SISTEMAS GRÁFICOS

PRÁCTICA 2: DISEÑO E IMPLEMENTACIÓN DE UN JUEGO BASADO EN *"MARIO KART"*

Jose Francisco López Rubio
Jorge Navarro Molina

02/05/2024

UNIVERSIDAD
DE GRANADA

Tabla de contenidos

1. Descripción del juego

En esta segunda práctica hemos desarrollado un juego basado en el *Mario Kart*, pero de una manera distinta. Lo hemos realizado con la herramiento *Three.js* para realizar todo el modelaje, la creación de luces y la aplicación de texturas y materiales. El código que dirige la funcionalidad del juego lo hemos realizado en *JavaScript*, y la interfaz con un simple fichero *HTML*.

En nuestra implementación, hemos seguido un sistema de 3 corazones (6 golpes, que se pueden ir reduciendo al obtener efectos específicos o recibir colisiones con obstáculos perjudiciales. Una vez el jugador llegue a 0 corazones, el juego acabará.

Habrá enemigos y obstáculos repartidos por todo el circuito, tanto sobre este como alrededor volando, y los voladores tendrán distintas velocidades. Estos enemigos, al derrotarse, otorgarán puntos y ciertas ventajas o desventajas. Habrá enemigos u objetos que otorguen más o menos puntos, y, de la misma forma, se perderán puntos si el personaje principal es alcanzado por algún obstáculo negativo.

2. Manual de usuario

Para comenzar, hay dos tipos de obstáculos que el jugador se va a encontrar a lo largo del circuito: los perjudiciales y los beneficiosos, otorgando efectos negativos o positivos en función de cual obtenga. Además de los objetos terrestres, el jugador se encontrará con enemigos voladores a los que deberá disparar para conseguir puntos.

2.1. Obstáculos terrestres perjudiciales

Estos obstáculos, al colisionar con ellos, otorgarán un efecto negativo al personaje principal. Los podemos diferenciar en dos de ellos:

1. **Planchas y bolas de pinchos**: estos dos tipos de obstáculos le quitarán medio corazón (una vida) al jugador, y además no desaparecerán del circuito incluso si se ha colisionado con estos.
2. **Bombas**: en caso de colisionar con estas, el personaje será ralentizado y se le apagará la luz solidaria a este durante 5 segundos. Estas si se eliminarán (explotarán) al ser colisionadas.

2.2. Objetos terrestres beneficiosos

Los objetos beneficiosos, al cogerlos, otorgarán un efecto positivo al personaje principal y desaparecerán tras su colisión con ellos, indicando que el efecto se ha dado y que el objeto se ha obtenido. Estos son:

1. **Escudo**: Otroga una cupula protectora al jugador, la cual lo hace invencible durante 20 segundos. Aunque inmune a todo daño, el personaje si seguirá sufriendo efectos de los enigmas y la ralentización de las bombas.
2. **Venus Atrapamoscas**: Hace que todos los enemigos voladores alcanzados puedan matarse de un solo click durante 30 segundos.
3. **Nitro**: Otorga un multiplicador x2 al jugador durante 30 segundos, que multiplicará la puntuación obtenida desde ese punto en adelante.
4. **Enigma**: El efecto de este objeto es variable, puede ser uno beneficioso o perjudicial. Se encuentra explicado en la sección 2.4.

2.3. Enemigos voladores

Este tipo de obstáculo las hemos representado mediante las **moscas**. Éstas estarán esparcidas por el circuito a una altura más elevada y con un movimiento alrededor del tubo. Se eliminarán haciendo click sobre ellas, y, en función de su tipo, tendrán más puntos de vida, velocidad u otorgarán un efecto u otro. Todas otorgarán más o menos puntos al ser eliminadas, desapareciendo de la escena una vez sus puntos de vida lleguen a 0. Los diferentes tipos son:

1. **Mosca corriente**: el tipo de mosca más básica. Tiene un único punto de vida (solo requiere un click sobre ella para morir) y proporciona solo 1 punto al personaje.
2. **Mosca Reina**: tiene 3 puntos de vida, y otorga 10 puntos al jugador una vez la mata.
3. **Mosca Agresiva**: tiene 2 puntos de vida, otorga 5 puntos al jugador una vez la mata, y esta se mueve mucho más rápido.
4. **Mosca de Luz**: tiene 2 puntos de vida, y otorga medio corazón más al jugador.
5. **Mosca Enigma**: tiene 1 único punto de vida, y otorga un efecto aleatorio de Enigma, explicado en la sección 2.4.

Cabe aclarar que la velocidad que la mosca tendrá para moverse alrededor del tubo es un valor aleatorio. Para el caso de la mosca reina el rango será menor (simulando que es la que más lento puede ir), para la agresiva será mayor (simulando ser de las más rápidas) y para el resto compartirán el mismo rango. Esto se ha hecho así para dar más variedad al juego, pues no todas las moscas tendrán la misma velocidad de rotación.

2.4. Enigmas

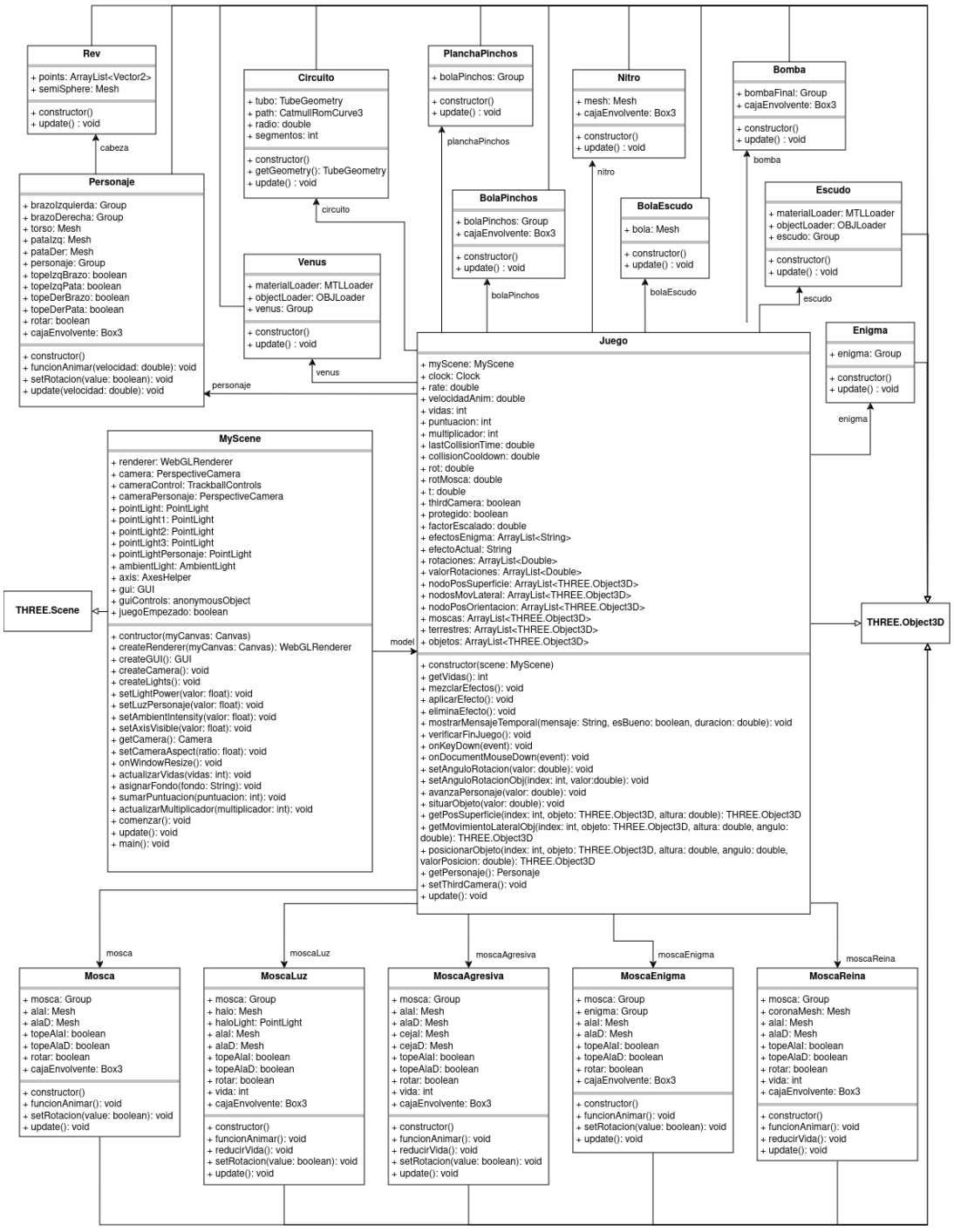
Hemos dejado para el final los enigmas, ya que, dependiendo de sus efectos, estos pueden ser considerados tanto beneficiosos como perjudiciales. Estos son los efectos posibles, tanto de los enigmas terrestres, como de las moscas Enigma:

1. **Efectos beneficiosos**:
   a) **Multiplicador x2**: Otorga un multiplicador al jugador que multiplicará toda la puntuación obtenida desde ese punto durante 5 segundos. Este multiplicador podrá acumularse en caso de haber cogido tambiñén una Venus Atrapamoscas.
   b) **Cura**: Otorga medio corazón más al jugador (una vida más).
   c) **Lentitud**: Ralentiza al jugador durante 5 segundos, para que este pueda esquivar y moverse con mas manejo.
2. **Efectos perjudiciales**:
   a) **Controles inversos**: Invierte las teclas de movimiento del jugador durante 5 segundos.
   b) **Daño**: Arrebata medio corazón al jugador (una vida menos).
   c) **Velocidad**: Aumenta la velocidad del jugador durante 5 segundos, haciendo el movimiento más dificil y menos predecible.

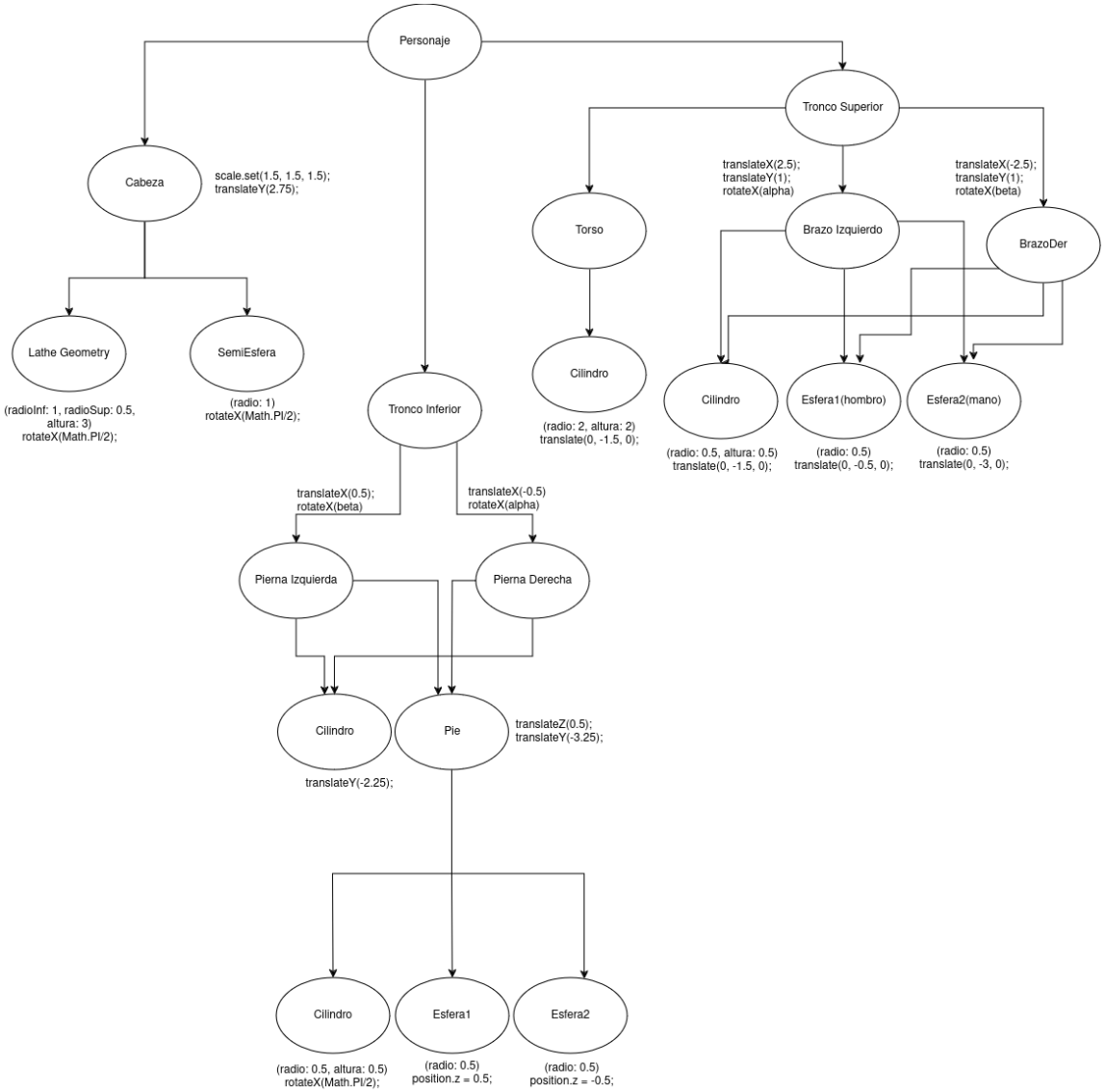
Claramente se puede ver nuestra intención al añadir este tipo de obstaculos: incluir una componente aleatoria en la partida, donde el jugador puede discutir sobre si arriesgar o no su partida para obtener más ventajas (por ello las moscas Enigma tienen un único punto de vida, para que el jugador no se lo pueda pensar dos veces).

3. Diseño de la aplicación.

3.1. Diagrama de clases.



3.2. Modelos jerárquicos.



3.3. Descripción de algoritmos.

3.3.1. Colisiones con *ray-casting*

Este ha sido el algoritmo que hemos utilizado para registrar las colisiones tanto del personaje con los obstáculos terrestres, como los proyectiles que este dispara a los obstáculos voladores. En el caso de las colisiones terrestres, hemos inicializado un rayo en el constructor de la siguiente manera:

```
this.rayo = new THREE.Raycaster( new THREE.Vector3(),
new THREE.Vector3(0,0,1), 0, 500);

this.personaje.getWorldPosition(posicion);
this.rayo.set(posicion, direccion.normalize());
```

Figura 3.3.1: Inicialización del rayo para la intersección de objetos terrestres.

Y una vez hecho esto, en el update obtenemos cada vez los objetos que se ha encontrado el rayo y en caso de existir, obtenemos el primer objeto intersectado del array. Dependiendo del tipo de objeto (su clase), se harán distintas acciones y se le aplicarán distintos efectos al personaje. Hemos añadido una linea de código que establece un *cooldown* tras haber colisionado con un objeto, para que el manejo de colisiones sea un poco menos caótico. Además, al haber creado la mayoría de los objetos como *Group*, tenemos que acceder al .ªbuelo"del objeto (es decir, el padre del padre, lo más alto de la jerarquía) para eliminar los objetos completos, "de raiz":

```
if(currentTime - this.lastCollisionTime > this.collisionCooldown){
      var impactados=this.rayo.intersectObjects(this.terrestres, true);
      if(impactados.length > 0){
        this.lastCollisionTime = currentTime;
        var padre = impactados[0].object.parent;
        var abuelo = padre.parent;
        if((abuelo instanceof BolaPinchos) ||
        (abuelo instanceof PlanchaPinchos) ||
        (abuelo instanceof Bomba)){
          if (!this.protegido){
            this.vidas--;
            this.myScene.actualizarVidas(this.vidas);
          }
        }
```

Figura 3.3.2: En caso de ser un obstáculo perjudicial, se resta una vida.

En el caso de las colisiones con los obstáculos voladores hemos hecho algo bastante parecido, solo que es mas parecido al algoritmo *Pick*. Hemos obtenido la posición actual del ratón (coordenadas x e y), y a través de dicha psoición hemos creado un rayo que se lanze desde la posición de la cámara del personaje, por ese punto y en la dirección en la que el personaje esta mirando. Lo demás es exactamente igual que el algoritmo de detección de colisiones terrestres, solo que aqui no se implementa un *cooldown*, ya que es mucho más manejable. Todo esto se ejecutará cada vez que se dé el evento de click del ratón. Según el objeto colisionado, se darán los distintos puntos y, en caso de haberlos, los efectos correspondientes al jugador.

```
onDocumentMouseDown(event){
    //Creamos el mouse y el rayo para el picking.
    this.mouse = new THREE.Vector2();
    this.rayoPick = new THREE.Raycaster();

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

    this.rayoPick.setFromCamera(this.mouse, this.myScene.getCamera());

    var pickedObjects=this.rayoPick.intersectObjects(this.moscas, true);

    if(pickedObjects.length > 0){
      var selectedObject = pickedObjects[0].object;

      var padre = pickedObjects[0].object.parent;
      var abuelo = padre.parent;
                              .
                              .
                              .
      else if(abuelo instanceof MoscaEnigma){ //1 vida
        this.efectoActual = this.efectosEnigma[0];
        this.aplicaEfecto();
        console.log("Efecto actual: " + this.efectoActual);
        this.mezclarEfectos();
        abuelo.remove(padre);
      }
```

Figura 3.3.3: En caso de ser una mosca Enigma, se aplicará un efecto aleatorio.

3.3.2. Posición, movimiento lateral y avance del personaje.

Para poder posicionar bien el personaje sobre el tubo, y que este se mueva tanto lateralmente como hacia delante encima de este, hemos creado varios nodos los cuales se añaden unos a otros. Primero creamos un nodo *posSuperficie*, al cual le añadimos el objeto Personaje, y una vez añadido, lo posicionamos en el eje Y según el radio del tubo Circcuito y la latura del objeto Personaje. Una vez hecho esto, pasamos a crear el siguiente nodo, *movimientoLateral*, al cual le añadimos el anterior nodo creado, *posSuperficie*, y establecemos el angulo de rotación de este nodo, el cual se actualizará cada vez que se pulsen las teclas de movimiento del personaje. Por último, creamos un último nodo, *nodoPosOrientTubo*, como siempre añadimos el nodo anterior, ejecutamos la función *avanzaPersonaje*, al cual le pasamos como parámetro *t*, que controla la posición del personaje conforme avanza. Esta función obtiene la posición en la que se esta en el circuito actualmente, se la atribuye al nodo *nodoPosOrientTubo*, se obtiene la tangente en esa posición, se añade a la posición obtenida anteriormente, y finalmente se calcula la normal del segmento actual del tubo para orientar bien al personaje. Y finalmente se añade este nodo que incluye a todos los anteriores a la escena, mostrando bien posicionado al personaje en cada frame que avanza.

4. Objetos importados.

Solo hemos importado dos objetos de todos los del juego:

1. **Escudo**: Enlace a escudo
2. **Venus**: Enlace a venus