Tecnología de Videojuegos - Team 6

ShutterEarth

Juan Casado Ballesteros
Daniel Fernández Diaz
Jorge Garcia Garcia
Pablo Pardo García

ÍNDICE

El videojuego.

Historia (resumida)

Nuestro juego cuenta la historia de Diana, una militar cuya hija ha sido raptada por un comandante Alien de las fuerzas enviadas a tomar La Tierra. Diana decide dejar su carrera militar y abandonar el ejército a pesar de la guerra, todo para recuperar a su hija.
Durante el juego nos enfrentaremos a toda clase de Aliens y a su más moderna flota, ¿logrará Diana salvar a su hija?

La historia completa se encuentra en los documentos de diseño.

Gameplay y controles.

El juego es un shooter-plataformas con una economía de guerra, (nunca mejor dicho).
Tiene un total de 10 niveles que descubriremos según avanza la historia, no obstante, podremos volver a jugar los niveles que ya nos hayamos pasado pues serán más fáciles y en ellos podremos conseguir balas con menos esfuerzo. Las balas no solo son la munición si no también el "dinero" del juego, debemos ser cuidadosos a la hora de disparar y tener en cuenta la vida de los enemigos para no disparar ni una sola bala de más, las armas más fuertes hacen más daño o tienen menor cadencia de disparo, pero eso solo significa gastar la munición más rápido.
Al pasarnos ciertos niveles obtendremos un arma mejor que podremos empezar a utilizar al seleccionarla dentro del inventario, (Q y E para cambiar de arma) no obstante, para mejorar las armas tanto como para aumentar la salud deberemos gastar balas en la tienda. Comprar un objeto mejor no siempre es una buena idea si ello significa perder toda la munición.
Como hemos dicho el juego es un shooter-plataformas en el que hemos incluido todos los requisitos que planteamos en el diseño inicial, así como alguno extra. Podemos mover al personaje con las WASD o con las flechas y dispara con el espacio, iremos al menú de pausa pulsando back o escape y pasaremos los diálogos con enter. El resto de la interface se utilizará con el ratón y la tecla tab para poder insertar texto en el log-in.
Nuestra misión es eliminar a todos los Aliens de la partida, para ello deberemos saltar y disparar sin que ellos nos toquen son más fuertes y tienen más vida, pero son muy lentos disparando.

Fases

Durante cada partida existirán varias fases en el juego, la idea es que Diana va por los edificios en los que los Aliens han establecido su zona de operaciones para busacar a su hija. Pero según está limpiando la zona los Aliens contraatacan con el grueso de su flota.

Cuanto más rápido derrotes a los soldados Alien más fácil tendrás pelear contra la flota, si somos lentos deberemos cambiar de estrategia, puede que sea mejor idea empezar a esquivar proyectiles antes que lanzarlos.
Al comandante Alien no solo le gusta raptar niñas humanas, también es un apasionado de la magia y jugará con nosotros a lo largo de los niveles hasta que logremos acabar con él.

Programación

En total hemos dividido el código en 5 paquetes:

shutterEarth

Este es el paquete inicial, en el se sitúan 3 clases que podrán ser utilizadas mediante métodos estáticos pues son de interés general para el conjunto del juego. La clase Game inicia el juego, y calcula las constantes de velocidad, tamaño de piso u otras para que el juego se adapte a cualquier tamaño de pantalla.
La clase Media es la encargada de cargar todas las imágenes y de almacenarlas en memoria para que no haya que cargarlas de disco a cada uso.
La clase SavingStation es la encargada de guardar y gestionar las cuentas de usuario para poder continuar la partida donde la dejamos, podremos tener tantas cuantas de usuario como deseemos, las cuentas creadas desde la interface no contarán con privilegios de administrador.

shutterEarth.screens

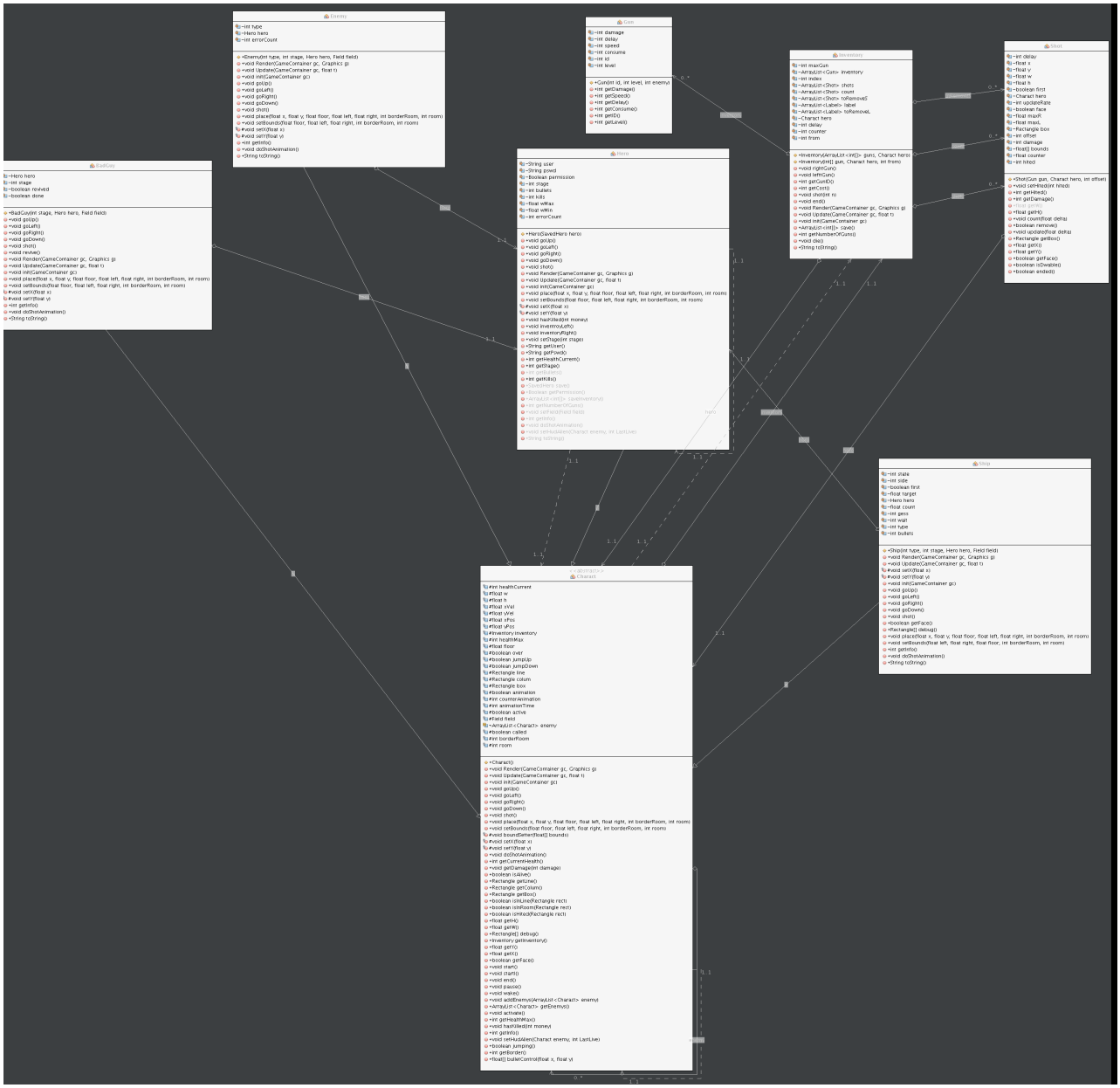
Contiene las clases que representan a cada pantalla de los menús, leen el ratón para saber si ha pulsado en los botones y muestran las imágenes correspondientes.
Destacan la tienda donde podemos comprar modificando los atributos del personaje y la clase Maper donde podemos elegir el nivel a jugar, esta clase tiene un papel activo en la historia y se modifica al avanzar en ella.
En la clase Access podemos entrar con nuestra cuenta y en Register crear una, recordemos que con tab podremos ingresar el texto en los fields.

shutterEarth.characters

Este paquete contiene los personajes, el héroe, una clase para los aliens de suelo, otra para las naves y otra para el malo final. Su IA es sencilla y simplemente intenta dispararnos aunque tienen un factor de aleatoriedad que les hará fallar los tiros, alejarse de nosotros si nos acercamos o esquivar las balas.

Cada personaje tiene implementada su IA dentro su misma clase por comodidad, más adelante explicamos como hacemos esto, cuando disparan generan objetos Shot que son controlados desde Inventory, cada personaje tiene su propia instancia de Inventory.

Todos los personajes heredan de Charact.



Vemos como los personajes heredan de Character, los malos contendrán una instancia del héroe, su parte abstracta es la que se comunica con el inventario y este con las balas y las arma, cuando disparamos empieza un timer para la latencia del arma y cuanto el disparo es apto para ser realizado avisa al personaje para que haga la animación.
Los personajes cobran vida gracias a sprites que los animan. Los controlamos modificando su velocidad de movimiento, de este modo producimos comportamientos realistas como gravedad o mismo tiempo en recorrer la pantalla sin importar el tamaño de esta. También realizamos un control para que los personajes se muevas siempre a la misma velocidad sin importar los fps.

shutterEarth.Map

Estas son las clases que van a formar elementos de mapa como el menú de textos donde se cuenta la historia, el menú de pausa, el HUD del juego, el mapa en si mismo o la clase BB que representa a la hija perdida.
Lo interesante de este paquete es la clase abstracta Map que define la forma de comunicación entre los personajes y los mapas, hemos usado varios en la fase de pruebas, pero al final nos hemos quedado con uno solo.

shutterEarth.Map.randomGenerator

En este paquete se define el mapa que utilizamos, es un mapa que se genera aleatoriamente para lo que usa distintos objetos. Cada generación de mapa será diferente completamente de la anterior. Este mapa, como los otros hereda de Map y se comunica con el personaje de forma estándar. La generación del nivel se explicará con más detalle a continuación.

Curiosidades de implementación

El aspecto más destacable de nuestro programa es que creamos una capa de abstracción encima de slick2d modificando el uso de su render, update e init, esto lo hacemos con la clase abstracta Scene. Esta clase en si es un ArrayList<Scene> en el que introduciremos todas las clases que implenten de ella según sea necesario, ahora el render, update e init de slick llamarán en orden a los render, update e init de las escenas, de este modo jugando con el orden de ellas en el ArrayList podemos hacer que lo que dibuja la clase personaje se haga encima del mapa y no debajo. Con ello logramos también que cada clase que lo necesite tenga sus propios métodos de actualización y dibujo, y no que haya uno enorme que haga llamadas a las clases.

Para el desarrollo y debuggeado hemos creado cuentas especiales de desarrollador que lanzaban un hilo que nos permitía mediante unos comandos ver parámetros del juego como las clases activas,

posiciones de objetos, el inventario del personaje… y modificarlos a nuestra voluntad en tiempo de ejecución lo cual ha sido un gran ayuda.

Generación de niveles

Se ha diseñado e imlementado 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.

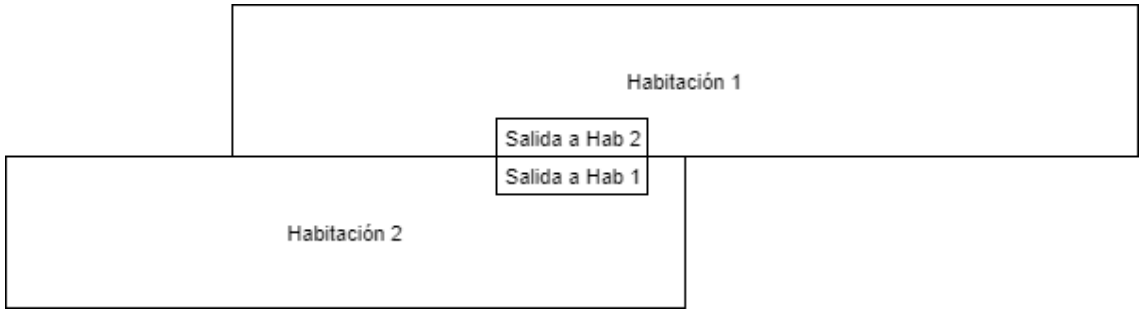
En cada instancia de esta clase encontramos, como atributos principales:

- 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.



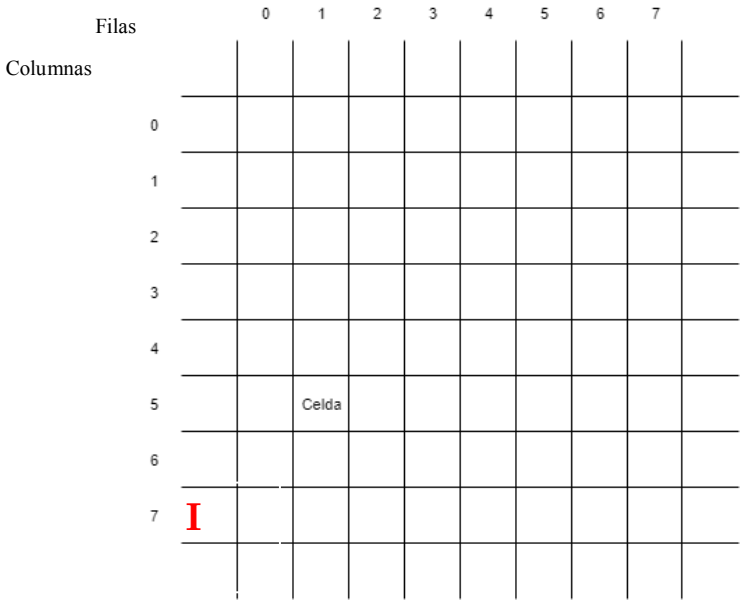
La generación:

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.

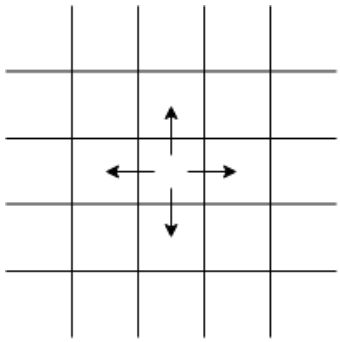
Inicializamos también una rejilla de Celdas, las cuales almacenarán si han sido visitadas y la Habitacion a la que han sido asignadas (además de su tesela de fondo como vimos antes), así como dos variables más que almacenarán la fila y la columna en la que nos encontremos en cada iteración.



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:

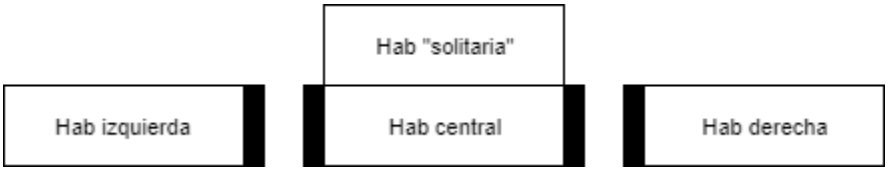
Con esto ya tendríamos una primera aproximación de nuestro nivel creada, pero a pesar de nuestras precauciones, aún hay situaciones durante el bucle en las que se pueden producir habitaciones de una sola celda. Así, la siguiente parte del algoritmo toma el doble array de Celdas con el que trabajamos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



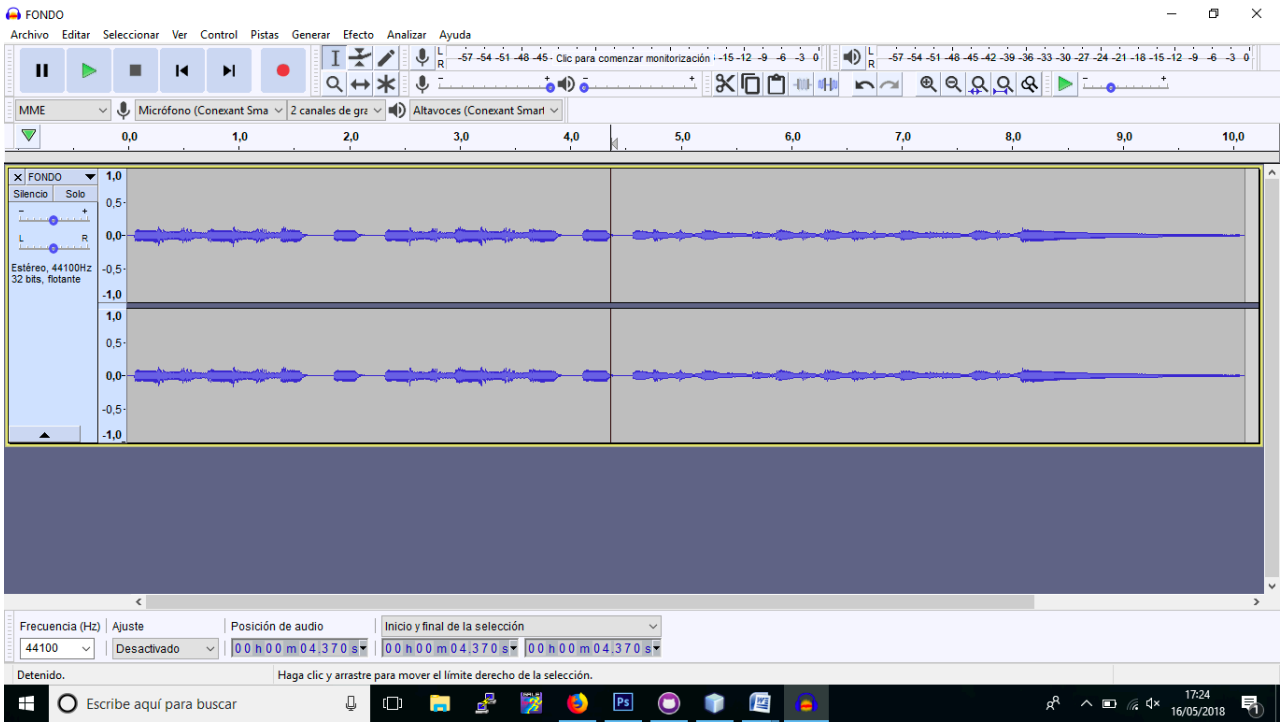
Para ello, guardamos primero una referencia a la que hasta ahora era la primera Habitacion del ArrayList; esta será donde spawneará el héroe y solo el héroe. A continuación, reordenamos el ArrayList de habitaciones obtenidas en la generación a través de un criterio compareTo de la clase Habitacion. Estas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

Media

Efectos de sonido y canciones

En cuanto a la banda sonora, para grabar el tema principal he utliiado un piano eléctrico y le he aplicado una serie de distorsiones y modifcaciones las cuales han servido para poder ambientar el juego en un mundo post-apocalíptco. Para realiiar esta tarea, me he basado en el programa “Audacity” el cual es Open Source y ofrece las opciones necesarias para poder desarrollar mi tarea. A contnuación se incluyedel proceso de edición del audio incorporado:

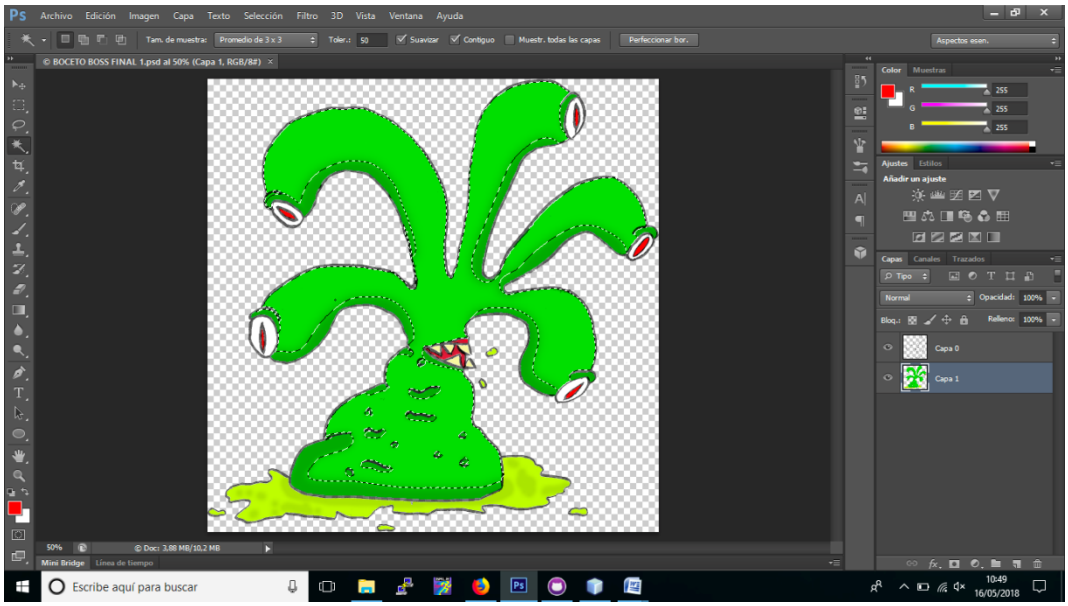


Con respecto al sonido hemos intentado en todo momento que fuera acorde con el tema del juego, mayor parte de él es creación propia, aunque algunos sonidos los hemos sacado de repositorios de sonidos sin copiright y de un canal de Youtube que realiza pequeños cortos de música en 8 bits también sin copiright.

Aliens y fondo

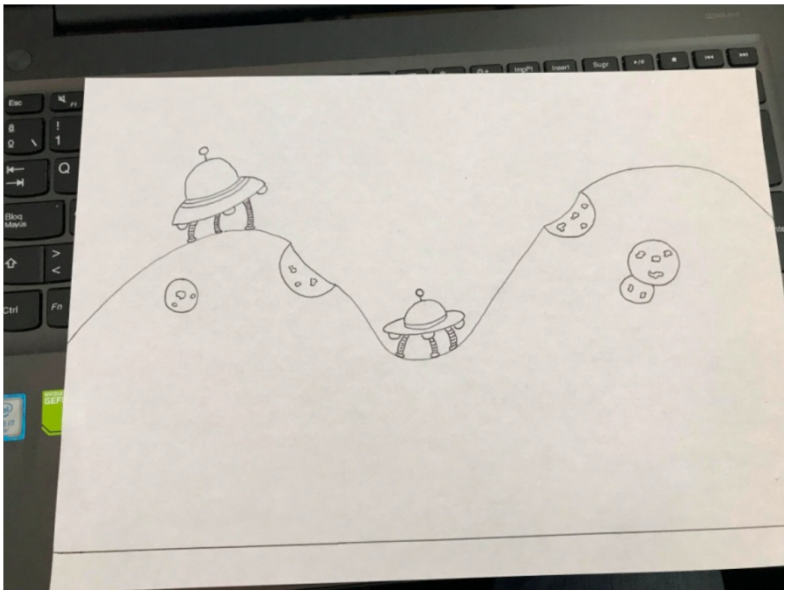
En cuanto al apartado gráfcoscabe destacar la creación de diferentes fondos, personajes, edifcios, etc… los cuales han sido creados de forma manual y posteriormente escaneados para su posterior adaptación en el videojuego. A continuación se muestran varios programas utilizados para el desarrollo de los mismos.

- Adobe Photoshop(Boceto del ENEMIGO BASE pasado a photoshop y pintado)

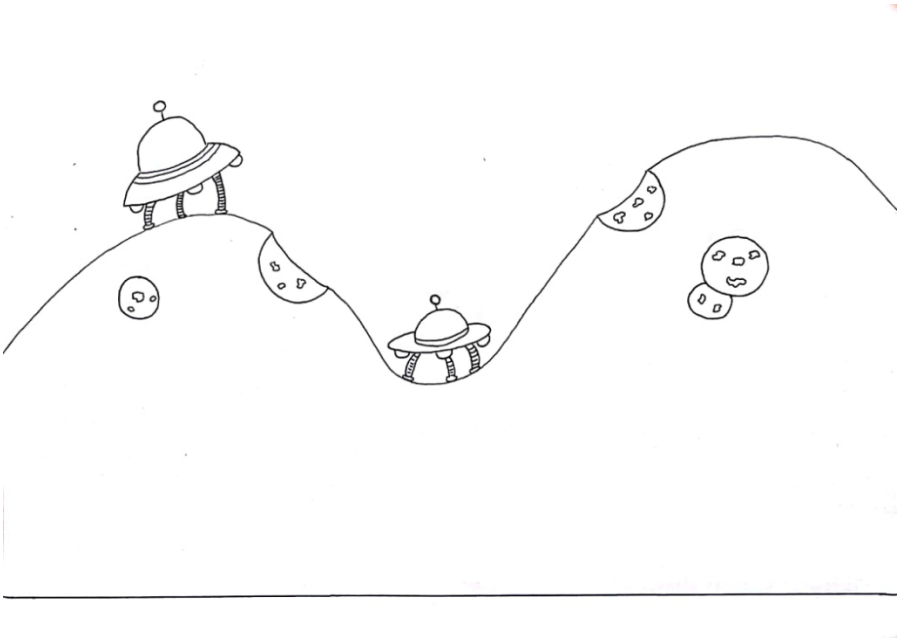


Los fondos los hemos creado de la siguiente forma:

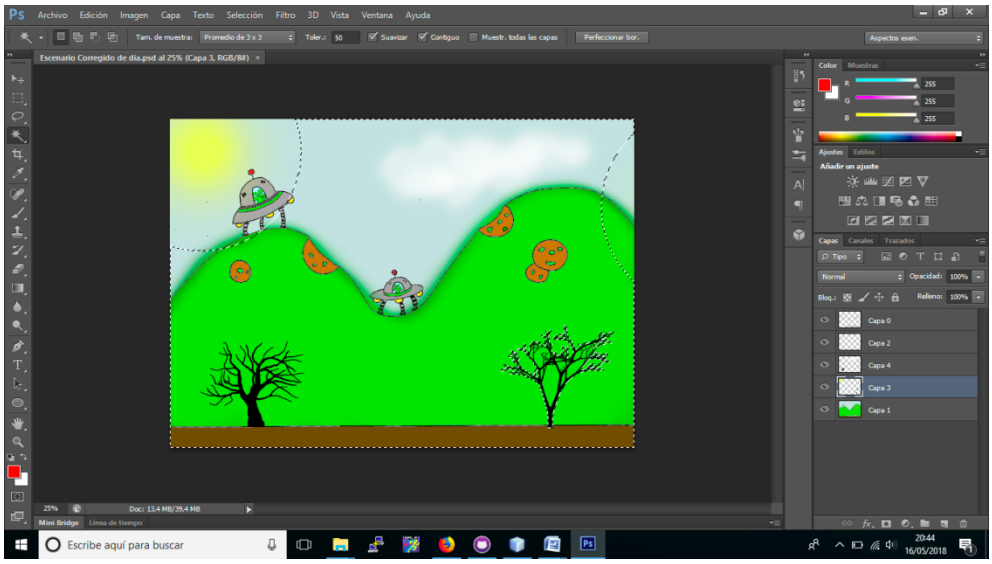
1.) Creación del boceto en papel.



2.) Escaneado.

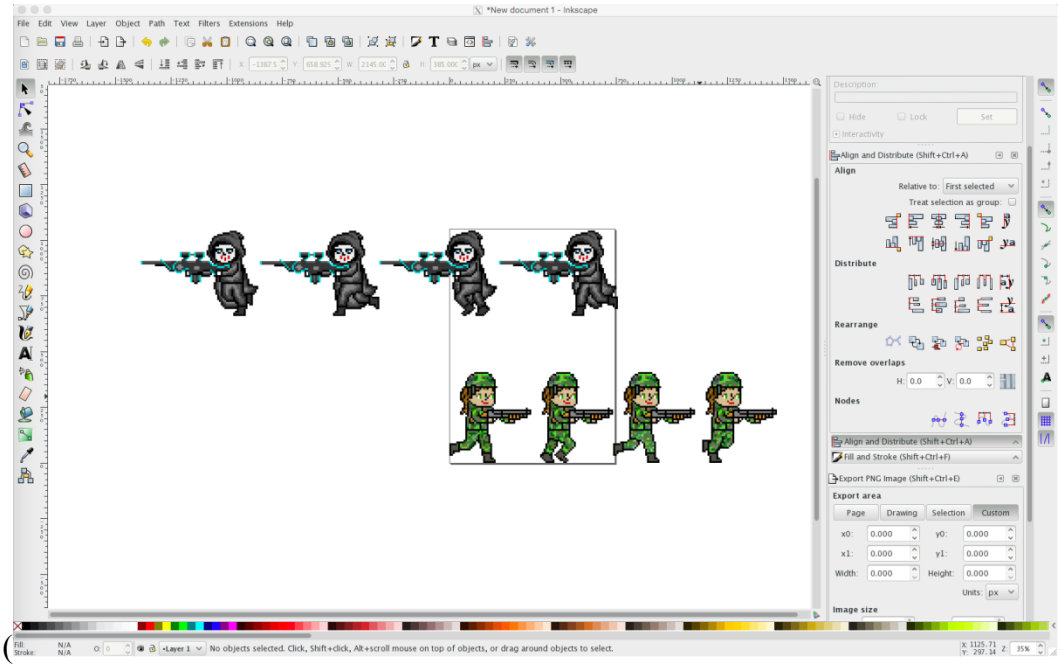


3.) Coloreado digital.



Montaje de Sprites

- InkScape



Con respeto a la creación de las imágenes hemos procurado en todo momento hacerlas de forma que al meterlas en el juego perdieran la menor calidad posible.

Diseño de los personajes

Para la creación de los personajes se ha usado un programa de pixel art llamado “Pixela.”
Es un editor de pixel art muy fácil de usar, con un sistema de capas, que se pueden superponer y que he encontrado perfecto para la creación de los Sprites del personaje ya que me permite juntar capas para crear el movimiento.

Diana:

Para la creación del personaje de Diana nos hemos basado en sprites ya creados por la comunidad. A partir de ahí hemos modificado la cara y vestuario para adaptarlo al personaje de la historia dando como resultado el personaje de la fotografía.

El Jefe Final:

Para la creación del jefe final nos hemos basado en unos modelos para crear la parte superior del vestuario (Capucha). También nos hemos inspirado en un modelo para el arma.
Para la creación de la parte inferior he reutilizado las capas ya creadas de Diana, hemos cambiado el color y añadido unas animaciones para el movimiento de la capa del personaje.

Las Armas:

También hemos diseñado de las armas de los personajes con Pixela

Armas para Diana:

- Una pistola débil

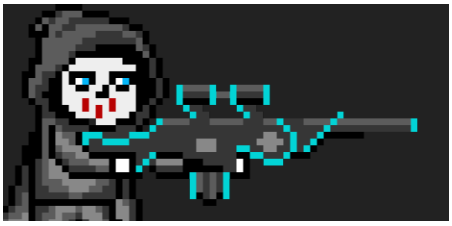
- Una pistola más potente.

-Una escopeta

-Un fusil

-Y por último un arma espacial inventada.

Arma del Jefe final:



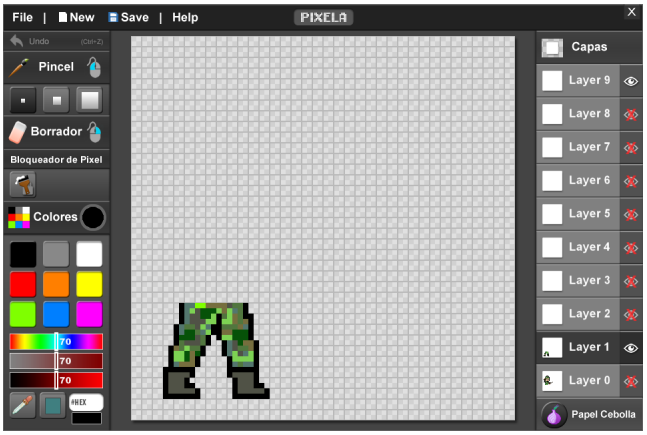
El proceso:

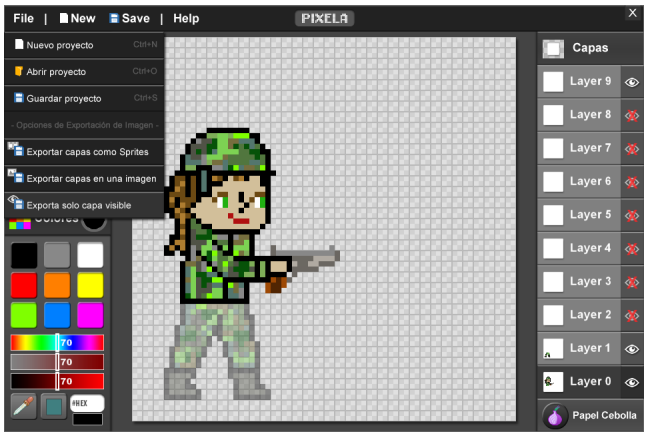
A continuación detallo, un ejemplo, en el proceso de creación de una imagen de Diana en el programa “Pixela.”

En la primera capa (capa 0) creamos el cuerpo de Diana.

Tenemos una paleta de colores predefinidos y la posibilidad de crear cualquier color con los marcadores que se ven abajo a la izquierda.

En la siguiente capa (capa 1) creamos la animación de las piernas, en este caso la animación de reposo.





Al final hacemos que ambas capas sean visibles. Para guardar la imagen utilizamos la opción de exportar solo la capa visible que uniría las dos capas que hemos creado, de este modo obtenemos un fichero formato “.png” que será posteriormente montado con otras imágenes para crear los Sprites.

Repetimos este proceso, creando nuevas imágenes de las diferentes posiciones de las piernas, en diferentes capas y las guardamos de la misma forma.
Para el montaje de las diferentes armas, el proceso ha sido editar solamente la primera capa (capa 1), cambiar el arma e ir uniendo la nueva capa 1 del cuerpo con las demás capas de las diferentes posiciones de las piernas ya creadas y de este modo crear los Sprites de cada movimiento con las diferentes armas.

Se pueden encontrar todo lo creado de medios audiovisuales en la carpeta de media en el proyecto de NetBeans o organizado en las carpetas de repositorio en la sección de Arte.