

UNTIL THE LAST NOTE



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

Until the Last Note

Hecho por Team 5:

Álvaro Zamorano Ortega

Miguel Ángel Losada Fernández

Ignacio Rubio Valverde

Abel de Coca Torres

Miguel Escuderos de la Morena

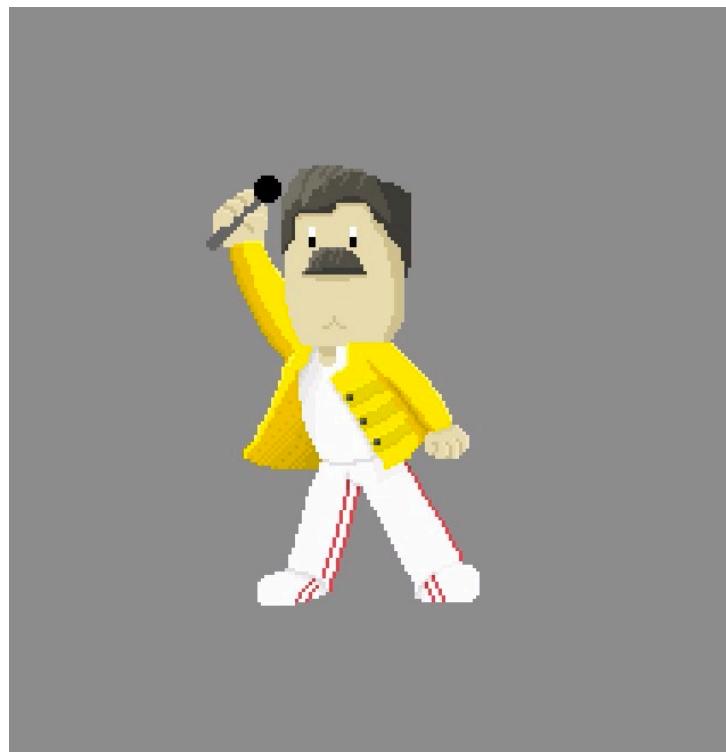
Objetivo del Juego

- * El juego es un RPG en donde el personaje avanzara por pasillos hasta encontrar a los jefes del juego, donde se medirá a ellos en un combate por turnos; en el cual sale vencedor aquel que deje sin puntos de vida al rival. El juego termina al derrotar al jefe final, pero cuidado, no podras guardar la partida en ningun momento.
- * Hay 3 personajes elegibles por el usuario con diferentes habilidades: Alfredo Mercurio, Moldova Sax y Ludwig Van Mozart.
- * Hay 3 jefes a los que hay que derrotar: Donald Trap, Luis Fonsi y Kim Jong-Dos.
- * Una vez derrotados a esos jefes, siendo el jefe final el nombrado Kim Jong-Dos, el juego habrá sido completado en su totalidad.

El Juego

- El juego lo comienzas eligiendo al personaje con el que quieras jugar, de los tres seleccionables.
- Una vez elegido, apareces en tu camerino, y ya es la hora de que comience tu aventura.
- Tras salir del camerino, vas avanzando por los pasillos y de repente llegas a un escenario... Aparece Luis Fonsi, y esta dispuesto a mandarte de vuelta para tu camerino con su odioso reggaeton.
- Le has vencido, te sientes poderoso y avanzas por los siguientes pasillos para proseguir con tu aventura y... Donald Trap esta construyendo un muro para no dejarte avanzar...
- Dificilmente logras vencerle y te encuentras con todo su botin y una pocima secreta para recuperar todos tus poderes. Por sorpresa, oyes el ruido de unos misiles y te diriges a ver que pasa...

- ¡Es Kim Jong-Dos! ¡Y está dispuesto a usar todo su arsenal nuclear contra ti!
- Gracias a haber recuperado todos tus poderes, logras desarmar todo el arsenal del líder del K-POP y sales victorioso de tu aventura.
- Llegó la hora de celebrar todos juntos la victoria sobre los nuevos géneros musicales que no son los mejores del mundo JAJAJAJA.
- Disfruta de esta aventura creada por el Team5, los cuales nos hemos divertido mucho haciendo este juego
- ¡GRACIAS POR JUGARLO!



Funcionamiento

- * El juego se basa en las librerías de Slick 2D, aplicadas al lenguaje de programación Java.
- * La estructura del juego se compone de 19 clases que realizan todas las funcionalidades de las que se compone el juego
- * Si pierdes un combate, vuelves al principio.

Clases

- * Las clases del videojuego son las siguientes: Ataque, Clase Estática, Estados de Batalla, Estados de Pasillos y Habitaciones, Estado de Menú, Personajes, Sprites, Punto y la clase Principal que contiene el método Main.

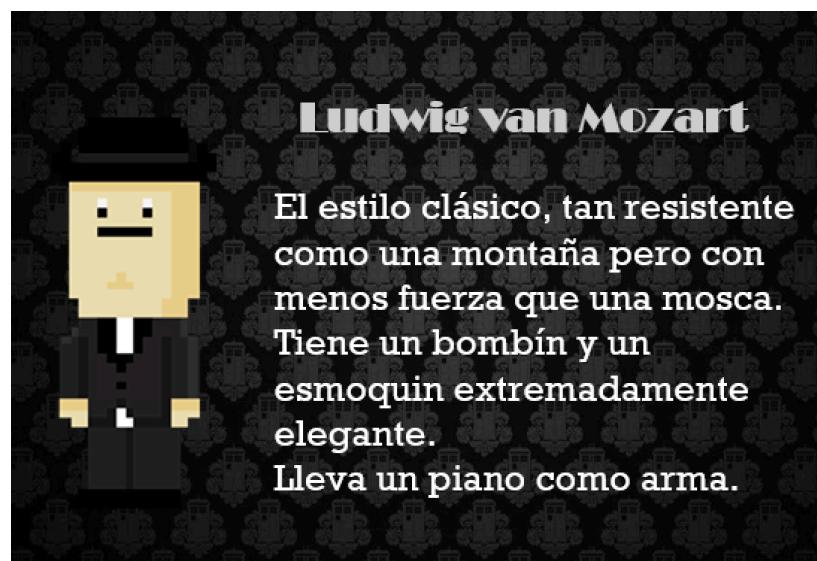


Ataque

- La clase ataque es la encargada de crear la funcionalidad de cada uno de los ataques de cada personaje, como su propio nombre indica.
- **Atributos de la clase:**
 - De tipo *int*: dmg (daño que aplica el ataque), usos (numero de veces restantes que se puede utilizar el ataque), usosMax (numero máximo de veces que puede usar el ataque) y probabilidadFallo (porcentaje de probabilidad de acierto del ataque en el turno actual).
 - De tipo *String*: nombre (nombre del ataque) y descripcion (descripcion de la funcionalidad del ataque).
 - De tipo *Sound*: efecto (sonido que emite el personaje al utilizar el ataque)

- Métodos de la clase:

- De tipo *get*, existe un metodo para obtener el valor de cada atributo de esta clase:
getNombre(), getUsosMax(),
getProbabilidadFallo(), getDmg(), getUsos(),
getDescripcion(), getEfecto().
- De tipo *set*, existe un metodo para definir el valor de cada atributo de cada objeto que sea del tipo de esta clase: setNombre(),
setUsosMax(), setProbabilidadFallo(),
setDmg(), setUsos(), setDescripcion(),
setEfecto().
- De tipo *boolean*, existe un metodo para ver si el ataque realizado: isAcertado().



Clase Estática

- La clase estatica es la encargada de guardar el estado del juego en cada momento del mismo.
- **Atributos de la clase:**
 - De tipo *personaje*: personaje y enemigo (los dos posibles tipos de personajes que hay en el juego)
 - De tipo *sound*: sonidoPaso, click y fail (tres sonidos que aparecen al jugar)
 - De tipo *music*: musicaMenu y musicSilence (musica que suena en cada momento del juego)
 - De tipo *String*: ultimoAtaque (registra el ultimo ataque usado en combate)
 - De tipo *boolean*: ataqueAcertado (guarda si el ultimo ataque ha acertado)

- Métodos de la clase:

- De tipo *get*, existe un metodo para obtener el valor de cada atributo de esta clase: getSonidoPaso(), getFail(), getUltimoAtaque(), getClick(), getPersonaje(), getEnemigo(), getMusicaMenu() y getMusicSilence().
- De tipo *set*, existe un metodo para definir el valor de cada atributo de cada objeto que sea del tipo de esta clase: setSonidoPaso(), setFail(), setUltimoAtaque(), setAtaqueAcertado(), setClick(), setPersonaje(), setEnemigo(), setMusicaMenu() y getMusicSilence().
- De tipo *boolean*, existe un metodo para ver si el ataque realizado; isAtaqueAcertado().



Estados de Batalla

- Las clases de los estados de batalla se basan en hacer funcionar el juego durante las batallas de nuestros personajes contra los jefes del juego. Estas clases se componen con la base de la programacion de videojuegos formadas por los metodos init, render, update y enter; que en este caso utilizamos los propios de Slick 2D.
- **Atributos de la clase:**
 - De tipo *Image*: fondo y hud(la imagen propia del escenario de cada batalla y el menu de batalla).
 - De tipo *Sprite*: puntero (la imagen del puntero para poder interactuar con el juego durante las batallas).
 - De tipo *Punto*: atacar, uir, a1, a2 y a3 (son las posiciones de las palabras del menu de batalla)
 - De tipo *int*: indicador, dato y tEspera (Datos de numeros para que funcione la batalla).

- De tipo *String*: texto, ataque, textoAtaque, textoHuir, message y textoAccion (son los textos que se muestran durante las batallas)
 - De tipo *UnicodeFont*: fuente (tipo de fuente)
 - De tipo *boolean*: turno (true si es del aliado)
-
- **Métodos de la clase:**
 - Init(): inicia el estado del juego en cada batalla, prepara la pantalla a la que se va a acceder estableciendo el valor de los atributos que se tienen que tener al inicio de cada batalla.
 - Render(): hace que se visualice cada elemento de la pantalla en cada momento, y tambien guarda los textos o imagenes que se mostraran a continuación.
 - Update(): actualiza a cada evento el estado de la batalla para que los elementos reaccionen en tiempo real.

- enter(): define los elementos que tienen que ser inicializados al entrar en esta habitación del juego
- getID(): sirve para saber que ID tiene este estado del juego para ser llamado por otro estado para ser ejecutado.
- mouseClicked(): que define la acción a realizar cuando hacemos click con el ratón en este estado del juego.
- vidaPersonaje(): es el contador de la vida de los personajes en la batalla.



Estado de Camerino

- La clase de estado de camerino hace que el juego siga funcionando al llegar a un camerino del mapa del juego. Su funcionamiento es prácticamente igual que las clases de los Estados de Batallas, siendo su base los métodos init, render, update y enter (explicado en los métodos de estas clases).
- **Atributos de la clase:**
 - De tipo *float*: x e y (datos numéricos para determinar la posición del personaje).
 - De tipo *Image*: fondo (imagen de fondo del camerino).
 - De tipo *Sound*: fail (sonido que realiza el personaje al chocarse contra las paredes).
 - De tipo *boolean*: derecha y mover (indicaciones para el movimiento del personaje).

- **Métodos de la clase:**

- Esta clase utiliza los mismos métodos de init(), render, update, enter, y getID().



Estados de Escenario

- Las clases de estados de escenario hacen que el juego continue su funcionamiento al llegar a los escenarios de batalla. Su funcionamiento es semejante al de las clases de los Estados de Batallas, siendo su base los métodos init, render, update y enter.
- **Atributos de la clase:**
 - De tipo *float*: personajex , personajey, enemigox, y enemigoy (datos numericos para establecer las posiciones de los personajes en el escenario).
 - De tipo *Sprite*: puntero (puntero del juego).
 - De tipo *Image*: fondo (imagen del escenario).
 - De tipo *boolean*: derecha y colision (detecta si el usuario se mueve hacia la derecha o colisiona con un elemento del escenario).

- De tipo *int*: estado (dato numerico del escenario).
 - De tipo *Sound*: step (sonido que emite el personaje al moverse).
-
- **Métodos de la clase:**
 - Esta clase utiliza los mismos métodos de init(), render(), update(), enter() y getID() de las clases anteriores, además de añadir el método leave(), para emitir un sonido para indicar que no se puede volver atrás.



Estado Info

- La clase Estado Info, proporciona al programa los atributos que se tienen que mostrar en cada momento determinado del juego
- **Atributos de la clase:**
 - De tipo *Image*: fondoClassic , fondoJazz, y fondoRock (fondos que se tienen que mostrar durante el juego).
 - De tipo *int*: indicador (para conocer el dato numerico de la situacion de objetos).
- **Métodos de la clase:**
 - Esta clase utiliza los mismos metodos de init(), render(), update(), enter() y getID() de las clase anteriores.

Estado de Menú

- La clase estado de menú es la encargada de llevar el correcto funcionamiento del menú principal del juego. Esta clase tambien tiene su base en los métodos init, render, update y enter.
- **Atributos de la clase:**
 - De tipo *Image*: fondo (fondo del menú).
 - De tipo *Sprite*: puntero (puntero del menú).
 - De tipo *Music*: musica (musica del menú).
 - De tipo *Punto*: JUGAR, INFO y SALIR (ubicacion de los botones del menú)
 - De tipo *int*: indicador (para conocer el dato numerico de la situacion de objetos).

- **Métodos de la clase:**

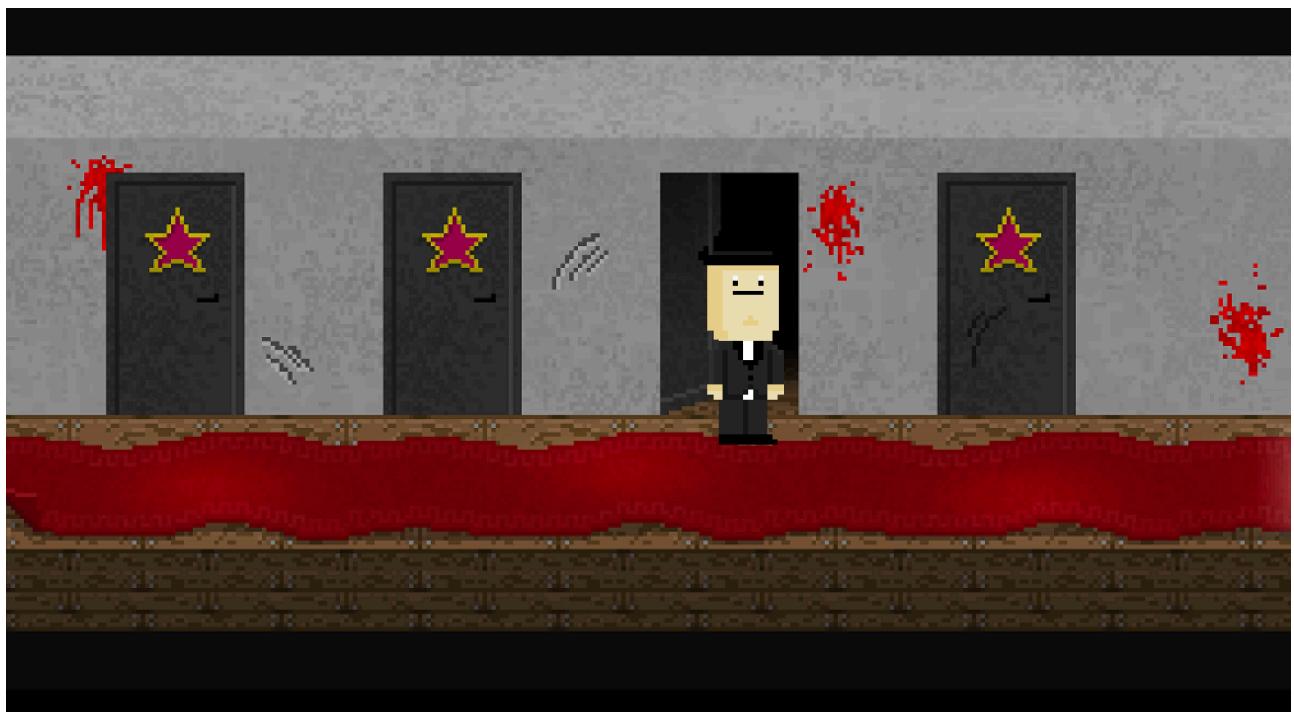
- Esta clase utiliza los mismos métodos de init(), render(), update(), enter() y getID() de las clases anteriores.



Estado de Pasillo

- La clase estado de pasillo sirve para que el juego siga su correcto funcionamiento dentro de los pasillos del juego. Esta clase, al igual que las anteriores tiene su base en los métodos init, render, update y enter.
- **Atributos de la clase:**
 - De tipo *float*: x e y (datos numéricos para establecer las posiciones del pasillo).
 - De tipo *Animation*: enemigoD, enemigoI y enemigoC (animaciones principales del pasillo).
 - De tipo *SpriteSheet*: spriteEnemigoD, spriteEnemigoI y spriteEnemigoC (sprites del enemigo en el pasillo).
 - De tipo *Image*: fondo (imagen del pasillo).
 - De tipo *boolean*: derecha (detecta si el usuario se mueve hacia la derecha).

- De tipo *Personaje*: enemigo (enemigo del escenario al que accede el pasillo).
- **Métodos de la clase:**
 - Esta clase utiliza los mismos métodos de init(), render(), update(), enter() y getID() de las clases anteriores.



Estado de Selección

- La clase estado de seleccion sirve para lograr que el usuario elija al personaje deseado y así jugar al juego con él. Esta clase, al igual que las anteriores tiene su base en los métodos init, render y update (esta clase no hace uso del método enter).
- **Atributos de la clase:**
 - De tipo *Sprite*: ALFREDO, MOLDOVA, LUDWIG y puntero (son los sprites de los tres personajes del juego y del puntero de selección).
 - De tipo *Image*: fondo, hudAlfredo, hudMoldova, hudMozart, backAlfredo, backMoldova y backMozart (imágenes de la selección de personajes).
 - De tipo *int*: indicador (número de referencia para elegir al personaje).
 - De tipo *Personaje*: AlfredoMercurio, LudwigvanMozart, MoldovaSax (personajes seleccionables).

- De tipo *SpriteSheet*: spriteAlfredoD, spriteAlfredol, spriteLudwigD, spriteLudwigl, spriteMoldovaD, spriteMoldoval, spriteAlfredoC, spriteLudwigC, spriteMoldovaC (sprites de los personajes para su selección).
 - De tipo *Animation*: alfredoD, alfredol, ludwigD, ludwigl, moldovaD, moldoval, alfredoC, ludwigC, moldovaC (animaciones de los personajes en la selección).
-
- **Métodos de la clase:**
 - Esta clase utiliza los mismos métodos de init(), render(), update() y getID() de las clases anteriores.

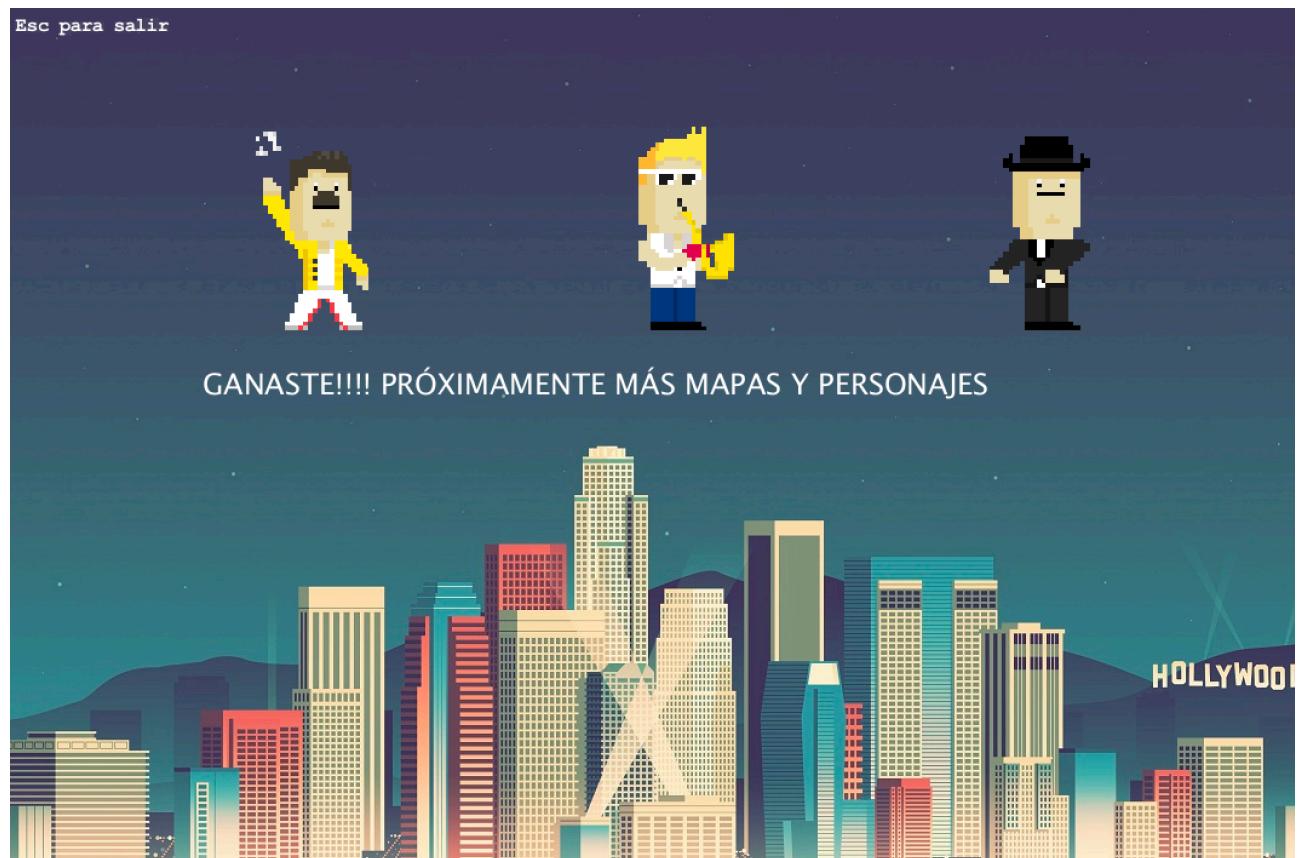


Estado Victoria

- La clase estado de victoria hace que al vencer al jefe final del juego se reproduzca la pantalla de victoria.
- **Atributos de la clase:**
 - De tipo *Image*: fondo (fondo de la pantalla de victoria).
 - De tipo *SpriteSheet*: Alfredo, Mozart, Moldova (Sprites de las poses de victoria de los personajes).
 - De tipo *Animation*: AlfredoDance, MozartDance, MoldovaDance (bailes de victoria de los personajes).
 - De tipo *Music*: musicaVictoria (musica que se reproduce al terminar el juego).

- Métodos de la clase:

- Esta clase utiliza los mismos métodos de init(), render(), update(), enter() y getID() de las clases anteriores.

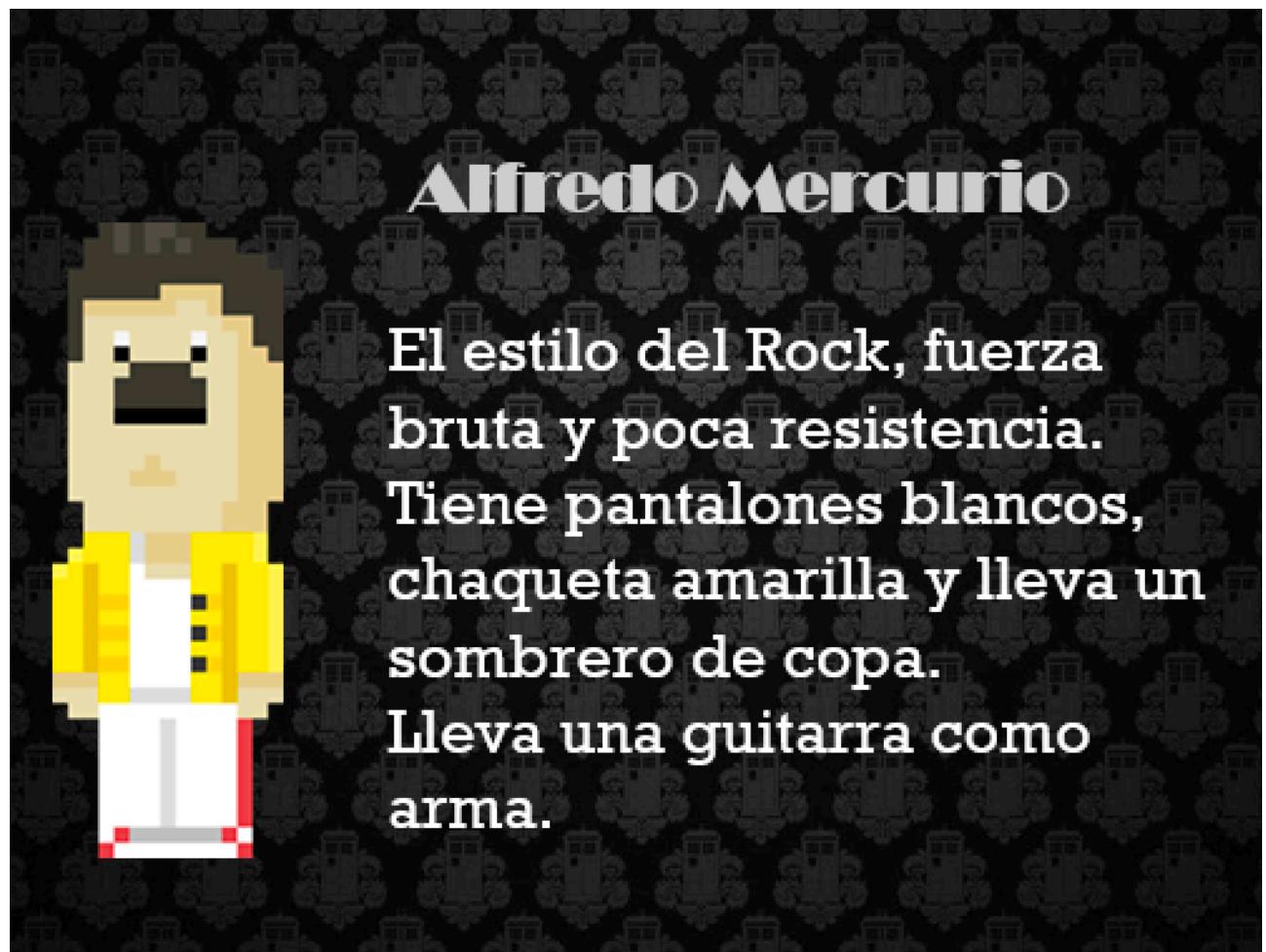


Personaje

- La clase Personaje, es la encargada de crear la funcionalidad de los personajes para que se puedan usar a lo largo del juego.
- **Atributos de la clase:**
 - De tipo *int*: vida (vida que tiene el personaje en ese momento) y vidaMax (numero máximo de vida que puede tener un personaje).
 - De tipo *String*: nombre (nombre del personaje).
 - De tipo *ArrayList*: ataques (lista de ataques del personaje).
 - De tipo *SpriteSheet*: spritePJ (sprite del personaje).
 - De tipo *Animation*: animD, animI y animC (animaciones al caminar del personaje)

- De tipo *Music*: musicB8, musicH8, musicBnormal, musicHnormal (musica de batalla y de pasillo de cada personaje).
 - De tipo *Image*: HUD (interfaz del personaje).
-
- **Métodos de la clase:**
 - De tipo *get*, existe un metodo para obtener el valor de cada atributo de esta clase: getAnimC, getHUD(), getMusicBnormal(), getMusicHnormal(), getMusicB8(), getMusicH8(), getAnimD(), getAnimI(), getVidaMax(), getSpritePJ(), getVida(), getNombre() y getAtaques().
 - De tipo *set*, existe un metodo para definir el valor de cada atributo de cada objeto que sea del tipo de esta clase: setAnimC, setHUD(), setMusicBnormal(), setMusicHnormal(), setMusicB8(), setMusicH8(), setAnimD(), setAnimI(), setVidaMax(), setSpritePJ(), setVida(), setNombre() y setAtaques()

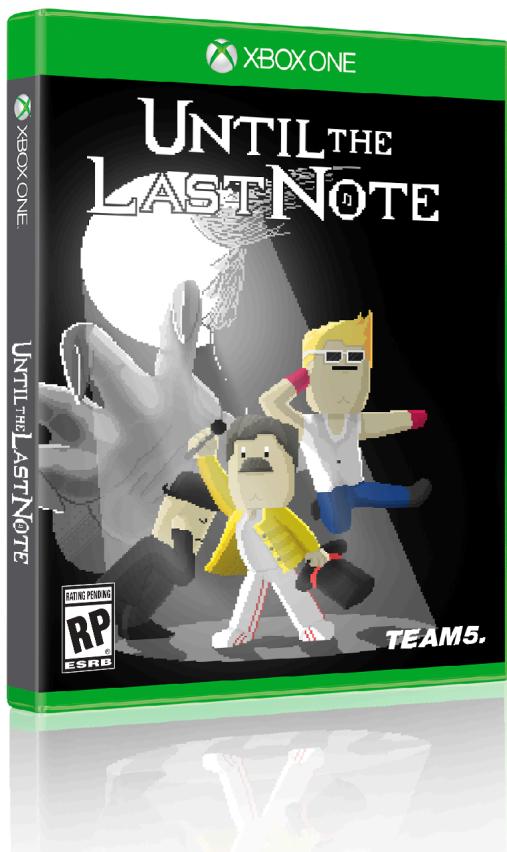
- De tipo *String*, existe dos métodos para realizar los ataques, tanto si es aliado o enemigo: atacar() y ataqueEnemigo().
- Tambien existen un metodo void que hace que se reestablezcan los atributos de los ataques al valor maximo: restaurarTodo().



Principal

- La clase principal es la encargada de el funcionamiento general del juego. Esta clase añade todos los estados del juego creados en las demás clases.
- **Atributos de la clase:**
 - De tipo *AppGameContainer*: contenedor (lugar donde se ejecutara el juego).
 - De tipo *Personaje*: personaje (inicializar un personaje para jugar).
- **Métodos de la clase:**
 - Principal(), es la clase que se encarga de inicializar el contenedor del juego.
 - initStatesList(), es la clase que se encarga de añadir todos los estados del juego al contenedor para su ejecución.

- main(), inicia el metodo principal.



“Toda una obra maestra.”
- IGN

**“Parece mentira que sea
un proyecto universitario.”**
- Eurogamer

**“El juego que revolucionará
la industria 8bit.”**
- Vandal.net

**“Lo más divertido que
hemos jugado en
mucho tiempo.”**
- HobbyConsolas

“Meh... no está mal.”
- Forocoches.es

“Se merecen un 10 de sobra.”
- Confucio

Punto

- La clase Punto, se encarga de obtener las posiciones de los elementos en el juego para relacionarse entre ellos y poder interactuar
- **Atributos de la clase:**
 - De tipo *float*: f0x y f1y (son dos elementos numericos para determinar la posicion x e y de cada elemento dentro del contenedor de juego).
- **Métodos de la clase:**
 - De tipo *get*, existe un metodo para obtener el valor de cada atributo de esta clase; getX(), getY().
 - De tipo *set*, existe un metodo para elegir el valor de cada atributo de esta clase; setX(), setY().

Sprite

- La clase Sprite, se encarga de obtener las imágenes correspondientes a cada elemento del juego y representarlo dentro del juego en una determinada posición x e y.
- **Atributos de la clase:**
 - De tipo *Punto*: posicion (sirve para determinar la posición x e y específica para dibujar el elemento deseado).
 - De tipo *String*: ruta (sirve para indicar la ruta de la imagen propia del sprite para que sea mostrada).
- **Métodos de la clase:**
 - De tipo *get*, existe un método para obtener el valor de cada atributo de esta clase: getPosicion().

- De tipo *set*, existe un metodo para elegir el valor de cada atributo de esta clase: setPosition().
- Tambien existe un metodo void para dibujar el sprite en el lugar deseado dentro del contenedor del juego.

