

UNTIL THE LAST NOTE



Álvaro Zamorano Ortega
Miguel Ángel Losada Fernandez
Ignacio Rubio Valverde
Abel de Coca Torres
Miguel Escuderos de la Morena

ÍNDICE

1.Introducción.....	3
2.Fases del videojuego.....	3
3.Código.....	5
4.Recursos artísticos.....	16
5.Conclusión.....	20

1. INTRODUCCIÓN

El videojuego que hemos desarrollado para esta asignatura se denomina *Until The Last Note*. Se trata de un PixelArt de perspectiva 2D, en el cual vamos investigando a través de varias salas (camerinos y pasillos) hasta encontrarnos con los “jefes”, en el que se desarrollará un combate por turnos, en el que el personaje principal atacará y el enemigo responderá.

El juego trata el problema actual de la música, la cual la tendremos que salvar de las terribles garras en la que se encuentra con los géneros actuales. Para ello, elegiremos al principio entre tres personajes distintos, el cual cada uno de ellos se caracterizará por representar un género musical (clásico) distinto. A partir de aquí, un narrador nos contará una pequeña introducción y nos guiará en esta intensa aventura, en la que tendremos que derrotar a los géneros musicales que están destruyendo el arte musical. A lo largo de estos duelos, tendremos que analizar a cada personaje y usar la estrategia como arma principal para derrotar hasta el último jefe.

Por otro lado, en el apartado relacionado con la programación, hemos desarrollado el videojuego en lenguaje Java, utilizando el motor Slick2D con las librerías de Java y usando Netbeans como IDE. Asimismo, el equipo artístico se ha ayudado de Adobe Photoshop y de Sibelius basándose en obras de 8bit, para dar más realismo y dinamismo al juego.

Por lo tanto, *Until The Last Note* se trata de un videojuego con un intenso apartado gráfico y sonoro, en el que se narra la aventura musical de un personaje que deberá vencer a varios enemigos para devolver a la música a su anterior estado de esplendor.

2. FASES DEL VIDEOJUEGO

Las clases principales del videojuego extienden de la clase `BasicGameState`, por lo que *Until The Last Note* es un videojuego en el cual cada sala será un estado diferente a los demás. Por lo tanto, cada sala (estado) se identificará con un número y será administrado de forma independiente a los demás para mayor comodidad a la hora de programar, manteniéndolos de forma más sencilla.

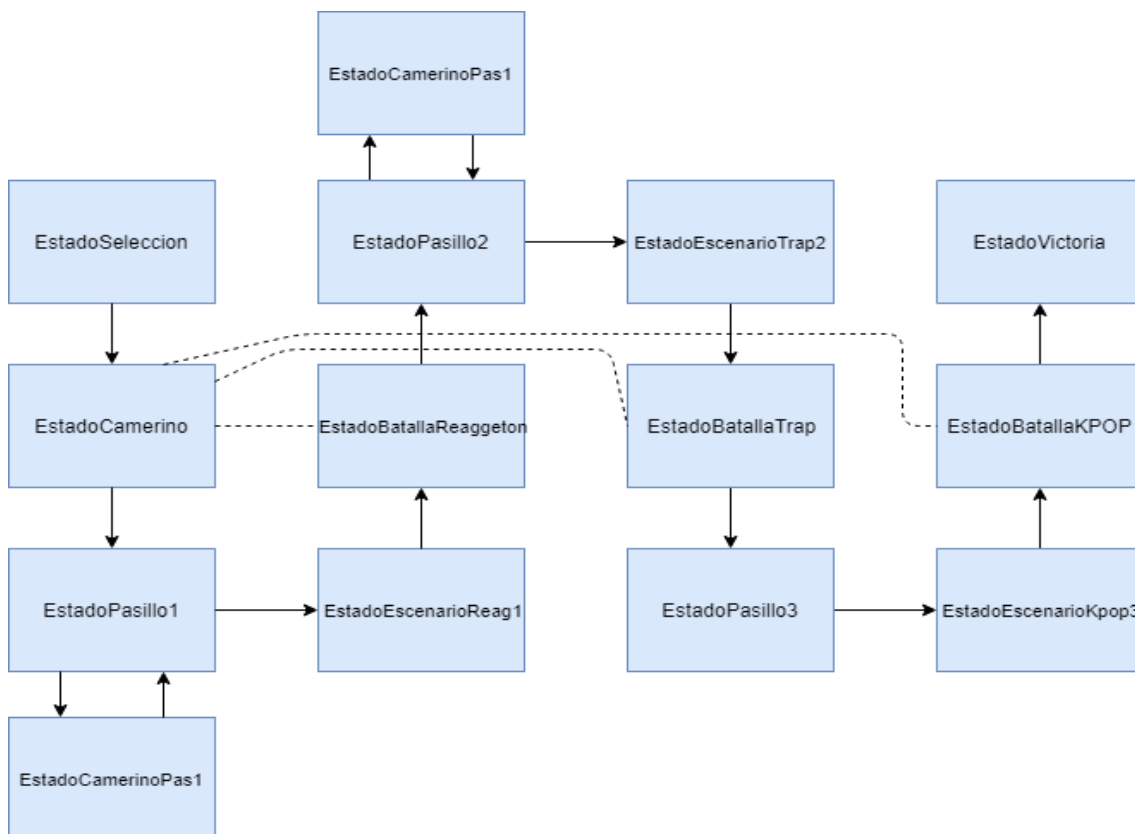
En primer lugar, en el menú principal hay tres opciones: JUGAR, PERSONAJES Y SALIR. En la opción JUGAR iremos directamente al juego en sí, que explicaremos posteriormente. En la segunda opción, nos saldrá la información de cada personaje del videojuego, con sus características principales y que diferencian a cada uno de ellos a la hora del combate contra los jefes. Y finalmente, en la opción SALIR, saldremos directamente del videojuego, sin ningún tipo de guardado.

Si la opción es la de JUGAR, inicialmente tendremos que elegir al personaje principal de esta aventura. Para ello, los controles serán ENTER para pasar los diálogos, las flechas del teclado para elegir el personaje y ENTER para seleccionar el personaje.

Posteriormente, el jugador saldrá en su camerino, en el cual nos hablará un narrador que nos acompañará a lo largo del videojuego. Para movernos usaremos las flechas del teclado o las teclas WASD para desplazar al personaje hasta la puerta. Como curiosidad, cada personaje tendrá un baile distinto, dando a la tecla B.

De este modo saldremos en el primer primer pasillo, y aquí tendremos que averiguar la entrada a otro camerino para encontrar una poción de vida que se usará mas adelante en los combates. Si seguimos por la puerta de la derecha del pasillo, entraremos al primer escenario, en el cual nos encontraremos al primer jefe. Las mecanicas de la batalla son bastante simples. Nos saldrá un HUD referido a los movimientos del personaje principal, y a través de las flechas del teclado elegiremos ATACAR o HUIR. Si la opción es ATACAR (seleccionando por medio de la tecla ENTER), elegiremos el ataque que más nos convenga en ese momento, atendiendo a los usos de cada ataque, su daño y la vida del enemigo. A su vez, podremos recuperar vida (si se posee pociones de regeneración) o aumentar el daño de cada ataque por medio de las teclas V o F respectivamente. Si no se posee usos del ataque seleccionado, se perderá el turno. Al realizar el ataque, el enemigo reponderá con otro o utilizará pociones, dependiendo de su vida. Finalmente, se pasará al siguiente estado si se gana la batalla, o se perderá todo lo realizado y se comenzará de nuevo en el camerino inicial.

Por tanto, esta mecánica de juego se repetirá con los dos jefes siguientes, en los que entremedias se podrá encontrar diferentes pociones de salud y de daño para poder vencer a los siguientes enemigos. Un dato importante es que, tras la batalla contra el segundo jefe, se regenerará la salud y los usos de los ataques del personaje principal.



Transición de estados

3. CÓDIGO

El código principal del juego se encuentra en el paquete *ideavidejuego*. Sus clases principales son las siguientes:

Personaje.java

Esta clase guarda todo lo referente a los personajes del videjuego, tanto al protagonista como a los enemigos. Para ello, almacena todos los atributos relacionados con los personajes:

- Nombre del personaje.
- Vida actual y vida máxima del personaje.
- Ataques del personaje
- Animaciones del movimiento del personaje, tanto el movimiento a la derecha, a la izquierda, así como el baile (atributos a null en enemigos).
- Música de pasillos y de combate del personaje (atributo a null en enemigos)
- Instancias de la clase Image (imagen) para el HUD de combate de cada personaje y su cuadro de diálogo (atributo a null en enemigos).
- Número de pociones de vida y de daño que posee en cada momento cada personaje.

Atendiendo a sus métodos, aparte de los correspondientes *get* y *set* para obtener y establecer los valores a cada atributo, posee los siguientes métodos:

- `public boolean useHealthPotion()` : este método es usado en el combate musical y sumará 75 puntos de vida a la vida actual. Para ello, retornará un valor booleano `true` si lo ha realizado con éxito (restando en uno las pociones de vida que tenga) o `false` si no le quedan pociones de vida.
- `public boolean useDmgPotion()` : al igual que el anterior, pero en vez de recuperar salud, aumentará el daño de cada ataque.
- `public String atacar(Personaje penemigo, int seleccion)` : se trata del método principal del ataque del personaje principal al enemigo. Para ello, se deberá meter como valores de entrada al personaje enemigo y un número que identificará el ataque que haya seleccionado el jugador. Si al ataque elegido le quedan usos, se le restará a uno el número de usos que tenga y, si ha sido realizado con éxito (explicación en la clase Ataque), el ataque podrá ser normal (daño normal) o daño crítico (explicación en la clase Ataque), restando el daño hecho al enemigo. De esta forma, sonará el sonido característico del ataque.
Finalmente, este método devolverá un texto (String), en el que aparecerá el ataque realizado, si ha sido realizado con éxito, si ha sido crítico y el daño que ha hecho al enemigo.

- `public String ataqueEnemigo(Personaje personajeBueno)` : se trata del método a partir del cual el enemigo ataca al personaje protagonista (valor de entrada).
Este método se diferencia del anterior en que el personaje enemigo elige un ataque de forma aleatoria, eligiendo siempre un ataque que le queden usos a través de una estructura *while()*. En lo referente a si ha sido acertado o no y si es crítico o no, es exactamente igual al método anterior.
- `public boolean Probabilidad(int x)` : se trata de un método que devolverá `true` o `false` en función de si 10 números generados aleatoriamente (1 ó 0) y su suma es mayor (`true`) o menor (`false`) que el valor de entrada `x`. Este método se usa cuando en la batalla, el personaje enemigo tiene una probabilidad de tomar un poción de salud o de daño.

Ataque.java

Esta clase guarda todo lo referente a los ataques de los personajes. Sus atributos principales son:

- Nombre del ataque
- Daño: número que indica el daño que resta vida al personaje.
- Usos y usos máximos: número de intentos actuales que se puede realizar el ataque y número de intentos máximos.
- Sonido que suena al realizar el ataque.
- Probabilidad de fallo del ataque.

Atendiendo a sus métodos, aparte de los correspondientes *get* y *set* para obtener y establecer los valores a cada atributo, posee los siguientes métodos:

- `public boolean isAcertado()` : este método devuelve `true` o `false` en función de si la suma de diez números (multiplicada posteriormente por 10) aleatorios entre el 0 y el 1 es mayor que la probabilidad de fallo (`true` si es mayor).
- `public boolean isCritico(int x)` : este método devuelve si el ataque es crítico o no (el daño del ataque se multiplica por 2). Al igual que en el anterior método, si la suma de diez números (multiplicada posteriormente por 10) aleatorios entre el 0 y el 1 es mayor que el valor de entrada `x` (probabilidad de fallo), se devolverá `true`.

ClaseEstatica.java

Esta clase guarda al personaje principal elegido y al enemigo actual. A su vez, nos sirve de ayuda para guardar la música que suena en el menú inicial del videojuego, el sonido de los pasos del personaje al moverse, y también en el

combate, guardando si el último ataque ha sido acertado o no, y en caso de haber sido acertado, se guardará el ataque realizado. Finalmente, también nos sirve de ayuda para guardar el último estado en el que se encontró el personaje protagonista para hacer más dinámica la transición entre estados.

ClasePunto.java

Esta clase guarda las coordenadas de los puntos en los que situamos las imágenes en los diferentes estados.

ClaseSprite.java

Esta clase hereda de la clase Image, y representa a los diferentes Sprites que colocamos en los diferentes estados. Tiene como atributo principal una instancia de la clase Punto, en el cual se guardan las coordenadas en las que se sitúa la imagen.

ESTADOS:

Principal.java

Se trata de la clase principal del juego, a partir de la cual se ejecuta el programa. Esta clase extiende de StateBasedGame, de modo que en esta clase añadimos todos los estados que conforman el videojuego.

En esta clase aplicamos el tamaño de la ventana en la que se ejecutará el videojuego. De este modo, fijamos el tamaño en 1080x720, y para una mayor comodidad y visualización, establecemos que no se visualicen los FPS.

EstadoMenu.java

Se trata del estado que visualiza el menú. Hereda de la clase BasicGameState, por lo que sus principales métodos son los de una clase típica del motor Slick2D. En el método *init* inicializamos el fondo, la música y las imágenes y en el bucle render lo visualizamos por pantalla.

En el bucle update, actualizamos la posición del cursor dependiendo de las opciones seleccionadas por el jugador, cambiando a otro estado cuando se teclaa la tecla Enter.

EstadoMenu.java

Este estado visualiza la información de cada personaje por pantalla, pudiendo cambiar de personaje con las teclas del teclado. Para poder volver al menú principal, basta con dar click a la tecla Escape.

EstadoSeleccion.java

Se trata del estado en el cual comienza el videojuego en sí. En este se inicializan todos los posibles personajes protagonistas que el jugador podrá elegir, así como sus ataques.

En primer lugar, habrá una pequeña introducción que se podrá pasar con la tecla Enter y después de haber pasado 1 segundo entre las diferentes frases. Esto lo hacemos gracias al valor de *delta* del método *update*. Posteriormente, se podrá elegir al personaje, el cual se representa por un indicador que va variando al cambiar de personaje con las teclas del teclado. Al seleccionarlo por medio de la tecla Enter, el personaje se guardará en la clase ClaseEstatica.java , y se producirá una transición al siguiente estado.

EstadoCamerino.java

En este estado se comienza con el personaje elegido anteriormente en un camerino, en el que el narrador nos cuenta una pequeña introducción que se podrá ir pasando con la tecla Enter. Al acabar esta introducción, nos podremos mover por el camerino y cambiar de sala (estado) por la puerta.

Todo el movimiento del personaje lo hemos realizado en el bucle update de cada estado (menos los estados de batalla), utilizando el valor de delta y multiplicándolo por una velocidad de 4f. De este método, los valores de x e y (coordenadas en las que se encuentra el personaje) se sumarán o restarán el valor de delta dependiendo si van a la derecha, izquierda, arriba o abajo. Si el personaje se mueve hacia la derecha, arriba o abajo, se utilizará las animaciones que miran hacia la derecha, y al revés si el personaje se mueve hacia la izquierda. Como curiosidad, si el jugador teclea la tecla B, el personaje hará un baile característico de su estilo musical.

Las colisiones que hemos realizado para que el personaje no se salga de los límites de la sala son simples. Se basan en que el personaje no puede moverse si sobrepasa un determinado valor de x e y, así como se cambiará de estado si las coordenadas en las que se encuentra el personaje coinciden con las coordenadas en las que se cambia de sala.

Por último, la música que sonará dependerá del personaje elegido, y estará será igual tanto para camerinos, pasillos y escenarios.

EstadoPasillo1.java, EstadoPasillo2.java, EstadoPasillo3.java

Estos estados los hemos agrupado debido a sus similitudes. En cada uno de ellos habrá una pequeña introducción y posteriormente el personaje se podrá mover, pudiendo entrar a otros camerinos y al siguiente escenario.

En estos estados se inicializan a los enemigos, así como a sus ataques.

Por último, al entrar en otros camerinos (EstadoCamerinoPas1), dependiendo del pasillo en el que se encuentre el personaje, pasará a un camerino o a otro, por lo que esto lo indicamos dándole el valor necesario al atributo UltimoEstado de la ClaseEstatica.

EstadoCamerinoPas1.java

Este estado representa a los diferentes camerinos a los que se pueden entrar en los pasillos. Por ello, cada camerino será distinto dependiendo del valor del atributo UltimoEstado de la ClaseEstatica.

Como en todos los estados, habrá una pequeña introducción y se podrá mover al personaje a través de él.

Dentro de este estado, existe otro tipo de colisiones entre el personaje y las diferentes pociones (este tipo de colisión también existe en los estados de escenarios). Básicamente, el personaje principal tiene en la parte de los pies un rectángulo (instancia de la clase Rectangle), al igual que la poción, y si estos rectángulos interseccionan entre sí (por medio del método *intersects()*), existirá una colisión, y en este caso, se sumará en uno el número de pociones del personaje, así como desaparecerá la imagen de la poción.

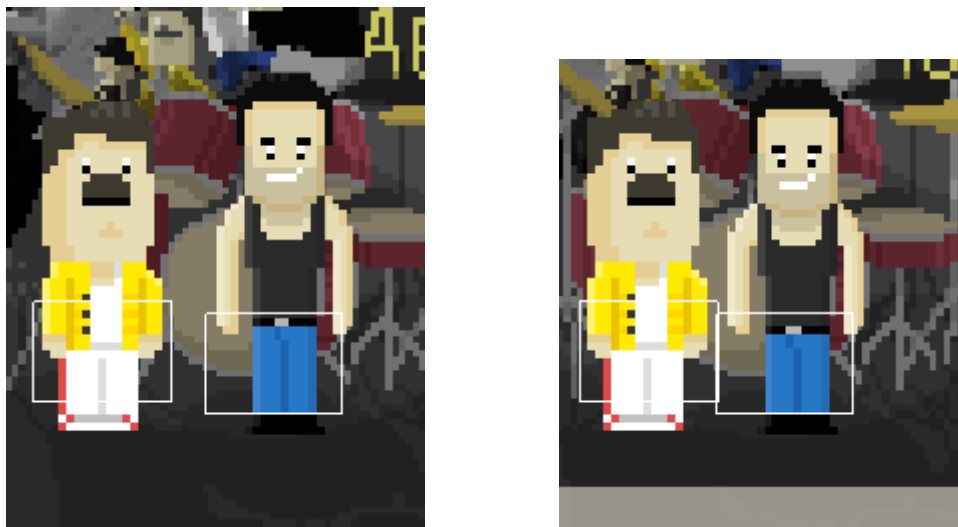


Simulación de la colisión de figuras geométricas entre el personaje y la poción

EstadoEscenarioReag1.java, EstadoEscenarioTrap2.java, EstadoEscenarioKPOP3.java

Estos estados representan a un escenario como parte inicial del combate entre el protagonista y el jefe. Para ello, habrá una pequeña introducción inicial y el personaje se podrá mover por el escenario. A su vez, también habrá pociones que el personaje podrá coger.

La colisión entre los diferentes personajes se realiza de forma exactamente igual a lo explicado anteriormente entre las pociones y el personaje, con la diferencia de que, en este caso, si existe una intersección entre los rectángulos, habrá un pequeño diálogo entre los personajes, para pasar posteriormente al siguiente estado (combate).



Simulación de la colisión de figuras geométricas entre personajes

EstadoBatallaReaggeaton.java, EstadoBatallaTrap.java, EstadoBatallaKPOP.java

En estos estados se desarrolla la batalla entre el protagonista y los diferentes bosses. Para ello, se visualizará por pantalla el HUD propio del protagonista (con sus respectivos ataques), y una animación del protagonista de espaldas y del boss de frente (tipo pokemon). A su vez, sonará la música de batalla característica del personaje protagonista.

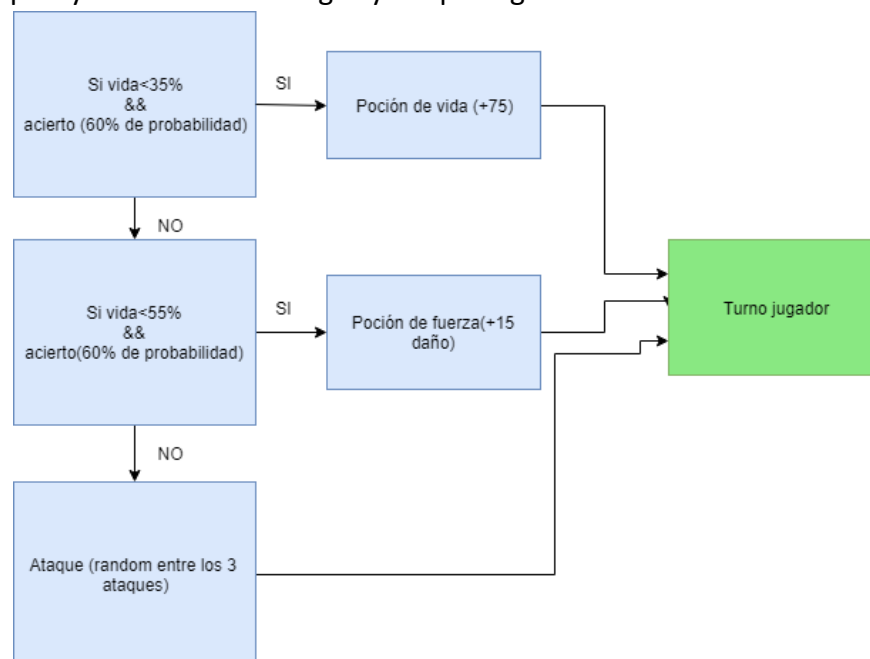
Para ello, inicialmente tendremos las opciones de atacar y huir (representadas por un indicador). Si la opción es huir, no hará nada, ya que por falta de tiempo no lo hemos implementado. Si se elige la opción de atacar, se podrá elegir entre los tres diferentes ataques que posee el protagonista. Al cambiar de opción, se podrá visualizar el daño del ataque, los usos que le quedan, el porcentaje de acierto del ataque, y las pociones de vida y de daño que le quedan al jugador. Al seleccionar por medio de Enter el ataque (o de las teclas V y F para usar pociones de vida o de fuerza), se llamará al método *atacar()* de la

clase personaje (o a los métodos *useHealthPotion()* o *useDmgPotion()*). Al realizar el ataque, sonará el sonido característico del ataque, y si ha sido fallado, el sonido de fallo, y se visualizará el daño realizado al enemigo, si ha fallado o si ha sido crítico. Finalmente, si elegimos un ataque que no dispone de usos o de una poción que no nos queda, perderemos el turno.

Para saber si el turno lo tiene el protagonista o el enemigo, nos ayudaremos de: un booleano que pondremos a false cuando el jugador haya realizado el ataque, y del atributo delta, que irá sumando al valor de un entero *dato* para fijar el tiempo que dura los sonidos de los ataques (si llega a un determinado tiempo, que es lo que dura el sonido del ataque, se pondrá el booleano a false o a true dependiendo de quien haya atacado), y que se pondrá a 0 cuando haya terminado el ataque del personaje.

Referido al ataque del enemigo, este usará una poción de vida si su porcentaje de salud es menor del 35% y si acierta (probabilidad de acierto del 60%). Si no lo ha realizado con éxito, podrá tomar una poción de fuerza si su porcentaje de vida es menor del 55% y si acierta (probabilidad de acierto del 60%). Si no se ha realizado ninguna de estas dos con éxito, llamará al método *ataqueEnemigo()*, y se visualizará el daño realizado al protagonista, si ha fallado o si ha sido crítico.

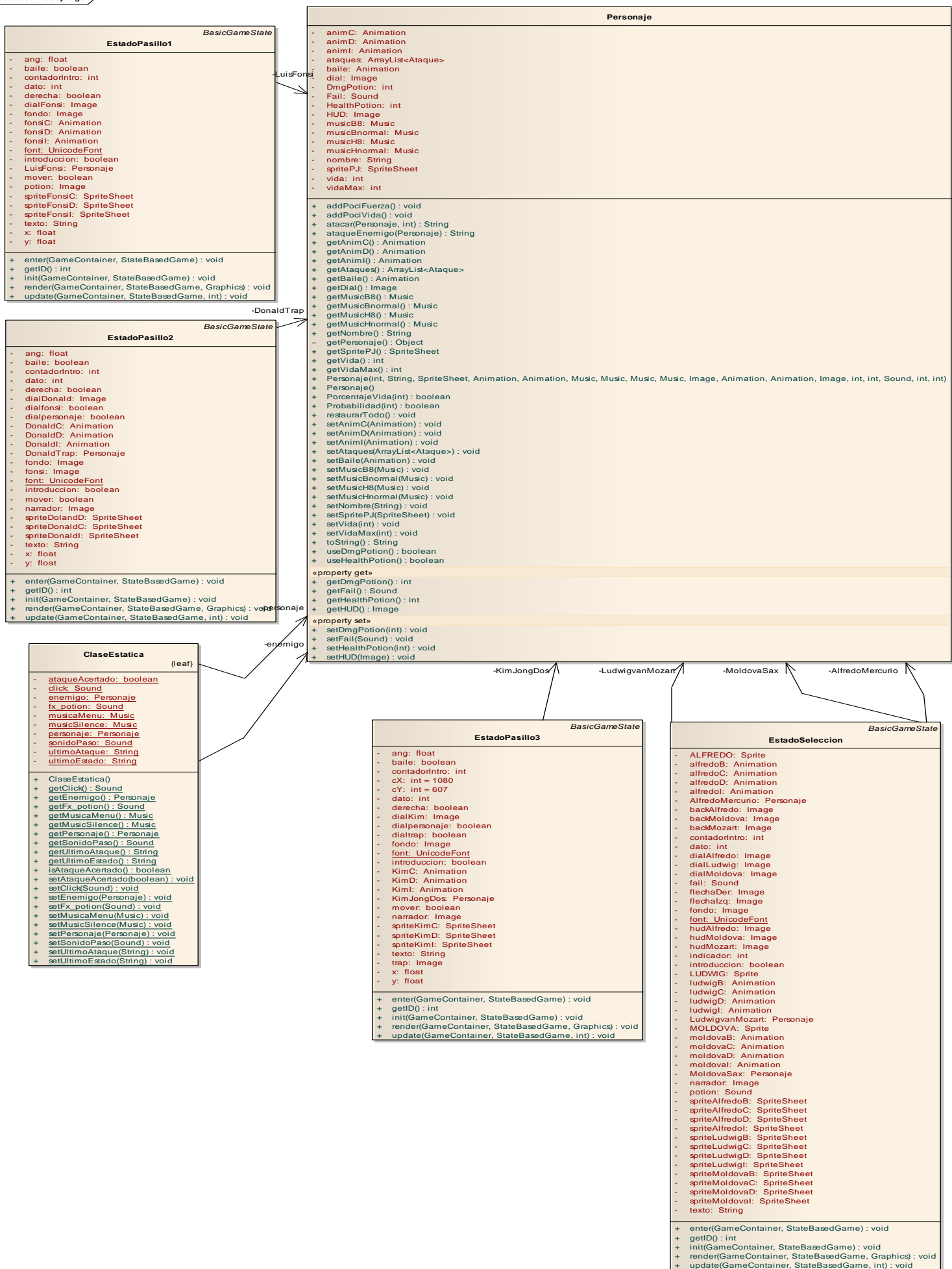
Finalmente, si se ha derrotado al enemigo, se pasará al siguiente estado correspondiente (pasillos), y en caso de derrotar al enemigo final, se producirá una transición al estado de victoria. Sin embargo, si cualquiera de los enemigos derrota al jugador, se pasará directamente al camerino inicial, restaurando todos los ataques y vida de los enemigos y del protagonista.

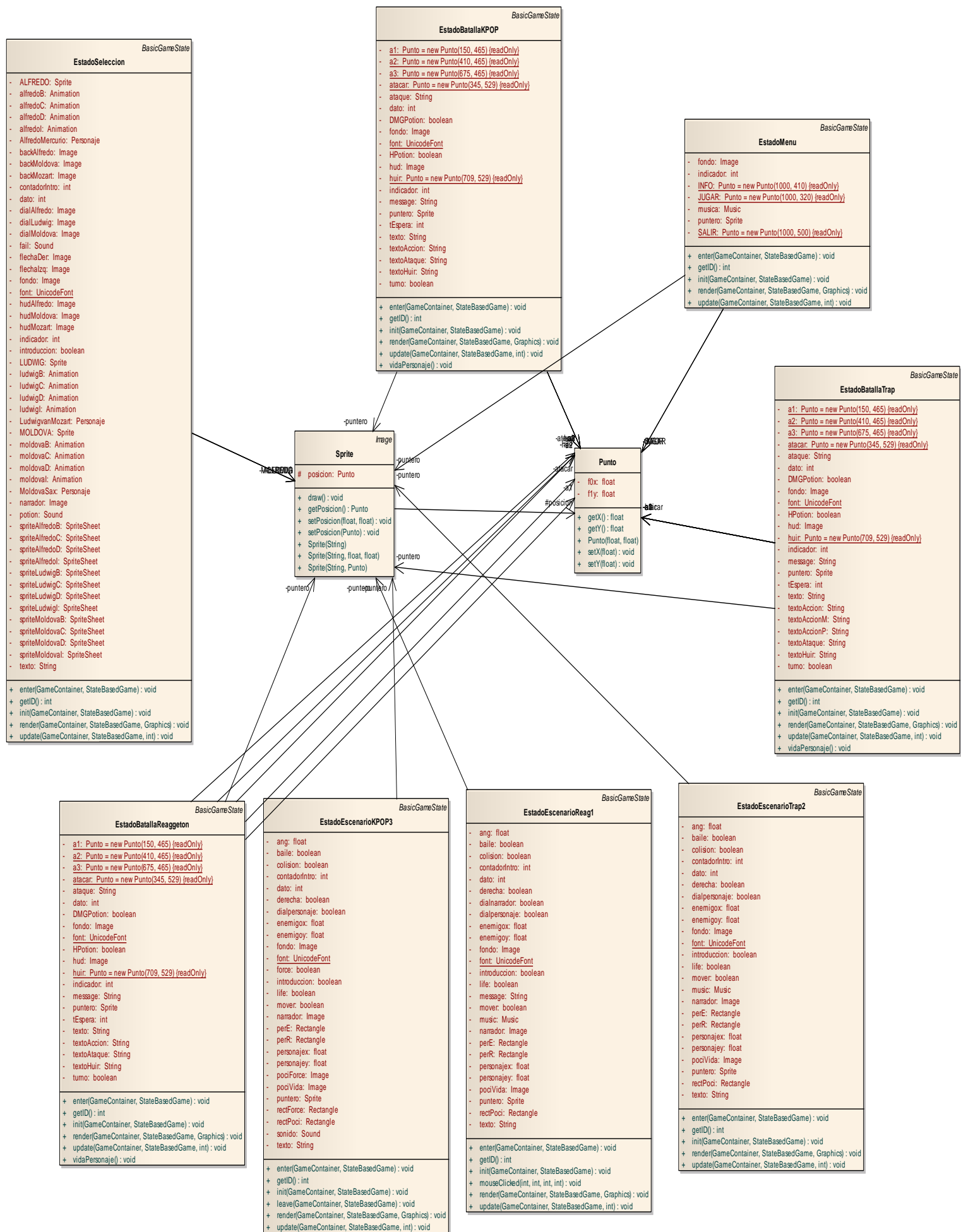


Esquema del turno del enemigo

EstadoVictoria.java

Se trata del estado final del videojuego tras ganar al boss final. En él saldrán los protagonistas bailando y en el que se visualizará que hemos llegado al final del juego. Para volver al estado del menú se deberá dar a la tecla Escape.





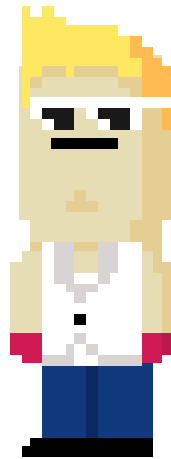
4. Recursos artísticos

Diseño de fondos, personajes y animaciones

Escenarios: Para el diseño de escenarios hemos elegido tres localizaciones (camerino, pasillo y escenario) con distintas modificaciones a medida que el jugador avance en el juego, principalmente, hacia un estado más deteriorado. Esta ambientación cada vez más tétrica viene dada por las texturas sucias de todas las superficies, sombras largas y oscuras, escasas luces y marcadas, salpicaduras de sangre, roturas, etc.



Personajes: La apariencia de los personajes se basa principalmente en la ropa que llevan, dejando de lado las expresiones faciales para mostrar una más seria e inexpressiva. De esta forma, conseguimos que la vestimenta de cada uno sea más simbólica y exclusiva, además de dar a cada personaje un color principal por el que podrán ser diferenciados muy fácilmente.



Para la vestimenta de **Alfredo Mercurio**, nos hemos basado en la ropa que llevaba el propio Freddie Mercury durante su más famosa improvisación sobre un escenario, donde destacan la chaqueta amarilla, y la abundancia del color blanco para camiseta, pantalón e incluso zapatillas. Además, le hemos dado ese tupé y bigote tan característicos.

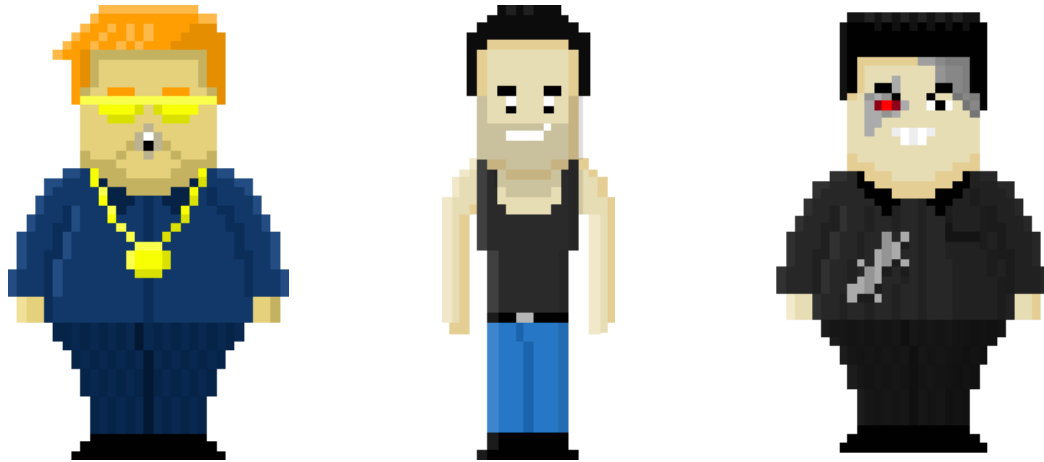
El diseño de **Moldova Sax** se fija en la ropa que vestía el famoso saxofonista moldavo durante su actuación en Eurovisión 2010, que sin duda sorprendió al mundo por su gran movimiento de cadera. Habría que destacar el chaleco blanco, pantalones azules y sobre todo, los guantes rojos y las gafas.

Por último, para crear a **Ludwig van Mozart**, nos hemos inspirado en un estilo más clásico y elegante, ilustrados en forma de bombín y traje negros con camisa blanca y corbata.

HUD: Para el HUD o menú de usuario, aunque solo se encuentra activo durante las batallas, hemos optado por un diseño bastante sencillo e intuitivo, que muestra al jugador los distintos ataques que tiene, dependiendo del personaje, la opción de atacar y la de huir; dejando también una caja de texto inferior para los registros de combate y otra en el lado derecho para mostrar el número de usos disponibles para cada ataque y su correspondiente daño en caso de acertar al enemigo.



Enemigos: Para el diseño de los enemigos hemos decidido guiarnos por los rasgos característicos de tres celebridades actuales, Luis Fonsi, Donald Trump y Kim Jong Un, dándole a Donald Trap esa papada, pelo y color de piel tan característicos; a Kim Jong-II, su tupé con raya en el medio, ojos con rasgos asiáticos y dientes saltones; y a Luis Fonsi, tupé, barba 'de tres días' y larga barbilla.



Animaciones: Cualquiera de las animaciones de bailar, andar o la propia pose estática en batalla, han sido creadas en Adobe Photoshop, diseñando cada frame individualmente y superponiendo a su vez cada uno con el frame anterior como referencia. Finalmente habría que juntar todos los frames en un mismo documento, respetando unos márgenes de separación por los que cortar en Slick2D para darle a cada sprite ese efecto animado.



Diseño de música

Al ser un juego donde la música es un rasgo importante, decidimos hacer tres bandas sonoras distintas: Rock, Jazz, y estilo Clásico. Cada estilo por personaje tiene un total de 4 canciones originales, y tiene 3 efectos de sonido propios. Durante todo el juego suena 8 bit, pero en el último momento, en la batalla final, suena Sonido Real, para darle emoción y profundidad. La única regla que había sobre como hacer las bandas sonoras es que tiene que pertenecer al estilo musical del personaje.

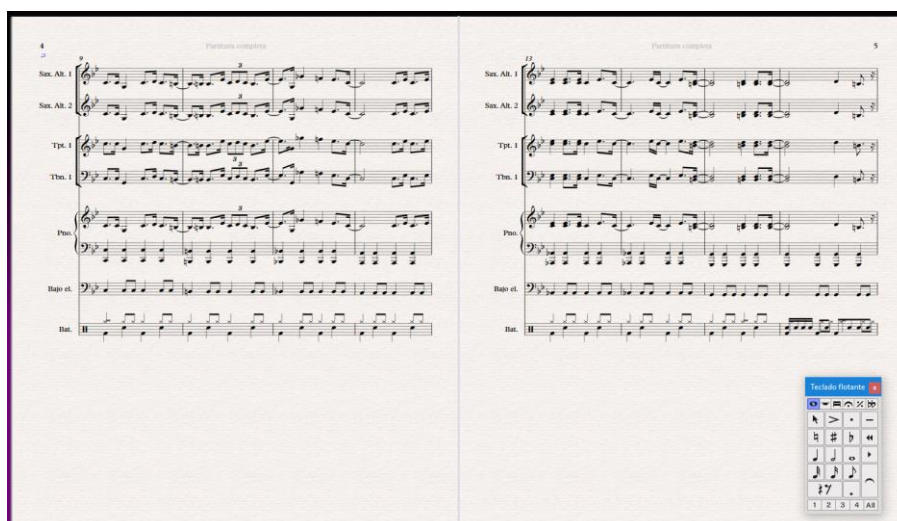
Para el jazz es el ritmo de swing, el uso de pianos de jazz, bajos eléctricos, vientos compuestos por Trompetas, Saxofones... Así pues, las referencias a “Sax Guy” o Riffs de Jazz como los descensos en la banda sonora de Batalla.

En el Rock, el ritmo base propio de cualquier canción rock, Bajos y Guitarras Eléctricas, Órganos de Rock... Esta inspirada en los inicios del Rock, o su relación con el Blues. El uso de frecuencias Ampliadas es frecuente en este estilo musical.

En la música clásica, el instrumento principal es el órgano de tubos, que prácticamente ocupa toda la banda sonora. En la música de batalla, también podemos encontrar un timbal, un piano de cola, triángulos, e instrumentos de cuerda. La música esta basada en Músicos clásicos: Bach, por su fuerte contrapunto (música de pasillo); y Beethoven, y su dinámica del ritmo (música de batalla).

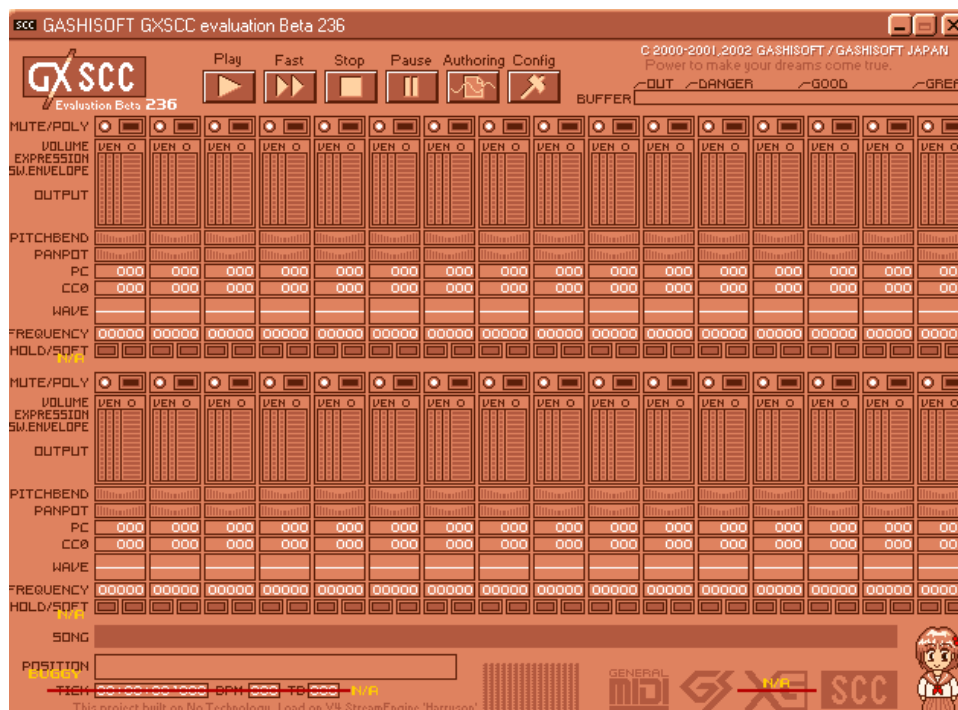
Realización de banda sonora

Todas las bandas sonoras están escritas sobre partitura, nota por nota, por medio de Sibelius, un programa de composición musical. Desde el propio programa, se pueden sacar archivos midi, para después personalizar cada instrumento de cada canción.



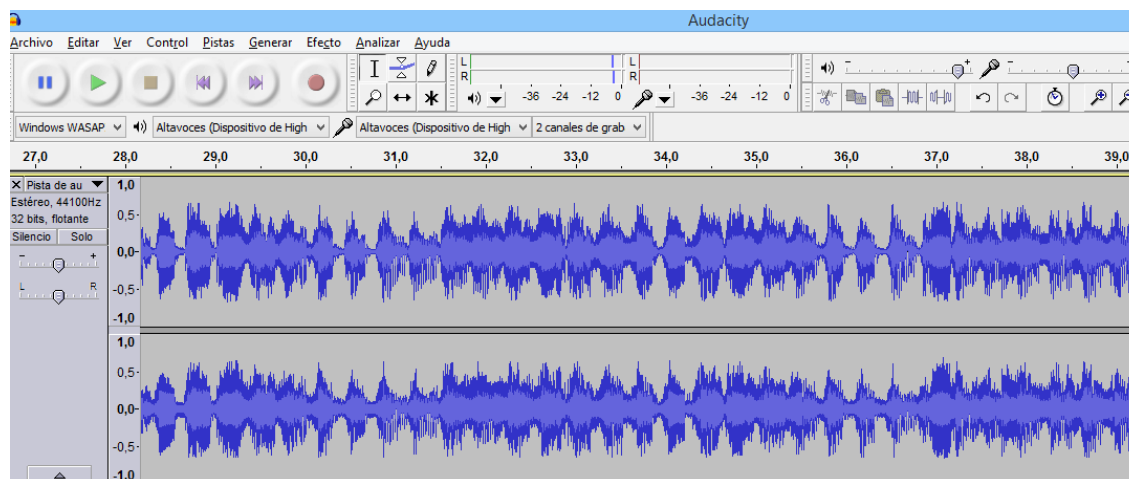
Una vez hecha la melodía principal y ya exportado un midi, dependiendo de que tipo de canción queremos se divide. En este caso, hay dos tipos de canción: música a sonido “8bit”, y sonido real:

-**Sonido “8bit”**: Gracias a GXSCC, un programa “Freeware” lanzado por GASHISOFT, importamos el midi. Automáticamente los instrumentos de ese midi reciben un sonido mecánico característico del sonido de videojuegos antiguos (8bit). A veces, se debe configurar, ya que hay frecuencias que son incómodas para el oído humano, y este programa suele sacar muchos sonidos incómodos.



Sonido real: Para conseguir el sonido real, usamos una herramienta online (<https://www.conversion-tool.com/midi>) totalmente de uso libre que nos permite configurar cada instrumento con una “fuente” de sonidos distinta, entre otras opciones. En las canciones del videojuego hemos usado “Merlin Vienna”.

Una vez ya tenemos nuestra canción, tenemos que pasarla por Audacity. Audacity es un editor de sonido con un gran abanico de herramientas. En las canciones hemos usado herramientas para que las frecuencias sean adecuadas, plugins de reverberación para otorgar profundidad, FadeOut, ampliación, etc.... Todo para llegar a la máxima calidad sonora. Una vez editado, se exporta como .ogg, listo para su uso en el videojuego.



Tanto la música como los efectos de sonido se modifican a partir de Audacity.

6. Conclusión

En primer lugar, describiremos el orden cronológico en el que ha sido desarrollado nuestro videojuego. Desde un primer momento, queríamos que nuestro juego fuera algo innovador y distinto a los típicos juegos de tipo shooter o rpg. Por ello, pensamos en algo que nos gusta, la música, y su actual problema con los géneros musicales que lo están “rompiendo”. Así, ideamos un juego que trataría de gustar a aquellas personas que piensan lo mismo que nosotros, a la vez que divertido con los personajes de Donald Trump y de Kim Jong-un. De este modo, buscamos referencias en aquellos videojuegos en los que podríamos sacar ideas para implementar nuestro videojuego: Pokemon, a la hora de los combates, y de Undertale, a la hora de los escenarios (8bit) y la perspectiva (2D).

Desde ese momento, nos dividimos el trabajo para implementarlo. El equipo de programación estuvo formado por Miguel Fernández y Álvaro, el equipo de diseño por Abel (música) y Miguel Escuderos (animación), y finalmente Ignacio se dedicó a la parte narrativa. Por ello, como reflejamos en el diseño del videojuego, *Until The Last Note* no tendría muchos elementos técnicos debido a nuestro desconocimiento en el motor usado, Slick2d. Por ello, hemos querido dar más importancia al apartado artístico debido a la temática del videojuego. A su vez, nos hubiera gustado implementar más habilidades, progresión de los personajes, una IA más sofisticada, pero, debido a la falta de tiempo y a la falta de experiencia en este tipo de proyectos, no hemos podido realizar todo lo que nos hubiera gustado hacer. Uno de los mayores problemas al que nos hemos enfrentado ha sido el de usar Slick2d, debido a nuestra falta de conocimiento acerca de él y a cómo usar sus librerías. Sin embargo, al ver tutoriales en internet y al recibir explicaciones por parte del profesor y de algunos compañeros, hemos podido aprender a como implementar en él un pequeño videojuego.

Con todo ello, hemos realizado un videojuego al que le hemos dedicado muchas horas desarrollarlo, y , a nuestro parecer, creemos que es divertido y que sacará algunas risas al escuchar a los personajes combatir.

En conclusión, al realizar este proyecto, hemos mejorado en nuestra faceta del trabajo en equipo (organización, comunicación, coordinación...). A su vez, hemos tenido nuestra primera experiencia al implementar un videojuego, y esto nos ha enseñado a que detrás de un videojuego hay un gran número de personas, de diferentes facetas (equipo artístico, programadores...) que han dedicado muchas horas al realizarlo.