Tecnológico de Monterrey

Light bikes - Gráficas Computacionales

Semestre Agosto-Diciembre 2019

Profesor: Octavio Navarro Hinojosa

Documento final

Rodrigo García Mayo	A01024595
Saúl Enrique Labra Cruz	A01020725
Manuel Guadarrama	A01020829

Objetivo

Crear una representación 3D del juego de Tron. El juego consiste en 2 jugadores en un mismo mapa viajando a la misma velocidad. Los jugadores estarán controlando motos de luz que generan un rayo de luz justo detrás de ellos mientras van avanzando. Esas barreras de luz tienen que ser evitadas por los jugadores para no ser destruidos. Las barreras de luz nunca desaparecen en ningún momento durante la duración de la ronda. Para ganar un juego, el jugador debe de ganar 3 rondas. El juego será persona contra persona.

Interacción: El jugador controla una moto moviendola únicamente a la izquierda o derecha.

Requisitos funcionales

El proyecto tiene las siguientes funcionalidades:

- Es jugador vs jugador
- El mapa tiene varios niveles de plataformas y los jugadores son capaces de moverse entre plataformas con el uso de rampas
- La luz emitida por las motos funciona de barrera para los demás jugadores, si un jugador colisiona con esta luz, perderán la ronda
- La luz se emite con la misma trayectoria que ha utilizado la moto
- El juego es controlado por las teclas AD (para jugador 1) y flecha izquierda y derecha (para jugador 2)
- La camara de cada jugador debe seguir la moto del jugador
- Para ganar un juego se tiene que ganar 3 rondas
- Las motos contienen animaciones de muerte que se activan cuando colisionan con una barrera o se salen del mapa
- El juego utiliza modelos 3D importados para las motos
- El juego contiene un menú
- Las motos se mueven automáticamente hacia delante
- Un jugador puede ser destruido con su propia línea de luz.
- Se puede pausar y reiniciar un juego

Requisitos no funcionales

- Tener js en la computadora
- Instalar las librerías de Three.js
- Tener un navegador web

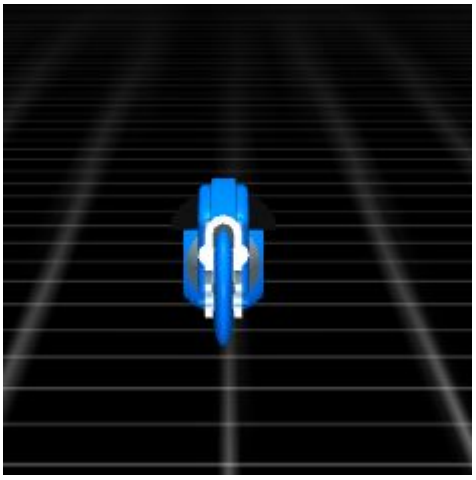
Temas utilizados

- Carga de modelos 3D
- Creación de clones de objetos
- Movimiento de objetos
- Animaciones

- Shaders
- Carga de música
- Uso de menús
- Puntuaciones
- Juego por rondas

Retos

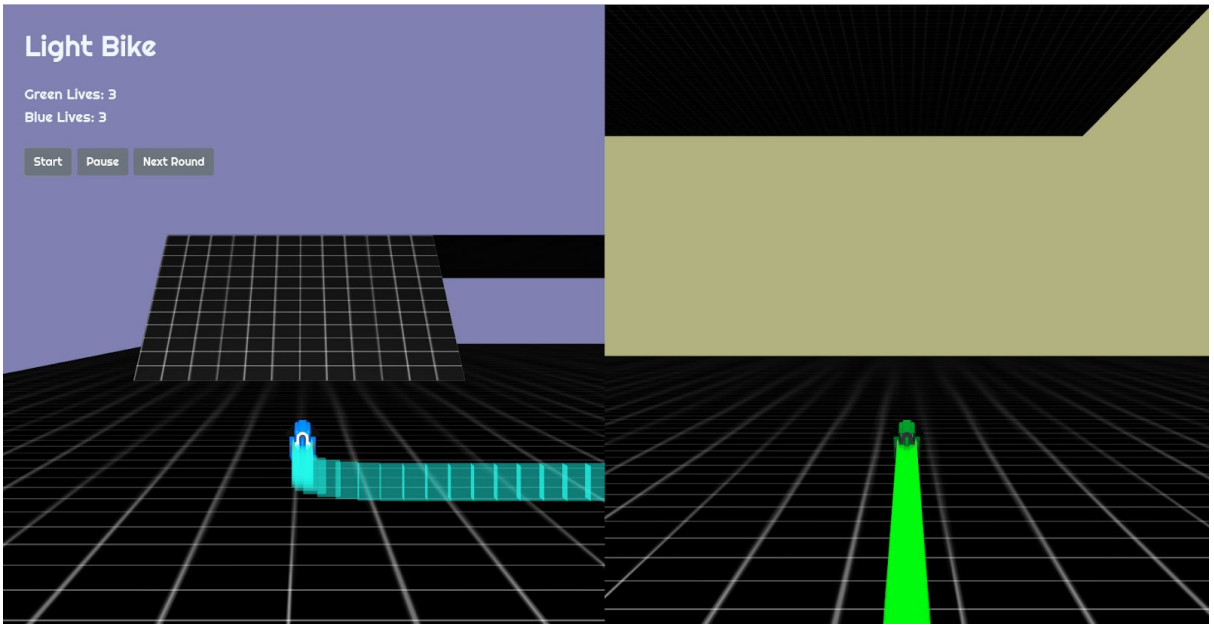
Obtención de modelos 3D: Se buscaron modelos 3D en internet gratuitos en formato .obj con texturas en .mtl para importarse al proyecto, usamos en conjunto dos loaders separados para cargar el mesh del objeto y después otro para cargar las texturas, todo esto se hizo dentro de una función asíncrona para evitar conflictos en el orden de ejecución de las instrucciones.



```
async function loadBlueBikeMTL() {
   var mtlLoader2 = new THREE.MTLLoader();

                                                    mtlLoader2.load(
'../models/classicTronBlue/classic-1982-tron-light-cycle-blue.mtl',
function( materials ) {
       materials.preload();

       var objLoader = new THREE.OBJLoader();
       objLoader.setMaterials( materials );


objLoader.load('../models/classicTronBlue/classic-1982-tron-light-cycle
-blue.obj', function ( object ) {
           tron_bike_blue = object;
           tron_bike_blue.add(camera1);
           tron_bike_blue.death = false;
           tron_bike_blue.deathAnim = false;
           tron_bike_blue.limits = true;
           tron_bike_blue.rampAnim = new KF.KeyFrameAnimator;
           tron_bike_blue.jumpAnim = new KF.KeyFrameAnimator;
           tron_bike_blue.up = 0;
```

```
            tron_bike_blue.color = 1;
            tron_bike_blue.initialPosition = 10;

            camera1.position.z += 10;
            tron_bike_blue.position.x = 10;
            tron_bike_blue.name = "tron_bike_blue";
            tron_bike_blue.rounds = 3;

            scene.add( tron_bike_blue );
        });
    });

}
```

El mapa tiene dos niveles: Se creó una función para generar los planos individuales y después juntarlos en un solo grupo de objetos y finalmente agregarlos a la escena. Al final se hicieron tres niveles para aumentar la complejidad del juego.

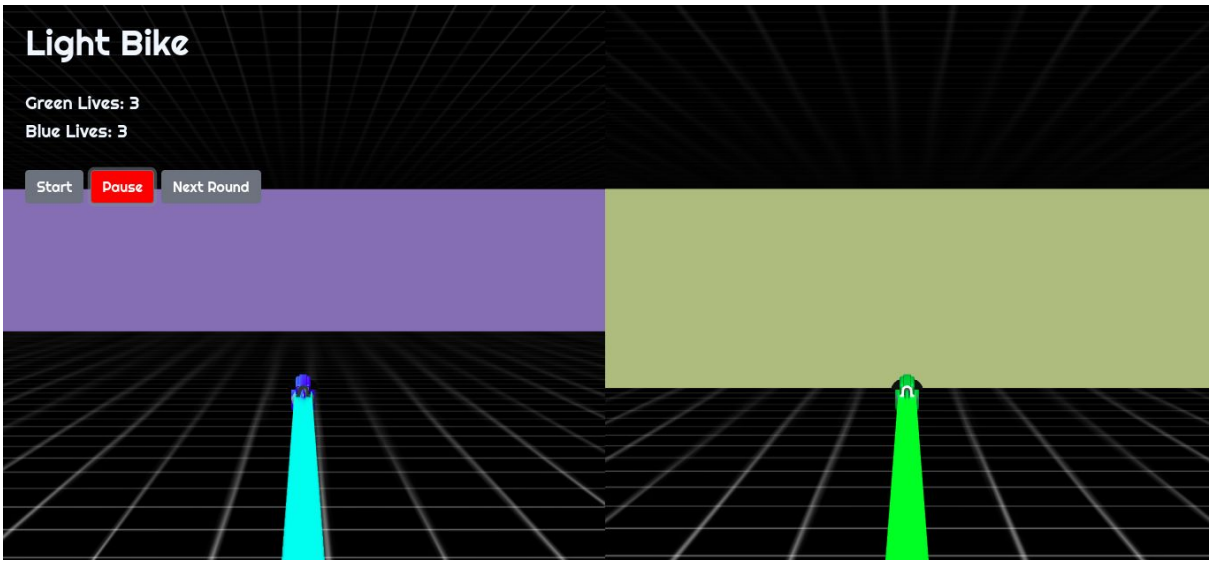



```
function createPlanes() {
    // Create a texture map for planes
    var plane_map = new THREE.TextureLoader().load(mapUrl);
    plane_map.wrapS = plane_map.wrapT = THREE.RepeatWrapping;
    plane_map.repeat.set(20, 20);

    // Create texture map for ramps
```

```
   var ramp_map = new THREE.TextureLoader().load(mapUrl);
   ramp_map.wrapS = ramp_map.wrapT = THREE.RepeatWrapping;
   ramp_map.repeat.set(2, 2);

   var color = 0xffffff;

   var planesGroup = new THREE.Object3D;

   // Level 1
   var plane_geometry_1 = new THREE.PlaneGeometry(200, 200, 100, 100);
     var     plane1     =     new     THREE.Mesh(plane_geometry_1,     new
THREE.MeshPhongMaterial({color:color,map:plane_map,side:THREE.DoubleSid
e}));
   plane1.rotation.x = -Math.PI / 2;

   //Ramp 1
   var ramp_geometry_1 = new THREE.PlaneGeometry(20, 20, 100, 100);
           var    ramp1    =    new    THREE.Mesh(ramp_geometry_1,    new
THREE.MeshPhongMaterial({color:color,                    map:ramp_map,
side:THREE.DoubleSide}));
   ramp1.rotation.x = -Math.PI / 2 + 0.52;
   ramp1.position.y = 5;
   ramp1.position.x = -80;
   ramp1.position.z = -20;

   //Ramp 2
   var ramp_geometry_2 = new THREE.PlaneGeometry(20, 20, 100, 100);
           var    ramp2    =    new    THREE.Mesh(ramp_geometry_2,    new
THREE.MeshPhongMaterial({color:color,                    map:ramp_map,
side:THREE.DoubleSide}));
   ramp2.rotation.x = Math.PI / 2 - 0.52;
   ramp2.position.y = 15;
   ramp2.position.x = 80;
   ramp2.position.z = -20;

   //Level 2
     var  plane_geometry_2  =  new  THREE.PlaneGeometry(180,  102.68,  100,
100);
```

```
          var    plane2    =    new    THREE.Mesh(plane_geometry_2,    new
THREE.MeshPhongMaterial({color:color,                    map:plane_map,
side:THREE.DoubleSide}));
   plane2.rotation.x = -Math.PI / 2;
   plane2.position.y = 10;
   plane2.position.z = -80;

   //Level 3
     var  plane_geometry_3  =  new  THREE.PlaneGeometry(80,  102.68,  100,
100);
          var    plane3    =    new    THREE.Mesh(plane_geometry_3,    new
THREE.MeshPhongMaterial({color:color,                    map:plane_map,
side:THREE.DoubleSide}));
   plane3.rotation.x = -Math.PI / 2;
   plane3.position.x = 50;
   plane3.position.y = 20;
   plane3.position.z = 40;

   planesGroup.add(plane1);
   planesGroup.add(ramp1);
   planesGroup.add(plane2);
   planesGroup.add(ramp2);
   planesGroup.add(plane3);

   return planesGroup;
}
```

Luz generada por motos: Para darle un aspecto más futurista al juego y que se asemejara más al juego original “tron” tuvimos que crear un material especial para la estela de luz que emite la motocicleta al avanzar para que esta pudiera ser transparente, esto se logró mediante la utilización de un “alpha”. A su vez para poder crear la estela creamos una clase “Panel” con una función llamada “createWall()” que tomaba como referencia la posición de la motocicleta e iba clonando conforme avanzaba nuevos cubos que en conjunto formaban a la estela de luz.

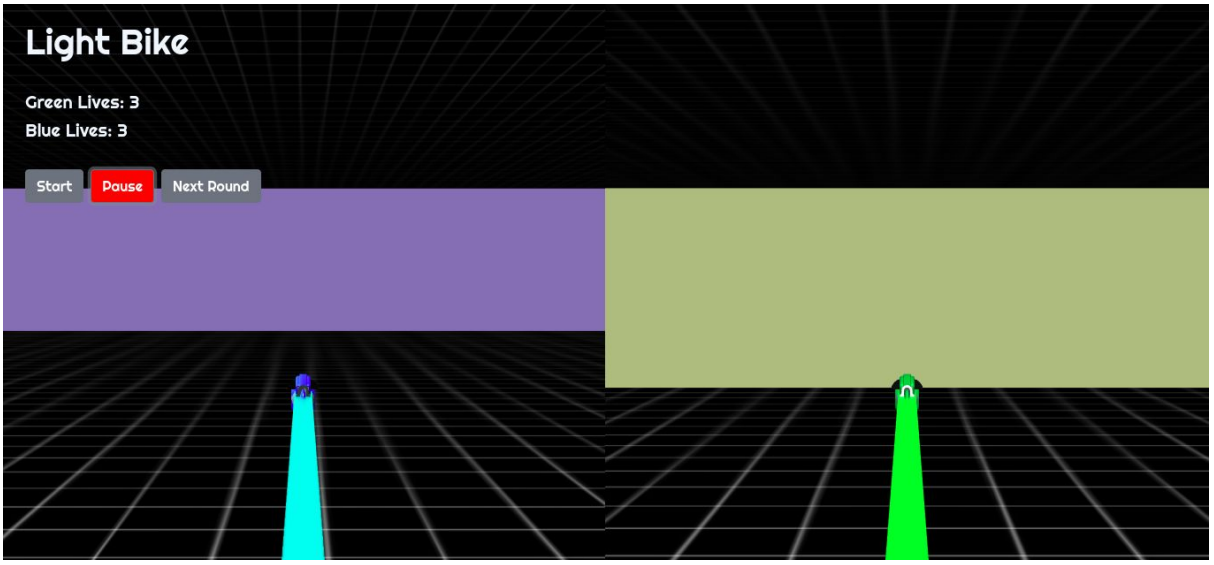



```
//Trail green
var trail_width = 0.5;
var geometry = new THREE.BoxGeometry( trail_width, 1.5, trail_width);
var material = new THREE.MeshBasicMaterial( {color: "#00FC0F", side:
THREE.DoubleSide, opacity: 0.5, transparent:true} );
var planeGreen = new THREE.Mesh( geometry, material );
var planeGreenArr = [];

//Trail2 blue
var material2 = new THREE.MeshBasicMaterial( {color: "#23F9EC", side:
THREE.DoubleSide, opacity: 0.5, transparent:true} );
var planeBlue = new THREE.Mesh( geometry, material2 );
var planeBlueArr = [];
```

```
class Panel {
   constructor() {
       //var lineMaterial = new THREE.LineBasicMaterial({color:
0xff00ff, linewidth:2});
   }

   createWall(bike, bikePosition) {
       if (bike != undefined) {
           if(bike.color == 0) {
               var newPlane = planeGreen.clone();
           }
           else
```

```
                var newPlane = planeBlue.clone();
            newPlane.position.x = bikePosition.x;
            newPlane.position.y = bikePosition.y;
            newPlane.position.z = bikePosition.z;
            arrPlaneCoords.push(newPlane.position);
            newPlane.name = "plane";
            scene.add( newPlane );
        }
    }
}
```

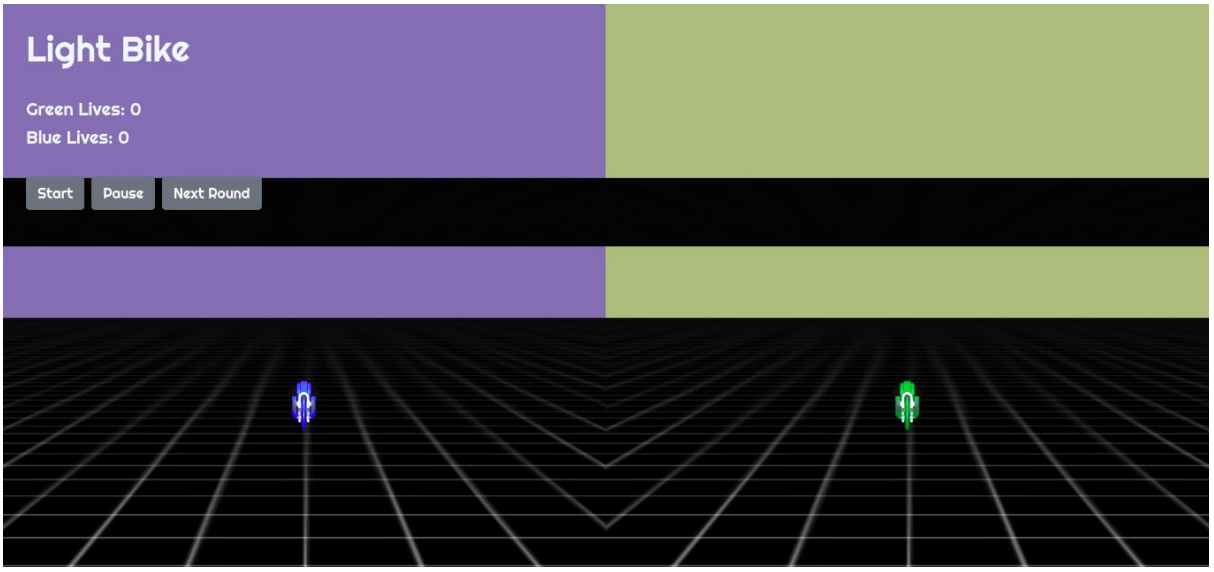
Creación de cámaras dobles: Al ser nuestro juego un juego multijugador se requería que el juego tuviera la pantalla dividida para cada uno de los jugadores tuviera una vista independiente. Para esto creamos dos cámaras con su respectivo recuadro y definimos las coordenadas en las que queríamos que se asignara cada una de las cámaras.




```
views = [
        {
            left: 0,
            bottom: 0,
            width: 0.5,
            height: 1.0,
            background: new THREE.Color( 0.5, 0.5, 0.7 ),
            eye: [ 0, 0, 20 ],
            up: [ 0, 0, 0 ],
            fov: 30,
            updateCamera: function ( camera, scene ) {
```

```
            camera.position.y = 4;
        }
    },
    {
        left: 0.5,
        bottom: 0,
        width: 0.5,
        height: 1.0,
        background: new THREE.Color( 0.7, 0.7, 0.5 ),
        eye: [ 0, 0, 20 ],
        up: [ 0, 0, 0 ],
        fov: 30,
        updateCamera: function ( camera, scene ) {
            camera.position.y = 4;
        }
    }
];
scene = new THREE.Scene();

var view = null;

view = views[0];
camera1 = new THREE.PerspectiveCamera( view.fov, window.innerWidth /
window.innerHeight, 1, 10000 );
camera1.position.fromArray( view.eye );
camera1.up.fromArray( view.up );
view.camera = camera1;

view = views[1];
camera2 = new THREE.PerspectiveCamera( view.fov, window.innerWidth /
window.innerHeight, 1, 10000 );
camera2.position.fromArray( view.eye );
camera2.up.fromArray( view.up );
view.camera = camera2;
```

Seguimiento de objetos con cámaras: Para lograr esto añadimos cada una de las cámaras de manera independiente a cada uno de los objetos de las motos para que estas pudieran seguir al jugador en todo momento.

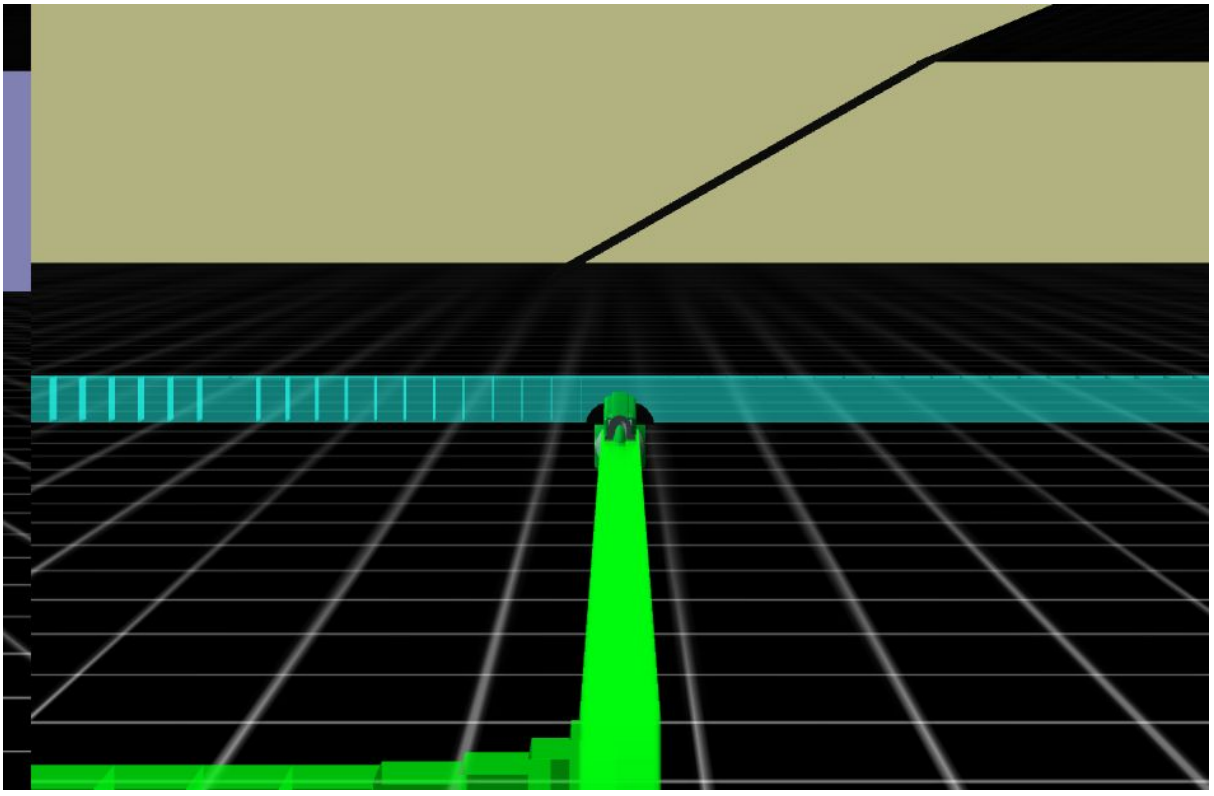



```
async function loadGreenBikeMTL() {

   var mtlLoader = new THREE.MTLLoader();
    mtlLoader.load(
'../models/classicTronGreen/classic-1982-tron-light-cycle-green.mtl',
function( materials ) {
       materials.preload();

       var objLoader = new THREE.OBJLoader();
       objLoader.setMaterials( materials );


objLoader.load('../models/classicTronGreen/classic-1982-tron-light-cycl
e-green.obj', function ( object ) {
           tron_bike_green = object;
           tron_bike_green.add(camera2);
           tron_bike_green.death = false;
           tron_bike_green.deathAnim = false;
           tron_bike_green.limits = true;
           tron_bike_green.rampAnim = new KF.KeyFrameAnimator;
           tron_bike_green.jumpAnim = new KF.KeyFrameAnimator;
           tron_bike_green.up = 0;
           tron_bike_green.color = 0;
           tron_bike_green.position.x += 10;
           tron_bike_green.initialPosition = -10;
           camera2.position.z += 10;
           tron_bike_green.position.x = -10;
```

```
            tron_bike_green.name = "tron_bike_green";
            tron_bike_green.rounds = 3;

            scene.add( tron_bike_green );
        });
    });
}

async function loadBlueBikeMTL() {
    var mtlLoader2 = new THREE.MTLLoader();

    mtlLoader2.load(
'../models/classicTronBlue/classic-1982-tron-light-cycle-blue.mtl',
function( materials ) {
        materials.preload();

        var objLoader = new THREE.OBJLoader();
        objLoader.setMaterials( materials );


objLoader.load('../models/classicTronBlue/classic-1982-tron-light-cycle
-blue.obj', function ( object ) {
            tron_bike_blue = object;
            tron_bike_blue.add(camera1);
            tron_bike_blue.death = false;
            tron_bike_blue.deathAnim = false;
            tron_bike_blue.limits = true;
            tron_bike_blue.rampAnim = new KF.KeyFrameAnimator;
            tron_bike_blue.jumpAnim = new KF.KeyFrameAnimator;
            tron_bike_blue.up = 0;
            tron_bike_blue.color = 1;
            tron_bike_blue.initialPosition = 10;

            camera1.position.z += 10;
            tron_bike_blue.position.x = 10;
            tron_bike_blue.name = "tron_bike_blue";
            tron_bike_blue.rounds = 3;

            scene.add( tron_bike_blue );
        });
```

```
    });

}
```

Colisiones dentro de Three.js: Para poder detectar cuando una moto se impactaba en la luz de otra utilizamos un arreglo de coordenadas que era verificado constantemente para comprobar si alguna de las motos había hecho contacto con alguna de las estelas de luz



```
function checkCollision(bike) {
    var xMin;
    var xMax;
    var zMin;
    var zMax;
    for (var i=0;i<arrPlaneCoords.length - 10;i++) {
        xMin = arrPlaneCoords[i].x - bikeSpeed;
        xMax = arrPlaneCoords[i].x + bikeSpeed;
        zMin = arrPlaneCoords[i].z - bikeSpeed;
        zMax = arrPlaneCoords[i].z + bikeSpeed;

        if (bike.position.x < xMin && bike.position.x > xMax) {
            if (bike.position.z < zMin && bike.position.z > zMax)
            {
                if (bike.position.y === arrPlaneCoords[i].y) {
```

```
                    bike.death = true;
                }

            }
        }

    }
}
```

Movimiento de objetos: Para el movimiento de las motos actualizamos constantemente la posición de la moto en “moveBike()” que se ejecutaba dentro de la función “run()” de nuestro programa. También implementamos la posibilidad de hacer cambiar de dirección a la motocicleta mediante teclas del teclado y que rotara 90 grados a la izquierda o a la derecha

```
var moveBike = function(bike) {

    if (bike != null && roundOver == false) {
        if(bike.color == 0)
        {
            document.getElementById("green_lives").innerHTML =
bike.rounds;
        }else{
            document.getElementById("blue_lives").innerHTML =
bike.rounds;
        }
        if(bike.death == true)
        {
            bike.rounds -= 1;

            deathBike(bike, bike.position.z);
            bike.death = false;
        }
        else
        {
            if(bike.limits == false)
            {
                bike.rounds -= 1;
                //BikeOffLimits(bike, bike.position.y)
                bike.death = true;
            }
            else
```

```
            {
                if(bike.death == false && bike.limits == true)
                {
                    bike.translateZ(bikeSpeed);
                    panel = new Panel();
                    panel.createWall(bike, bike.position);

                    if(bike.position.y == 0)
                    {
                        if(bike.position.z <= -100 || bike.position.z >=
100 || bike.position.x <= -100 || bike.position.x >=100)
                        {
                            bike.limits = false;
                        }
                        else
                        {
                            if(bike.position.z >= -12 && bike.position.x
>= -90 && bike.position.z <= -11 && bike.position.x <= -70)
                                prev_pos = 1;
                            if(bike.position.z >= -11 && bike.position.x
>= -90 && bike.position.z <= -10 && bike.position.x <= -70)
                            {
                                if(prev_pos > 0)
                                    bike.death = true;
                                else
                                {
                                    prev_pos = 0;
                                    BikeonRamp1(bike, bike.rampAnim);
                                }
                            }
                        }
                    }
                    if(bike.position.y == 10)
                    {
                        if(bike.position.z <= -130)
                        {
                            bike.limits = false;
                        }
                        else
                        {
```

```
                            if(bike.position.z >= -31 && bike.position.x
>= -90 && bike.position.z <= -30 && bike.position.x <= -70)
                            {
                                if(bike.up > 2)
                                {
                                    DownBikeonRamp1(bike,
bike.rampAnim);
                                    bike.up = 0;
                                }
                                else
                                    bike.up++;
                            }
                            if(bike.position.z >= -30 && bike.position.x
>= 70 && bike.position.z <= -29 && bike.position.x <= 90)
                            {
                                BikeonRamp2(bike, bike.rampAnim);
                            }
                            else
                            {
                                if(bike.position.x <= 69 &&
bike.position.x>= -69 && bike.position.z >= -31 && bike.position.z <=
-30)
                                {
                                    JumptoFloor(bike, bike.jumpAnim);
                                }
                            }
                        }
                    }
                    if(bike.position.y == 20)
                    {
                        if(bike.position.z >= -11 && bike.position.x >=
70 && bike.position.z <= -10 && bike.position.x <= 90)
                        {
                            if(bike.up > 2)
                            {
                                DownBikeonRamp2(bike, bike.rampAnim);
                                bike.up = 0;
                            }
                            else
                                bike.up++;
```

```
                        }
                        else
                            {
                                if(bike.position.x <= 10 ||
bike.position.x>= 90 || bike.position.z >= 90)
                                {
                                    JumptoFloor2(bike, bike.jumpAnim);
                                }
                                else
                                {
                                    if(bike.position.z >= -11 &&
bike.position.x <= 69 && bike.position.z <= -10)
                                    {
                                        JumptoFloor2(bike,
bike.jumpAnim);
                                    }

                                }
                            }
                    }

                }
            }
        }
    }

}
```

```
function onDocumentKeyDown(event) {
    event.preventDefault();
    var keyCode = event.which;
    if (keyCode == 65 && !blueRotateAnim.running) {
        //tron_bike_blue.rotateY(Math.PI/2);
        rotateBike(tron_bike_blue, Math.PI/2, blueRotateAnim);
    }
    if (keyCode == 68 && !blueRotateAnim.running) {
        //tron_bike_blue.rotateY(-Math.PI/2);
        rotateBike(tron_bike_blue, -Math.PI/2, blueRotateAnim);
    }
```

```
    if (keyCode == 74 && !greenRotateAnim.running) {
        rotateBike(tron_bike_green, Math.PI/2, greenRotateAnim);
    }
    if (keyCode == 76 && !greenRotateAnim.running) {
        rotateBike(tron_bike_green, -Math.PI/2, greenRotateAnim);
    }

    if(keyCode == 38) {
        console.log("pushed up");
        camera1.position.z += 10;
    }

    if(keyCode == 40) {
        console.log("pushed down");
        camera1.position.z -= 10;
    }

    if(keyCode == 37) {
        console.log("pushed left");
        camera1.position.x -= 10;
    }

    if(keyCode == 39) {
        console.log("pushed right");
        camera1.position.x += 10;
    }
};
```