Memoria Pr2-Proyecto

Crazy Cube

Universidad de Granada

3ºGII-Sistemas Gráficos

Autores:

Rafael Cano Expósito

Luis Molina Castillo

Fecha:

Mayo 2022

Sumario

1. Descripción de la aplicación.....2
2. Diseño de la aplicación.....2
2.1 Diagrama de clases.....2
3. Manual de usuario.....2

1. Descripción de la aplicación

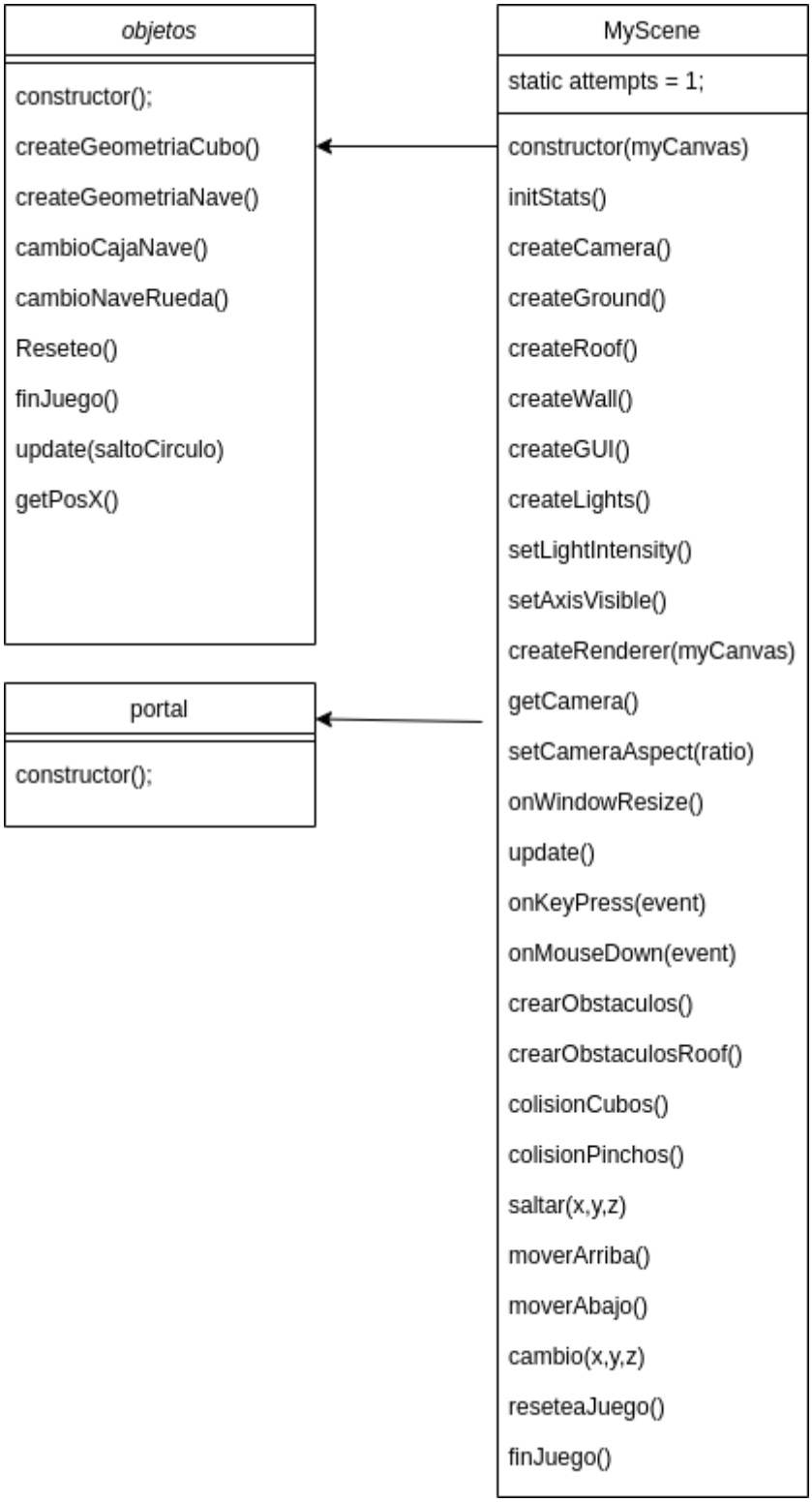
En la realización de este proyecto, hemos implementado alguna de las funcionalidades que ofrece el juego “Geometry Dash”, juego en 2D en el que un objeto se mueve a lo largo de un mapa con obstáculos, que debe superar sin chocarse con ellos para llegar al final con vida.

La funcionalidad que nosotros hemos incluido en este proyecto, no ha sido la completa que tiene el juego original. Primero, el juego se inicia con un cubo que esquiva los obstáculos colocados en el suelo con un movimiento arqueado. Al pasar el primer portal (un objeto creado con la geometría del toroide), el objeto se convierte en una nave (creada mediante varias geometrías haciendo uso de CSG) en la que va montada el cubo inicial, que se moverá arriba y abajo en el eje y según se pulse W o S. Por último, al pasar por el segundo portal, el objeto se convierte en una esfera con pinchos, que rota constantemente en el eje z, y cuyo movimiento es cambiar del suelo al techo para esquivar los distintos obstáculos.

2. Diseño de la aplicación

En este apartado, pasamos a explicar el proceso de diseño llevado a cabo para implementar los objetos, construir el mapa completo con los obstáculos y las paredes, el procesamiento de colisiones para resetear el juego en caso de choque con algún obstáculo y la animación del final del juego

2.1 Diagrama de clases



Como podemos apreciar en la imagen, tenemos tres clases en nuestro proyecto.

La clase *objetos*: es la encargada de la construcción de los objetos principales del juego, los que se mueven en el mapa (el cubo, la nave y la esfera con pinchos). En los métodos de cambio de objeto, simplemente hacemos un *remove* del objeto que tuviésemos hasta ese momento, y añadimos el que nos corresponda después. Al método "Reseteo", se le llama cuando ocurre una colisión con algún obstáculo. Por ello, este método hace uso de *clear*, para eliminar el modelo que tuviésemos en el momento y la memoria de la página, para añadir el objeto inicial (cubo). Como el último objeto, gira constantemente en su eje z, nos encontrábamos con el error de que al chocarnos, el cubo a veces se nos iniciaba girado, por lo que para resolver este error, hicimos un bucle para ver cuantas vueltas le quedarían por dar al objeto para estar recto, y así tener el cubo en la posición correcta. En el método "update" simplemente incrementamos la posición de x en 0.3 constantemente, y en caso de tener a true el booleano, es que estamos con el último objeto y vamos rotando en z todo el rato.

La clase *portal*, en la que simplemente se crean dos objetos con la geometría del toroide, que serán los "portales" por los que cuando pase el objeto del juego, se modificará su aspecto a otro objeto diferente.

La clase *MyScene:*. En ella creamos todo lo que va a ser nuestro juego completo. Para ello, en el constructor creamos el suelo y el techo con sus obstáculos, unas paredes que encerrarán el mapa con una textura cogida de https://es.dreamstime.com/stock-de-ilustraci%C3%B3n-fondo-futurista-del-juego-image54259579, y los portales de cambio de objeto. Muchos de los métodos de esta clase son los necesarios para visualizar la escena correctamente. Es el ejemplo de "createCamera", aunque para este juego lo hemos modificado, ya que la cámara debe mirar en todo momento al objeto en pantalla. Para ello, hacemos que el lookAt sea a la posición en el eje x del objeto en memoria en cada momento. Pero, hay algunos de gran interés que pasamos a mencionar a continuación:

- update

  En este método vamos actualizando la escena constantemente. La cámara, se mueve en el eje x según la posición del modelo (porque apunta a él en todo momento). En caso de superar unas posiciones concretas en x, hacemos las llamadas a los cambios de objeto (que actualizan los respectivos booleanos para el control), y en caso de superar 350 (hemos llegado al final), llamamos a la función del fin de juego. Actualizamos las animaciones TWEEN que hemos creado para los movimientos, y comprobamos que no haya colisiones con los obstáculos. Como el círculo queremos que empiece desde el suelo, al cambiar de nave a círculo, hacemos una comprobación en la que si tenemos el booleano correspondiente a true, lo posicionamos para estar en el suelo (algo poco visual, pero que es de la única manera que hemos podido posicionarlo como queríamos).

- onKeyPress(event)

  Este listener, llama a las diferentes funciones de movimiento en base a la tecla pulsada, y al booleano del objeto que esté a true.

- onMouseDown(event)

  Este listener, llama a las diferentes funciones de movimiento en base al ratón clickado, y al booleano del objeto que esté a true.

- colisionCubos() y colisionPinchos()

  Estos métodos son los encargados de comprobar si hay colisiones con los diferentes obstáculos del mapa (son cubos y conos). En un principio, teníamos idea de hacer las colisiones con cajas englobantes, pero al final cambiamos a hacer uso de *RayCaster*. Por ello, creamos un rayo que sale de nuestro modelo en el eje x (dirección del movimiento), y en caso de encontrarnos con que el rayo interseca con alguno de los objetos almacenados en los arrays que guardan los obstáculos, imprimimos un mensaje por pantalla y reseteamos el juego.

  Adicionalmente, añadimos dos rayos para comprobar con la nave que no nos metíamos por debajo o por encima de los obstáculos, pero al hacerlo y chocarnos, el juego no se reseteaba bien, no se actualizaba la posición en x por algún motivo que no supimos resolver. Por ello solo dejamos la colisión en el eje x.

- saltar(x,y,z) y cambio(x,y,z)

  En estas dos funciones la idea es la misma, se crean los recorridos de las 2 animaciones existentes.

En saltar se hace el camino del salto del cubo, una parábola creada a partir de la posición en la que se encuentra el cubo en el mismo instante en el que se presiona el botón para realizar la animación. Además se maneja un booleano para poder evitar que la animación no se interrumpa hasta que este terminada, con el método onComplete().

En cambio la idea es la misma, la diferencia es que aquí manejamos 2 animaciones, una de subida de la rueda y otra de bajada. Usando booleanos se intercalan estas 2 para que no se repita 2 veces seguidas la misma

- reseteaJuego()

  Aquí se llama al método Reseteo del modelo de la clase *Objetos*, y se coloca el mismo modelo en la posición inicial del juego. Además se aumenta en uno el número de intentos realizados. Por último se actualizan todos los booleanos del juego

- finJuego()

  Para el final del juego se pone una frase de agradecimiento, mostrando el número de intentos utilizados, y además parecen 2 botones, unos para volver a jugar y otro para salir del juego

3. Manual de usuario

Para poder jugar al juego lo primero es abrir en la carpeta Juego-SG-main/FINAL/juego una terminal y poner la orden $*python -m http.server* o la orden $*python -m SimpleHTTPServer* en función de la versión de python que se esté utilizando.

Despues hay que abrir el navegador y poner en la barra de búsqueda *localhost:8000* y se abrirá directamente el menú principal del juego donde habrá 2 botones, uno para empezar a jugar y otro para ver los controles del juego que son los siguiente:

Controles del juego:

Click Izquierdo ratón --> Salto normal y cambio de suelo (objeto círculo)

Spacebar --> Salto normal y cambio de suelo (objeto círculo)

W/S --> Sube y baja la nave