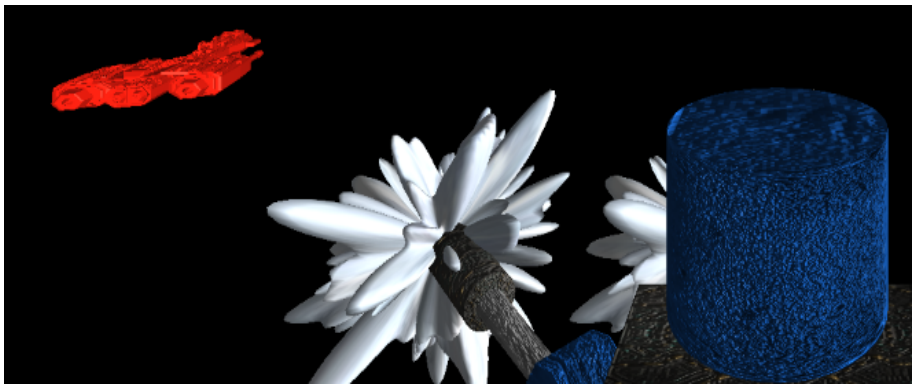


Figura 20: Shoot

8.5 Explosion

The explosion is the characteristic animation of the destruction of objects in the scene, in particular way, the earth, the robot and the spaceship.
For these animations is required the *perlin* library to generate noise in the vertices of the figure, in this case, a sphere.

```
var sphere_geometry_explo = new THREE.SphereGeometry(1, 128,
    128);
var material_explo = new THREE.MeshLambertMaterial( { map:
    texture_explo } );
sphere_explo = new THREE.Mesh(sphere_geometry_explo,
    material_explo);

var k = 3;
for (var i = 0; i < sphere_explo.geometry.vertices.length; i++)
    {
     var p = sphere_explo.geometry.vertices[i];
     p.normalize().multiplyScalar(1 + 0.3* noise.perlin3(p.x * k
        + start_explo, p.y * k, p.z * k));
}

sphere_explo.geometry.verticesNeedUpdate = true;
sphere_explo.geometry.computeVertexNormals();
```

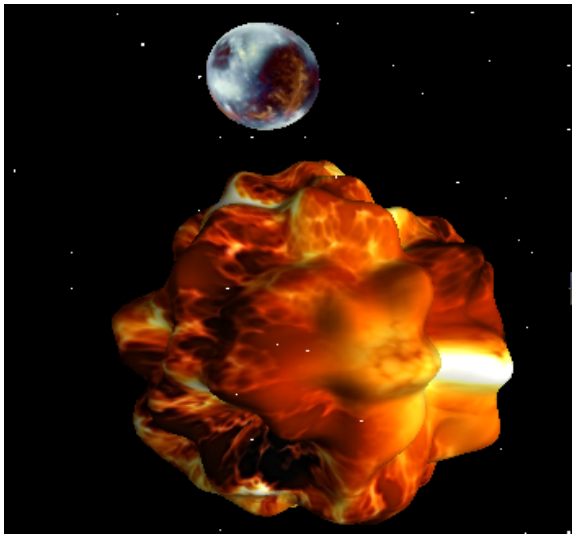


Figura 21: Explosion

After creating the sphere with 16258 vertices, to create a very dynamic animation, all the vertices of the sphere are disturbed. In partcular way, the vertices of the sphere are iterated and normalized with the normalize function normalize and multiScalar, the first convert the vector passed to a unit vector (with the same direction, but length 1) and the second multiplies this vector by scalar x, passed to the function. In the function a new function is called to produce noise by passing the vertex in question, at each iteration the variable *start_explo* is incremented to generate movement.

9 Loss / Victory

The user can win the game if, as first thing, he destroys the spaceship, immediately after the destruction starts a timer that lasts 160 seconds, if the user manages to keep the earth alive until the timer expires then he win, otherwise he lost.
In every case, the user is redirected in the specific page.

(a) Loss

(b) Victory

10 Conclusions

I realized the game entirely by myself, making the most of the knowledge acquired during the course. I implemented all the required objectives, trying to mix the technique with the imagination to entertain the user in every phase of the game. Obviously, I tried to find the right compromise between the nice animations and the computational capacity of my machine.

10.1 Libraries

I used different threejs libraries to realize the whole game: **Three.js**, **Texture.js**, **Raycaster.js**, **OBJLoader.js** and **Perlin.js**

10.2 Result

This is the complete game:

Figura 23: Save The Earth