JRV
PRACTICA 2: LABERINTO 3D

Francisco Andrés Sanchís Pérez
Juan Rubén Segovia Vilchez
Joaquín Vegara García

Índice de contenido

1. Resumen

Este juego consiste en resolver una serie de laberintos/puzzle. Para moverte por el mapa utiliza las flechas. Cuidado porque si caes una distancía superior a un piso mueres y vuelves a empezar (no hay transición, directamente vuelves a empezar el nivel). Si se te acaba el tiempo vuelves a empezar el nive.

Para pasar niveles tienes que conseguir coger el ascensor que te llevara muuy alto. Y de pronto desapareces y empiezas otro nivel. No, no hay transicion...

Habrá veces que si tomas decisiones equivodas el laberinto ya no tenga solución... mala suerte y piensalo mejor la próxima vez. Bueno, puedes pulsa en cualquier momento la tecla R para volver a empezar el nivel.

Para cambiar el color del GUI usa las teclas T,G (rojo) Y,H (verde) U,J (azul). La primera letra de cada par aumenta la compontente mientras que la segunda letra la disminuye.

Para acercar o alejar la minivista utiliza las teclas Z,X. Y para también las teclas Q,A para acercar o alejar la vista en un único eje.

Por último, para ver el juego en vista estereográfica se puede pulsar la tecla E. Pulsa la tecla M para volver a la visión monográfica en cualquier momento. Aunque también regresaras a la visión monográfica automáticamente cuando pases de nivel, mueras, o reinicies el nivel. La visión estereográfica consiste en dos renders pequeños uno al lado del otro, correspondiente a lo que ve cada uno de los ojos del personaje. Esto enviado a unas gafas de realidad virtual produciría un entorno 3D. Si uno es hábil puede cruzar los ojos, y jugar en un entorno 3D simulado, lo he probado y queda muy bien, lastima que de dolor de cabeza... quiero unas gafas de realidad virtual!

2. Clases

2.1 Juego

Esta clase contiene el bucle principal, y se encarga de gestionar el cambio de niveles.

2.2 Nivel

El constructor recibe un nombre de fichero a cargar, y se encarga de parsearlo y generar un nivel según su información. Después recibe el flujo desde Juego en cada iteración, en los métodos Actualizar y Pintar.

2.3 Item

Esta clase es la base para la mayoría de elementos del nivel.

2.4 Jugador

Esta clase se encarga de recoger el estado de las flechas y mover al jugadro en función de eso. El método Pintar pinta un bounding box, pero solo es pintado en la minivista.

2.5 Camara

El nivel tiene un objeto camara, que se asocia al jugador. Lo único que hace es obtener la posición y angulos del jugador, y generar las llamadas opengl necesarias para colocar la cámara en modod 1ª persona del jugador. Esto lo hace en el método Camara::Pintar, de modo que este es el primero en ser llamado al renderizar la escena.

2.6 Ascensor, Losa, Puerta

Representan los objetos interactuables del escenario. En realidad solo es interactuable

directamente, la losa (Anillo mágico), con los otros se interactua por medio de losas. Redefinen el método Actualiar y Pintar conforme lo necesitan.

2.7 Bloque

Los objetos bloque de un nivel representan la geometría del nivel, ha sido necesario crearlos, para que el movimiento a traves del nivel se realice utilizando el mismo sistema de colisiones utilizado para las colisiones con otros items (ascensores, losas, puertas). Es decir, no se mueve al jugador, comprobando la altura de la celda en la que esta, sino que se crean bloques con distintas alturas para cada celda y el jugador cae constantemente, pero como colisiona con estos bloques, no cae.

2.8 Luz

El level script language permite configurar la iluminacion de los niveles, desde la iluminacion ambiental, hasta situar luces por el mapa (con color configurable). Esta clase representa estas luces, y en el método pintar realizan las llamadas opengl para situar la luz (Las luces en opengl se guardan en coordenadas de ventana, así que si cambia la posición de la cámara es necesario volver a situarlas, para que opengl vuelva a hacer las transformaciones correspondientes)

3. Flujo

3.1 Inicialización

La inicialización de recursos se realiza en Juego::Inicializar, donde además se llama a los métodos Inicializar de las clases que necesiten inicializar algo.

3.2 Blucle Principal

Primero, se limpia el frame buffer y el depth buffer (Se borra la pantalla para pintar en ella, evidentemente el back buffer, no el front buffer). Tras esto se llama a nivelActual->Actualizar() (y que a su vez se llama a Actualizar de todos los objetos del nivel) y después a nivelActual->Pintar (que a su vez llama a Pintar de todos los objetos del nivel). Y por ultimo glFlush() y allegro_gl_flip(). Ya se ha actualizado y pintado el mundo.

Tras esto se comprueba si se pulsa la tecla **ESC** (Salir del juego) o **R** (resetear el nivel). Además se comprueba si el nivel está en estado *Muerto* (Resetear el nivel) o *Superado* (Pasar de nivel).

Al pasar de nivel, si no quedan mas niveles, se produce una excepción y se sale del juego, haría falta arreglarlo, pero lo único que cambiaría sería en que se saldría del juego sin decir el motivo, de este modo por lo menos sabes que es porque ya no quedan más niveles. Las solución óptima sería mostrar algún mensaje de que se ha superado el juego, etc... pero ya no hemos querido meternos en más trabajo para la práctica, y dedicarnos al proyecto.

3.3 Losa::Actualizar()

Y un último método del que creo conveniente dar una explicación es Losa::Actualizar ya que refleja como funciona el proceso de Item::Excitar, Item::Desexcitar e Item::Intercambiar.

Primero se obtiene la posicion actual del jugador con nivel->GetPosicionJugador y nivel->GetAlturaJugador, para determinar si el jugador ha pisado la losa. En caso de que así sea, llama a nivel->ExcitarCodigo(objetivo), donde *objetivo* es el objetivo que se le ha definido en el fichero del nivel.

En Nivel::ExcitarCodigo, lo que se hace es recorrer la lista de Items, se compara el codigo de cada item con el valor recibido, y si son iguales, se le invoca al método Item::Excitar. Si por ejemplo, hay una ascensor con ese codigo, se llamara al método de la clase derivada Ascensor::Excitar, que cambiará el estado interno del ascensor a Excitado, de tal forma, que

cuando se ejecute Ascensor::Actualizar (para este ascensor concreto) empezar a ascender (o descender, según la posición excitada esté por encima o por debajo).

Es conveniente hacer notar, que cualquier número de Items pueden compartir el código (es decir, e código no es de ningun modo un valor representativo o identificativo, solo sirve para recibir estos mensajes) de modo que sean activados por la misma losa. Por supuesto, losas distintas también pueden actuar sobre el mismo código.

E incluso, las losas, como derivadas de Item que son, pueden recibir mensajes. Una losa desexcitada está desactivada, y una losa excitada está activada, una losa que está desactivada no se pinta ni hace nada.

Por último, es perfectamente posible, que una losa actue sobre si misma, si su objetivo es igual a su código. De modo que una losa podría actuar sobre algún elemento del mapa intercambiando su estado, y además sobre si misma, intercambiando su estado, por lo que, la losa se desactivaria y no se podría volver a pisar. Estas es la forma de hacer losas que sólo actuan una vez.

4. Partes específicas

4.1 Sistema de colisiones

Las colisiones son calculadas a nivel de Item (clase base de la mayoría de elementos, y desde luego, de todos aquellos elementos que tengan que intervenir en las colisiones), de hecho, no haría falta crear los objetos Bloque (derivado de Item) para la geometría del escenario, puesto que el escenario en realidad es pintado desde Nivel::Pintar, sin embargo, se crean estos objetos con el único objetivo de participar en las colisiones, perimitiendo de esta forma al jugador poder explorar el escnario, en lugar de caer en el vacío.

Existen dos propiedades de todo item (y que son asignadas en las respectivas clases derivadas) que maracarán su comportamiento en las colisiones: *obstaculo* y *empujable*. El primero indica que este objeto es un obstaculo, cuando por ejemplo el jugador intente moverse a una nueva posición, se calculará colisiones con todos los Items que sean obstaculos. El segundo, empujable, indica que este item es empujado por otros en su movimiento, el único item que es empujable es el jugador (aunque si se incluyesen por ejemplo armas en el escenario, enemigos, o alguna clase de item para recoger, sería también empujable). Debe notarse, que no todos los items son *empujadores*; no hay en realidad ninguna propiedad de itemp para *empujador*, puesto que el empuje no se hace a nivel de Item sino de clase derivada. Los ascensores y puertas son empujadores, por ejemplo si un ascensor está subiendo, y el jugador esta encima del ascensor es empujado hacía arriba, y lo mismo sucede con la puerta.

Para el movimiento del jugador, se calcula el vector de movimiento 2D según las flechas de movimiento, y se le añade una componente vertical hacía abajo (aceleración de la gravedad), que claro, si hay suelo, será eliminada por la colisión, y el jugador no caera, pero si no hay suelo...

Para las colisiones se utilizan colisiones entre bounding boxs, y para su explicación recomiendo leer el codigo directamente en Item::Colisionar y Item::Empujar, ya que cualquier explicación que ponga aquí va a ser un embrollo más que otra cosa.

4.2 Level Script Language

Los ficheros de nivel son scripts sencillos, en lo que hemos llamado Level Script Language. Se puede observar en los niveles de ejemplo del juego (directorio niveles) la estructura de los niveles, ya que están autodocumentados, explicando cada directiva y sus parámetros.

4.3 Anillo mágico (Losa)

Los anillos mágicos (o losas) estan hechos con un quad, que tiene una textura trasparente con el dibujo de las “runas” en blanco para poder pintarla del color que se quiera, (ver Gui para una

explicación del color track).

Los anillos se pintan de color azul, y cuando son activados, se aumenta la propiedad del material (con glMaterial). Lo que se hace concretamente es añadirle emision (GL_EMISSION) de color verde. Esto ciertamente queda más vistoso en niveles oscuros.

Además, los anillos están girando, y cuando los pisas, se le aplica una aceleración angular, hasta alcanzar una velocidad angular crítica (realmente no se ha implementado una función de movimiento resistiva, sino que se ha simulado la velocidad crítica con un if jeje)

El valor de emision verde es proporcional a la velocidad actual de giro.

4.4 Gui

La gui se compone de un quad con textura que es pintado después de pintar la escena. Para que se pinte por encima de todo se desactiva el GL_DEPTH_TEST.

La textura del gui (y de todos los textos del gui) son de color blanco, de modo que se pueda pintar del color que se quiera cambiando el track color (cambiando el color actual con glColor), para esto tiene que estar activado GL_COLOR_MATERIAL.

Esto ha permitido que se pueda cambiar el color de la GUI. Con las teclas **T Y U** se sube el valor de las compontentes roja, verde y azul respectivamente, mientras que con las teclas **G H J** se reducen.

Los textos son escritos con el mismo color, pero encima de la gui, por lo tanto, hace el efecto de verse "menos transparentes" debido a la forma en que funciona el blending.

4.5 Minivista

La minivista se observa en la esquina superior derecha. Se pinta encima del gui.

Para pintar la minivista se repinta toda la escena de nuevo cuando ya esta todo pintado, en un viewport que lo delimita a la zona pequeña de la minivista. Antes de hacer esto hay que borrar el depth buffer para que se pinte, ya que si no le podría afectar otras cosas que hubiesen abajo y no se pintaría bien.

La vista no es desde arriba, ya que hay suelo en todas las celdas, unicamente diferenciados por la altura, y con una vista vertical desde arriba se confundiría todo, por lo que se ha optado por una vista inclinada de 45º, en la que se observa el nivel girando. Se pueda aumentar y disminuir el zoom con **Z X**, y se puede alejar y acercar el nivel con **Q A**.

4.6 Visión estereográfica

Se ha incorporado la posibilidad de ver el juego en una vista estereográfica. Se activa pulsando **E** y se desactiva pulsando **M** (monográfica), en la vista estereográfica se observa dos imagenes del nivel, correspondientes cada una a un ojo. Si se cruzan los ojos hasta conseguir juntar las imágenes se podrá jugar en 3D simulado.

Evidentemente, si se dispusiera de unas gafas de realidad virtual, esto se podría hacer sin necesidad de quedarse "bizco". OpenGL parece soportar el renderizado a contextos estereoscópicos (como lo pudirean ser unas gafas de realidad virtual) con la funcion glDrawBuffer, pasandole GL_LEFT o GL_RIGHT.

Actualmente, lo que se hace es partiendo de la posición del jugador, renderizar desde una posición un poco a la izquierda, con un angulo un poco a la derecha, y desde una posición un poco a la derecha, con un ángulo un poco a la izquierda. El angulo es fijo, sin embargo lo conveniente sería hacer que el ángulo variase para enfocar los ojos (el punto en que se cruzarían las trayectorias) justo en el punto central de la pantalla (dependiendo de la profundidad de este)