Memoria

Tecnología de videojuegos – Team 6

Fernández Díaz Daniel

# Fase de diseño

Durante la fase de diseño fui el encargado de realizar una pequeña historia de ciencia ficción sobre la que basar el argumento del videojuego, puede verse completa en el documento de diseño de este proyecto.

Se trata de un relato corto que utiliza el tópico de las invasiones alienígenas y está fuertemente inspirado en la invasión encubierta de los franceses a España y la posterior rebelión de los españoles contra ellos.

Además, en esta fase realicé varios concept-art de la protagonista y algunas armas, tratando de buscar ese enfoque militar y futurista.

# Fase de desarrollo

Durante esta fase, mi responsabilidad ha sido la de diseñar e implementar un sistema de generación procedural de niveles, así como una serie de métodos para comunicar y conectar el nivel generado con el resto del juego.

El nivel generado se compone de una serie de habitaciones interconectadas por una serie de salidas. Este es su componente clave, la Habitacion. Esta clase contienen toda la información necesaria para que los elementos del juego como pueden ser los personajes y las balas interactúen de forma correcta con el nivel.

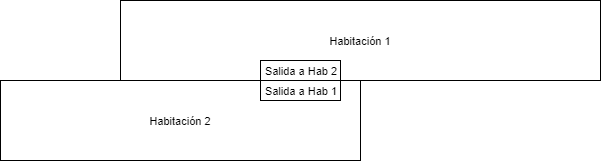
En cada instancia de esta clase encontramos, como atributos principales:

* El objeto en sí, que hereda de la clase Rectangle y, por ello, nos facilita una serie de parámetros. Los principales son los que nos permiten obtener tanto sus **x** e **y** máximas y mínimas como los que nos devuelven directamente su altura y su ancho.

Solamente con estos parámetros y la correcta generación previa, ya disponemos de la mayor parte de información que se suministrará a los personajes y a otros componentes del programa para implementar la interacción y la jugabilidad.

* Las salidas inferiores y superiores de dicha habitación. Se describirán con más detalle a continuación.
* Las celdas por las que está formada la habitación. Tienen un papel clave en la generación, pero a este nivel son importantes porque permiten crear el fondo de la mazmorra: a cada una se le asigna una imagen o tesela de entre las siete posibles y el fondo de la habitación es la combinación de estos.

En cuanto a las salidas, se trata de otra clase que hereda de rectángulo y, por tanto, permite identificar si otra figura geométrica de slick2d está intersectando con ella o, lo que es lo mismo, colisionando. De esta forma los métodos de la clase Nivel permiten al personaje llamador saber si al saltar puede o no cambiar de habitación, comprobando dicha colisión. Además, cada salida contiene una referencia a la habitación a la que conduce, facilitando este cambio de parámetros a los personajes.



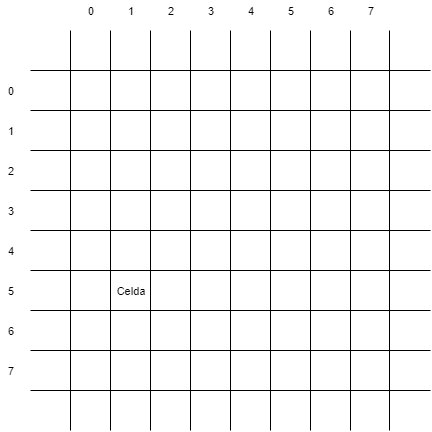
## Generación de niveles

Estos niveles compuestos de habitaciones interconectadas se generan de forma procedural a través de un algoritmo de toma de decisiones. Se explica su funcionamiento a continuación.

### Paso 1:

Establecemos en un entero un contador de celdas y en otro el valor máximo que permitiremos en dicho contador. Este será el número mínimo de celdas que queremos visitar antes de que finalice el algoritmo y una de las condiciones esenciales para su convergencia.

Inicializamos también una rejilla de Celdas, las cuales almacenarán si han sido visitadas y la Habitacion a la que han sido asignadas (además de su tesela de fondo como vimos antes), así como dos variables más que almacenarán la fila y la columna en la que nos encontremos en cada iteración.



Filas

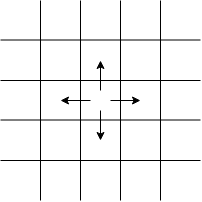
Columnas

**I**

Marcamos como visitada la celda inferior izquierda según la ventana del juego, la cual corresponde a la fila 0 y la columna 7 de nuestra rejilla; instanciamos la primera habitación con ella; y pasamos al bucle del algoritmo.

### Paso 2

Dentro del bucle do-while distinguimos dos zonas. En la primera se decide aleatoriamente si el algoritmo irá a la derecha, a la izquierda, hacia arriba o hacia abajo dentro de la tabla de Celdas. Esta decisión puede ser totalmente aleatoria o estar limitada al movimiento horizontal o vertical solamente en función del número de celdas que formen ya parte de la habitación y el número de celdas ya visitadas en ese “piso” del edificio.



En la segunda zona, tenemos un switch, el cual contiene las acciones a realizar en función del movimiento que finalmente se ha decidido hacer.

* En movimientos horizontales, la nueva celda que se visita se añade a la habitación de la celda anterior. Con las restricciones de movimiento que mencionamos antes se intenta que ninguna Habitación tenga menos de dos celdas y que tenga un número máximo de cuatro.
* Los movimientos verticales en este algoritmo implican la creación de una nueva Habitación si la celda siguiente no ha sido visitada. Intentamos dar así al nivel una apariencia más laberíntica.

Así, antes de realizar este movimiento se verifica que el número de celdas libres en ese piso sea superior a uno si dicha celda no ha sido ya visitada, tratando de evitar de nuevo la aparición de habitaciones con una sola celda.

Una vez superada esta verificación, se crea la nueva Habitacion si la celda no había sido visitada y se añaden a esta y a la Habitacion de la que venimos las correspondientes salidas superior e inferior con las referencias a su vecina.

Al final de cada uno de estos Case actualizamos las variables de fila o columna para continuar desde la nueva Celda en la siguiente iteración. Mencionar también que, en cualquiera de los cuatro casos, si la Celda a la que nos dirigimos ya ha sido visitada simplemente se vuelve a visitar, pero, como es obvio, no se añade de nuevo a ninguna habitación.

Estas sentencias están contenidas dentro de un bloque try-catch para no tener que preocuparnos de los IndexOutOfBoundsException. Simplemente se vuelve al principio del bucle y se vuelve a intentar otro movimiento con un nuevo número aleatorio.

El bucle termina cuando el contador de celdas visitadas es igual o mayor que el límite mínimo que impusimos, siempre y cuando la habitación en la que nos encontremos tenga más de una celda asignada.

### Paso 3:

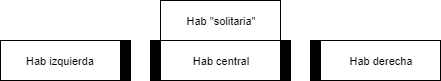
Con esto ya tendríamos una primera aproximación de nuestro nivel creada, pero a pesar de nuestras precauciones, aún hay situaciones durante el bucle en las que se pueden producir habitaciones de una sola celda. Así, la siguiente parte del algoritmo toma el doble array de Celdas con el que trabajamos y lo recorre buscando entre las habitaciones asignadas a las Celdas visitadas aquellas formadas por una sola Celda.

Recorremos dicho array de arriba abajo y de izquierda a derecha según la posición de estas en la pantalla del juego, intentando preferentemente combinar las Habitaciones de una Celda vecinas. Si la Habitación en cuestión no tiene ninguna vecina con la misma casuística, se combina aleatoriamente con la de su izquierda o su derecha. Todo esto controlando los índices del array para evitar excepciones.

De esta forma la Celda es añadida a la otra Habitación y dicha Habitación es asignada a la Celda. Mencionar por último que se efectúa también un cambio de todas las referencias de las Salidas implicadas con esta Habitacion unitaria que desaparece y se combina con la otra.

### Paso 4:

Ya contamos con un nivel correcto, solo falta añadir una referencia que nos permita distinguir si una Habitacion está sola en su piso, está completamente rodeada o está en el lado izquierdo o derecho. Con esto podemos implementar un aspecto clave de la jugabilidad: el hecho de que las balas mueran en los tabiques interiores, pero puedan salir al exterior en los extremos, permitiéndonos disparar a las naves que nos atacan desde el exterior y estas a nosotros.



Para ello, guardamos primero una referencia a la que hasta ahora era la primera Habitacion del ArrayList; esta será donde spawneará el héroe y solo el héroe. A continuación, reordenamos el ArrayList de habitaciones obtenidas en la generación a través de un criterio compareTo de la clase Habitacion. Estas se ordenan de arriba hacia abajo y de izquierda a derecha en Habitaciones en el mismo piso.

Recorremos ahora el ArrayList ordenado comprobando si cada una de las habitaciones tiene una vecina a la izquierda, a la derecha, a ambos lados o en ninguno. Con esto obtenemos la información deseada, la cual se almacena en un array de dos booleanos.