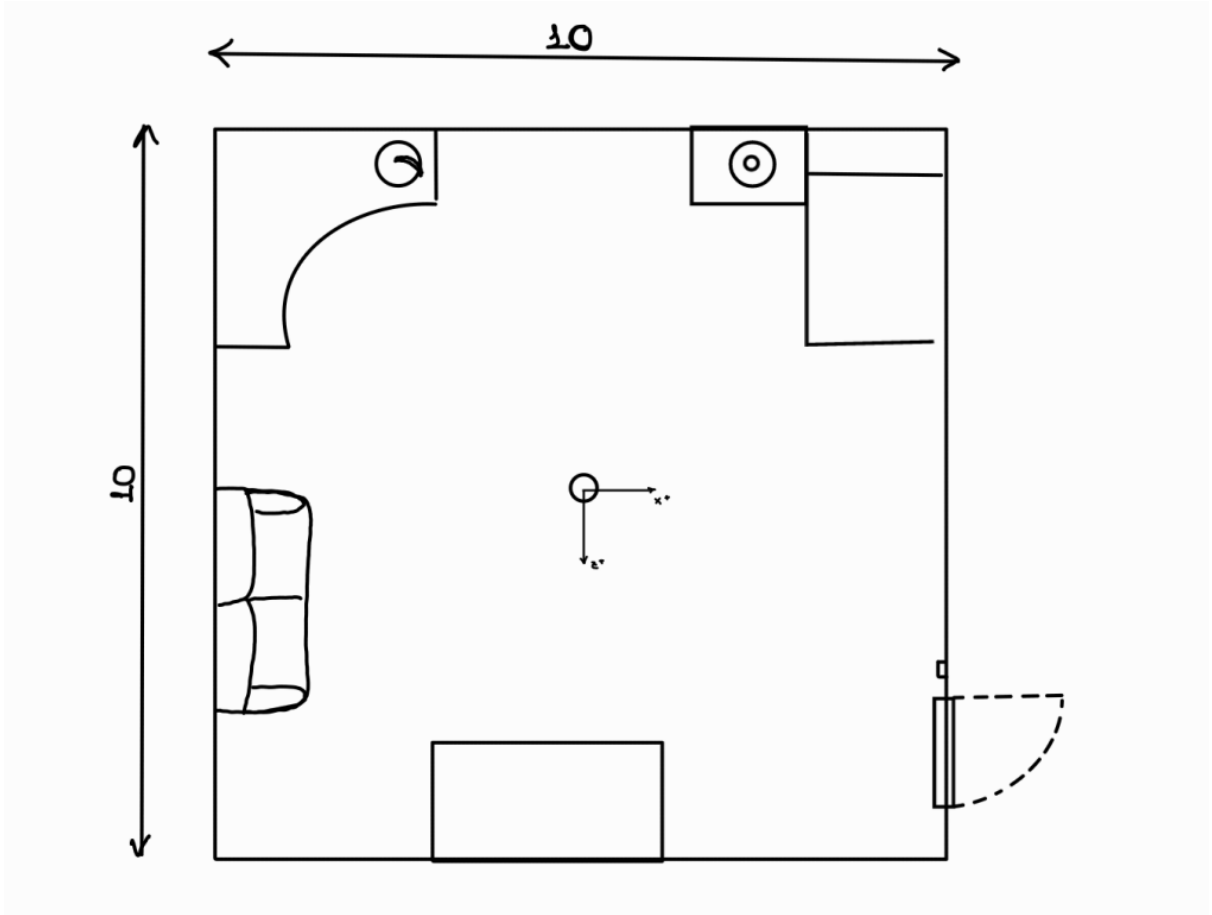
Memoria Práctica 2

1. Descripción del juego

La descripción de nuestro *escape room* no ha diferido mucho con respecto al original. Durante las diferentes defensas con el profesor, hemos ido puliendo ciertos detalles acerca de texturas, luces, interacciones, etc. pero nada que afectase directamente al modelo planteado inicialmente en la descripción original.

Los objetos han sido los mismos que se especificaron en el documento original: el armario, la cama, la mesita de noche, el sofá, la puerta, el globo terráqueo… El único que no hemos incluido en comparación con la propuesta original ha sido el flexo del escritorio; lo hemos cambiado por una lámpara en la mesita de noche.

La disposición de los diferentes muebles sin embargo sí que difiere del boceto original. Esto se debe a que, al juntar todos los objetos y posicionarlos en el cuarto, no quedamos completamente con la disposición original y decidimos reposicionar algunos muebles. La disposición resultante de nuestro cuarto es el siguiente:



Cabe comentar que, como se comentó en la última defensa, la lámpara que se encuentra sobre la mesita de noche en ocasiones aparece cortada según la posición y dirección de la cámara.

Con respecto a los requisitos mínimos requeridos para la práctica, a continuación listamos todos ellos (sacados del guión de la práctica 2) y dónde se pueden encontrar en nuestro cuarto:

☑ *“El jugador deberá cumplir al menos 3 condiciones para poder salir de la habitación.”*

El jugador debe:

1. Encender la luz del cuarto para poder interactuar con los objetos.
2. Encontrar la llave para abrir el cajón.
3. Coger la llave para abrir la puerta.

☑ *“Tendrá al menos 1 habitación cerrada: 4 paredes, techo y suelo. Con 1 puerta (con pomo) en una de las paredes que se abrirá para poder salir cuando se haya resuelto el juego.*”

☑ *“Diversos objetos 3D que se hayan modelado usando diferentes técnicas: extrusiones, CSG, modelos cargados desde disco, etc.”*

Se han utilizado diversas técnicas para modelar los objetos, como por ejemplo:

- CSG para el armario
- Modelo cargado de disco para el sofá
- Extrusión junto con geometría 2D para una parte del modelo jerárquico
- Revolución para la lámpara de la habitación
- Geometrías básicas como THREE.BoxGeometry (para las paredes) o THREE.SphereGeometry (para la bola terráquea).

☑ *“Al menos 1 objeto articulado (debe incluirse su modelo jerárquico en la documentación)”*

El objeto articulado que está continuamente animado es la bola del mundo (WorldBall) que se encuentra en el escritorio. Su modelo jerárquico se muestra más adelante en este documento.

☑ *“Alguna parte del modelo jerárquico debe tener un movimiento continuo.”*

La parte que tiene un movimiento continuo es la esfera que tiene el mapa del mundo.

☑ *“La puerta de salida se abrirá cuando, tras reunir las condiciones para poder salir, se haga clic con el ratón en el pomo de la puerta. La apertura se hará con una animación que debe durar exactamente 2 segundos.”*

Esto es así en nuestro juego; lo hemos conseguido controlando la animación con TWEEN y añadiendo únicamente el pomo de la puerta al array de objetos con los que se puede interactuar.

☑ *“El jugador se moverá por la habitación con las teclas del cursor.”*

Sí, el jugador puede moverse con las flechas del teclado, y girar la cámara bloqueando el puntero con la tecla de control y moviendo el ratón.

☑ *“La selección de objetos de la habitación (coger cosas, abrir cajones, encender luces, etc.) se hará haciendo clic con el ratón en dichos objetos.”*

Sí, se realiza el pick con click izquierdo sobre el objeto que se desee seleccionar (siempre y cuando este se pueda seleccionar, como por ejemplo el pomo de la puerta, las puertas del armario, los cajones de la mesita de noche, el interruptor, etc.).

☑ *“Para abrir la puerta y finalizar el juego se debe hacer clic sobre el pomo de la puerta (no siendo válido hacer clic en cualquier otro sitio de la puerta).”*

☑ *“Debe haber materiales basados en un color.”*

Sí, estos materiales se pueden observar por ejemplo en las sábanas de la cama, el pomo de la puerta, las llaves o el sofá.

☑ *“Debe haber materiales basados en texturas para el canal difuso.”*

Los materiales de este tipo se pueden encontrar en el suelo, el techo, los pósteres, el mapa del mundo y más objetos.

☑ *“Debe haber al menos un material con una textura en el canal de relieve.”*

Estos materiales se pueden encontrar en las paredes principalmente, aunque también en algunos muebles de madera como la puerta, la mesita de noche o el armario.

☑ *“Debe haber luces de diferentes colores.”*

En nuestro juego hay tres luces: una luz ambiental, la luz del cuarto la cual es un THREE.PointLight de color blanco, y la luz de la mesita de noche que es un THREE.PointLight de color rojo (con menos intensidad que la luz del cuarto).

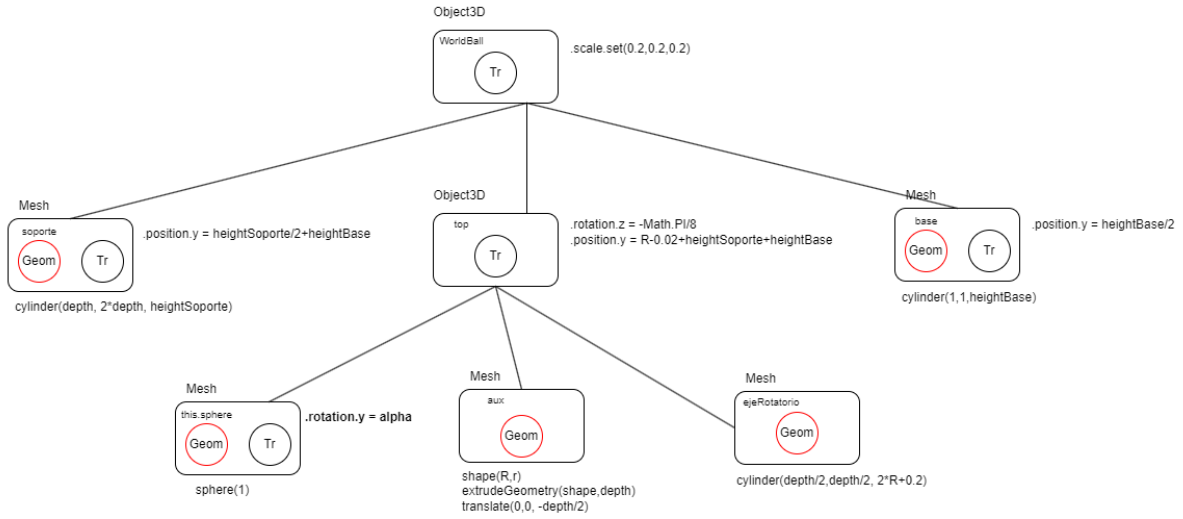
☑ *“Debe haber al menos una luz que cambie en el juego (por ejemplo, que se vaya apagando sola, que se encienda al clicar un interruptor, etc.)”*

Dos de las tres luces cambian en el juego. Al interactuar con la lámpara se enciende/apaga la luz roja, y al interactuar con el interruptor se enciende/apaga la luz blanca que ilumina todo el cuarto.

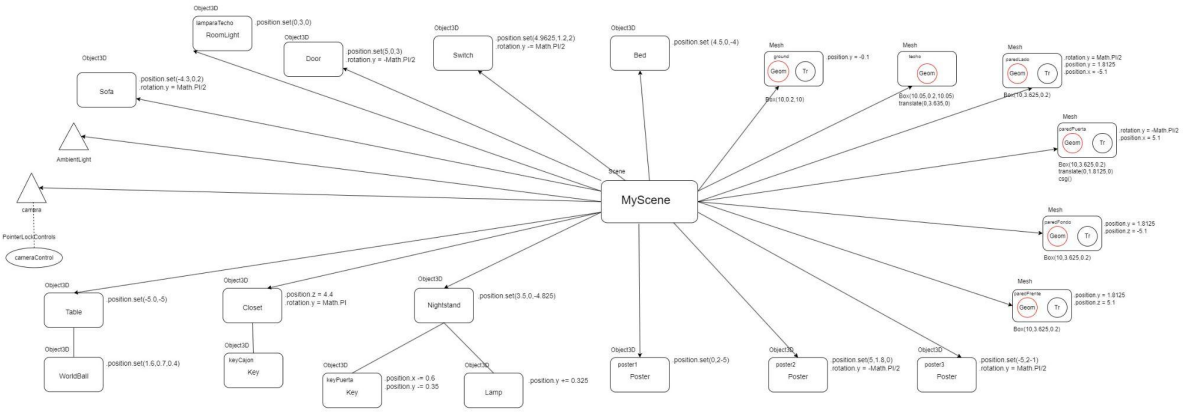
☑ *“El juego es en primera persona, la cámara serán los ojos del personaje protagonista del juego.”*

2. Diseño de la aplicación

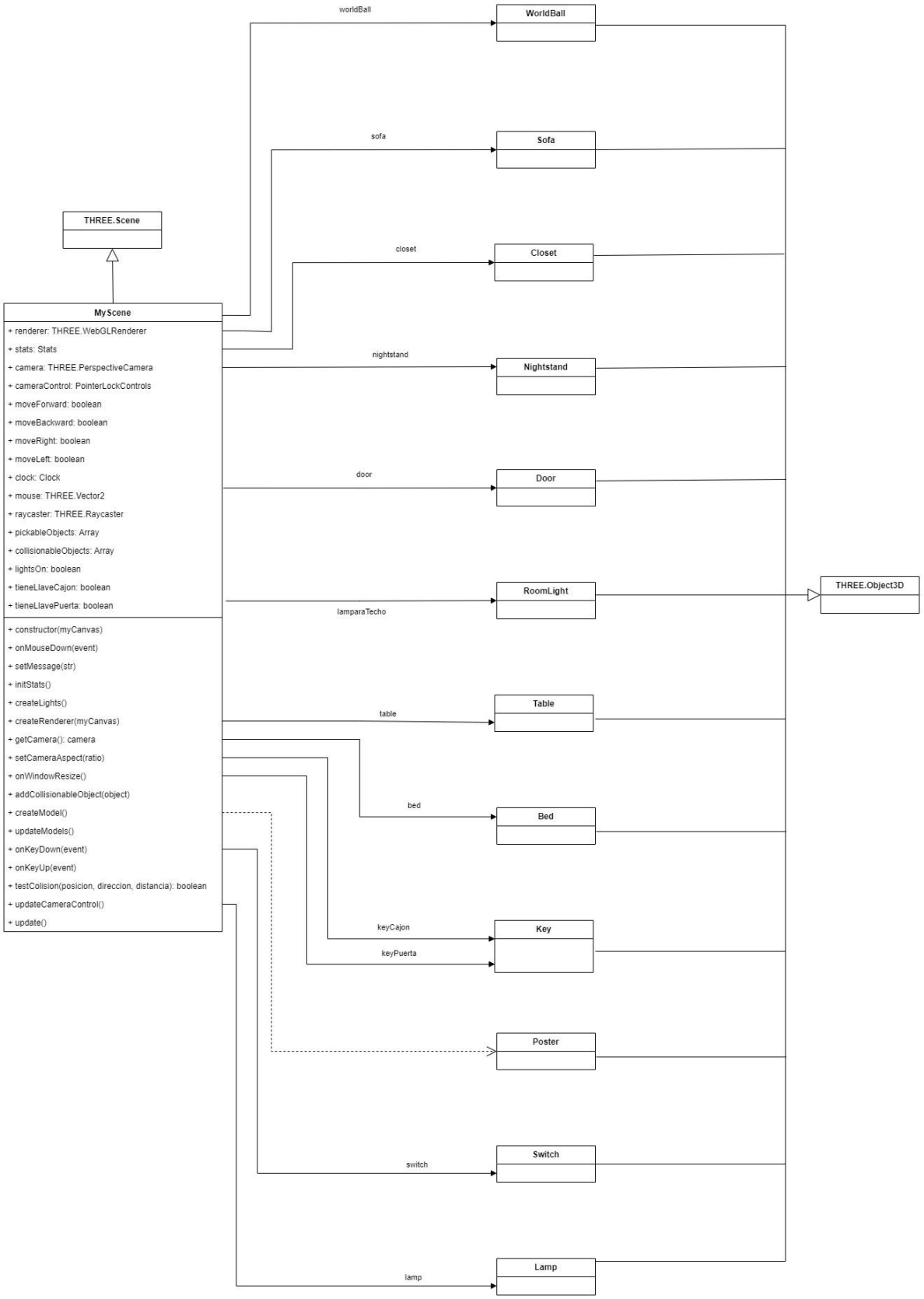
Aquí se incluye el modelo jerárquico del objeto animado que en el caso de nuestra práctica es la bola del mundo.



Aquí encontramos el grafo de escena de la práctica



Y por último el diagrama de clases incluyendo la escena y todos los objetos



En la entrega de la práctica, junto con este documento, hemos incluido las imágenes para poder observarlas con una mayor calidad y resolución.

A continuación explicaremos los algoritmos usados en nuestra práctica.

“Picking”

El primero será el algoritmo del método “onDocumentMouseDown()” implementa la funcionalidad de selección de objetos en la escena utilizando la técnica de “pick”.

En general, el proceso de selección de objetos en una escena 3D implica lanzar un rayo desde la cámara hacia el escenario y determinar qué objetos, si los hay, se intersectan con ese rayo. Esto se logra mediante el uso de la clase “Raycaster”. El método “onDocumentMouseDown()” se activa cuando el usuario hace clic con el botón izquierdo del ratón en la ventana del navegador. A continuación, vamos se describe paso por paso el algoritmo:

1. Se verifica si el clic del ratón fue con el botón izquierdo (“event.button === 0”).
2. Se obtienen las coordenadas normalizadas del clic del ratón en relación con la ventana del navegador. Estas coordenadas se utilizan para calcular la dirección del rayo que se lanzará desde la cámara.
3. Se actualiza el rayo con la posición y la dirección basadas en las coordenadas normalizadas del clic del ratón.
4. Se realiza la intersección entre el rayo y los objetos de la escena que son seleccionables (“pickableObjects”). Esto se hace mediante el método “intersectObjects()” de la clase “Raycaster”. El resultado es un array de objetos que han sido alcanzados por el rayo, ordenados por distancia.
5. Si se enciende la luz de la mesita de noche, muestra un mensaje por pantalla y enciende la luz de la misma, pero no nos sirve para coger el resto de objetos, indicándose con un mensaje en la pantalla
6. Si se selecciona el interruptor, se enciende la luz del cuarto y se muestra un mensaje por pantalla.
7. Si la luz del techo está encendida, se procede a determinar qué acción realizar en función del objeto seleccionado.
8. Si se selecciona un objeto que no es el interruptor (como un cajón, una puerta, una llave), se realiza la acción correspondiente en función del objeto seleccionado. Esto incluye abrir cajones, abrir puertas, recoger llaves, etc.
9. Se muestra un mensaje en la pantalla según la acción realizada o el estado de la interacción.

En resumen, solo se permite seleccionar objetos si la luz del techo está encendida. Esto implica que la interacción con otros objetos en la escena, como cajones, puertas o llaves, solo es posible cuando la luz del techo está encendida.

Colisiones

Para explicar el algoritmo relacionado con colisiones primero debemos explicar el método “addCollisionableObject()” que es utilizado para añadir objetos al array “collisionableObjects”, que contiene los objetos con los que se puede colisionar en la escena. Este método recorre de forma recursiva todos los hijos de un objeto y añade los “Mesh” al array. Aquí se explica como funciona el método:

1. El método toma un parámetro “object”, que representa el objeto al que se desea añadir los objetos colisionables.
2. Se verifica si el “object” es una instancia de un Mesh utilizando el operador “instanceof”, es decir, si es un objeto con geometría y material.
3. Si el “object” es un Mesh, se añade al array “collisionableObjects” utilizando el método “push()”.
4. Luego, se recorren de forma recursiva los hijos de “object” utilizando un bucle for. El bucle itera sobre los hijos del objeto, accediendo a ellos mediante la propiedad “children” de “object”.
5. Para cada hijo, se llama de forma recursiva al método “addCollisionableObject()”, pasando el hijo como parámetro.
6. El método se ejecuta recursivamente hasta que se han recorrido todos los objetos en la jerarquía del objeto inicial.

En resumen, el método “addCollisionableObject()” recorre de forma recursiva la estructura de objetos, añadiendo los objetos “Mesh” al array “collisionableObjects”. Esto permite tener un control sobre los objetos con los que se puede colisionar en la escena y realizar operaciones de detección de colisiones de manera eficiente. Es importante remarcar que muchos objetos tienen incluidos en ellos una caja transparente igual de alta que el techo, puesto que algunos de ellos son demasiado bajos para colisionar con el rayo si no se les está mirando directamente.

Una vez establecidos los objetos con los que es posible colisionar, podemos hablar de la función “testColision(posicion, direccion, distancia)”. Se encarga de realizar una prueba de colisión entre un rayo lanzado desde una posición y en una dirección específica, con respecto a los objetos que se consideran colisionables. Aquí está el proceso que sigue:

1. Se crea un rayo utilizando el objeto raycaster con la función “set(posicion, direccion)”. “posicion” representa el punto de origen del rayo y “direccion” indica la dirección en la que se lanza el rayo.
2. El rayo se lanza utilizando “intersectObjects(this.collisionableObjects, true)” para comprobar si colisiona con algún objeto en el array “collisionableObjects”. Esta función devuelve un array de objetos que colisionan con el rayo.
3. Se verifica si hay al menos una colisión (“collisionObjects.length” > 0) y si la distancia a la colisión más cercana es menor o igual a la distancia máxima permitida (“distancia” + 0.2). Si se cumple esta condición, significa que ha ocurrido una colisión y se devuelve true, de lo contrario se devuelve false.

Esta función se llama en el método “updateCameraControl()” que es el encargado de actualizar el control de la cámara en función de la lógica de colisiones implementada. Aquí está el proceso que sigue:

1. Se obtiene el tiempo transcurrido en segundos desde la última actualización mediante “this.clock.getDelta()”. Esto se utiliza para calcular la distancia recorrida por la cámara en ese tiempo en función de la velocidad.
2. Se crea un vector “posicion” y se copia la posición actual de la cámara en ese vector.
3. Se crea un vector “direccion” para almacenar la dirección hacia la que se mueve la cámara.
4. Si “this.moveForward” es true, se obtiene la dirección actual de la cámara utilizando “this.cameraControl.getDirection(direccion)”. Luego, se verifica si hay alguna colisión en esa dirección utilizando la función “this.testColision(posicion, direccion, distancia). Si no hay colisión, se mueve la cámara hacia adelante utilizando “this.cameraControl.moveForward(distancia)”.
5. Si “this.moveBackward” es true, se realiza un proceso similar al punto anterior, pero se niega la dirección para mover la cámara hacia atrás.
6. Si “this.moveRight” es true, se obtiene la dirección actual de la cámara y se calcula la dirección hacia la derecha utilizando el producto vectorial con un vector (0,1,0) y normalizándolo, calculando el perpendicular. Esto permite que la cámara se mueva hacia la derecha en relación con su orientación actual. Nuevamente, se verifica la colisión y se mueve la cámara si no hay colisión.
7. Si “this.moveLeft” es true, se realiza un proceso similar al punto anterior, pero se niega la dirección para mover la cámara hacia la izquierda.
8. Si hay una colisión, se muestra un mensaje en la consola indicando que se ha producido una colisión.

En resumen, el código implementa una función de prueba de colisión y actualiza el control de la cámara en función de las colisiones detectadas. Si no hay colisión, la cámara se mueve hacia adelante según la velocidad establecida. Si hay una colisión, se registra en la consola un mensaje de colisión.

Movimientos

A continuación se explica qué pasos sigue el movimiento de la puerta una vez se pulsa sobre el pomo de la misma teniendo la llave:

1. Primero, se verifica si hay alguna animación en curso. Si no hay ninguna animación en curso, se procede con el movimiento de la puerta.
2. Se obtiene la posición actual de la puerta antes de moverla. En este caso, la puerta sólo puede estar en dos posiciones: izquierda o derecha..
3. Se definen dos objetos, origen y destino, que contienen la propiedad “p” que representa la rotación en el eje Y de la transformación de la puerta.

4. Si la puerta está en la posición izquierda (con “origen.p” igual a 0), se establece el destino de la rotación en 1.57 (aproximadamente 90 grados en radianes) para mover la puerta hacia la derecha. Si la puerta está en la posición derecha (con “origen.p” diferente de 0), se establece el destino de la rotación en 0 para mover la puerta hacia la izquierda.
5. Se crea una animación utilizando la biblioteca “TWEEN”. Esta animación cambia gradualmente el valor de la propiedad “p” de origen hacia el valor de destino en un período de 2 segundos.
6. Durante cada actualización de la animación, se actualizan las posiciones de la puerta y su transformación según el valor actual de “origen.p”. La posición de la puerta se establece en “x”=0.4, se define otro nodo “transformación” que contiene a puerta, la rotación de la transformación se establece en “origen.p”, y la posición de la transformación se establece en “x=-0.4”. Esto se hace así para respetar el orden de THREE Escalado>Rotación>Translación.
7. Cuando la animación se completa, se ejecuta la función “onComplete()”. En este caso, se establece “this.animacion” en falso para indicar que no hay ninguna animación en curso y se muestra una alerta al usuario indicando que ha completado el juego.
8. Antes de iniciar la animación, se establece “this.animacion” en verdadero para indicar que hay una animación en curso.
9. Se inicia la animación llamando al método “start()”.

La función “update()” se encarga de actualizar la animación en cada cuadro de la aplicación. Si this.animacion es verdadero, se llama al método TWEEN.update() para actualizar la animación en curso.

Los movimientos de la mesita de noche y el armario son análogos. El movimiento del interruptor es simplemente una rotación del mismo para que aparezca al revés.

3. Modelos externos y librerías

Con respecto a material externo hemos utilizado:

- Nuevas imágenes para texturas. Todas ellas han sido sacadas de Google Imágenes puesto que son imágenes genéricas que hemos utilizado para los pósters del Real Madrid. La única imagen que hemos añadido a la carpeta */imgs* y que no había sido proporcionada ya por el profesor o sacada de internet ha sido *wood-bump.jpeg* . Esta última la hemos creado nosotros, simplemente aplicando un filtro blanco y negro a la imagen original *wood.jpg*, para poder asignar un material con relieve a los muebles.

- Librería para el control en primera persona de la cámara. Hemos utilizado la librería PointerLockControls, que realmente no se puede considerar como material externo puesto que el profesor la proporcionó por PRADO. No obstante se puede conseguir, al igual que el resto de librerías, en la página oficial de THREE.js (https://threejs.org/).

- El modelo cargado. Este lo hemos descargado del siguiente enlace: https://free3d.com/es/modelo-3d/sofa-801691.html

4. Manual de usuario

Para ejecutar correctamente el juego, se debe lanzar un servidor web con python desde la carpeta que contiene el juego junto con las librerías y las imágenes (en nuestro caso, lanzarla desde la carpeta */juego*.

Controles

El movimiento de la cámara se realiza con las flechas, el picking se activa pulsando el click izquierdo del ratón (importante recordar que la distancia al objeto y si la luz del techo está encendida o no afectan al mismo) y el control de la cámara se hace pulsando la tecla “Ctrl” y moviendo el ratón a la vez.

Resolución

Para resolver el juego debemos realizar los siguientes pasos:

1. Encontrar una luz.
2. Si la luz que hemos encendido es la de la lámpara de la mesita de noche hay que encontrar el interruptor.
3. Si la luz que hemos encendido es la del techo, podemos empezar a interactuar con el resto de objetos.
4. Hay que buscar la llave abriendo la puerta izquierda del armario. Esta llave se usa para abrir el cajón más cercano al suelo de la mesita de noche.
5. Abrir el cajón indicado de la mesita, importante no confundir con el de arriba
6. Coger la llave de dentro del cajón.
7. Abrir la puerta una vez conseguida la llave. Para abrirla debemos pulsar sobre el pomo, en cualquier otro lugar dará error.

Una vez realizados todos estos pasos, cuando se acabe la animación de la puerta, saltará una alerta indicando que hemos acabado el juego.

5. Autores

Práctica realizada por Carlos Quesada Pérez y Jorge Lombardo Bergillos.