

**Documentación de Juanito Espadas**

**Creado por:**

Fryda Renata Lara López

Marvin Hazzam Torres Garcia

**Entregado:**

28 de mayo del 2024

**Introducción**

Se realizó la creación de un videojuego llamado “Juanito Espadas” del cual fue creado en java.

Se implementaron recursos como encapsulamiento, herencia e implementación de clases abstractas, así como de interfaces.

**Documentación de las clases y sus métodos**

Se cuenta con 2 clases ya predeterminadas, las cuales ya venían implementadas de forma automática en el IDE solicitado para la creación del proyecto.

**World y actor**

De estas 2 clases ya se cuenta con mucha documentación en línea ya que son las clases principales de las cuales parte la creación del videojuego.

**GameOverScreen**

Esta clase hereda de la clase padre World, no cuenta con atributos y solo cuenta con 1 método:

*GameOverScreen {*

Es el encargado de mostrar lo que sucede en la pantalla

*}*

Esta clase tiene como propósito el dar una retroalimentación de cuando el jugador se ha quedado sin vida y por lo tanto a perdido.

**GameWorld**

Esta clase hereda de la clase padre World, cuenta con varios atributos y solo cuenta con 2 método además de los getters y setters

*LoadMap {*

se encarga de leer un archivo de texto con una matriz en su interior que cada número representa un sprite diferente

*}*

*checkGameComplete{*

verifica que el usuario se encuentre en la dungeon y que haya derrotado a todos los enemigos que se encuentran en ella, para pasar a el final del juego

*}*

**CreditsScreen**

igual que las anteriores hereda de la clase padre world, esta clase cuenta con 2 métodos y ningún atributo

*CreditsScreen{*

además de instanciar la clase, este método es el encargado de dar el feedback de que el juego a llegado a su fin

*}*

**GameCompleteScreen**

esta clase no cuenta con atributos y tiene 2 métodos

*GameCompleteScreen{*

además de instanciar el objeto,este muestra una pantalla intermedia que espera la entrada de la tecla ‘enter’ del teclado para poder seguir adelante

*}*

**StartScreen**

al igual que las anteriores no cuenta con ningún atributo pero sí con métodos

*StartScreen{*

además de instanciar el objeto este método se encarga de mostrar a el usuario el inicio del juego

*}*

**Button**

esta clase hereda de la clase actor cuenta con los 2 atributos y diversos métodos

*atributos{*

private String label;

private GreenfootSound cursor;

*}*

*act{*

este método decide si dependiendo de qué botón sea pulsado la acción a realizar en el juego

*}*

**InventoryWindow**

esta clase es la encargada del inventario del jugador cuenta con atributos de otras clases y métodos muy interesantes

*atributos{*

private Protagonic player;

private ArrayList<SuperObject> items;

private GreenfootImage image;

private boolean itemClicked = false;

private SuperObject selectedItem;

private boolean showingItemOptions = false;

*}*

*updateInventory{*

este método verifica si el jugador tiene colisión con un item del suelo para agregarlo a el inventario

*}*

*useSelectedItem{*

este método detecta el tipo de ítem que tienes el el inventario y al pulsar con el el mouse le aplica su efecto al jugador, además de que lo remueve del inventario de el mismo

*}*

**Monster**

esta clase de los esqueletos enemigos los cuales tienen los siguientes atributos y métodos

*atributos{*

*private int speed;*

*private int agroRange; // Rango de detección del jugador*

*private int attackRange; // Rango de ataque al jugador*

*private int attackDamage; // Daño del ataque*

*private int max\_life;*

*private int actual\_life;*

*}*

*attackPlayer{*

*este método es el encargado de que el enemigo al estar a cierta distancia sea capaz de atacar a el jugador para infligir daño*

*}*

*detectPlayer{*

este método se encarga de determinar si el jugador está en la distancia mínima para que el enemigo lo comience a perseguir

*}*

**Protagonic**

esta clase es la que controla al personaje y permite al jugador interactuar con él además de controlarlo sus atributos y unos de sus métodos más relevantes son

*atributos{*

private int speed;

private int max\_life;

private int life;

private int level;

private int experience;

private int experienceThreshold;

private int money;

private int attack;

private int defense;

*}*

*act{*

este método se encarga de reconocer los input del teclado para saber de qué manera el jugador quiere controlar al personaje

*}*

*moveAndAnimate{*

este se encarga de cambiar los sprites del personaje para simular una animación y dar la ilusión al jugador de que es personaje se mueve de manera independiente

*}*

*checkCollisions{*

este detecta y determina si el bloque en el que esta y estara el personaje es un bloque libre o solido

*}*

*attack{*

este método es el que cambia el sprite del personaje y también manda una señal a otro método para determinar si golpeo a algo mientras se realizaba el ataque

*}*

**StatsWindow**

esta clase es la que muestra en la pantalla la información del jugador solo cuenta con 1 atributo que es el jugador y él siguientes métodos

*drawStats{*

se encarga de crear el recuadro y distribuir la información en forma de lista

*}*

**SuperObject**

Esta es la clase padre de todos los demás objetos del juego cuenta con los siguientes atributos. Para no mencionar a todos los demás objetos se referencian aquí mismo

*atributos{*

public GreenfootImage image;

public String name;

public boolean collision = false;

public int worldX, worldY;

*}*

*swords{*

esta subclase le añade daño a el personaje, dependiendo del tipo de espada que sea se añade más o menos daño

*}*

*shields{*

esta subclase le da defensa ante el daño a el personaje, dependiendo del tipo de escudo la defensa añadida

*}*

*potions{*

esta subclase le da un efecto al personaje, depende del tipo de poción el efecto añadido

*}*

*coins{*

añade valor a los bolsillo del personaje,depende del tipo de moneda el valor de cada una

*}*

**Tile**

esta clase es la que instancia el escenario dando su sprite a cada uno de los puntos en el mapa, además de determinar si es un bloque de colisiones o no cuenta con los siguientes atributos y métodos

*atributos{*

private boolean collision;

private int tileType;

*}*

*getImageNameForTileType{*

este método es el que regresa la dirección del archivo al cual se debe colocar ese sprite en esa consulta del método

*}*

*isSolid{*

determina si es un bloque sólido(con colisiones) en la consulta de ese método

*}*

**EnemiSetter**

esta clase es la encargada de determinar la posición de los enemigos y los escenarios donde aparecerán tiene los siguientes atributos y métodos

*atributos{*

private World world;

public Monster [] moustros;

*}*

*enemiMap{*

se encarga de determinar si entro en un escenario donde pueden aparecer los enemigos

*}*

*placeEnemi{*

coloca a los enemigos en la posición que se le pase como parámetros(x,y)

*}*

**MapManager**

esta clase es como un diccionario de las intersecciones de un escenario con los demás tiene los siguientes atributos y métodos

*atributos{*

private GameWorld world;

private String[] mapFiles;

private int [] [] mapConnections;

*}*

*changeMap{*

determina qué escenario cargar al llegar a cierta posición del mapa actual

*}*

**MusicManager**

esta clase se encarga de la colocación de la música en los momentos exactos tiene los siguientes atributos y métodos

*atributos{*

private static GreenfootSound backgroundMusic;

private static GreenfootSound dungeonMusic;

private static GreenfootSound gameOverMusic;

private static GreenfootSound gameCompleteMusic;

private static GreenfootSound currentMusic;

*}*

*playBackgroundMusic{*

este se encarga de poner la musica de fondo en un loop

*}*

*stopAllMusic{*

este se encarga de pausar la música actual para darle entrada a colocar otra diferente

*}*

**ObjectSetter**

esta clase se encarga de asignar en que lugar y escenario se generarán los cofres con los siguientes atributos y métodos

*atributos{*

private World world;

private List<Chest> cofres;

private List<Boolean> cofresAbiertos;

*}*

createRewardsForMapIndex{

*determina las recompensas de cada cofre en el juego*

}

*placeObjects{*

coloca los cofres en la posición dada como parámetro(x,y)

*}*