

PROYECTO PROGRAMACIÓN 2022/2023  
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

# HARRY POTTER

## EL LABERINTO ENCANTADO

---

CARLOS COLLADO GARRIDO



# INDICE

<b>INDICE.....</b>	<b>2</b>
<b>INTRODUCCIÓN.....</b>	<b>3</b>
<b>OBJETIVO.....</b>	<b>3</b>
<b>EVALUACIÓN.....</b>	<b>3</b>
<b>CONTEXTO.....</b>	<b>3</b>
<b>Fase 1.....</b>	<b>4</b>
1. Pared.....	5
2. Mapa.....	6
Código del programa principal para probar los mapas en esta fase.....	6
<b>Fase 2.....</b>	<b>8</b>
3. Harry.....	8
Pseudocódigo del backtracking para la búsqueda del camino.....	10
<b>Fase 3.....</b>	<b>11</b>
4. Adversidad.....	11
Código en el que se crea el mapa con las paredes y adversidades.....	11
Resultados de la simulación con los mapas descritos anteriormente.....	13
<b>Fase 4.....</b>	<b>14</b>
5. GUI.....	14
6. registro.log.....	15
7. JavaDoc.....	16

## INTRODUCCIÓN

En este documento se describen los detalles para la construcción de un proyecto de programación usando el lenguaje Java. El proyecto se divide en varias fases que se irán detallando en este mismo documento. Cada fase supondrá la incorporación de nuevas partes al proyecto y/o la modificación de algunas existentes.

## OBJETIVO

El desarrollo de este proyecto tiene como fin proporcionar al alumnado una visión general y práctica de todo o casi todo lo estudiado durante el curso en el módulo de programación.

Además, también servirá como refuerzo para aquellos alumnos que tienen dificultades en la comprensión de conceptos de cara a la convocatoria ordinaria.

## EVALUACIÓN

La evaluación de la práctica se divide fundamentalmente en dos partes:

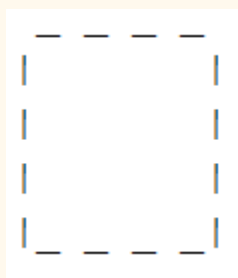
1. Evaluación continua: seguimiento semanal para comprobar el trabajo de forma escalonada e individualizada del proyecto. De este modo permite al alumno evitar agobios, estrés y posibles fallos que deriven en problemas futuros por dependencias de código u otros motivos. También permite el conocimiento del profesor sobre el esfuerzo realizado y el estado actualizado del proyecto de cada alumno.
2. Evaluación final y defensa: una vez finalizado el proyecto, se hará una entrega de la práctica completa y se realizará una defensa en la que el profesor podrá hacer diversas cuestiones al alumno para comprobar que el alumno comprende el trabajo realizado y es capaz de responder a dudas o posibles cambios en el código. **Superada la defensa** se informará de la nota obtenida en la práctica.

## CONTEXTO

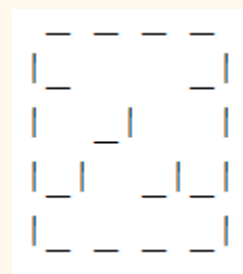
El desarrollo consiste en la simulación de un juego sobre Harry Potter. El personaje se adentrará en un laberinto e intentará llegar a la salida superando todas las adversidades que se puedan producir. Esta simulación podrá variar en función de una configuración inicial que hará el mapa más o menos grande y otorgará al personaje mayor o menor capacidad de lucha.

## Fase 1

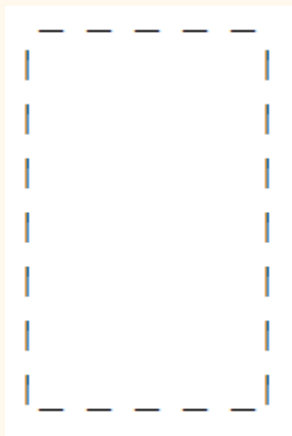
Llegados a este punto es importante definir las bases para comenzar la elaboración del código. Como se ha mencionado, la simulación se desarrolla sobre un laberinto que se construye como un mapa de dimensión variable. Inicialmente, el mapa se crea indicando el ancho y el alto pero se crea vacío. Una vez creado, se añaden paredes que definen la forma final del laberinto.



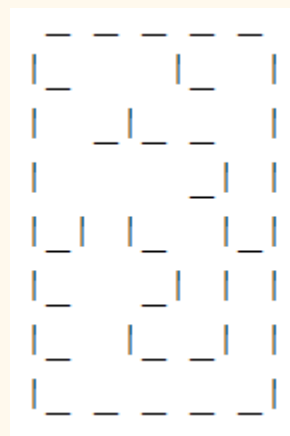
*Mapa 4x4 vacío*



*Mapa 4x4 con paredes*



*Mapa 7x5 vacío*



*Mapa 7x5 con paredes*

**\*\*\*Ejemplos de mapas de diferente tamaño antes y después de añadir las paredes\*\*\***

Las paredes que se añaden al mapa estarán descritas en un fichero que define la configuración inicial de la simulación. Existirán varios ficheros de carga dando lugar a distintas simulaciones con diferentes mapas.

Es importante tener en cuenta que Harry siempre inicia su camino en el laberinto por la parte superior izquierda y para conseguir la victoria y su respectiva salida del laberinto debe llegar a la esquina inferior derecha.

## 1. Pared

Cada pared estará definida por dos posiciones  $x$  e  $y$ , de modo que Harry no podrá pasar de forma directa de la posición  $x$  a la  $y$ . Estas posiciones son numéricas y absoluta, por lo que, por ejemplo, para un mapa de  $4 \times 4$ , tendremos un total de 16 posiciones que irán de la 0 a la 15, tal y como se muestra en la siguiente imagen:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

*Mapa de  $4 \times 4$  con las posiciones*

Es importante destacar que sobre una posición absoluta siempre es posible calcular las coordenadas (fila y columna) en las que se ubica y viceversa a través de las siguientes normas:

$$\text{pos} = f * \text{ancho} + c$$

$$f = \text{pos} / \text{ancho}$$

De este modo, la posición 9 en el mapa representado anteriormente de  $4 \times 4$  ocuparía las coordenadas:

$$f = 9 / \text{ancho} = 9 / 4 = 2$$

$$c = \text{pos} - f * \text{ancho} = 9 - 4 * 2 = 9 - 8 = 1$$

Conocido lo anterior, observar la siguiente figura:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

*Mapa de 4x4 con posiciones y paredes*

En este caso se señalan dos paredes definidas por las posiciones 5,9 (roja) y 10,11 (verde).

Por último, sobre las paredes, es necesario concretar que la pared 5,9 es exactamente la misma que la 9,5. Este aspecto es especialmente relevante en el apartado del mapa.

## 2. Mapa

Para crear un mapa será necesario especificar el ancho y el alto. Almacenará un conjunto de paredes y debe proporcionar la capacidad de pintar usando los caracteres de barra baja '\_' y barra vertical '|' representados por los valores 95 y 124 en ascii respectivamente.

### Código del programa principal para probar los mapas en esta fase

```
Mapa m = new Mapa(4,4);
System.out.println(m);
m.anadirPared(0,4);
m.anadirPared(3,7);
m.anadirPared(5,6);
m.anadirPared(5,9);
m.anadirPared(8,9);
m.anadirPared(8,12);
m.anadirPared(10,11);
m.anadirPared(10,14);
```

```
m.aniadirPared(11,15);  
System.out.println(m);
```

```
m = new Mapa(7,5);  
System.out.println(m);  
m.aniadirPared(0,5);  
m.aniadirPared(2,3);  
m.aniadirPared(3,8);  
m.aniadirPared(6,7);  
m.aniadirPared(6,11);  
m.aniadirPared(7,12);  
m.aniadirPared(8,13);  
m.aniadirPared(13,14);  
m.aniadirPared(13,18);  
m.aniadirPared(15,16);  
m.aniadirPared(15,20);  
m.aniadirPared(16,17);  
m.aniadirPared(17,22);  
m.aniadirPared(18,19);  
m.aniadirPared(19,24);  
m.aniadirPared(20,25);  
m.aniadirPared(22,23);  
m.aniadirPared(22,27);  
m.aniadirPared(23,24);  
m.aniadirPared(25,30);  
m.aniadirPared(26,27);  
m.aniadirPared(27,32);  
m.aniadirPared(28,29);  
m.aniadirPared(28,33);  
System.out.println(m);
```

## Fase 2

Tras haber construido las clases Pared y Mapa y haber conseguido pintar el laberinto, en esta fase se trata de introducir a Harry en el laberinto para que aprenda a encontrar el camino hasta la salida. Esta tarea no será sencilla, debido a que no se proporciona la ruta que se debe seguir, por lo que tendrá que avanzar a base de prueba y error... ¿O no? ¿Se podría usar algún algoritmo que encuentre el camino de Harry a la salida? Pues sí...

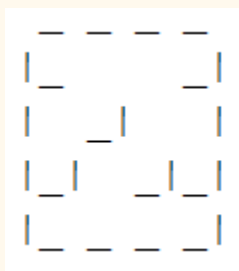
### *BACKTRACKING*

Este algoritmo se aplicará a modo de recurso mágico, **antes de iniciar su camino en el laberinto**. De este modo conoce la ruta a seguir cuando inicia el viaje.

### 3. Harry

Aunque en fases posteriores se incluirán más cosas en el mapa, en este momento hay que centrarse en el protagonista, Harry. Como se explicó, debe recorrer un camino que le lleve desde la sala superior izquierda (0) hasta la sala inferior derecha ( $\text{alto} \cdot \text{ancho} - 1$ ). Cada paso que da, lo hace en una dirección concreta (N, S, E u O) por lo que la ruta quedará como un conjunto de direcciones.

Así pues, la ruta para el mapa de la figura siguiente sería:



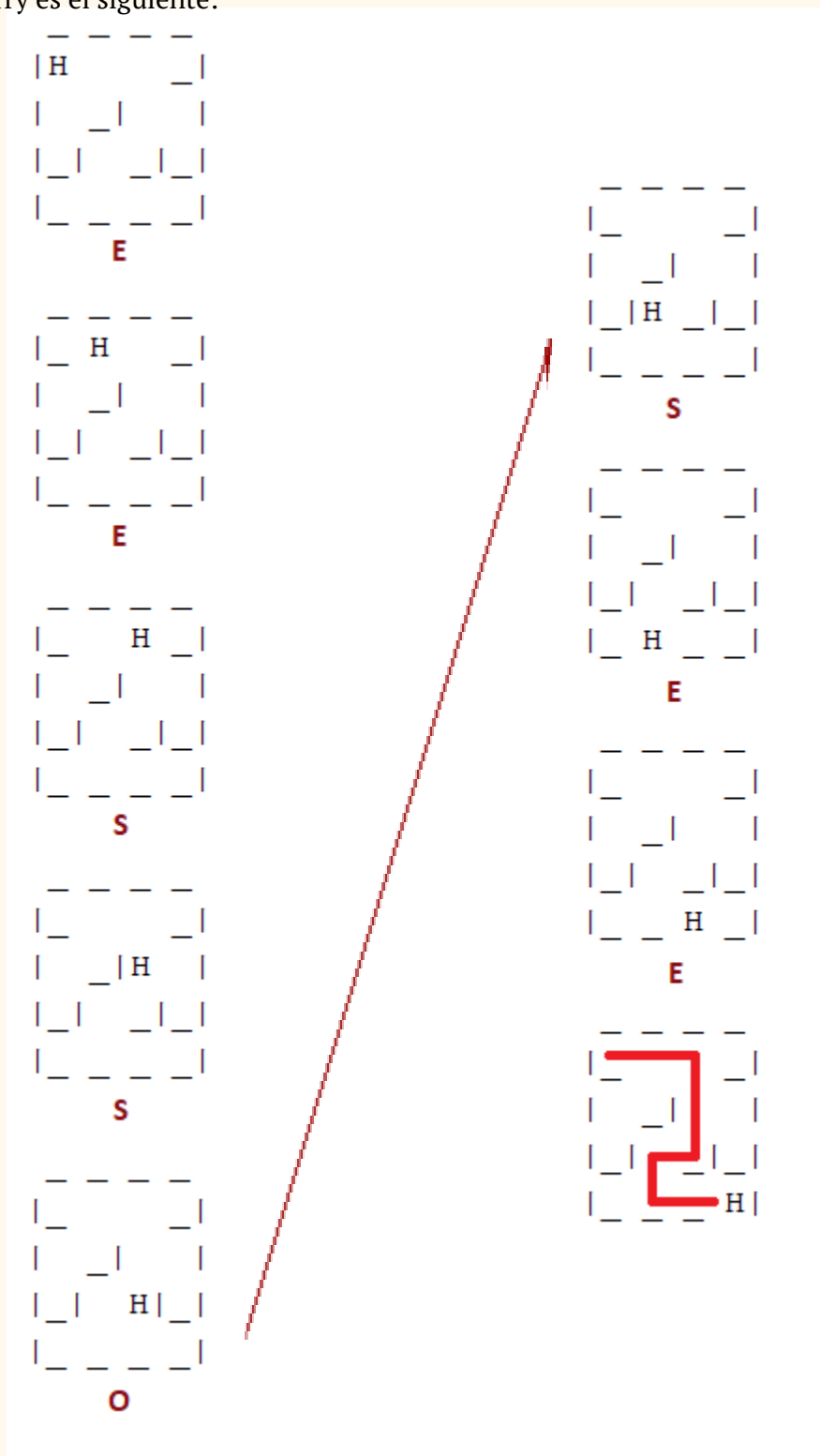
E,E,S,S,O,S,E,E

Resumiendo, el objetivo consiste en ser capaces de generar una ruta válida para llegar desde la sala inicial hasta la sala final usando un algoritmo BACKTRACKING.

El personaje Harry se representará en el mapa con el carácter H, y hay que tener en cuenta que por simplicidad y limitaciones de la consola, cuando se escribe 'H' no se puede escribir '\_' por lo que en aquella sala en la que se encuentre Harry no se podrá



pintar una pared con la sala inferior. De este modo, para el mapa de 4x4, el movimiento de Harry es el siguiente:



## Pseudocódigo del backtracking para la búsqueda del camino

```
boolean calcularRuta(m, salasVisitadas) {  
    Si (estoyEnUltimaSala) {  
        ResetearSalaActual  
        return true;  
    } Sino {  
  
        Si (noEsUltimaFila Y noHayParedConSalaSur Y noHePasadoPorSalaSur) {  
            añadir salaSur a visitadas  
            añadir a la ruta DireccionSur  
            salaActual = salaSur  
            Si (calcularRuta(m, salasVisitadas) es false) {  
                salaActual vuelve a la que era;  
                borrar la ultima dirección añadida a la ruta  
            }Sino  
                return true;  
        }  
  
        Repetir este Si para las 3 direcciones restantes en el orden E,O,N  
    }  
    return false;  
}
```

## Fase 3

Llega el momento de la verdad. El camino de Harry por el laberinto puede presentar adversidades que intentarán impedir su llegada a la sala final. Esas adversidades estarán ubicadas en algunas salas del mapa, en el que se insertarán del mismo modo que se hace con las paredes.

### 4. Adversidad

Como se ha descrito anteriormente, representan una oposición al personaje para su avance en el mapa hasta la sala destino. Al iniciar una adversidad en el mapa, ésta permanecerá estática (en la misma sala donde se define) durante toda la simulación, a excepción de que el personaje llegue a la sala donde la adversidad se encuentra. En ese caso, se lanza un ataque al personaje y la adversidad desaparece. Se podrán dar dos tipos de adversidades:

1. Dementores: Teniendo en cuenta que Harry comienza la simulación con 100 de salud, los dementores consiguen restar 30 a dicha salud cuando realizan el ataque al encontrarse en la misma sala.
2. Viento: Esta adversidad ataca al personaje trasladándolo a la sala inicial, es decir, como si la simulación comenzase de cero (recalculando la ruta), pero manteniendo su salud en el mismo nivel.

Ojo al cansancio. No es una adversidad definida como tal, pero hay que tener en cuenta que a medida que pasa el tiempo, Harry sufre cansancio y eso afecta a su salud. De este modo cada vez que realiza 10 movimientos resta 20 a su salud.

Cabe destacar que desde el punto de vista del mapa, las adversidades (sea la que sea) realizan ataques al personaje, cada una a su manera. Dicho lo cual, es conveniente pensar bien la forma en la que se va a construir esta parte del código.

### Código en el que se crea el mapa con las paredes y adversidades

```
Mapa m = new Mapa(4,4);
```

```
m.aniadirPared(0,4);
```

```
m.aniadirPared(3,7);
```

```
m.aniadirPared(5,6);
```

```
m.aniadirPared(5,9);
m.aniadirPared(8,9);
m.aniadirPared(8,12);
m.aniadirPared(10,11);
m.aniadirPared(10,14);
m.aniadirPared(11,15);

m.nuevaAdversidad(new Viento("Viento Norte", 2));
m.nuevaAdversidad(new Viento("Viento Sur", 13));
m.nuevaAdversidad(new Dementor(6));
m.nuevaAdversidad(new Dementor(10));

/*
//Mapa de 7x5
Mapa m = new Mapa(7,5);

m.aniadirPared(0,5);
m.aniadirPared(2,3);
m.aniadirPared(3,8);
m.aniadirPared(6,7);
m.aniadirPared(6,11);
m.aniadirPared(7,12);
m.aniadirPared(8,13);
m.aniadirPared(13,14);
m.aniadirPared(13,18);
m.aniadirPared(15,16);
m.aniadirPared(15,20);
m.aniadirPared(16,17);
m.aniadirPared(17,22);
m.aniadirPared(18,19);
m.aniadirPared(19,20);
m.aniadirPared(19,24);
m.aniadirPared(20,25);
m.aniadirPared(22,23);
m.aniadirPared(22,27);
```

```
m.aniadirPared(23,24);
m.aniadirPared(25,30);
m.aniadirPared(26,27);
m.aniadirPared(27,32);
m.aniadirPared(28,29);
m.aniadirPared(28,33);

m.nuevaAdversidad(new Viento("Viento Norte", 2));
m.nuevaAdversidad(new Viento("Viento Sur", 32));
m.nuevaAdversidad(new Dementor(10));
m.nuevaAdversidad(new Dementor(21));
m.nuevaAdversidad(new Dementor(26));
m.nuevaAdversidad(new Dementor(19));

*/
```

## Resultados de la simulación con los mapas descritos anteriormente

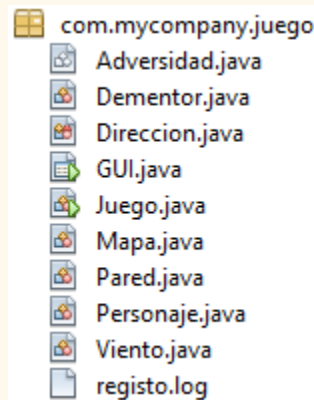
En el aula virtual, se encontrarán dos ficheros denominados “*Resultado4x4.txt*” y “*Resultado7x5.txt*”. Contienen el OutPut de la simulación con los mapas de 4x4 y 7x5 respectivamente.

## Fase 4

En este último apartado se trata de incorporar al proyecto dos nuevas funcionalidades: una interfaz gráfica y la generación de un archivo .log que registre lo ocurrido en el juego.

Teniendo en cuenta todo el trabajo que habrá que realizar para el proyecto, el resultado del explorador tras ejecutar esta última fase, debería ser el siguiente:

- 10 archivos de los cuales:
  - 1 archivo es un tipo enumerado
  - 7 son clases (1 abstracta y 1 ejecutable que contiene el **main()**)
  - 1 es la ventana gráfica (también ejecutable)
  - 1 es un archivo .log a modo registro que almacena la salida por pantalla

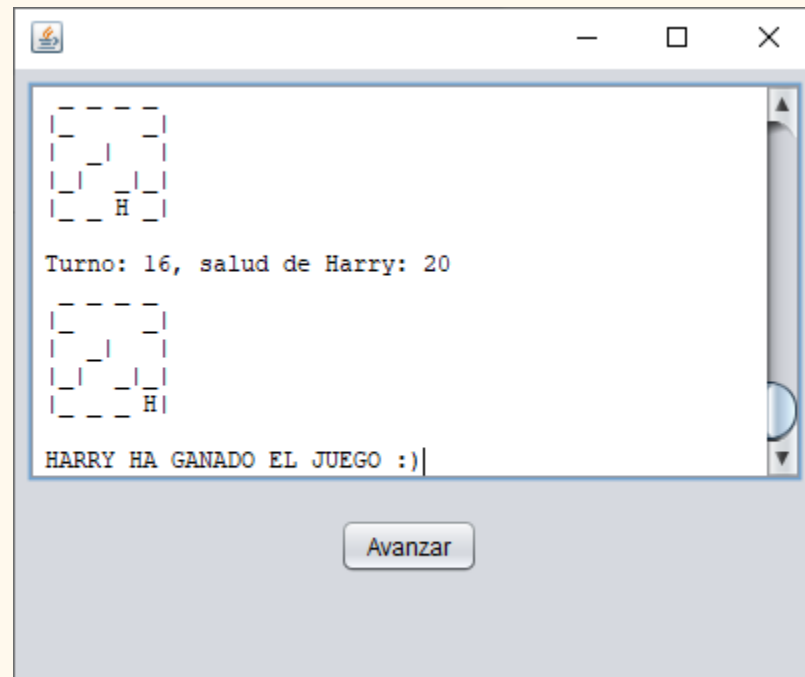


## 5. GUI

La interfaz gráfica que se debe incorporar es muy sencilla. Consiste en una ventana de 400 x 300 que contiene un área de texto en el que se irá representando la información que salía por consola, y un botón de avance que permite al usuario ir ejecutando turno a turno.

Cuando la simulación acaba, el botón de avance ya no realiza ninguna acción, por lo que el usuario deberá cerrar la ventana para finalizar la ejecución.

A continuación se muestra una imagen tras realizar la simulación del mapa de 4x4:

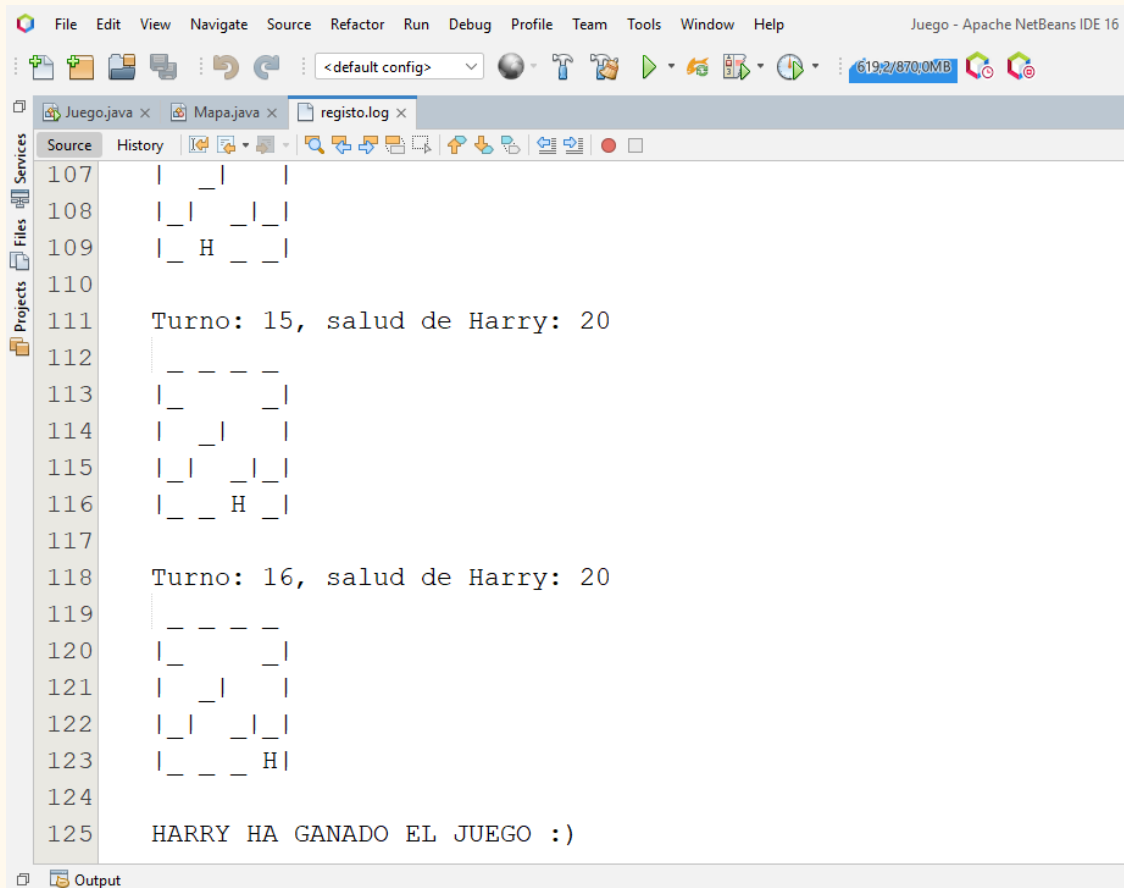


## 6. registro.log

Por definición en informática, se usa el término registro, log o historial de log para referirse a la grabación secuencial en un archivo o en una base de datos de todos los acontecimientos que afectan a un proceso particular. De esta forma constituye una evidencia del comportamiento del sistema. Por tanto, en este caso tiene como objetivo quedar registrada toda la información que se ha mostrado en la ventana gráfica. De este modo, si en algún momento se diera una situación inapropiada, se podrían realizar comprobaciones y verificaciones del resultado que se obtuvo.

Cada vez que se ejecuta la simulación, se debe generar el .log en la misma ubicación que el resto de clases del proyecto, teniendo en cuenta que si el fichero ya existía, se reemplaza por el nuevo.

Cabe mencionar que tanto la salida por ventana gráfica como el registro log deben contener exactamente lo mismo, y que **si todo es correcto, existirá una coincidencia total con los outputs proporcionados en la fase 3.**



```
107 | _ |
108 | _ | _ |
109 | _ H _ |
110
111 Turno: 15, salud de Harry: 20
112 | _ _ _ |
113 | _ | _ |
114 | _ | _ |
115 | _ | _ |
116 | _ _ H _ |
117
118 Turno: 16, salud de Harry: 20
119 | _ _ _ |
120 | _ | _ |
121 | _ | _ |
122 | _ | _ |
123 | _ _ _ H |
124
125 HARRY HA GANADO EL JUEGO :)
```

## 7. JavaDoc

Es importante recordar que es la última entrega del proyecto y, por lo tanto, debe estar totalmente completo, incluyendo la documentación interna en formato JavaDoc.

*FIN*