Diseño del Juego

José Ramón Plaza Plaza Jaime Parra Jiménez

Índice

1. Diagrama de Clases

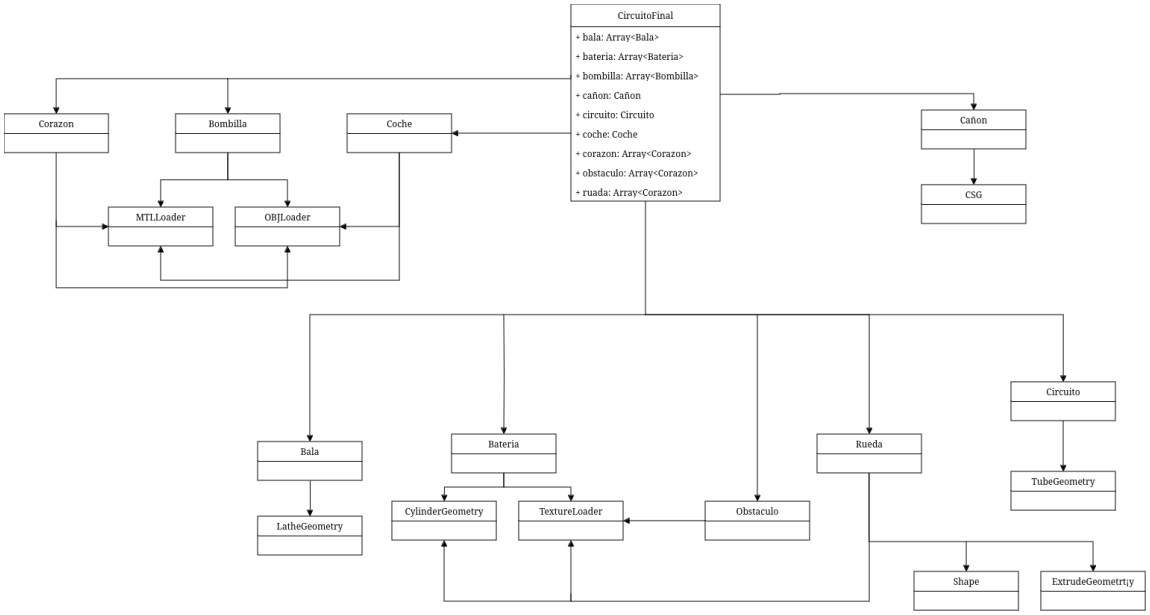

Figura 1: UML del juego.

2. Modelo Jerárquico del Cañón

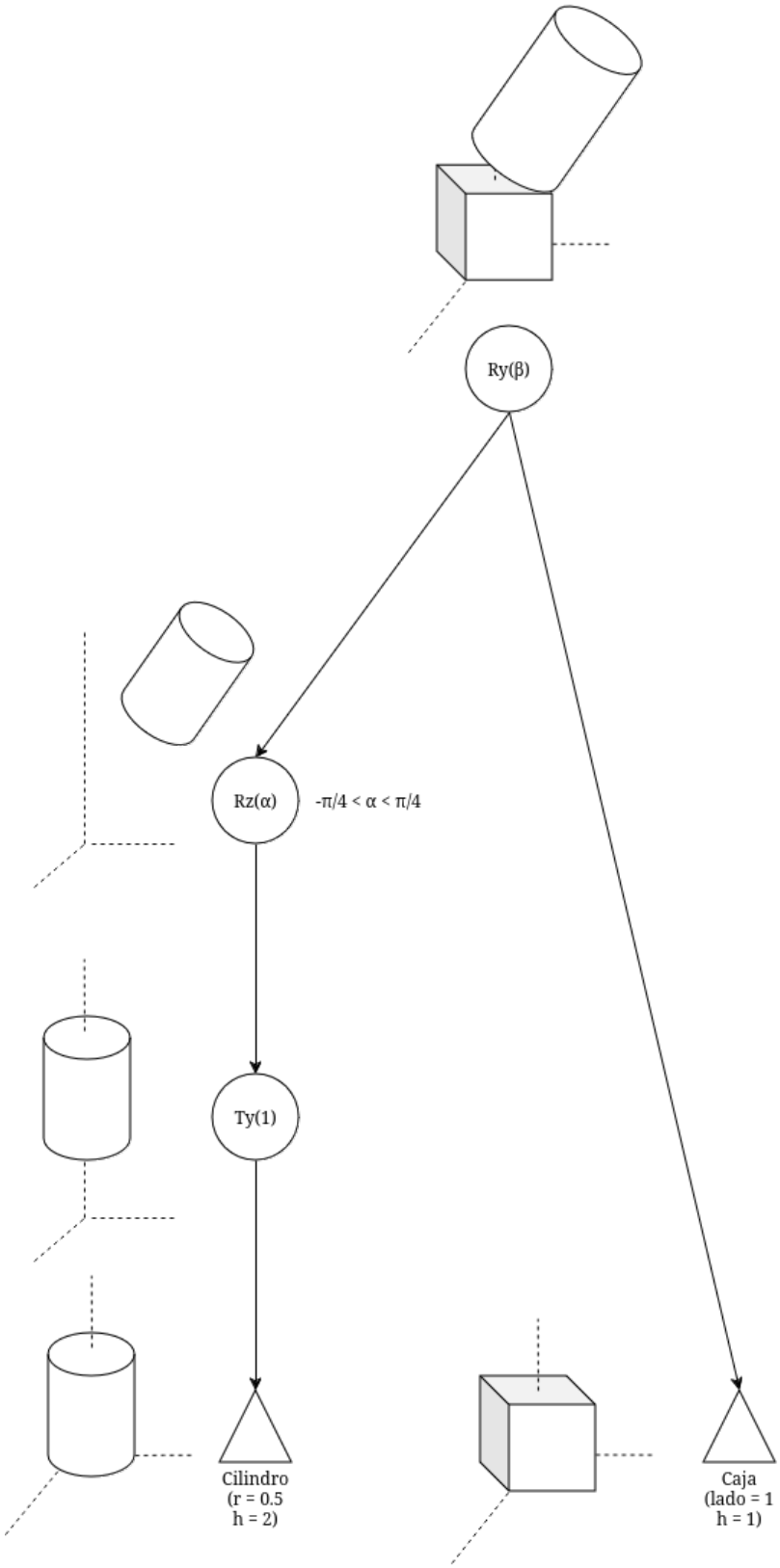

Figura 2: Modelo jerárquico.

3. Algoritmos más importantes

3.1. Constructor Inicial

En el constructor inicial declaramos las variables principales para nuestro juego como son las variables para controlar la velocidad de nuestro coche (Figura 3) y las variables para la jugabilidad del juego (Figura 4).

```
// Varriables para el control del coche y el cañón
this.reloj = new THREE.Clock();
this.velocidad = 1/70;
this.t = 0;
this.rotacion = 0;
this.sentido = 1;
this.velocidadRot = Math.PI / 300;
this.limiteSup = Math.PI / 4;
this.limiteInf = -Math.PI / 4;
```

Figura 3: Variables para controlar el movimiento del coche y el cañón.

```
// Varriables para la jugabilidad
this.puntuacion = 0;
this.balas = 25;
this.vidas = 3;
```

Figura 4: Variables para la jugabilidad.

Además en el constructor creamos todos los objetos que irán dentro de nuestro circuito como son los muros, las bombillas, las vidas... también los objetos voladores, en nuestro caso las ruedas, y el propio circuito por el cual circula nuestro coche. Una vez creados los objetos, los añadimos al circuito con la función *'colocarEnCircuito'* a la cual le pasamos el objeto y su posición en éste. Por último creamos las cajas de cada objeto para realizar las colisiones de los objetos con el coche y añadimos los objetos a la escena.

3.2. Picking

Hemos implementado el picking para obtener puntos simulando que disparamos a los objetos voladores que se encuentran por la escena (en nuestro caso las ruedas). Para ello hemos recogido un listener para obtener las posiciones del ratón al hacer *click* sobre la escena. Una vez tenemos las coordenadas, las pasamos al raycaster junto con la cámara, y por último pasando al raycaster los objetos que son clicables con el ratón, nos devolverá un array con los objetos que han sido alcanzados. Cuando ya tenemos este último array comprobamos si se ha alcanzado alguno de los objetivos e implementamos la función que sumará los puntos y disminuirá las balas.

```
onDocumentMouseDown(event){

  this.mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  this.mouse.y = 1 - 2 * (event.clientY / window.innerHeight);

  this.raycaster.setFromCamera(this.mouse, this.getCamera());

  var pickedObjects = this.raycaster.intersectObjects(this.pickableObjects, true);

  if(pickedObjects.length > 0){
    if(this.balas>0){
```

Figura 5: Implementación del picking.

3.3. Colisiones

Para las colisiones, creamos una caja que engloba a todos los objetos y a nuestro coche para controlar las colisiones mediante cajas englobantes (Figura 6):

```
crearCaja(objeto){

  var caja = new THREE.Box3();
  caja.setFromObject(objeto);

  var caja2 = new THREE.Box3Helper(caja, 0x000000);
  this.add(caja2);
  caja2.visible = true;

  return caja;
}
```

Figura 6: Método para crear la caja de un objeto.

Una vez creada la caja de cada uno de los objetos (la creamos en el constructor de nuestra escena) solo falta implementar las funciones, que irán en el método update, para ir comprobando constantemente si la caja de nuestro coche colisiona con la caja de cualquier otro objeto para realizar la función implementada en cada caso:

```
choqueBalas(cajaBala,cajaCoche){

  if(cajaBala.intersectsBox(cajaCoche)){
    this.cambiarBalas(25);
    console.log("Balas recargadas");
  }
}
```

Figura 7: Método para comprobar si se produce un choque entre cajas englobantes.

En este ejemplo (Figura 7), vemos que con el método *'instersectsBox'* se comprueba si una de las cajas colisiona con la otra que se le pasa como parámetro a la función y ya se implementa lo que se quiera que pase en el caso de que haya colisión.

3.4. Cámaras

En nuestro juego, hemos implementado dos cámaras que podremos ir alternando cuando pulsamos la tecla del espacio. Una de las cámaras es en primera persona (Figura 9) y se añade al nodo del coche para que siga por el circuito, la otra cámara es una cámara general (Figura 8) con la cual podremos interaccionar para ver toda la escena.

```
createCameraGeneral(){
  // Para crear una cámara le indicamos
  //   El ángulo del campo de visión en grados sexagesimales
  //   La razón de aspecto ancho/alto
  //   Los planos de recorte cercano y lejano
  this.camera = new THREE.PerspectiveCamera(90, window.innerWidth / window.innerHeight, 0.1, 50);
  // Recuerda: Todas las unidades están en metros
  // También se indica dónde se coloca
  this.camera.position.set (4, 2, 4);
  // Y hacia dónde mira
  var look = new THREE.Vector3 (0,0,0);
  this.camera.lookAt(look);
  /* this.add (this.camera); */

  // Para el control de cámara usamos una clase que ya tiene implementado los movimientos de órbita
  this.cameraControl = new TrackballControls (this.camera, this.renderer.domElement);
  // Se configuran las velocidades de los movimientos
  this.cameraControl.rotateSpeed = 5;
  this.cameraControl.zoomSpeed = -2;
  this.cameraControl.panSpeed = 0.5;
  // Debe orbitar con respecto al punto de mira de la cámara
  this.cameraControl.target = look;

  return this.camera;
}
```

Figura 8: Cámara general.

```
createCamaraSubjetiva(){

  this.camara = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight, 0.1, 50);

  const distanciaDetras = 0.8;
  const altura = 0.7;
  this.camara.position.set(this.coche.position.x, this.coche.position.y + altura, this.coche.position.z - distanciaDetras);

  const puntoDeMiraRelativo = new THREE.Vector3(0, 0.5, 0);

  const target = new THREE.Vector3();
  target.add(puntoDeMiraRelativo);
  this.camara.lookAt(target);

  return this.camara;
}
```

Figura 9: Cámara priemra persona.

Para el cambio de las cámaras, hemos definido una variable que llamamos *'camaraGeneral'* que le iremos asignando una cámara diferente cada vez que se pulsa el espacio. Para ello antes de nada creamos un listener y luego se comprueba que la tecla haya sido el teclado y se actualizan las cámaras.

```
cambiarCamara(event){
  if(event.which === 32){
    if(this.camaraActiva === this.camaraSubjetiva){
      this.coche.remove(this.camaraSubjetiva);
      this.camaraActiva = this.camaraGeneral;
      this.add(this.camaraGeneral);
    }
    else if(this.camaraActiva === this.camaraGeneral){
      this.remove(this.camaraGeneral);
      this.camaraActiva = this.camaraSubjetiva;
      this.coche.add(this.camaraSubjetiva);
    }
  }
}
```

Figura 10: Función cambio de cámara.

Como se observa en la Figura 8 la cámara subjetiva que es la cámara en primera persona, se añade al coche para que esta siga el movimiento del coche en todo momento, mientras que la cámara general se añade a la escena.

3.5. Colocación de los objetos en el circuito

Para colocar los objetos en el circuito hacemos uso de la función *'colocarEnCircuito'* que coloca un objeto en una posición específica a lo largo de una trayectoria definida por un path y orienta el objeto de acuerdo con la geometría de la trayectoria.

```
colocarEnCircuito(objeto , t){

  var posTmp = this.path.getPointAt(t);
  objeto.position.copy(posTmp);

  var tangente = this.path.getTangentAt(t);
  posTmp.add(tangente);
  var segmentoActual = Math.floor(t*this.segmentos);
  objeto.up = this.tubo.binormals[segmentoActual];
  objeto.lookAt(posTmp);

}
```

Figura 11: Función para colocar un objeto en el circuito.

Como podemos ver en la Figura 9, obtenemos el pnuto *'t'* en la trayectoria, obtenemos la tangente en dicho punto para indicar hacía donde debe mirar el objeto y la orientación del mismo.

3.6. Animación de los objetos

Para la animación de los objetos, en nuestro caso, la animación de la rueda que es la única se mueve siguiendo una trayectoria, hemos utilizado la biblioteca TWEEN que nos permite realizar una animación mediante caminos especificando el tiempo que queremos que se tarde en realizar dicho camino.

Nuestra función *'animaciónRueda'* es la que usamos para la animación de las ruedas, en ella creamos la animación que varía *'t'* durante un tiempo en milisegundos. Después mediante *'.onUpdate'* vamos actualizando la posición, orientación y dirección de la rueda en cada actualización de la animación. Cuando la animación se ha completado, volvemos a reiniciar la *'t'* para que se comience de nuevo la animación y se repita indefinidamente con *'.repeat(Infinity)'*.

```
var animacion1 = new TWEEN.Tween(origen1).to(fin1, tiempoDeRecorrido1)
.onUpdate(() => {
    var posicion1 = spline.getPointAt(origen1.t);
    rueda.position.copy(posicion1);
    var tangent1 = spline.getTangentAt(origen1.t);
    posicion1.add(tangent1);
    rueda.up = binormales1[Math.floor(origen1.t * segmentos1)];
    rueda.lookAt(posicion1);
})
.onComplete(() => {origen1.t=0.1;
    animacion1.start();
})
.repeat(Infinity)
.start();
```

Figura 12: Creación de la animación de la rueda.

3.7. Animación del coche

Para la animación del coche, a diferencia de las ruedas, no hemos hecho uso de la biblioteca TWEEN ya que debiamos de ir variando la velocidad de nuestro coche cuando suceden diferentes eventos y con esta biblioteca no se podía hacer.

Hemos implentado una función para la velocidad independiente del ordenador y que se ejecutará en el método *'update()'* para que se ejecute en cada frame y se vaya actualizando el objeto a lo largo de una trayectoria.

Primero, colocamos el objeto en la posición inicial (de igualmanera que colocamos cualquier otro objeto en el circuito) y después vamos a ir modificando la *'t'*, es decir la posición del coche, dependiendo los segundos transcurridos en cada frame y una variable velocidad.

```
if(this.empezar){

  this.segmentos = 100;
  this.binormales = this.splineCoche.computeFrenetFrames(this.segmentos,true).binormals;

  var segundosTrancurridos = this.reloj.getDelta();

  if(this.t === 0){
    console.log('aumento velocidad');
    this.setVelocidad();
  }

  this.t += this.velocidad * segundosTrancurridos;

  if(this.t >= 1){
    this.t = 0;
  }
}
this.coche.actualizarCaja();

this.nodo.add(this.coche);
this.nodo.add(this.cañon);
```

Figura 13: Animación del coche.

3.8. Giros del coche

Se ha implementado los giros del coche a izquierda y derecha pulsando las teclas **'a'** o **'flecha izquierda'** para girar a la izquierda y las teclas **'d'** o **'flecha derechaz'** para girar a la derecha.

```
giroCocheDown(event){
  if (event.which === 37 || event.which === 65) {
    this.left = true;
  } else if (event.which === 39 || event.which === 68) {
      this.right = true;
  }
}

giroCocheUp(event){
  if (event.which === 37 || event.which === 65) {
    this.left = false;
  } else if (event.which === 39 || event.which === 68) {
      this.right = false;
  }
}
```

Figura 14: Control de las teclas para los giros.

Mediante un *'listener'* controlamos las pulsaciones de las teclas con el teclado y realizamos una pequeña rotación respecto al eje z. Además para complicar la jugabilidad de nuestro juego, si se realiza un aumento en la velocidad, también se aumenta la velocidad de giro de nuestro objeto.

```
updateMovimientoCoche(){
  if(this.left){
    this.moverIzquierda();
  }
  if(this.right){
    this.moverDerecha();
  }
}
```

Figura 15: Se gira mientras se tiene pulsada la tecla.

```
moverIzquierda(){
  this.coche.rotation.z -= 1 * this.velocidad;
  this.cañon.rotateZ(-1 * this.velocidad);
}

moverDerecha(){
  this.coche.rotation.z += 1 * this.velocidad;
  this.cañon.rotateZ(1 * this.velocidad);
}
```

Figura 16: Rotación simulando el giro del coche y el cañon.

3.9. Luces

Para las luces hemos creado dos situaciones, una cuando es de día en la cual está encendida solo la luz ambiental y la otra situación es cuando se hace de noche al chocar con un muro. En este segundo caso, la luz ambiental general se apaga y parecen destintos focos de colores sobre la escena, uno de ellos se inserta en el nodo y sigue el movimiento del coche. Tambien jugamos con una imagen de fondo que cambia entre la noche y el dia.

```
apagarLuces() {
  this.setAmbientIntensity(0);
  this.setAmbientIntensity1(0);
  this.loader.load('../imagenes/noche.jpg', (texture) => {
    this.background = texture;
  });
}
```

Figura 17: Función para apagar las luces.

```
encenderLuces() {
  this.setAmbientIntensity(2);
  this.setAmbientIntensity1(1);
  this.loader.load('../imagenes/dia.jpg', (texture) => {
    this.background = texture;
  });
}
```

Figura 18: Función encender las luces.

```
setAmbientIntensity1 (valor) {
  if (valor === 0) {
    this.add(this.redLight);
    this.add(this.blueLight);
    this.add(this.greenLight);
    this.specularLight.intensity = 1;
  } else {
    this.remove(this.redLight);
    this.remove(this.blueLight);
    this.remove(this.greenLight);
    this.specularLight.intensity = 0;
  }
}
```

Figura 19: Función que añade las luces.