Informática gráfica 2014-2015
Etapa 7: Maze Of Doom

Balaguer Gimeno, Pere
41573414V

Boyanov Koev, Petar
Y0081060E

Índice

1. INTRODUCCIÓN.

Acabas de despertar en un lugar oscuro y siniestro, a tu alrededor solo hay muros de piedra maciza. No sabes donde estas, no sabes donde ir, bienvenido al **laberinto de la fatalidad** donde morir es la opción mas sencilla.

Se ha elegido una ambientación “retro”, siniestra y poco iluminada intentando rendir un tributo a aquellos juegos de antaño donde lo importante no era la calidad gráfica o de la historia sinó cuan lejos eras capaz de llegar sin morir. Como se puede intuir, hemos puesto esmero para que el jugador sienta el pavor de no saber lo que se avecina en la próxima esquina, si su perdición o la salida de dicho infierno.

El objetivo primordial del juego no es otro que conseguir escapar del laberinto. Como todo buen juego, es importante puntuar a los jugadores y eso se hace cronometrando el tiempo que tardan en salir (si es que lo consiguen, no todos lo hacen).

2. CONTROLES

En la pantalla de inicio, victoria o muerte:

Control	Acción
Click derecho del mouse	Despliega el menú de inicio. Desde el puede iniciarse una nueva partida o cambiar la dificultad de la próxima.

Dentro del juego:

Control	Acción
W, w	Avanzar
S, s	Retroceder
A, a	Desplazamiento a la izquierda
D, d	Desplazamiento a la derecha
Flecha 'up'	Mirar hacia arriba
Flecha 'down'	Mirar hacia abajo
Flecha 'left'	Mirar a la izquierda
Flecha 'right'	Mirar a la derecha
1	Modo primera persona
2	Modo tercera persona
3	Vista ortogonal del laberinto en conjunto
M, m	Cambia el modelo de renderizado (flat / smooth)
G, g	GodMode
ESC	Vuelve a la pantalla de inicio
Click derecho del mouse	Ídem que en el caso anterior

3. DETALLES DE IMPLEMENTACIÓN.

3.1. LABERINTO

Para generar el laberinto, se ha recurrido a una versión randomizada del algoritmo de generación de árboles minimales de Robert C. Prim (conocido como algoritmo de Prim).

Dicho algoritmo garantiza que el laberinto es connexo (es decir, desde cualquier punto se puede llegar a la salida) pero, como se especifica en wikipedia (de donde surgió la idea), la llegada de un punto cualquiera a la salida, no es tan sencilla.

A fin de simplificar el algoritmo (implementado en el fichero *mazeGenerator.c* y definido en *mazeGenerator.h*) se ha definido una estructura de datos auxiliar (Lista) que permite la elección y eliminación (aleatoria) de un elemento en tiempo constante O(1) y en 2 fases separadas (no entraré en detalles, si se considera adecuado, el codigo fuente puede ser consultado). Dicho código se halla descrito en los ficheros *list.h* y *list.c*

Como último detalle, cabe decir que la creación del laberinto se hace sobre memoria dinámica, cosa que implica la posiblidad de definir laberintos de tamaño “ilimitado”.

3.2. OBJETOS DE LA ESCENA

La definición de las estructuras de datos auxiliares de cada objeto se hace en el fichero *sceneObjects.h* y su implementación, juntamente con una serie de funcionalidades que simplifican su inicialización y eliminación en el fichero *sceneObjects.c*

Es relativamente importante decir que la información sobre el material de los objetos se ha comprimido. Se entrará en detalles más adelante.

Por último, al igual que en el caso del laberinto, los objetos también se construyen sobre memória dinámica, por ello, su número seria “ilimitado”.

3.3. PINTADO DE LA ESCENA

El pintado de la escena se lleva a cabo en el fichero *drawSceneObjects.c* (y su correspondiente *header*). Como cabría esperar, en dicho fichero está todo el trabajo relacionado con la carga de texturas en la escena.

Poco puede decirse de las rutinas de pintado ya que se ha sacado partido a las bondades de la libreria Glut. Respecto a la compresión de los materiales, la fórmula es:

especular = material → especular;
ambiente = material → especular – 0.8;
difusa = material → especular – 0.2;

También mencionar que el Clipping Plane se ha situado a relativamente poca distancia del jugador para que este, al encontrar un pasillo largo (mas de 10 unidades), crea que está llegando a la salida y se ilusione hasta que finalmente, aparezca un muro ante él.

3.4. RENDERIZADO CONJUNTO

El renderizado completo de la escena, se lleva a cabo en el fichero *etapa7.c*. Es un fichero muy extenso donde se especifican luces, gestión de la lógica del juego, llamadas a funciones de los otros ficheros, creación y destrucción de objetos, detección de colisiones, etc.

Cabe mencionar que pese a que tal vez la parte de optimización relacionada con openGl no es tan "buena" como podría ser, se ha puesto esmero en conseguir que el código general sea lo mas eficiente posible (excepto en la gestión de memoria, no se ha considerado demasiado importante que el juego tarde unos instantes más en iniciarse porque está reservando/liberando memoria).

Por poner un egemplo de la esmentada optimización, para evitar comprobar en cada movimiento de cada objeto si dicho objeto colisiona con el jugador / con un muro, etc. Se ha mapeado el laberinto sobre un array bidimensional que contiene en cada momento el elemento que hay en cada una de las casillas así, solo hay que aplicar el algoritmo de colisiones exhaustivo (distancias entre rádios) en el caso de que en la celda destino haya el jugador o un obstáculo. Con el propósito de mantener en todo momento el estado del laberinto, se usa un array auxiliar donde cada objeto (que tiene su propio id) mantiene registrado el contenido de la última casilla visitada a fin de, una vez la abandone, poder restaurar su valor anterior.

3.5. OTROS DETALLES

Como puede intuirse, el código ha sido íntegramente desarrollado en C, esencialmente debido a su flexibilidad y velocidad de ejecución (además de un apego especial que tenemos mi compañero y yo a dicho lenguaje).

Para cargar las texturas, se ha usado la librería Simple OpenGL Image Library (SOIL), disponible en: http://www.lonesock.net/soil.html. Dada su antiguedad, es recomendable (en mi caso ha sido imprescindible) compilar la librería en la máquina destino (viene con un makefile incluido en projects/) antes de intentar compilar nuestro proyecto o de lo contrario, pasarán cosas terribles.

Se ha adjuntado con el proyecto un makefile que se encarga de compilar automáticamente el codigo fuente. Es importante destacar que NO se ha testeado sobre ninguna máquina Windows (irónicamente, no disponemos de tal herramienta) aunque suponemos que no habrá ningún problema (y si lo hubiese, podemos ejecutar la práctica sobre alguno de nuestros ordenadores o en uno de los del aula linux).

4. CAPTURAS DE PANTALLA