MEMORIA DE DISEÑO Y EXPLICACIÓN DE LA PRÁCTICA

Práctica Obligatoria de Programación Avanzada:

BubbleTrouble

Grado en Diseño y Desarrollo de Videojuegos – 2018/2019 – URJC
Denis Gudiña Nuñez

Acerca del documento

Este documento pretende explicar todas las decisiones de diseño escogidas para realizar esta práctica, explicando los conceptos aplicados en ella y su relación con la práctica. Se analizarán las clases que influyen en de la ejecución del programa y que se hayan creado o modificado.

Explicación de clases

Clases utilizadas sin modificación

Las clases “Vector3D”, “Pared”, “Camara” y “CamaraFPS” se han utilizado sin ninguna modificación en el código base pues no se consideró necesario.

Clase Solido

Esta clase maneja la creación de cualquier objeto sólido en el juego, por lo que será la clase padre de otras clases. Se parte de la base dada en el motor de clase y se añaden ciertos cambios para encajar con los requerimientos de la práctica y el proyecto elegido. Los cambios son:

- Dos nuevas variables: un “string” que indica el tipo del sólido y un “bool” que indica si ha sido colisionado o no.
- Los correspondientes métodos de acceso de estas dos variables.
- Cambios en el “update” para que actualice a aquellos objetos no manipulados por el jugador y que se muevan de forma autónoma, dando un tipo de comportamiento dependiendo del tipo de objeto que sea. Se aplica a las balas y las pelotas (se verá más adelante).

Clase Jugador

Esta clase será instanciada como el sólido que representa a nuestro personaje, que se mostrará como un cuadrado. Hereda directamente de “Solido” por lo tanto recibe sus métodos al ser públicos. La clase consta de:

- Tres variables: Un “double” que indica el valor del lado del cuadrado a generar (la forma del personaje), un “int” que indicará la puntuación y un “bool” que indica si el personaje está vivo o muerto.
- Los constructores pertinentes (por defecto y por valor).

- Un destructor.
- Los métodos de acceso pertinentes para tratar las variables.
- Un método “render”, que se encargará del pintado por pantalla.
- Un operador “ostream”.

Clase Bala

Esta clase será instanciada como el sólido que representa a las balas que actuarán como proyectiles manejables en el juego. Hereda directamente de “Solido” por lo tanto recibe sus métodos al ser públicos. La clase consta de:

- Dos variables: Un “double” que indica el valor del radio de la esfera a generar (la forma de la bala) y un “bool” que indica si la bala está en movimiento.
- Los constructores pertinentes (por defecto y por valor).
- Un destructor.
- Los métodos de acceso pertinentes para tratar las variables.
- Un método “render”, que se encargará del pintado por pantalla.
- Un operador “ostream”.

Clase Pelota

Esta clase será instanciada como el sólido que representa a las pelotas que actuarán como objetivo a eliminar en el juego. Hereda directamente de “Solido” por lo tanto recibe sus métodos al ser públicos. La clase consta de:

- Dos variables: Un “double” que indica el valor del radio de la esfera a generar (la forma de la pelota) y un “int” que indica el número de capas que tiene la pelota y los subniveles en que se puede dividir.
- Los constructores pertinentes (por defecto y por valor).
- Un destructor.
- Los métodos de acceso pertinentes para tratar las variables.
- Un método “render”, que se encargará del pintado por pantalla.
- Un operador “ostream”.

Clase Escena

Esta clase recoge los objetos anteriores y los condensa en una sola clase que tendrá todas las herramientas de actualización de posición, de colisiones, estructuras para almacenar las instancias en pantalla y métodos para manipularlas. La clase consta de:

- Cuatro variables: Un vector de punteros a objetos “Pared” (delimitan la región de juego), un vector de punteros a objetos “Pelota” (guardan los objetivos), un puntero a “Jugador” y un puntero a “Bala”.
- Un constructor vacío.
- Su destructor.
- Los correspondientes métodos de acceso para las variables.
- Un método “add” sobrecargado que guarda pelotas o paredes dependiendo de la entrada.
- El método “render”, para mostrar por pantalla todas las instancias anteriores.
- Los métodos de colisión entre pelota-jugador y pelota-bala. Estos métodos calculan a través de razonamientos matemáticos de vectores cuando se encuentra en contacto una pelota con un jugador o una pelota con una bala. En caso de que se produzca colisión, en el caso del jugador y la pelota, se pone el “bool” de muerte del jugador a “true” y esto se comprueba en el “main”, que termina la partida. En el caso de la pelota con la bala, se elimina la pelota y la bala vuelve a su posición original, produciendo un sonido.

Main del proyecto

Aquí se encuentran las inicializaciones de variables del proyecto y distintas funciones útiles para el juego. Consta de:

- Las variables usadas en el juego.
- Distintos métodos de acceso para variables del “main”.
- Métodos de lectura y escritura en fichero para almacenar las puntuaciones en un fichero de texto persistente.
- Un método menú que a través de entradas de texto nos permite elegir distintos modos de dificultad o la consulta de las puntuaciones.
- Métodos de control del ratón para actuar sobre la cámara y la rotación de esta.
- Un método “displayMe” que se encarga de configurar lo que ve la cámara.
- Un método para controlar las teclas del teclado y sus funciones. Aquí se recogen todos los comandos que se van a usar en el juego, como el movimiento, disparos, pausa, control de cámara, salida y escritura básica.
- Un método “init” que inicializa las variables del juego en un principio.
- El método “idle” para que la ejecución esté en un “loop” hasta que se indique lo contrario.

- Un método “reshape” para que el “viewport” se ajuste al tamaño de ventana y no se deforme la información de pantalla.
- Y por último el “main” que recoge lo anterior y lo pone en marcha.

Decisiones de diseño

En este proyecto se ha intentado tocar un poco todo lo visto en clase para demostrar el conocimiento de estos conceptos.

La herencia es parte fundamental del proyecto pues todos los objetos de juego se crean a base de herencias (pelotas, jugador, bala, paredes…), y esto permite un control de distintos parámetros comunes y específicos de las instancias de manera sencilla.

En cuanto al polimorfismo se destaca la función “render” de los sólidos, heredada del padre y que cada objeto modifica después de acuerdo a lo que se ha de mostrar por pantalla. También se han aplicado métodos sobrecargados, como el “add” de escena que añade distintos tipos de objetos a su respectiva estructura de datos dependiendo de la entrada del método.

Todas las variables se han utilizado a modo de punteros y se han optimizado de acuerdo a su función. Por ejemplo, en vez de crear muchas balas, solo se crea una y se actualiza su posición cuando se detecta colisión o se dispara, ahorrando memoria. Lo mismo se aplica respecto a las pelotas, pues cuando una se destruye genera otras dos a través del mismo puntero, por lo que están ordenadas en memoria y ocupan menos.

En cuanto a las actualizaciones de colisión y posición se decidió usar dos enfoques diferentes. Para las colisiones con objetos inmóviles se usan colisiones simples por posición pues es el enfoque más sencillo para ese problema. Sin embargo, cuando se habla de colisiones con objetos móviles se consideró más efectivo la utilización de principios matemáticos de vectores y distancias entre vectores como solución más adecuada. Al final las colisiones se adaptan a aquello a lo que hacen referencia de la manera más sencilla que se ha podido encontrar para cada caso.

Se ha implementado un menú por texto también para que la experiencia de juego sea algo más completa, mostrando unas opciones de modos de dificultad que el usuario puede seleccionar. La selección se guarda para motivos explicados más adelante, y a partir de esta se crea la pelota inicial con distinto nivel dependiendo del modo elegido.

El uso de ficheros también ha sido algo fundamental en la práctica, pues el sistema de puntuaciones funciona a través de la manipulación de ficheros. La información guardada a través de la selección de modo del jugador se guarda junto a la puntuación final después de ganar o perder en el juego y se guarda en el fichero, que se accede en modo “append” para evitar que se borre lo que está escrito y escriba la nueva información a continuación de la antigua. Con esto podemos guardar el tipo de partida

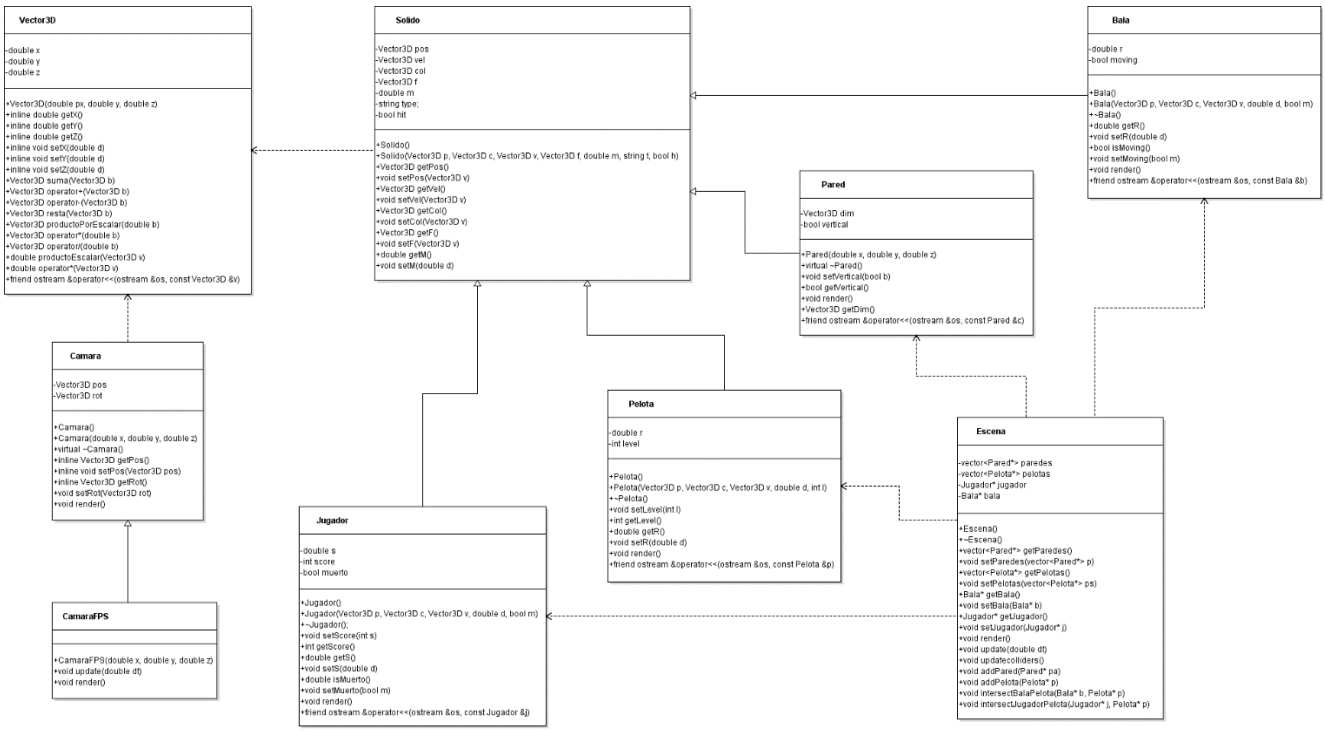
jugada y la puntuación. A la hora de leer, si el usuario selecciona la opción de mostrar las puntuaciones aparecerán por texto todas las puntuaciones.

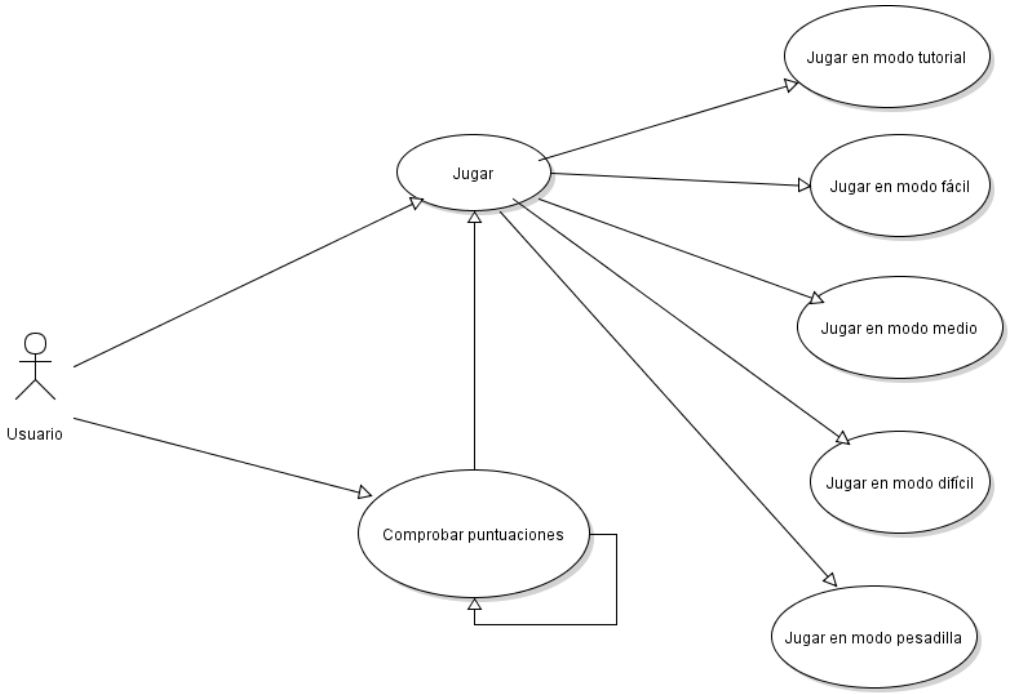
Así mismo dentro del juego se añadió también una sencilla pausa a través de un “bool” que para la actualización automática del juego hasta que vuelva a modificarse su valor, siguiendo el esquema de soluciones sencillas pero efectivas que se ha querido llevar a cabo con esta práctica.

En cuanto a controles de teclado se buscaba que se asemejase lo más posible a un juego arcade y se han añadido los controles necesarios de control de movimiento, disparo, pausa y movimiento/rotación de la cámara, que se consideran fundamentales.

Diagrama de clase y de casos de uso

Los diagramas mostrados se encuentran en la carpeta de documentación en formato más grande para mejor visualización. Se muestran el diagrama de clases y de casos de uso:




Jugar en modo tutorial
Jugar
Jugar en modo fácil
Jugar en modo medio
Usuario
Jugar en modo difícil
Comprobar puntuaciones
Jugar en modo pesadilla