Memoria Práctica 2 SG

Raúl Durán Racero

31 Mayo 2022

1 Introducción

En esta práctica habrá que aplicar todo lo aprendido en clase para desarrollar un sistema gráfico usando Javascript con la biblioteca Three.js. En mi caso, he elegido desarrollar un pequeño videojuego, cumpliendo los requisitos mínimos indicados en el guión de la práctica.

2 Descripción

La aplicación consiste en un videojuego donde el personaje que controlamos, el Bimbot, tendrá que esquivar distintas trampas para alcanzar un objetivo, y luego volver sobre sus pasos para completar el nivel. La cámara seguirá en todo momento al Bimbot, orientada siempre hacia la misma dirección, alterando sólo su posición en los ejes "X" y "Z".

3 Diseño de la aplicación

La aplicación está estructurada en: una carpeta con los scripts que crean los distintos modelos (MisModelos), y, además de las carpetas de las librerías e imágenes, una carpeta (BimbotRush) donde se usan estos modelos para crear el juego. En esta carpeta se encuentra el script para crear el personaje principal, **Bimbot.js**, el script **Objetos** que crea y coloca todos los modelos de la carpeta MisModelos, el script principal, **main.js**, donde se añaden los objetos y el Bimbot y se dan todas las funcionalidades de colisiones, movimiento, animación, gestión de la vida y coleccionables del robot y los checkpoints y final o reinicio del juego. Por último, tenemos el script de la escena **MyScene.js**, donde se añade todo lo creado en el main.js y se añade el suelo, las luces, el renderer, los límites del mapa, y se obtiene la cámara, la cual no se crea en la escena, sino que es hija del Bimbot, para que así pueda seguir al personaje por el mapa. Mencionar que para las luces he usado una **HemisphereLight** y 2 **SpotLights**, una principal que ilumina toda la escena, y otra roja para iluminar la zona final del nivel. Para lanzar las sombras, he tenido que activar dicha propiedad tanto en la luz, el renderer, el suelo y en el resto de los objetos.

3.1 Diagramas de clases

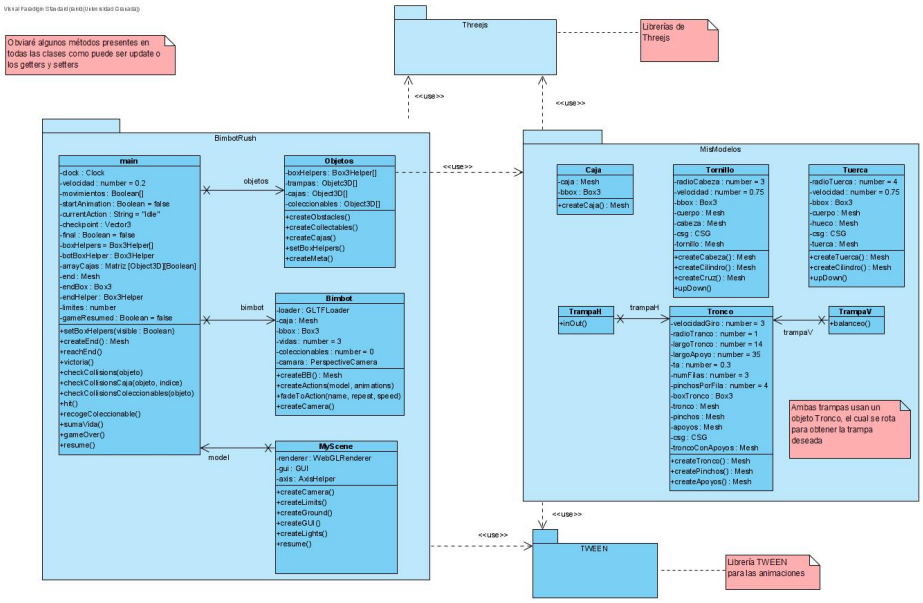

Figure 1: Diagrama de clases

3.2 Aspectos más importantes

Los aspectos que considero más importantes y que más quiero destacar son:

1. **Colisiones:** Para que cada objeto, tanto los coleccionables, cajas, trampas y el Bimbot, pudieran interactuar entre sí, era necesario añadirles alguna forma de detectar colisiones. Una opción inicial era utilizar el **Raycasting**, pero al poder recibir colisión desde varias direcciones distintas, decidí que habría una mejor forma. Intenté utilizar el motor de físicas visto en teoría, pero tras un tiempo trabajando en ello resultó que el modelo GLTF no era compatible con este motor de físicas. Así que opté finalmente por usar las BoundingBoxes que ofrece Threejs (Box3). Para trabajar mejor con estas cajas, me serví de las Box3Helpers, las cuales permiten visualizar cada bounding box fácilmente.
   Para crear estas cajas, en cada modelo creo un Box3, le digo al mesh final que se va a utilizar que calcule la Bounding Box de su geometría, y actualizo al Box3 en el update del objeto:

```
1     //En el contructor:
2     this.bbox = new THREE.Box3();
3     mesh.geometry.computeBoundingBox();
4     //En el update del objeto:
5     this.bbox.copy(this.tornillo.geometry.boundingBox)
```

```
6       .applyMatrix4(this.tornillo.matrixWorld);
```

Luego, guardo todos estos objetos por su tipo en 3 arrays: trampas, coleccionables y cajas, que luego utilizo en el main.js para detectar las colisiones con la BoundingBox del Bimbot gracias a la función de Box3 *intersectsBox(box3 : Box3)*, la cual devuelve *true* o *false*, dependiendo si la caja pasada como argumento choca o no con esta caja.
En cuanto a las BoxHelpers, creo todas las de los objetos en la clase Objetos, donde las guardo en un array. Luego en el main, creo las BoxHelpers del Bimbot y de los checkpoints, para que se vean correctamente. Gracias a utilizar un array, puedo cambiar fácilmente desde el main.js la visibilidad de todas las cajas con mi función *setBoxHelpers(bool : Boolean)*.

2. **Movimiento y Animación:** El movimiento y animación de las trampas y coleccionables ha sido sencillo, utilizando lo ya visto en la primera práctica. La dificultad la encontré con el modelo GLTF, ya que logré rápidamente que se moviera y girara en 8 direcciones, usando los Listeners y comprobando las teclas de movimiento (WASD) para girar o desplazar al robot. Pero el problema vino con la animación, ya que no lograba hacer que se repitiera correctamente la animación de correr. Tras un tiempo, di con una solución, la cual consiste en tener un array de booleanos, uno por cada tecla, para comprobar cuáles se están presionando. Este array sirve para organizar mejor las entradas por teclado. Para la animación, declaré 2 variables:

   - **startAnimation**: Booleano que marca si una animación ha empezado o no.
   - **currentAction**: String que contiene el nombre de la animación que está ocurriendo actualmente.

   Con estas variables, en el update del main.js, compruebo si, cuando se aprieta una tecla, tiene que empezar la animación de correr o no, para así evitar que la animación esté continuamente empezando.
   Más tarde añadí otro evento, el click izquierdo del ratón para dar un puñetazo. Con la base de animación correcta, simplemente tuve que añadir un bool al array, y comprobar que no se repita la animación *Punch* del modelo.

3. **Diseño de nivel:** En cuanto al diseño del nivel, mi base era partir de la idea original del videojuego **Crash Bandicoot**: un nivel estrecho y largo, con varios obstáculos entre el punto de salida y el objetivo, y algunos coleccionables para recoger.
   Por ello, he diseñado un suelo con un tamaño en el eje X reducido, pero largo en el eje Z. Luego, hago un reparto de los distintos objetos por todo el recorrido, el cual viene limitado artificialmente por unas variables que impiden al Bimbot pasar de ellas. El fondo es un fondo falso, ya

que se tratan de varios Boxes cuyo material es el *MeshToonMaterial*, que proporciona un color bastante particular. He decidido usar esto en lugar de un skyBox porque así oculto al jugador cosas como los apoyos de las trampas de pinchos horizontales, que sobresalían bastante del suelo, lo cual me parece pobre en cuanto a diseño visual se refiere. Luego todo lo relacionado con las vidas del robot, los coleccionables y los checkpoints son funciones simples, cambiando los atributos del objeto Bimbot o variables de la clase principal.

3.3 Referencias

El modelo del personaje principal, el **Bimbot**, lo he obtenido de la librería de ejemplos de Threejs.org:
https://threejs.org/examples/#webgl_animation_skinning_morph.

3.4 Manual de usuario

El objetivo es recoger la **Gran Tuerca**, la cual está al final del recorrido. Una vez recogida, habrá que volver a la línea de meta de la que hemos partido para ganar el juego. Los controles del Bimbot son los siguientes:

1. W - Movimiento hacia delante.
2. A - Movimiento hacia la izquierda.
3. S - Movimiento hacia detrás.
4. D - Movimiento hacia la derecha.
5. Click izquierdo - Puñetazo.

Nuestro Bimbot comienza con 3 vidas y ningún coleccionable (ambos datos se muestran en la interfaz). Cuando el Bimbot colisiona con una trampa, perderá una vida y volverá al último **checkpoint** (punto de guardado de su posición) alcanzado. Hay 2 checkpoints: el punto de salida, y el punto donde está la Gran Tuerca. El jugador puede consumir un coleccionable (tuerca o tornillo) para sumar una vida al Bimbot haciendo click izquierdo sobre el botón **USAR** de la interfaz. En caso de que el Bimbot se quede sin vidas, terminará el juego, mostrando un mensaje de **"Game Over"** con la opción de reiniciar el juego.

Además, si encuentra cerca de una caja, el jugador puede romper dicha caja con el botón izquierdo. No importa hacia dónde esté mirando, si está chocando con la caja y pulsa el click izquierdo, la destruirá. Adicionalmente, en la GUI se puede tanto cambiar la intensidad de la luz principal, activar y desactivar los ejes, y otro botón dentro de la carpeta BIMBOT de la GUI para activar y desactivar las Box Helpers de todos los objetos.