ZOMBIESURVIVOR

Guillermo Sierra Castejón

DESARROLLO DEL VIDEOJUEGO VERSIÓN 1.4

ÍNDICE

Hoja De Versiones

Fecha de Entrega	Versión del Juego	Autor de los Cambios	Modificaciones
18/11/2024	Versión 1	GSC	Creación de la base del videojuego con un menú y controles de errores.
10/01/2025	Versión 1.1	GSC	Otra creación de otro juego nuevo ya que el otro lo borré y en este tiene lo mismo solo que hay funcionalidad para jugar y aparte he añadido arrays.
17/01/2025	Versión 1.2	GSC	He utilizado la modularización en el código.
27/01/2025	Versión 1.3	GSC	Actualización de la documentación, fichero readme y he añadido comentarios encima de cada método
10/02/2025	Versión 1.4	GSC	Agregación de clases con herencia y separar el código en más clases para que no esté todo en la main
17/02/2025	Versión 1.5	GSC	SOLO LA TENIAN QUE HACER LOS QUE RECUERABAN
07/03/2025	Versión 1.6	GSC	Creación del fichero Utilidades.java y he dividido el código en paquetes
17/03/2025	Versión 1.7	GSC	Corrección de errores y creación de la clase partida que almacena datos sobre la partida
28/04/2025	Versión 1.8	GSC	Corrección de errores, agregación de Colecciones, ficheros y BBDD

Descripción del videojuego

Tema y Objetivo:

El tema de este videojuego es sobre zombies y el objetivo es llegar al lugar seguro sin que los zombies te quiten mucha vida, el lugar seguro está en alguna parte del mapa, si llegas con vida y energía al lugar seguro ganarás la partida.

Descripción Narrativa o Contexto:

Aparecerás en un lugar totalmente aleatorio y vas a tener que ir moviéndote hacia el **Norte (N)** dirección arriba, **Noreste (NE)** dirección arriba a la derecha, **Este (E)** dirección a la derecha, **Sudeste (SE)** dirección abajo a la derecha, **Sur (S)** dirección abajo, **Sudoeste (SO)** dirección abajo a la izquierda, **Oeste (O)** dirección izquierda, **Noroeste (NO)** dirección arriba a la izquierda. Dependiendo de la casilla que elijas tendrás opción de recuperar o perder parte de las estadísticas que tienes como vida o energía que estarán con x% desde un inicio.

Si pierdes toda la vida o toda la energía acabas muriendo y se termina la partida, en la siguiente versión del juego al tener la vida al 0% morirás, pero si llegas a tener la energía al 0% lo más normal es que acabes muriendo ya que habrá que hacer tareas o escoger caminos en los que necesites energía para realizar las cosas.

Debes guiarte por la posición en la que te encuentras para saber dónde estás ya que actualmente no hay ningún mapa por el que guiarte.

Audiencia y Estilo de Juego:

La audiencia de este juego es para niños mayores de 10/11 años que tengan el conocimiento suficiente para moverse por el tablero a través de las coordenadas Norte, Noreste, Sur, Sudoeste, Este, Sudeste, Oeste, Noroeste.
El estilo del juego es tipo arcade de supervivencia en el que tendrás que llegar al lugar seguro para ganar la partida.

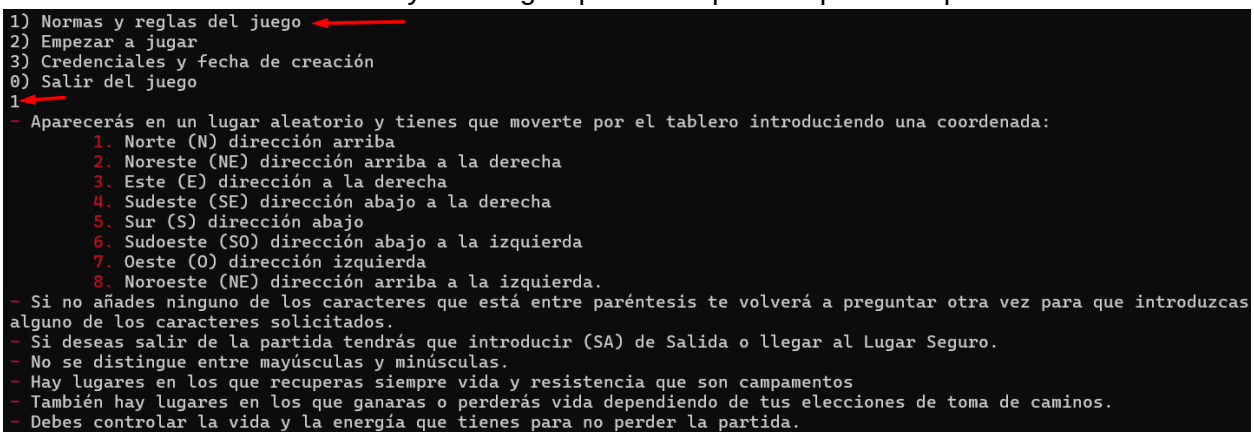
Reglas del Juego:

- Aparecerás en un lugar aleatorio y tienes que moverte por el tablero introduciendo una coordenada:

1. **Norte (N)** dirección arriba
2. **Noreste (NE)** dirección arriba a la derecha
3. **Este (E)** dirección a la derecha
4. **Sudeste (SE)** dirección abajo a la derecha
5. **Sur (S)** dirección abajo

6. **Sudoeste (SO)** dirección abajo a la izquierda
7. **Oeste (O)** dirección izquierda
8. **Noroeste (NE)** dirección arriba a la izquierda.

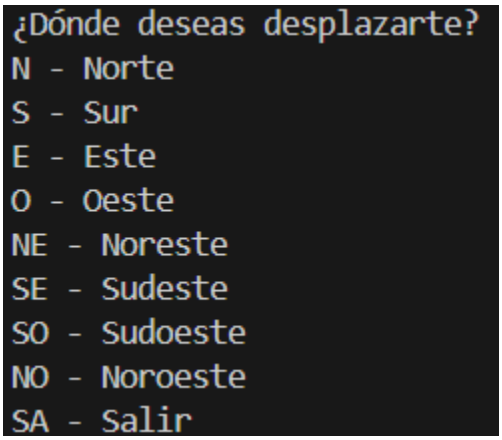
- Si no añades ninguno de los caracteres que está entre paréntesis te volverá a preguntar otra vez para que introduzcas alguno de los caracteres solicitados.
- Si deseas salir de la partida tendrás que introducir **(SA) de Salida** o llegar al **Lugar Seguro.**
- No se distingue entre mayúsculas y minúsculas.
- Hay lugares en los que recuperas siempre vida y resistencia que son campamentos
- También hay lugares en los que ganaras o perderás vida dependiendo de tus elecciones de toma de caminos.
- Debes controlar la vida y la energía que tienes para no perder la partida.



Mecánicas y Jugabilidad:

Controles y Movimientos (si aplica):

Los movimientos que puedes hacer son los de las coordenadas geográficas indicadas en las reglas del juego y solo puedes moverte de una en una casilla por cada pregunta.



Interacciones con Elementos del Juego (si aplica):

Mapa y Ubicaciones:

- El juego utiliza un array bidimensional que es el mapa y cada celda de este representa un lugar en el juego como: Casa, Camino, Hospital, Lugar seguro…
- Cada ubicación tiene atributos que afectan la vida y la energía del jugador, definidos en mapaVida y mapaEnergia.

Desplazamiento del Jugador:

- El jugador puede desplazarse en diferentes direcciones (N, S, E, O, NE, SE, SO, NO), afectando su posición en el mapa (posX, posY).
- Al moverse, la vida y la energía del jugador se actualizan según el lugar al que se mueva (mapaVida y mapaEnergia).

Condiciones del Juego:

- Hay lugares que afectan positivamente o negativamente la vida y la energía.
- Si la vida o la energía del jugador llegan a 0, el juego termina.
- Si el jugador llega al "Lugar seguro", gana el juego.
- Si el jugador escribe “sa” o “SA” termina la partida y sale del juego.

Interacción con el Usuario:

- El jugador selecciona acciones a través del menú.
- Durante el desplazamiento, se solicita al jugador que elija una dirección explicada más arriba en desplazamiento del jugador.

Generación Dinámica de Elementos:

- El mapa inicial se genera aleatoriamente (como la posición inicial en el array LUGARINICIAL).

Desafíos y Obstáculos:

El desafío es llegar al lugar seguro y los obstáculos son que puedes ir perdiendo vida si no eliges la opción correcta. Para ello habrá unos lugares específicos en los que si caes ahí puedes recuperar vida, energía o perderla dependiendo de la opción que elijas.

```
Te has movido a la posición X1, Y2 que es un/a: Lugar seguro y varía la vida: 90 y la energia: 90
Por tanto tienes vida: 100 y energia: 100
Enhorabuena, has llegado al lugar seguro.
```

```
Te has movido a la posición X1, Y2 que es un/a: Lugar seguro y varía la vida: 90 y la energia: 90
Por tanto tienes vida: 100 y energia: 100
Enhorabuena, has llegado al lugar seguro.
```

```
Te has movido a la posición X0, Y1 que es un/a: Casa y varía la vida: -30 y la energia: -40
Por tanto tienes vida: 0 y energia: 10
Te has quedado sin vida.
¡Has perdido el juego!
```

Progresión o Niveles:

La progresión es ir moviéndote por el mapa intentando sobrevivir hasta llegar al lugar seguro.

Requisitos funcionales y no funcionales

Requisitos Funcionales:

- **Caso de Uso: Menú Inicial**
  - **ID:** CU01
  - **Título:** Menú Inicial
  - **Descripción:** Ofrece al usuario múltiples alternativas para seleccionar
  - **Precondiciones:** El juego debe estar en la pantalla de inicio.
  - **Entrada:** Opción seleccionada por el usuario
  - **Salida:** Ver las normas, iniciar partida, visualizar credenciales, cerrar el juego
  - **Boceto:** Menú con 4 opciones a elegir
  - **Estado:** Implementado (v1, v1.1, v1.2)
- **Caso de Uso: Entorno Virtual**
  - **ID:** CU02
  - **Título:** Entorno Virtual
  - **Descripción:** El jugador explora un entorno 3D que impacta su vida y energía.
  - **Precondiciones:** Ninguna
  - **Entrada:** Coordenadas del jugador en el entorno tridimensional.
  - **Salida:** Cambios en la vida y energía
  - **Boceto:** Entorno 3D que muestra la posición y efectos del jugador.
  - **Estado:** Implementado (v1.1, v1.2)
- **Caso de Uso: Desplazamiento del usuario**
  - **ID:** CU03
  - **Título:** Desplazamiento del usuario
  - **Descripción:** Permite moverse de formas distintas en el juego
  - **Precondiciones:** El jugador debe tener vida y energía para moverse
  - **Entrada:** Dirección seleccionada por el jugador.
  - **Salida:** Nueva posición del jugador actualizada en el terreno.
  - **Boceto:** El tablero con la posición actual del jugador indicada.
  - **Estado:** Implementado (v1, v1.1, v1.2)

- **Caso de Uso: Manejo de Vida y Energía**
  - **ID** CU04
  - **Título** Modificar vida y energía
  - **Descripción** La vida y energía del jugador se ajustan según las condiciones del terreno explorado.
  - **Precondiciones** Tener siempre más de 1 de vida y energía.
  - **Entrada** Ninguna.
  - **Salida** Actualización de los valores de vida y energía del jugador.
  - **Boceto** Indicador visual que muestra los valores de vida y energía.
  - **Estado** Implementado (v1, v1.1, v1.2)
- **Caso de Uso: Reglas para ganar o perder**
  - **ID** CU05
  - **Título** Evaluar victoria o derrota
  - **Descripción** El juego determina si el jugador gana al llegar al "Lugar seguro" o pierde al quedarse sin vida o energía.
  - **Precondiciones** Ninguna.
  - **Entrada** Estado actual del jugador (vida, energía y posición).
  - **Salida** Victoria, derrota o seguir jugando.
  - **Boceto** Mensaje visual que indica si el jugador gana o pierde.
  - **Estado** Implementado (v1.1, v1.2)
- **Caso de Uso: Representación de la Situación del Jugador**
  - **ID** CU06
  - **Título** Mostrar estado del jugador
  - **Descripción** Muestra información sobre la ubicación, vida, energía y lugares actual del jugador.
  - **Precondiciones** Ninguna.
  - **Entrada** Solicitud para mostrar información del jugador.
  - **Salida** Visualización en pantalla de la información del jugador.
  - **Boceto** Pantalla con indicadores de vida, energía y lugares explorados.
  - **Estado** Implementado (v1, v1.1, v1.2)
- **Caso de Uso: Reglas del Juego**
  - **ID** CU07
  - **Título** Mostrar reglas
  - **Descripción** Se muestran las reglas y requisitos para jugar.
  - **Precondiciones** Que el usuario seleccione la opción de reglas del juego.
  - **Entrada** Lectura del dato introducido por el usuario.
  - **Salida** Reglas y requisitos mostrados en pantalla.
  - **Boceto** Pantalla inicial con las reglas del juego organizadas de forma clara.
  - **Estado** Implementado (v1, v1.1, v1.2)
- **Caso de Uso: Información del Creador**
  - **ID** CU08

  - **Título** Mostrar información del creador
  - **Descripción** Se muestra información sobre el autor del juego y la fecha de inicio de la creación.
  - **Precondiciones** Que el usuario seleccione la opción de credenciales.
  - **Entrada** Lectura del dato introducido por el usuario.
  - **Salida** Las credenciales mostradas en pantalla.
  - **Boceto** Muestra el nombre del creador, fecha y descripción.
  - **Estado** Implementado (v1, v1.1, v1.2)
- **Caso de Uso: Interacción con zombis**
  - **Identificación:** CU09
  - **Título:** Interacción con Zombis
  - **Descripción:** Los zombies pueden atacar al jugador y reducir su vida. El jugador puede esquivar, atacar o escapar.
  - **Precondiciones:** El jugador debe encontrarse en una ubicación donde haya zombies.
  - **Entrada:** Acción del jugador (esquivar, atacar, huir).
  - **Salida:** Resultado de la acción (reducción de vida, eliminación del zombie, escape exitoso).
  - **Boceto:** Representación del combate con zombies en pantalla.
  - **Estado:** Pendiente de implementación.
- **Exploración de lugares**
  - **Identificación:** CU10
  - **Título:** Exploración de Lugares
  - **Descripción:** El jugador puede explorar diferentes ubicaciones en el mapa y descubrir recursos o peligros.
  - **Precondiciones:** El jugador debe tener energía para moverse.
  - **Entrada:** Dirección o ubicación seleccionada por el jugador.
  - **Salida:** Información sobre el lugar descubierto.
  - **Boceto:** Representación del mapa y los lugares disponibles.
  - **Estado:** Pendiente de implementación.
- **Gestión de inventario**
  - **Identificación:** CU11
  - **Título:** Gestión de Inventario
  - **Descripción:** El jugador puede recoger, usar y descartar objetos en su inventario.
  - **Precondiciones:** El jugador debe haber encontrado algún objeto.
  - **Entrada:** Selección de objeto y acción correspondiente.
  - **Salida:** Cambio en el inventario del jugador.
  - **Boceto:** Interfaz con lista de objetos y opciones de uso.
  - **Estado:** Pendiente de implementación.
- **Sistema de Guardado y Carga de Partidas**
  - **Identificación:** CU12
  - **Título:** Guardado y Carga de Partidas

  - **Descripción:** Permite al jugador guardar su progreso y continuar en otro momento.
  - **Precondiciones:** Debe haber una partida en curso.
  - **Entrada:** Opción de guardar o cargar.
  - **Salida:** Estado del juego guardado o restaurado.
  - **Boceto:** Menú con opciones de guardado y carga.
  - **Estado:** Pendiente de implementación para hacerlo con ficheros.
- **Generación Procesal del Mapa**
  - **Identificación:** CU13
  - **Título:** Generación Procesal del Mapa
  - **Descripción:** El mapa se genera dinámicamente en cada nueva partida para variar la experiencia.
  - **Precondiciones:** Inicio de una nueva partida.
  - **Entrada:** Parámetros de generación aleatoria.
  - **Salida:** Nuevo mapa generado.
  - **Boceto:** Vista del mapa inicial.
  - **Estado:** Pendiente de implementación.

Requisitos No Funcionales:

1. **Interfaz de Usuario (UX):**
   - Mensajes claros y colores para mejorar la experiencia del jugador.
   - Diseño de menús y mensajes intuitivos para navegación fluida.
   - **Estado:** Implementado (v1, v1.1, v1.2)
2. **Rendimiento:**
   - Procesamiento de entradas y movimientos en tiempo real.
   - Respuesta inmediata en cada interacción del usuario.
   - **Estado:** Implementado (v1, v1.1, v1.2)
3. **Portabilidad:**
   - Ejecución en sistemas Windows, macOS y Linux.
   - **Estado:** Implementado (v1, v1.1, v1.2)
4. **Escalabilidad:**
   - Tablero adaptable a diferentes tamaños.
   - Posibilidad de agregar nuevos niveles o mapas.
   - **Estado:** Implementado (v1, v1.1, v1.2)

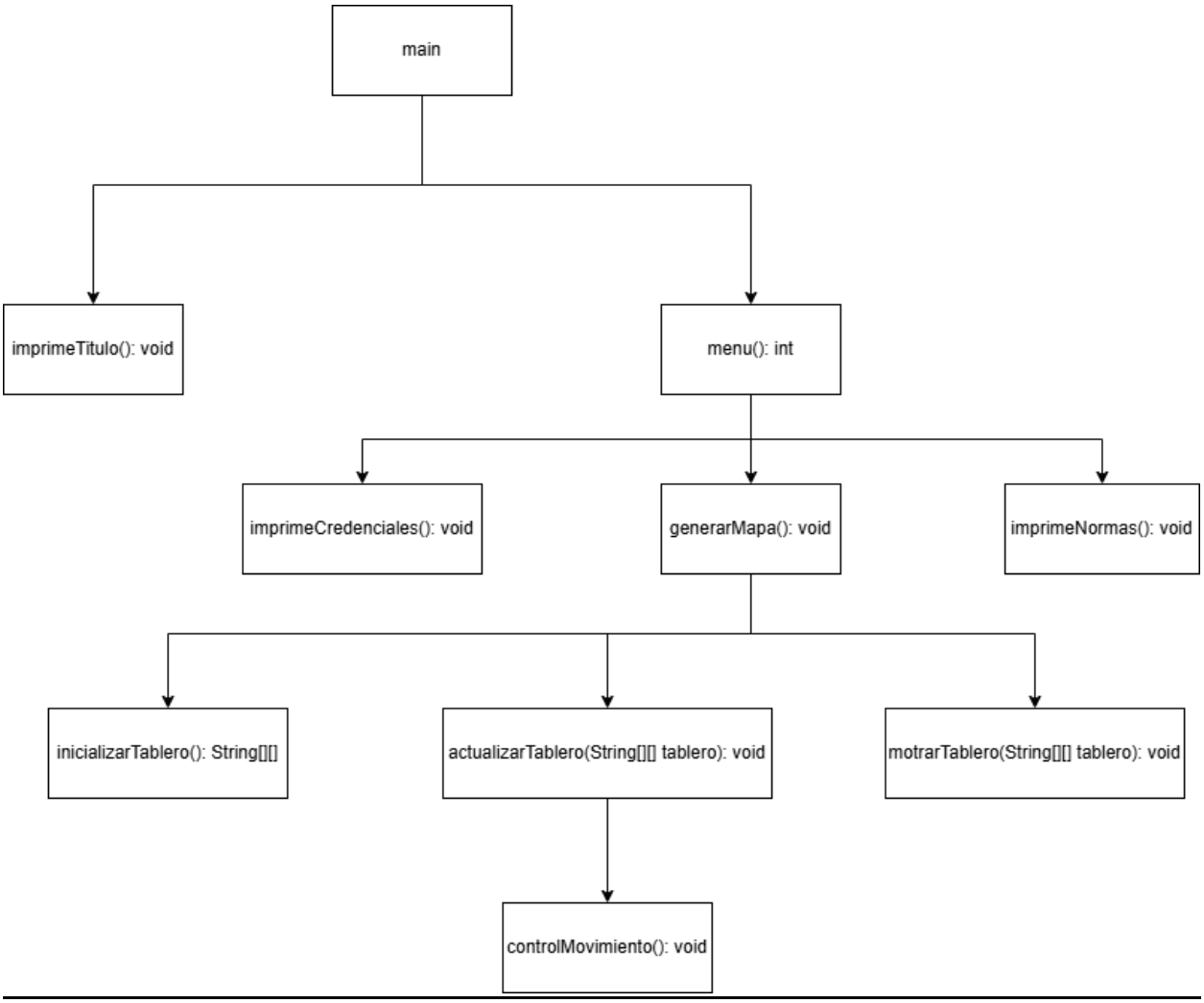
5. **Fiabilidad:**
    - Gestión robusta de entradas inválidas.
    - Control de bordes y validación de coordenadas para movimientos.
    - **Estado:** Implementado (v1, v1.1, v1.2)
6. **Accesibilidad:**
    - Soporte para colores y mensajes inclusivos.
    - Independencia de mayúsculas y minúsculas en las entradas.
    - **Estado:** Implementado (v1, v1.1, v1.2)

Herramientas de Desarrollo y Recursos Adicionales

1. **JDK 21**: Para la compilación y ejecución del código.
2. **Visual Studio Code**: Como entorno de desarrollo integrado (IDE).
3. **ASCII art generator**: Para las figuras y títulos
4. **ENTER**: Este enlace para pulsar solo enter

Análisis del Proyecto

Descripción de los módulos identificados:



imprimeTitulo(): void

Objetivo: Imprimir por pantalla el título del videojuego.
Entrada: Ninguna.
Salida: Solamente imprimir el título.

menu(): int

Objetivo: Muestrar el menú principal del juego y permitir que el usuario seleccione una opción válida.
Entrada: Solicita un num al usuario y es leído con Scanner. El num se guarda en opcionMenu
Salida: Retorna un entero que representa la opción seleccionada por el usuario.

imprimeNormas(): void
Objetivo: Mostrar las reglas y normas del juego al usuario.
Entrada: Ninguna.
Salida: Solamente imprimir las normas del juego.

imprimeCredenciales(): void
Objetivo: Mostrar las credenciales el juego al usuario
Entrada: Ninguna.
Salida: Solamente imprimir las credenciales del juego.

generarMapa (): void
Objetivo: Inicializar el mapa del juego, asignando ubicaciones, valores de vida y energía a cada posición.
Entrada: Ninguna.
Salida: Modifica las variables globales mapa, mapaVida, y mapaEnergia

controlMovimiento(): void
Objetivo: Ajustar las posiciones del jugador para que permanezcan dentro de los límites del mapa.
Entrada: Ninguna directamente (usa las variables globales posX y posY).
Salida: Ninguna (modifica posX y posY para mantenerlas dentro de los límites).

inicializarTablero(): String [][]
Objetivo: Crear y devolver un tablero vacío para el juego.
Entrada: Ninguna.
Salida: Un array bidimensional de cadenas (String[][]) representando un tablero vacío.

motrarTablero(String[][] tablero): void
Objetivo: Mostrar el tablero actual en la consola con formato y colores.
Entrada: Un array bidimensional de cadenas (String[][] tablero).
Salida: Imprime el tablero en la consola.

actualizarTablero(String[][] tablero): void
Objetivo: Actualizar el contenido del tablero con la posición actual del jugador.
Entrada: Un array bidimensional de cadenas (String[][] tablero).
Salida: Modifica directamente el tablero recibido como argumento.

Descripción técnica

Clase Main

1. Fichero que tiene la main y ejecuta el proyecto.

Clase ZombieSurvivor

Variables:

1. **opcionMenuSeleccionado (int):** Dependiendo de que introduce el usuario hace una cosa del switch u otra.
2. **opcionMenu (int):** Lee la opción que introduce el usuario.

Constantes:

3. **COLOR_RESET, COLOR_RED, COLOR_PURPLE:** Contantes para dar color.
4. **FINALIZARPROGRAMA (int):** Sirve para finalizar el programa al pulsar 4.

Clase Partida

Variables:

1. **Nombre (String):** Nombre de la partida o del jugador asociado a la partida.
2. **fechaInicio, fechaFin (LocalDateTime):** Fecha de inicio y de fin de la partida jugada.
3. **numMovimiento (int):** Almacena los movimientos del usuario.
4. **Victoria (boolean):** Muestra si se gana o si se pierde la partida.

Clase Mapa

Variables:

1. **tamanioMaxX, tamanioMaxY (int):** Tamaño del mapa.
2. **posInicialX, posInicialY (int):** Posición inicial del jugador.
3. **posJugadorX, posJugadorY (int):** Posición en la que se encuentra el jugador en todo momento.

Constantes:

4. **COLOR_RESET, COLOR_RED, COLOR_CYAN (String):** Contantes para dar color.
5. **MOVIMIENTO_SALIR, MOVIMIENTO_CORRECTO, MOVIMIENTO_ERRONEO (int):** Estados de movimiento.

Arrays:

6. **mapaLugares (Lugar[][])**: Array bidimensional que representa el mapa con los distintos lugares.
7. **tablero (String[][]):** Representación gráfica del mapa.

Clase Lugar

Variables:

1. **nombre (String):** Nombre del lugar.
2. **esLugarSeguro (boolean):** Booleano para saber si es lugar seguro o no

3. **costeEnergia (int):** Energía requerida para acceder al lugar.
4. **posX, posY (int):** Coordenadas del lugar en el mapa.
5. **tieneZombie (boolean):** Booleano para saber si hay zombie en el lugar
6. **vidaObtenida, energiaObtenida (int):** Vida y energía que se obtiene en el lugar.

Constantes:

7. **COSTE_ENERGIA_POSITIVO (int):** Coste de energía en positivo para ir a este lugar
8. **COSTE_ENERGIA_NEGATIVO (int):** Coste de energía en negativo para ir a este lugar
9. **VIDA_ENERGIA_OBTENIDA (int):** Energía que obtienes por ir a este lugar
10. **LUGARES (String[]):** Array de nombres de los lugares posibles.

Arrays:

11. **grupoZombie (Zombie[]):** Array de zombies presentes en el lugar.

Clase Personaje (abstracta)

Variables:

1. **vida, energia (int):** Vida y energía del personaje.

Clase Zombie

Constantes:

5. **COLOR_RESET, COLOR_GREEN, COLOR_YELLOW (String):** Contantes para dar color.
6. **VIDA_ENERGIA_POSITIVO (int):** La vida y energía que quita el zombie en positivo.
7. **VIDA_ENERGIA_NEGATIVO (int):** La vida y energía que quita el zombie en negativo.

Clase Jugador

Variables:

2. **nombre (String)**: Almacena el nombre del jugador.

Constantes:

3. **MAX_VIDA, MIN_VIDA, MAX_ENERGIA, MIN_ENERGIA:** Límites de vida y energía del jugador.

Clase Utilidades

Variables:

1. **Sc (Scanner):** Leerá los datos introducidos por el usuario

Constantes:

2. **COLOR_RESET, COLOR_GREEN (String):** Contantes para dar color.

Clase Juego

Variables:

1. **jugador (Jugador):** Declara un objeto tipo Jugador
2. **partida (Partida):** Declara un objeto tipo Partida

3. **mapa (Mapa):** Declara un objeto tipo Mapa
4. **historialPartidas (ArrayList<Partida>):** Una lista static para el histial de las partidas jugadas.

Constantes:

5. **COLOR_RESET, COLOR_RED, COLOR_GREEN, COLOR_YELLOW, COLOR_BLUE, COLOR_PURPLE:** Contantes para dar color.

Clase ConexionBBDD

Variables:

1. **Cnx (Connection):** Es la conexión activa con la base de datos.
2. **textoInsert (String):** Es la consulta SQL preparada para insertar una nueva partida en la tabla partidas
3. **sentenciaInsert (PreparedStatement):** Crea una sentencia preparada a partir de la conexión activa (cnx) y el SQL que hemos definido.
4. **listaPartidas (ArrayList<Partida>):** Lista donde se van almacenando todas las partidas recuperadas desde la base de datos.
5. **textoSelect (String):** Consulta SQL que selecciona todos los datos relevantes de la tabla partidas, ordenados por la fecha de inicio.
6. **sentenciaSelect (PreparedStatement):** Prepara la sentencia para ejecutarla de forma segura sin riesgo de inyección SQL.
7. **filasBBDD (ResultSet):** Ejecuta la consulta y guarda los resultados en un objeto ResultSet, que permite recorrer fila por fila.

Constantes:

1. **URL (String):** Define la URL de conexión a la base de datos MySQL. Incluye la dirección del servidor (localhost), el puerto (3306) y el nombre de la base de datos (zombiesurvivor).
2. **USER (String):** Guarda el nombre de usuario para acceder a MySQL. En este caso, es el usuario por defecto: root.
3. **PASSWORD (String):** Es la contraseña que se usa para conectarse con el usuario root a la base de datos. Debe coincidir con la configurada en tu sistema.

Operadores:

Operadores aritméticos:

He utilizado los operadores +, - como operadores aritméticos en las casillas a las que te mueves para restar o sumar vida o energía.

```
mapa[0][1] = "Casa";
mapaVida[0][1] = -30;
mapaEnergia[0][1] = -40;
```

Operadores de comparación:

He utilizado estos operadores:

1. **>, <** Los signos de mayor o menor los suelo utilizar para hacer una acción si algo es mayor o menor que otra cosa.

```
if (vida > 100) {
    vida = 100;
}
if (vida < 0) {
    vida = 0;
}
```

2. **==** El de comprobación lo he utilizado para mostrar un mensaje por pantalla indicando que si las coordenadas del mapa son == a “Lugar seguro” entonces le muestre que ha ganado.

```
if (mapa[posX][posY] == "Lugar seguro") {
    System.out.println(x:"Enhorabuena, has llegado al lugar seguro.");
}
```

3. **!=** Los distintos de los he utilizado mayormente para salir de los bucles.

```
} while (opcionMenuSeleccionado != 0);
```

Operador de asignación:

En operadores de asignación solo he utilizado el = y lo he usado para asignar valores en las variables o incluso en las matrices para sumar o restar vida.

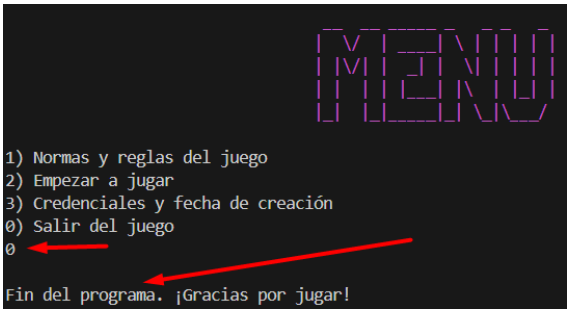
```
mapa[0][0] = LUGARINICIAL[numeroSecreto];
mapaVida[0][0] = 30;
mapaEnergia[0][0] = 30;
```

```
posX = 0;
posY = 0;
```

Estructuras de control:

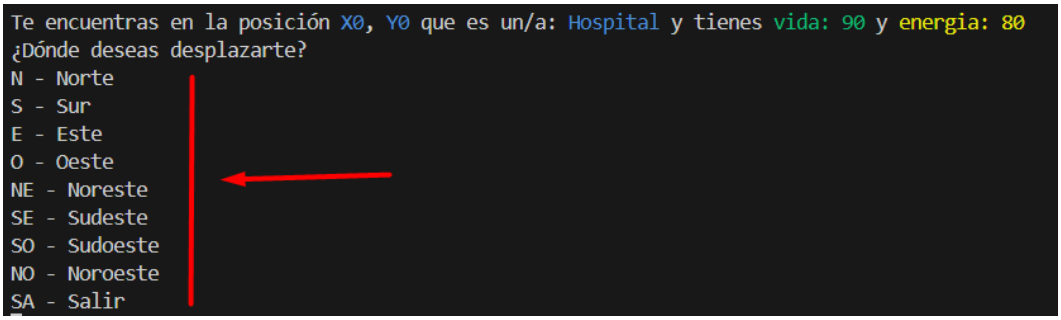
Estructura do-while:

- El do-while lo he usado en el menú principal para controlar el programa hasta que el jugador seleccione salir que es el 4 (opcionMenuSeleccionado != 0).
- También lo he utilizado para para manejar el movimiento del jugador



Estructura switch-case:

- Lo he utilizado para gestionar las opciones del menú ya que hay que tener un control.
- También lo he utilizado para los movimientos del jugador.



Estructura if o if-else:

- He utilizado if o if-else para que la vida y energía no pasen ni de 0 ni de 100
- También lo he utilizado para comprobar la victoria o derrota
- Y para posiciones válidas en el mapa.

```
if (coordenadaPermitida.contains(desplazamiento.toUpperCase())){
    vida += mapaVida[posX][posY];
    if (vida > 100) {
        vida = 100;
    }
    if (vida < 0) {
        vida = 0;
    }
```

Manejo de excepciones (try-catch):

- Usado en el método menu() para capturar entradas no válidas (InputMismatchException) y así garantizo que el programa no falle ante errores escritos por el usuario.
- También en imprimeNormas e imprimeCredenciales para manejar errores al leer la entrada de continuar (System.in.read).

```
try {
    System.out.println(COLOR_PURPLE + "\t\t\t\t\t  __   __ _____ _     _ _     _ \n" +
                                      "\t\t\t\t | |  \\/  | ___| \\ | | | | |\n" +
                                      "\t\t\t\t | |\\/| |  _| |  \\| | | | |\n" +
                                      "\t\t\t\t | |  | | |___| |\\  | |_| |\n" +
                                      "\t\t\t\t |_|  |_|_____|_| \\_|\\___/ \n" + COLOR_RESET + "");
    System.out.println("1) Normas y reglas del juego" +
    "\n2) Empezar a jugar" +
    "\n3) Credenciales y fecha de creación" +
    "\n0) Salir del juego");
    opcionMenu = sc.nextInt();  // Leemos la opción

} catch (InputMismatchException e) {
    System.out.println(x:"Caracter no válido. Introduzca un número del 0 - 3");
    sc.next();  // Limpiar el buffer en caso de un error inesperado
}
```

Estructura del proyecto

4. **ZOMBIESURVIVOR_GUILLERMO_SIERRA/**: Carpeta principal del proyecto.
    - **.vscode/**: Contiene configuraciones específicas del entorno de Visual Studio Code, como preferencias del editor y configuración del depurador.
        - **settings.json**: Archivo de configuración del entorno en VS Code. Actualmente, no se han modificado configuraciones específicas.
    - **bin/**: Carpeta que contiene los archivos compilados del proyecto.
        - **Main.java:** Archivo resultante de la compilación del código fuente.
        - **bbdd/**
            - **ConexionBBDD.class:** Archivo resultante de la compilación del código fuente.
        - **mapa/** Paquete creado para el mapa del juego
            - **Mapa.class:** Archivo resultante de la compilación del código fuente.
            - **Lugar.class:** Archivo resultante de la compilación del código fuente.
        - **personaje/** Paquete creado para los personajes del juego
            - **Personaje. class:** Archivo resultante de la compilación del código fuente.
            - **Jugador. class:** Archivo resultante de la compilación del código fuente.
            - **Zombie.class** Archivo resultante de la compilación del código fuente.
        - **utilidades/** Paquete creado para leer los datos introducidos por el usuario
            - **Utilidades.class:** Archivo resultante de la compilación del código fuente.
        - **zombieSurvivor/** Paquete que incluye el control del juego
            - **ZombieSurvivor.class**: Archivo resultante de la compilación del código fuente.
            - **Juego.class**: Archivo resultante de la compilación del código fuente.
            - **Partida.jclass:** Archivo resultante de la compilación del código fuente.
    - **lib/**: Carpeta que contiene los .jar
        - **mysql-connector-j-8.0.33.jar:** Sirve para poder conectarse con la BBDD
    - **src/**: Carpeta que contiene los paquetes y .java del proyecto.
        - **Main.java:** Fichero que llama a controlarJuego que está en ZombieSurvivor y desde este Main.java se ejecuta todo ya que tiene la main.

  - **bbdd/** Paquete creado para la conexión con la BBDD
    - **ConexionBBDD.java:** Clase que conecta, desconecta, inserta, muestra contenido de la BBDD
  - **mapa/** Paquete creado para el mapa del juego
    - **Mapa.java:** Clase que gestiona el tablero de juego y el movimiento del jugador. También se encarga de determinar qué casillas contienen zombis y de asignar lugares seguros.
    - **Lugar.java:** Clase encargada de definir los distintos lugares dentro del mapa del juego. Estos lugares afectan la vida y energía del jugador y pueden contener zombies.
  - **personaje/** Paquete creado para los personajes del juego
    - **Personaje.java:** Clase abstracta que representa a cualquier personaje en el juego. Es la clase base de Jugador y Zombie, proporcionando atributos comunes como vida y energía.
    - **Jugador.java:** Clase que representa al jugador en el juego. Se encarga de controlar su vida y energía, asegurando pase de 100 no bajen de 0.
    - **Zombie.java:** Clase que representa a los zombies en el juego. Su función principal es atacar al jugador, quitándole vida si lo encuentra en su camino.
  - **utilidades/** Paquete creado para leer los datos introducidos por el usuario
    - **Utilidades.java**: Fichero creado para leer los datos introducidos por el usuario, int, String…
  - **zombieSurvivor/** Paquete que incluye el control del juego
    - **ZombieSurvivor.java**: Es el fichero que contiene el control del juego, gracias al menú.
    - **Partida.java:** Fichero que va a contener los datos de la partida como la fecha, hora inicio/fin, movimientos o si gana o pierde.
    - **Juego.java**: Fichero creado para controlar la funcionalidad del juego.
- **documentación/:**
  - **Especificacion_Analisis_Guillermo_Sierra.docx**: Documento en formato Word que contiene la especificación y análisis del proyecto.
  - **Especificacion_Analisis_Guillermo_Sierra.pdf**: Versión en PDF del documento de especificación y análisis del proyecto.
  - **ZombieSurvivor.mdj**: Es el fichero UML de mi videojuego actual.
  - **ZombieSurvivorUML.pdf:** Contiene una imagen del UML
- **ZombieSurvivorMain.jar**: Archivo JAR del proyecto, empaquetado y listo para ser ejecutado.
- **README.md**: Archivo con una descripción del proyecto, instrucciones de instalación, ejecución y detalles adicionales.
- **Partidad.txt:** Es el fichero que se crea si no existe al jugar una partida, una vez este creado las partidas se guardan en este.
- **sqlZombieSurvivor.sql:** Contiene la BBDD del juego, se puede modificar, borrar, crear desde este fichero .sql.

Estado de implementación

Caso de Uso	Estado	Detalles
Creación del Mapa	Implementado	Generación del mapa, inicializando correctamente las áreas y atributos.
Movimiento del Jugador	Implementado	Se permite movimiento en múltiples direcciones, con control de límites, actualización de estado y validación de entrada.
Verificación de Victoria	Implementado	El jugador llega al "lugar seguro" y se activa el mensaje de victoria.
Verificación de Derrota	Implementado	Si los puntos de vida o energía llegan a cero, se activa el mensaje de derrota y se termina el juego.

Requisito	Estado	Detalles
Interfaz de Usuario (UI)	Implementado	La salida en consola utiliza colores para mejorar la legibilidad y ofrece mensajes claros.
Rendimiento	Implementado	Las operaciones responden de forma rápida, actualizando en tiempo real el estado del jugador y la visualización del mapa.
Portabilidad	No Comprobado	Actualmente probado solo en un entorno de línea de comandos, no se ha verificado la compatibilidad en otras plataformas.
Escalabilidad	Implementado	El diseño del código permite futuras expansiones, como aumentar el tamaño del mapa, añadir más lugares...
Fiabilidad	Implementado	Manejo de entradas no válidas con mensajes amigables, evitando que el programa se cierre por errores inesperados.

Posibles mejoras futuras

Ampliación del tablero:

- Permitir que el tamaño del tablero sea configurable, por ejemplo, 20x20 o 50x50.
- IMPLEMENTADO

Generación de Lugares Aleatorios:

- Permitir que los lugares que hay ahora por defecto en las casillas se pongan aleatoriamente, incluido el Lugar seguro para que cada partida sea distinta.
- IMPLEMENTADO

Modificación de Vida y Energía dependiendo de tu elección:

- Al caer en una casilla que te haga varias preguntas y dependiendo de tu elección tu vida y energía variara en positivo o negativo.
- SEMI IMPLEMENTADO

Minijuegos:

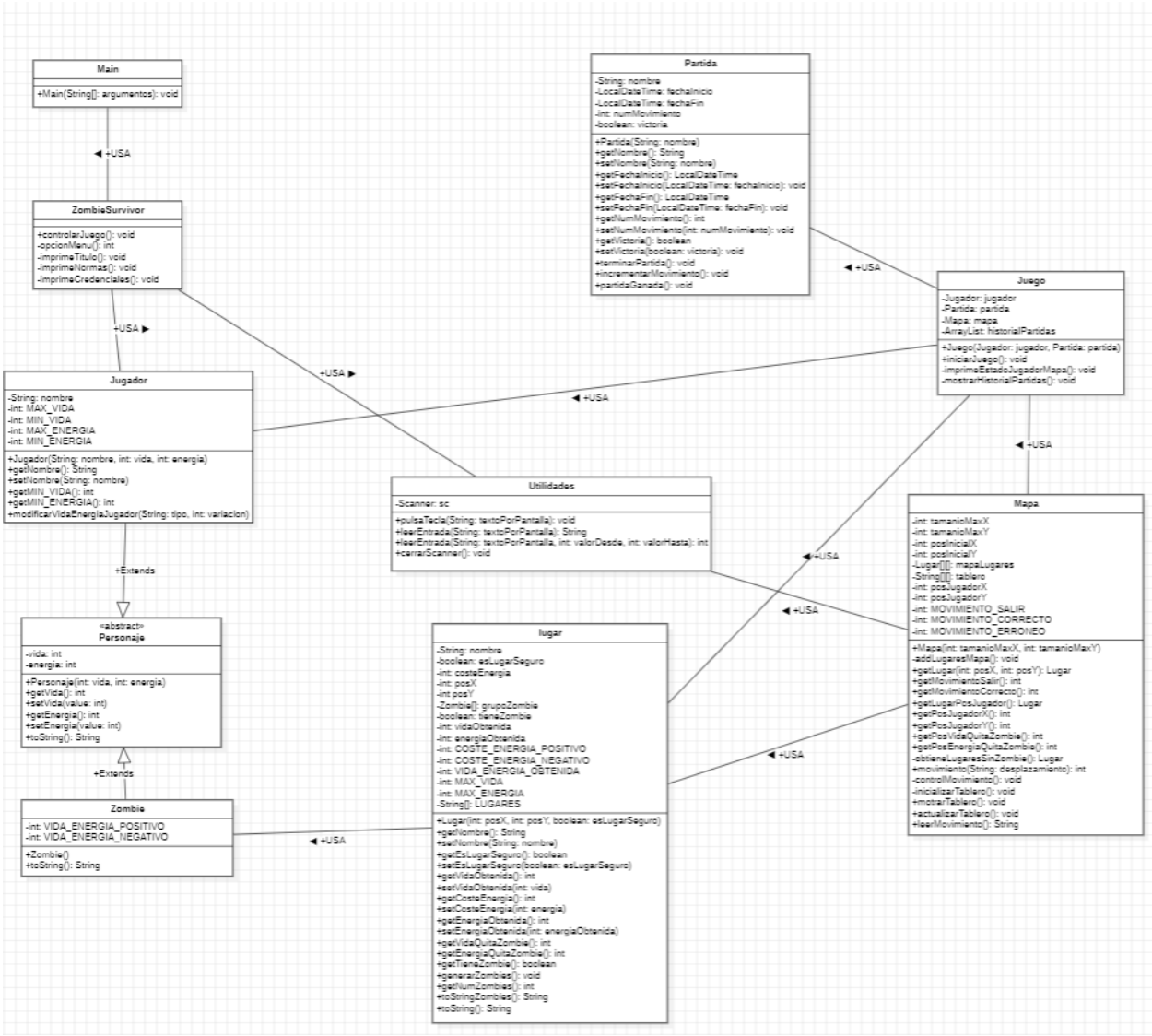
- Al igual que la posible futura mejora anterior en la cual se hacen preguntas pues esta mejora quiero que al caer en alguna casilla te haga algún minijuego ara ganar o perder vida y energía como:
    1. Adivinar un número del 1 al X en 5 intentos
    2. Un ahorcado
    3. Un wordle

Y algunos más que se me ocurrirán a lo largo del tiempo

Minas/Obstáculos en el Mapa:

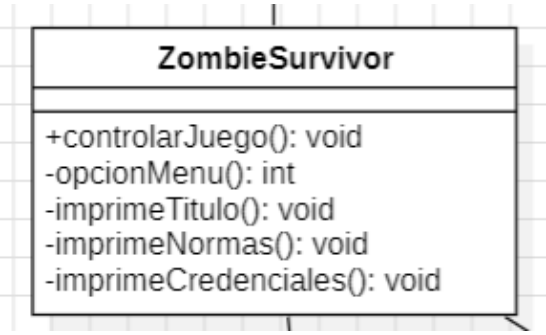
- Al igual que quiero que los lugares se pongan aleatoriamente también quiero que se pongan minas u obstáculos como:
    1. Personas que matan
    2. Minas
    3. Grupos enormes de zombies
    4. Barrancos

Diagrama UML

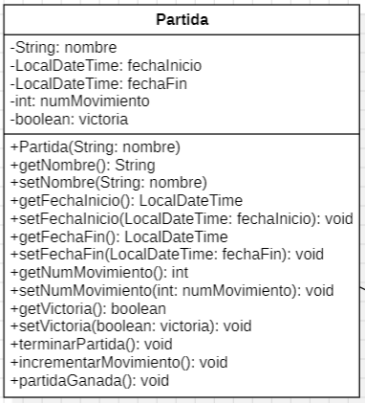


Clases y el uso de cada una:

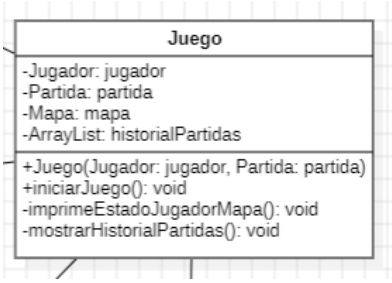
1. **ZombieSurvivor.java**: Clase ZombieSurvivor. Controla el menú, permite iniciar una partida, mostrar normas y credenciales. Maneja la creación del jugador y la partida, e imprime información en la consola con formato de colores.



2. **Partida.java:** La clase Partida representa una sesión de juego en ZombieSurvivor. Permite registrar el inicio y fin de la partida, contar los movimientos realizados y determinar si el jugador ha ganado o perdido. Proporciona métodos para finalizar la partida, incrementar el número de movimientos y marcar la partida como ganada.

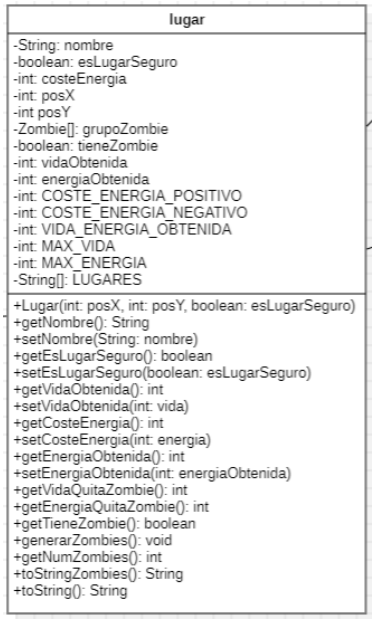


3. **Juego.java**: La clase Juego gestiona la lógica principal del juego ZombieSurvivor. Se encarga de inicializar al jugador, generar el mapa y controlar el flujo del juego, permitiendo el desplazamiento del jugador y la actualización de su estado (vida y energía) en función de los eventos que ocurran en el mapa. También maneja las condiciones de victoria y derrota.

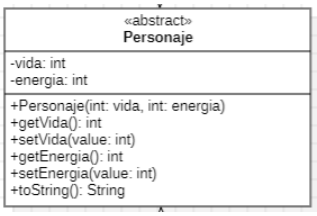


4. **Mapa.java:** Representa el mapa del juego, compuesto por una cuadrícula de lugares donde el jugador puede moverse. Gestiona la generación de lugares, la colocación de zombies y el control del movimiento del jugador. También maneja la representación visual del mapa mediante un tablero.

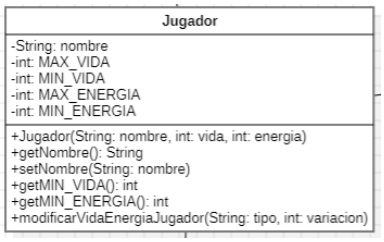
5. **Lugar.java:** Representa una ubicación en el mapa del juego. Puede ser segura o peligrosa, afectar la vida y energía del jugador y contener zombies que infligen daño. También gestiona su nombre, posición y costos de energía.



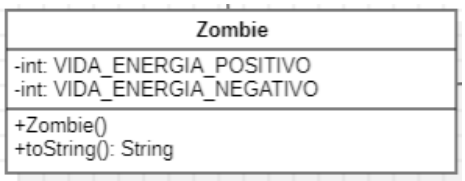
6. **Personaje.java:** Clase Personaje, es de tipo abstracta porque en un futuro añadiré métodos abstractos, también es la clase madre de la que heredarán las variables de vida y energia las clases Jugador y Zombie



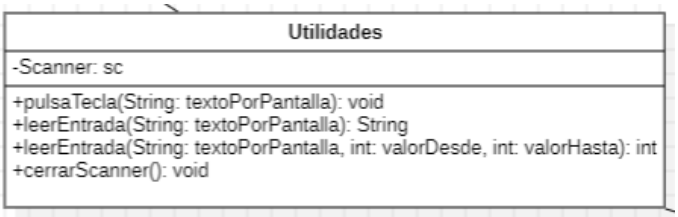
7. **Jugador.java:** Representa al jugador en el juego, heredando de la clase Personaje. Gestiona su nombre, vida y energía, permitiendo modificar estos atributos en función de las acciones dentro del juego.



8. **Zombie.java:** Clase Zombie, hereda algunas variables de la clase madre que en este caso es Personaje, la función de la clase Zombie es quitar vida al jugador



9. **Utilidades.java**: Fichero creado para leer los datos introducidos por el usuario, int, String…



10. **Main.java:** Clase principal que inicia la ejecución del juego ZombieSurvivor. Llama al método que controla el flujo del juego desde la clase ZombieSurvivor.