Prácticas Sistemas Gráficos

UNIVERSIDAD DE GRANADA

Miguel Bravo Campos

Documentación

• Documentación: El documento debe incluir, al menos, las siguientes secciones:

◦ La descripción del juego entregada al principio de la asignatura, con las correcciones incorporadas por el profesor.

◦ El diseño de la aplicación. Incluyendo diagrama de clases, modelos jerárquicos y la descripción de los algoritmos más importantes de la misma.

◦ Si habéis usado modelos u otro material descargados de internet debéis incorporar a la documentación una sección con todas las referencias.
Por ejemplo: ▪ La nave la he descargado de http://modeloslibres.com/naveChula.zip
▪ He usado tal biblioteca, distinta a las vistas en clase, para tal cosa descargada de http://bibliotecas-superutiles.com/biblioteca.js

◦ Un manual de usuario que explique, claramente, todo lo indicado en el punto 6 de la sección "Desarrollo".

• En la carpeta de documentación se incluirá una demo en vídeo (captura de pantalla) con una duración de entre 20 y 40 segundos que muestre lo mejor del juego.

Primera Descripción del Juego

Primero, para aclarar, no tengo compañero, por lo tanto, voy a hacer el proyecto solo. Mi juego va a ser simplemente una copia de Mario Kart en el que vamos a tener los objetos básicos iguales que en el juego original, como la moneda y el champiñón. Además, voy a incluir objetos como un muelle que permitirá al usuario saltar. Estos tres objetos mencionados serán buenos y harán que el coche aumente su velocidad de manera infinita o finita según cada uno. Igualmente, voy a implementar objetos negativos, así como las columnas de Mario Kart que adjuntaré una foto y estarán por el mapa moviéndose. Un fantasma, en el caso de que se toque alguna columna, entonces se recibirá una penalización ya sea permanente o finita, igual que con los objetos buenos.

En el caso de los objetos voladores, no he entendido muy bien a qué se refiere, pero supongo que serán iguales que los otros objetos tanto buenos como malos. La idea que me ha surgido es unas alas para objeto volador bueno y un caparazón con alas como objeto volador malo.

Mi personaje será un personaje típico característico de Mario Kart, seguramente será Toad, ya que lo tengo medio modelado con el champiñón que ya tengo, el cual adjuntaré más tarde, al igual que la moneda y un primer intento del futuro muelle con la técnica de barrido que haré.

El objeto articulado será un brazo gigante que estará en constante movimiento, tanto el brazo desde el codo como si estuviera dando un golpe de karate, y los dedos como si estuviera cerrando el puño. También puede ser que, en el caso de que el personaje se choque con el brazo, obtenga una penalización y por tanto tenga que esquivarlo.

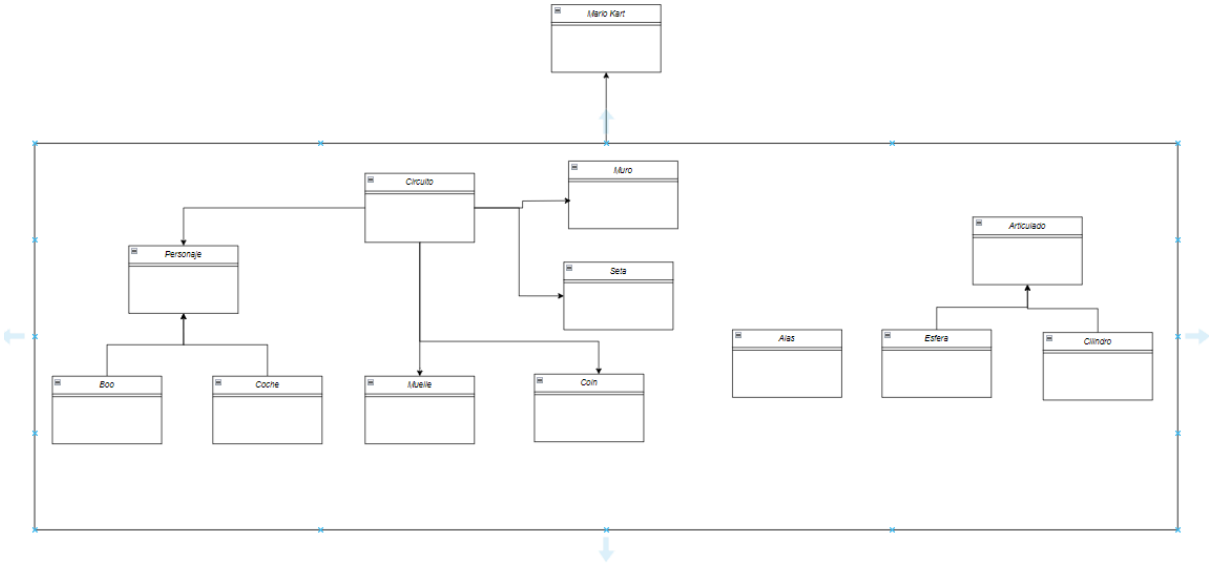
-Correcciones del profesor

No dispongo del correo con las correcciones del profesor, ya que el webmail de la ugr no me deja ir tanto para atrás en el tiempo, pero creo recordar que estuvo de acuerdo en las decisiones que dije que iba a tomar. Los cambios que han habido con respecto a mi prototipo de juego han sido los siguientes:

- El personaje ya no es Toad si no que es Boo.
- El muelle no produce un salto si no que incrementa la velocidad
- Solo se ha creado un objeto volador
- Los movimientos del brazo son golpe de karate y abrir/cerrar el codo

Esos son los únicos cambios que se han realizado con respecto a la propuesta.

Diagrama de Clases



Descripción del Diagrama de Clases:

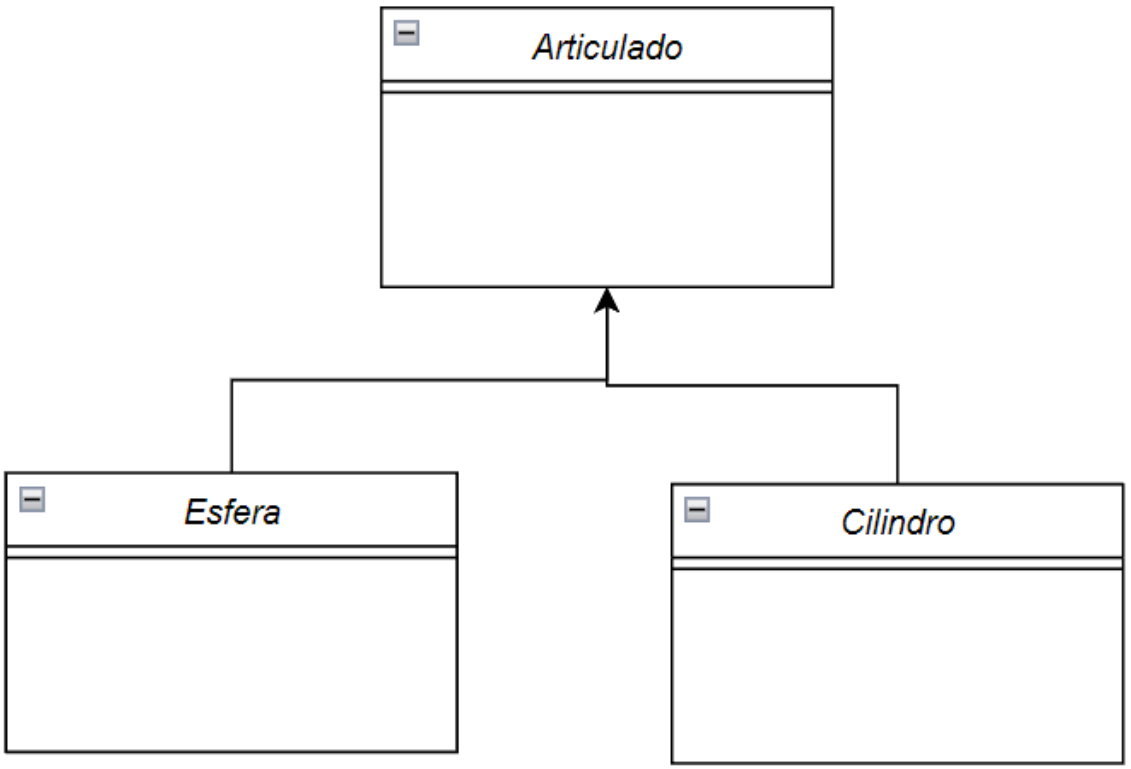
Personaje: Está compuesto por la clase Boo y la clase Coche para su creación y para su movimiento requiere de la clase Circuito.

Circuito: El circuito principal de esta clase requieren Muro,Seta,Coin,Muelle y Personaje.

Articulado: Está compuesto simplemente por Cilindros y Esferas.

Mario Kart: Es el juego completo está compuesto por todas las clases descritas anteriormente

Modelo Jerarquico



El modelo jerárquico en su totalidad simplemente se ha creado con esferas para simular tanto el hombro, codo y muñeca es decir para simular las articulaciones.
Cilindros para simular el bíceps, antebrazo, dedos para simular los huesos que conectan las articulaciones.

A la hora de grados de libertad cuenta con dos:

- Primer grado de libertad en el hombro que permite subir y bajar el brazo completo
- Segundo grado de libertad en el codo que permite girar hacia un lado u otro el antebrazo, muñeca y dedos

Para que también se pueda mover de manera manual y poder realizar esos movimientos se han hecho uso de las teclas:

```
onKeyDown(event) {
  switch (event.key) {
    case 'w':
      this.rotateShoulder(Math.min(this.shoulderAngle + 0.1, Math.PI / 2));
      break;
    case 's':
      this.rotateShoulder(Math.max(this.shoulderAngle - 0.1, -Math.PI / 2));
      break;
    case 'a':
      this.rotateElbow(Math.max(this.elbowAngle - 0.1, -Math.PI / 2));
      break;
    case 'd':
      this.rotateElbow(Math.min(this.elbowAngle + 0.1, Math.PI / 2));
      break;
    default:
      break;
  }
}
```

Por último con respecto al modelo jerárquico se ha realizado una animación que involucra ambos movimientos tanto el de subir bajar el brazo con el de abrir/cerrar el codo.

```
animateObject() {
  var origen = { y: -(Math.PI / 2), z: 0 }; // Inicia desde -pi/2 grados en Y y 0 grados en Z
  var destino = { y: Math.PI / 2, z: Math.PI / 2 }; // Gira hasta pi/2 (90 grados) en Y y Z

  var movimiento = new TWEEN.Tween(origen)
      .to(destino, 5000) // Duración de 5000 ms (5 segundos)
      .easing(TWEEN.Easing.Quadratic.InOut) // Utiliza una función de interpolación suave
      .onUpdate(() => {
          // Actualiza la rotación del objeto en los ejes Y y Z
          this.rotation.y = origen.y;
          this.rotation.z = origen.z;
      })
      .repeat(Infinity) // Repite infinitamente
      .yoyo(true); // Hace que la animación vuelva en la dirección opuesta

  // Iniciar la animación
  movimiento.start();
}
```

Se ha realizado a través de la librería Tween.js.
En cuanto a los algoritmos más importantes no se si se refiere al Picking o a la detección de colisiones pero eso lo abordaré en otro punto de la documentación que es **Un manual de usuario que explique, claramente, todo lo indicado en el punto 6 de la sección "Desarrollo",** para poder concretarlo todo más preciso.

Recursos Utilizados

Modelo Cargado

El modelo cargado que he utilizado ha sido para el coche y lo puede encontrar aquí:

-Coche Mario Kart

Librerías

- Librería tween.esm.js
- Librería THREE.js
- Librería CSG-v2.js
- Librería MTLLoader.js
- Librería OBJLoader.js

No he utilizado ninguna librería aparte de las proporcionadas en el código.

Manual de Usuario

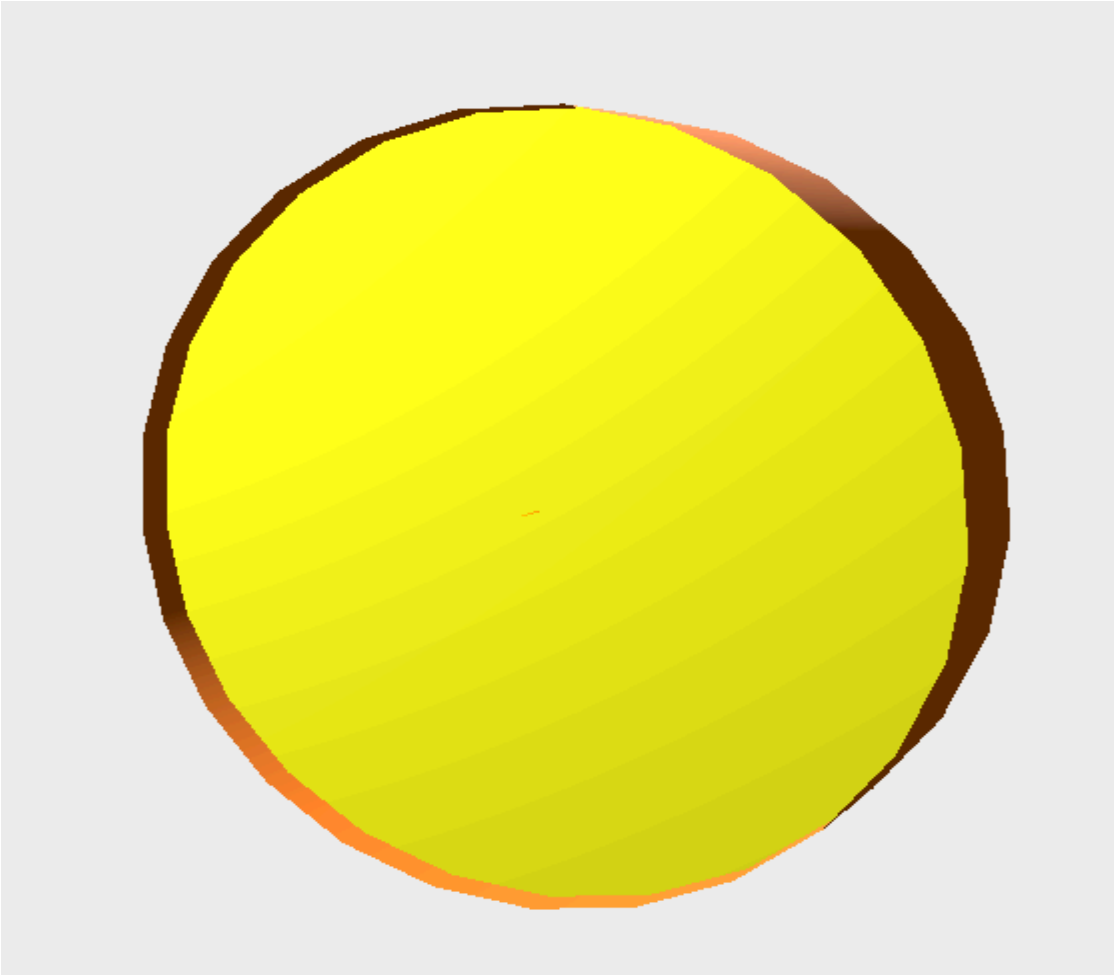
Tal como se pide en la documentación voy a proceder a explicar todo lo indicado en el punto 6 de la sección “Desarrollo”:

Realizar un manual de usuario que explique claramente cómo afecta cada elemento del juego al personaje: tanto los elementos que habría que atrapar y qué ventajas obtiene el personaje por atraparlos como los elementos con los que no tendría que colisionar y en qué medida se vería perjudicado el personaje por dicha colisión. Explicar también los beneficios que aportan al personaje disparar a los objetos voladores.

Vamos a describir cada objeto 1 por 1:

Coin

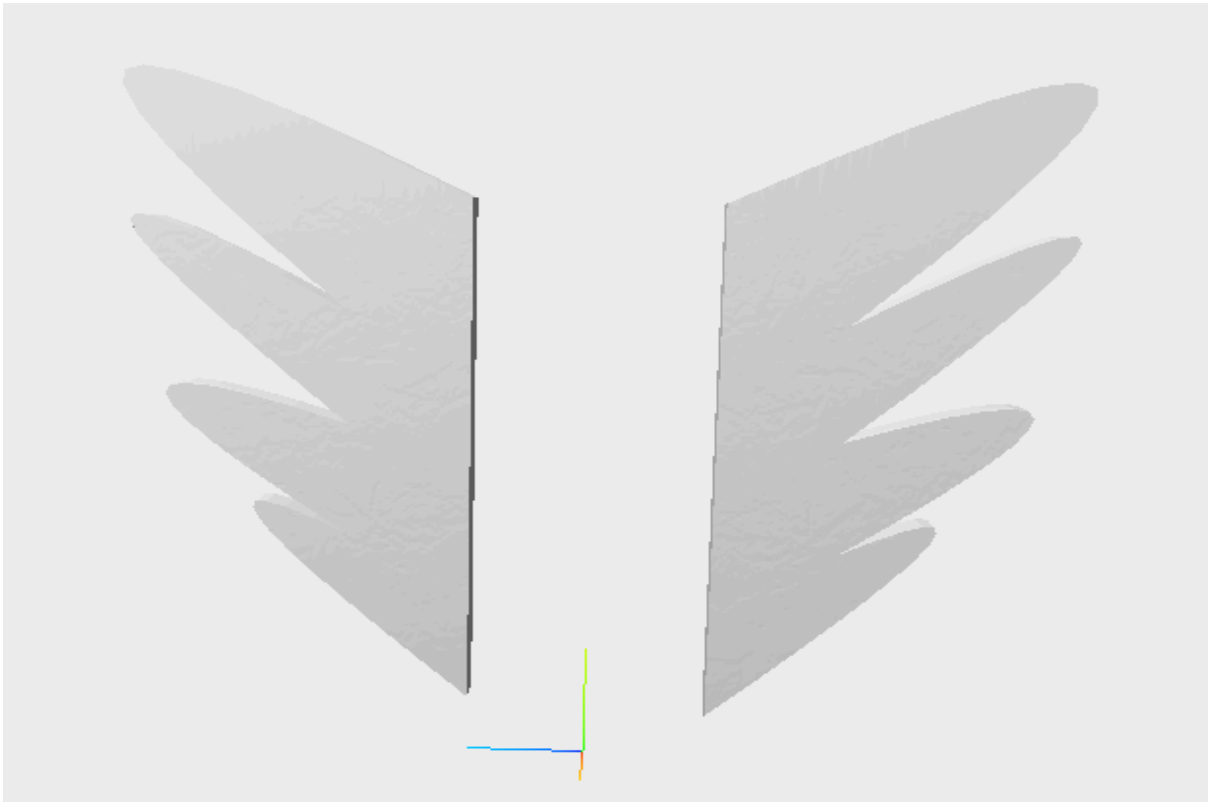
El objeto Coin fue el primer elemento que diseñe simplemente utilizando geometría básica, son dos cilindros uno vacío por dentro solo el contorno exterior y otro entero.
Se puede encontrar andando de manera normal por el tubo y tiene un efecto positivo ya que si el coche colisiona con él se aumenta su velocidad de manera permanente.



Alas

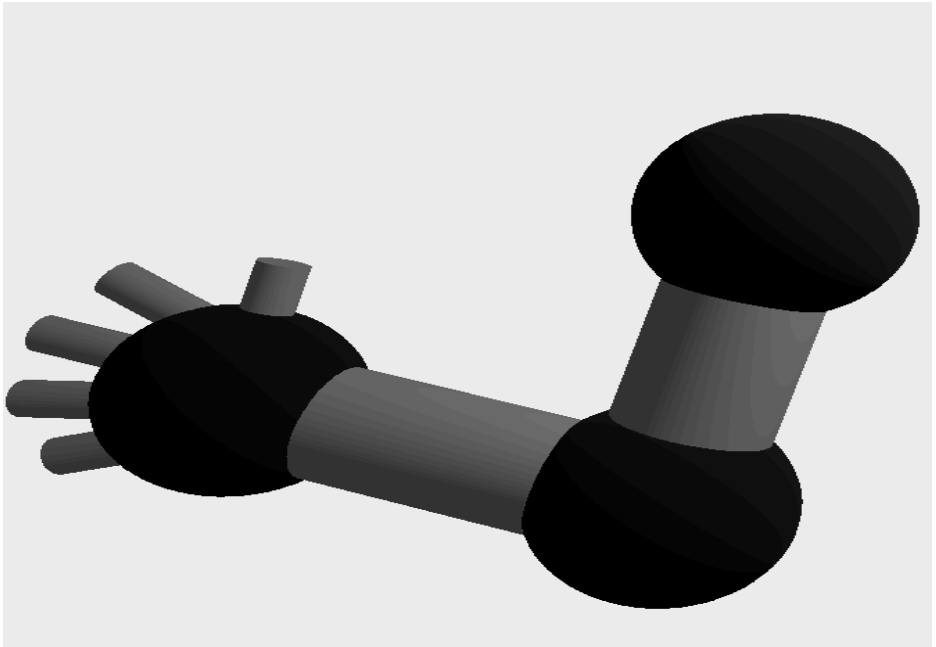
El método utilizado para crear las alas ha sido extrusión, primero hemos definido un shape con diferentes curvas con quadraticCurveTo y después le hemos hecho extrusión.
Le hemos aplicado una textura en el canal de relieve que da la impresión de que fuera un papel arrugado, una vez que se realiza un ala simplemente se le realiza un clone y se rota.

El impacto que tiene en el juego es el siguiente, se puede interactuar con ellas mediante picking pulsando encima de ellas, lo que produce que aumente la puntuación que se encuentra arriba a la izquierda.



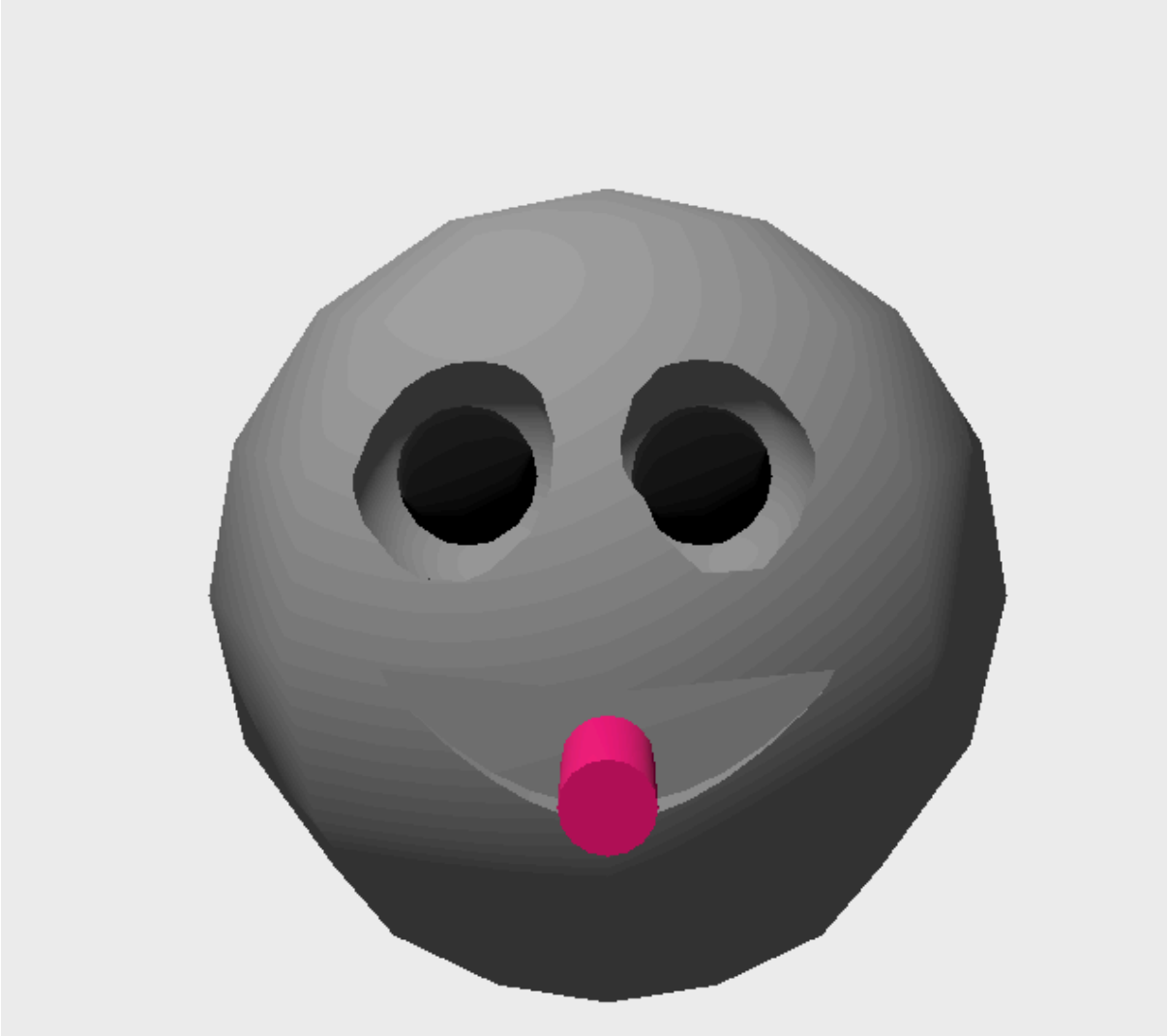
Articulado

El objeto articulado la forma de su creación se ha descrito anteriormente por tanto vamos a describir el impacto que tiene en el juego, en caso de que el personaje impacte contra la mano perderá velocidad de manera permanente.



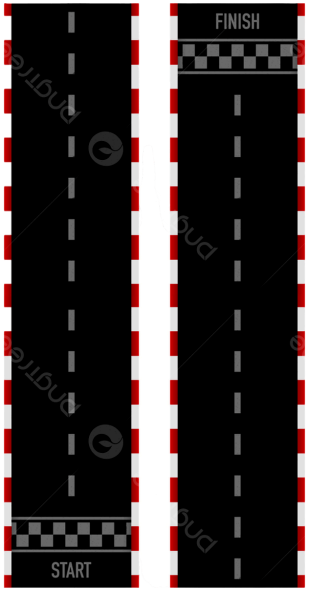
Boo

El objeto Boo representa una parte del personaje se ha realizado utilizando CSG con uniones, diferencias y todo tipo de técnicas.
El impacto que tiene en el juego simplemente es ser parte del personaje.



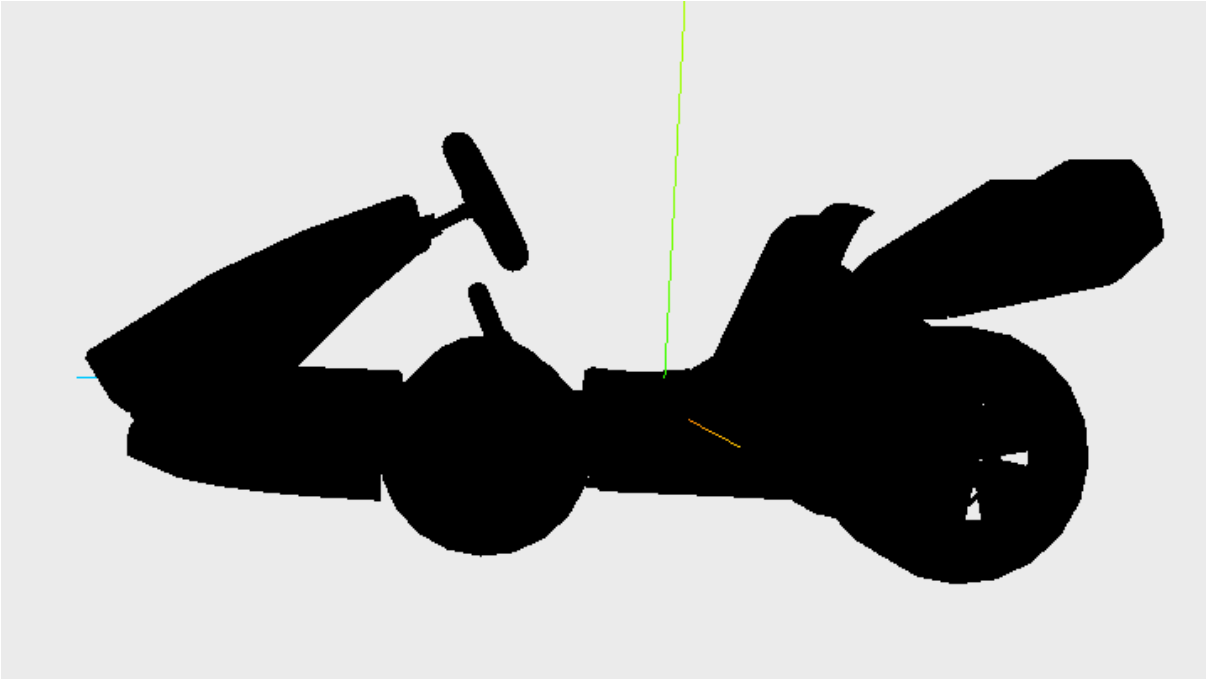
Circuito

El objeto circuito tal como su nombre indica es simplemente el circuito a través del que van a circular la mayoría de los objetos, simplemente se ha realizado con un tubeGeometry y se le ha aplicado una textura encima que es esta:
La interacción que tiene en el juego es simplemente dar sus datos para que los demás objetos puedan circular por encima suya.


FINISH
START

Coche

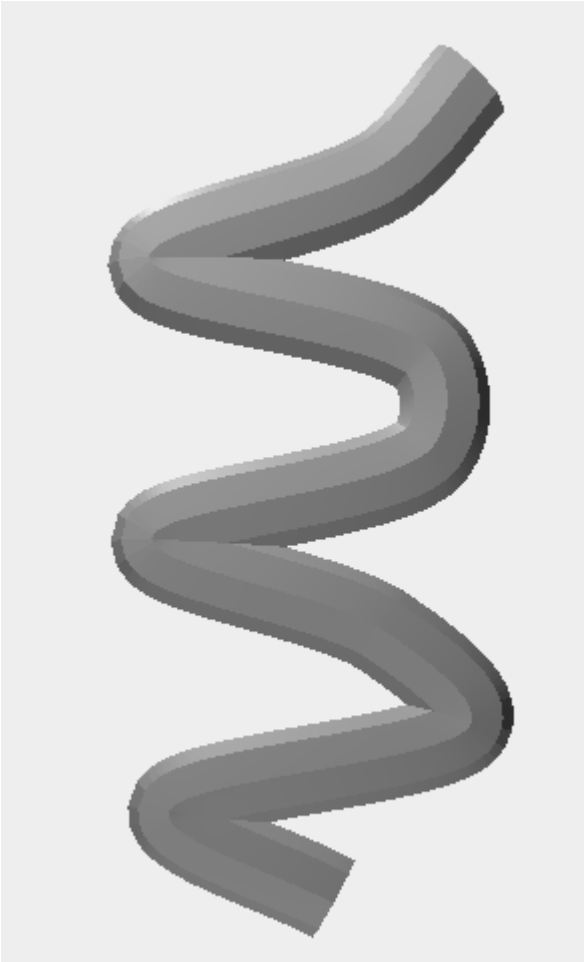
El objeto coche es un modelo cargado de la ubicación que he enlazado anteriormente, simplemente, no tiene otra interacción con el juego que juntarse con Boo para así formar el personaje.



Muelle

El objeto muelle se ha construido a través de barrido simplemente dando una geometría 2D inicial y una línea de puntos a la que seguir, se ha construido más bien en 2D que en 3D ya que simplemente sube y no se enrosca como haría un verdadero muelle.

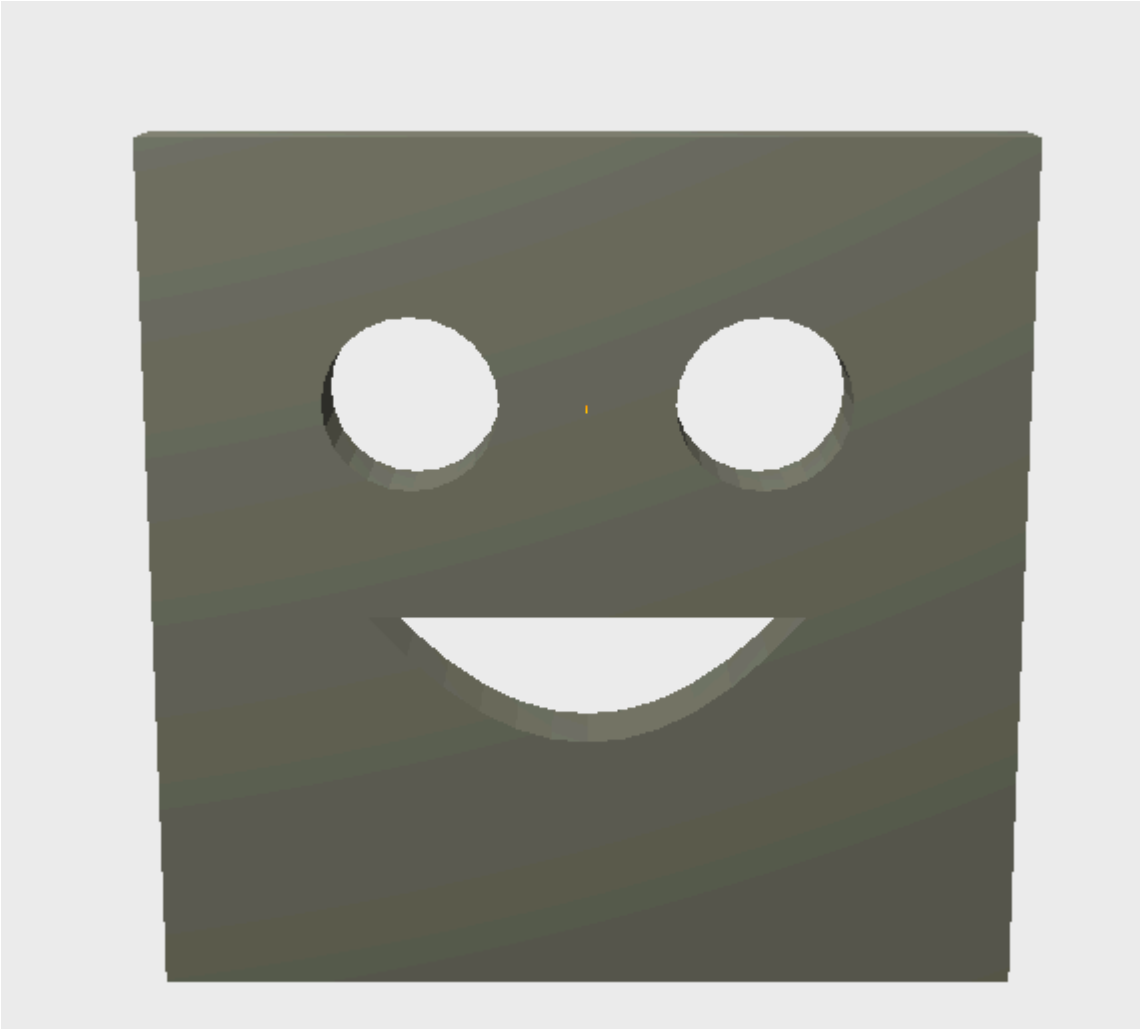
El impacto que tenía en el juego inicialmente era de producir un salto y hacer que el coche vuele pero se cambió su propósito por incrementar de forma permanente la velocidad del personaje.



Muro

El objeto Muro se realizó de manera muy simple utilizando CSG es una réplica de un objeto del juego de Mario Kart:

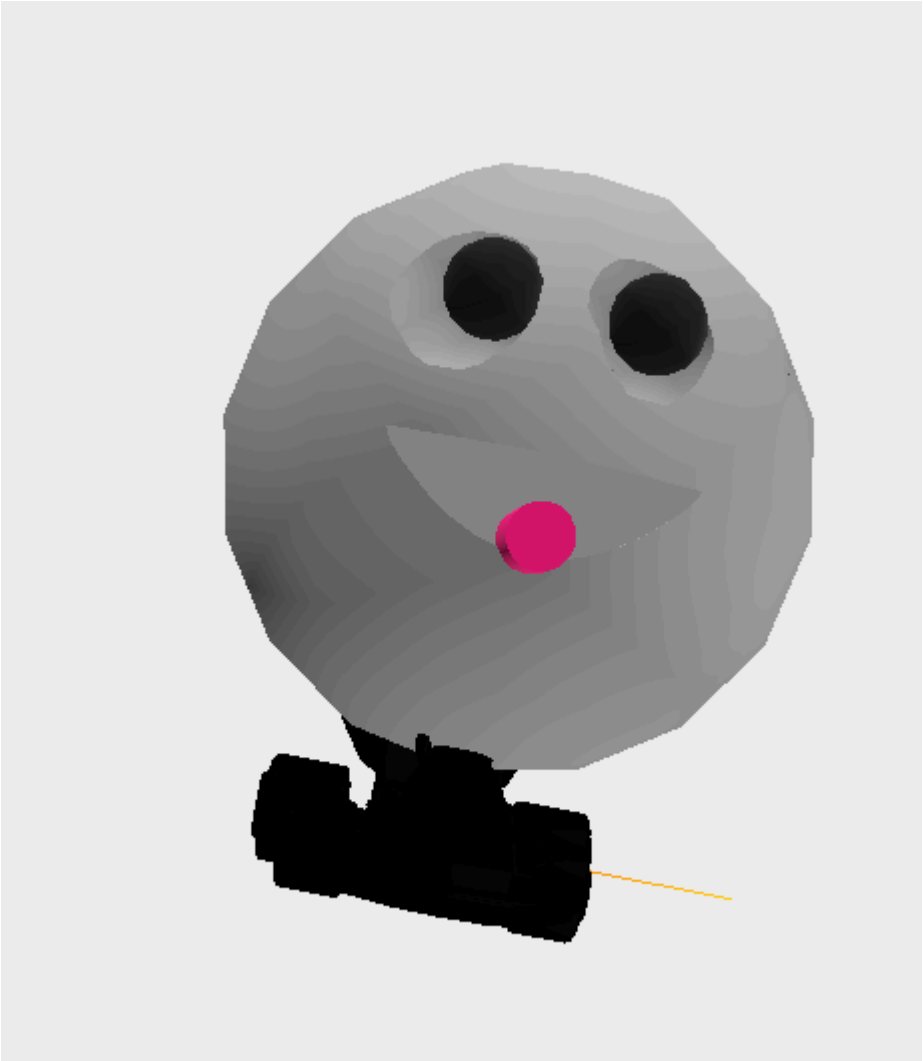
El impacto que tiene en el juego es negativo ya que si el personaje entra en contacto con este objeto sufrirá una penalización permanente en su velocidad.



Personaje

El objeto Personaje es el principal de todo el juego es a través del cual se van a realizar todas las interacciones tanto las de colisión, como el picking, poder moverse de un lado a otro, pero esto vamos a verlo parte por parte:

Su creación ha sido simplemente la combinación de Coche y Boo, no hay nada nuevo en eso.



Detectar Colisiones

```
function detectarColisiones(personaje, sceneObjects,sceneObjects2,scene) {
```

Variables Globales

`objetosChocadosDuranteAtravieso` es una variable global que almacena los objetos con los que el personaje ha colisionado y todavía está intersectando.

Función `detectarColisiones`

La función `detectarColisiones` toma los siguientes parámetros:

- `personaje`: El objeto del personaje principal.
- `sceneObjects`: Array de objetos en la escena con los que se detectan colisiones positivas.
- `sceneObjects2`: Array de objetos en la escena con los que se detectan colisiones negativas.

- `scene`: La escena completa, utilizada para acceder a métodos del personaje.

Crear Caja Envolvente del Personaje

Se crea una caja envolvente (`BoundingBox`) alrededor del personaje para detectar colisiones.

Detección de Colisiones con `sceneObjects`

Para cada objeto en `sceneObjects`, se verifica si no está en `objetosChocadosDuranteAtravieso`. Si no lo está, se crea una caja envolvente para el objeto y se verifica la intersección con la caja envolvente del personaje. Si hay intersección:

- Se registra la colisión.
- Se llama al método `colision_positiva` del personaje.
- Se añade el objeto a `objetosChocadosDuranteAtravieso`.

Detección de Colisiones con `sceneObjects2`

La lógica es similar a la anterior, pero para `sceneObjects2`. Se llama al método `colision_negativa` del personaje en caso de colisión.

Actualización de la Lista de Objetos Chocados

Se verifica si el personaje ha salido de la intersección con los objetos en `objetosChocadosDuranteAtravieso`. Si ya no hay intersección:

- El objeto se elimina de la lista.
- Se ajusta el índice del bucle para no saltar elementos después de eliminar uno.

Picking

```
function onDocumentMouseDown(event, personaje, pickableObjects, camera, raycaster, scene) {
```

Descripción del Método

Parámetros de la Función

La función `onDocumentMouseDown` toma los siguientes parámetros:

- `event`: El evento del mouse que contiene información sobre la posición del clic.
- `personaje`: El objeto del personaje principal (no utilizado directamente en esta función).
- `pickableObjects`: Array de objetos en la escena que pueden ser seleccionados mediante clics del mouse.

- `camera`: La cámara utilizada para renderizar la escena.
- `raycaster`: El objeto `Raycaster` de `Three.js` utilizado para detectar intersecciones entre rayos y objetos en la escena.
- `scene`: La escena completa, utilizada para acceder y modificar su estado.

Calcular las Coordenadas Normalizadas del Mouse

Se crea una variable `mouse` de tipo `THREE.Vector2` y se calculan las coordenadas normalizadas del mouse basándose en la posición del clic y las dimensiones de la ventana.

Actualizar el `Raycaster`

El `raycaster` se actualiza con la posición del mouse y la cámara para proyectar un rayo desde la cámara hacia la dirección del clic.

Encontrar Intersecciones con Objetos Clicables

Se utilizan las coordenadas del mouse para encontrar intersecciones con los objetos en `pickableObjects` mediante `raycaster.intersectObjects`, que devuelve una lista de objetos intersecados.

Verificar Intersecciones y Realizar Acciones

Si se encuentran intersecciones (`pickedObjects` no está vacío):

- Se accede al primer objeto intersecado, que es el más cercano a la cámara.
- Se obtiene el punto de intersección en coordenadas del mundo.
- Se cambia el color del objeto intersecado a rojo para indicar la selección.
- Se ajusta la intensidad de la luz del sol en la escena a 0.
- Se incrementa el puntaje (`score`) de la escena y se actualiza el texto en la pantalla para reflejar el nuevo puntaje.

Restaurar el Color Original

Se establece un temporizador para restaurar el color original del objeto y la intensidad de la luz del sol después de 1 segundo.

Movimiento Lateral

El movimiento lateral se ha realizado utilizando dos listener uno para la tecla “a” y otro para la tecla “d”:

```
// Listener para la tecla "a"
$(window).on("keydown", (event) => {
  if (event.key === "a") {
    this.personaje.rotateCar('left');
  }
});

// Listener para la tecla "d"
$(window).on("keydown", (event) => {
  if (event.key === "d") {
    this.personaje.rotateCar('right');
  }
});
```

Simplemente implementamos esa rotación en el nodo correspondiente el cual es movimientoLateral:

```
rotateCar(direction) {
  // Ajusta la rotación del coche según la direc
  if (direction === 'left') {
    this.movimientoLateral.rotation.z += 0.0001;
  } else if (direction === 'right') {
    this.movimientoLateral.rotation.z -= 0.0001;
  }
}
```

Estos son los métodos que se aplican sobre todo al personaje pero también al resto de objetos que producen incrementos o decrementos en la velocidad de este, así como movimiento por la superficie del circuito y el aumento de la puntuación a través del picking con el objeto volador.

Seta

El último objeto del juego es una seta que quiere imitar una seta de Mario Kart:

Seta se ha creado a traves de un shape utilizando varios quadraticCurvesTo para generar una línea de punto a la cual después le aplicamos revolución.
El impacto que tiene en el juego es que tras su colisión con este objeto el personaje recibirá un incremento de la velocidad de manera permanente.