

# Práctica 2

## Entornos Servidor: Maze Game



Eduardo González Lorenzo  
2º año Desarrollo de Aplicaciones Web  
Desarrollo Web en Entorno Servidor  
Curso 2022-2023

# Índice

<b>1. Introducción:</b>	<b>4</b>
1.1 Explicación de la aplicación:	4
1.2 Especificaciones de la información:	4
1.2.1 Comunicación entre frontend y backend:	4
1.2.2 Base de datos:	4
<b>2. Recursos utilizados</b>	<b>5</b>
2.1 IntelliJ	5
2.2 VisualStudio Code	5
2.3 Apache Tomcat	5
2.4 Maven	5
2.5 Docker	6
2.6 MySQL	6
2.7 Phpmyadmin	6
2.8 Navegador	6
2.9 GitHub	6
<b>3. Instrucciones</b>	<b>7</b>
3.1 Selector de nivel	7
3.2 Movimiento del jugador	7
3.2.1 Muros	8
3.2.2 Puertas	8
3.3 Coger objetos	8
3.3.1 Monedas	8
3.3.2 Llaves	9
3.3.3 Inventario:	9
3.4 Interacción con lados	9
3.4.1 Muros	9
3.4.2 Puertas	9
3.5 Cómo ganar	10
3.5.1 Meta	11
3.5.2 Registrar puntuación	11
3.6 Funciones extra.	11
3.6.1 Reiniciar	11
3.6.2 Nueva partida	12
<b>4. Los niveles:</b>	<b>13</b>
4.1 Tutorial	13
4.2 Juego	13

<b>5. Estructura teórica de la aplicación</b>	<b>14</b>
5.1 Nivel Cliente	14
5.2 Nivel Servidor	14
5.3 Nivel Base de datos	15
<b>6. La aplicación</b>	<b>16</b>
6.1 Nivel cliente	17
6.2 Nivel Servidor	19
Controllers:	19
Services:	20
Model:	21
DAO:	21
Filter:	21
Utils:	22
Excepciones:	22

# 1. Introducción:

## 1.1 Explicación de la aplicación:

En esta práctica se ha creado tanto el front como el back de un juego. El juego consiste en una mazmorra de la cual el jugador debe escapar en el menor tiempo posible. Las mazmorras están compuestas de varias habitaciones, las cuales a su vez tendrán 4 paredes que pueden o no tener puertas. Estas puertas podrán estar abiertas o cerradas. Las habitaciones además pueden estar vacías o tener una moneda y/o una llave. Las monedas son necesarias para coger las llaves, las cuales se usan para abrir puertas. Las puertas conectan habitaciones.

Este juego soporta la existencia de varias partidas (sesiones) de forma simultánea. Además del juego en sí, la aplicación cuenta con un selector de niveles, pantallas de error, un ranking con los mejores jugadores y un formulario para, una vez superada la mazmorra, un jugador pueda registrar su resultado y guardarlo en la base de datos.

## 1.2 Especificaciones de la información:

A continuación se va a explicar el procesamiento de la información que realiza la aplicación, tanto para transmitirlo como para crearlo y guardarlo.

A nivel servidor la información se crea y gestiona a nivel interno, siendo *intellij* el encargado de tener la información en la memoria RAM mientras el servidor esté activo. El envío de información de *frontend* a *backend* se realizará mediante formularios, haciendo uso tanto del método post como del método get. El almacenamiento de los ganadores y el envío de la información del *backend* al *frontend* tienen un tratamiento especial que se explicará a continuación:

### 1.2.1 Comunicación entre frontend y backend:

Para la transmisión de información desde el servidor al cliente se ha utilizado el formato JSON. Este formato permite convertir objetos a formato *String*, el cual luego puede ser nuevamente convertido en objetos por javascript, facilitando la transmisión de información del servidor al cliente, permitiendo utilizar en ambos apartados el mismo "formato".

### 1.2.2 Base de datos:

La base de datos se encarga de almacenar la información de los jugadores que han conseguido superar una mazmorra. La información almacenada es el id del ganador, el nombre del jugador, el nombre de la mazmorra superada y el tiempo utilizado. Esta información se guarda en MySQL, permitiendo la persistencia de datos aún cuando el servidor se apaga.

## 2. Recursos utilizados

Para la realización de esta práctica, así como las pruebas y su ejecución, se han utilizado los siguientes recursos y herramientas:

### 2.1 IntelliJ

Editor de texto de Java, el cual se ha utilizado para realizar toda la parte servidor de la práctica, así como para implementar las librerías Maven y para ejecutar el programa en el servidor Tomcat, recursos que se explicarán más adelante.

### 2.2 VisualStudio Code

Editor de texto global que se ha utilizado para la parte cliente, creando los archivos jsp, css y js necesarios para el correcto funcionamiento de la interfaz de usuario, incluido el canvas. También se ha utilizado para crear copias html de los ficheros jsp, facilitando su edición y permitiendo realizar pruebas estéticas y de funcionalidad con más facilidad.

### 2.3 Apache Tomcat

Es un contenedor web utilizado para implementar las especificaciones de servlets y de las páginas JSP. Se encarga de compilar los archivos JSP para convertirlos en servlets.<sup>1</sup>

Un servlet es una clase de Java que ejecuta pequeños programas en el contexto de un navegador web, permitiendo generar páginas web de forma dinámica a partir de los parámetros recibidos del navegador.<sup>2</sup>

Tomcat puede funcionar como servidor web por si mismo, cosa que hace en esta práctica. Debido a que Tomcat está escrito en Java cuenta con la ventaja de ser multiplataforma.

Tomcat se ha utilizado en esta práctica para ejecutar el backend de la aplicación

### 2.4 Maven

Es un software usado para la construcción y gestión de proyectos en Java basado en XML. Su funcionamiento se basa en la descripción del proyecto a construir, sus dependencias de otros módulos y componentes externos y el orden de construcción de los elementos. Además cuenta con unos objetivos predefinidos para realizar ciertas tareas definidas, como la compilación o el empaquetado del código.

Una de las características más importantes de Maven es que está listo para usarse en red, además de poder descargar plugins de su repositorio.<sup>3</sup>

Maven se ha utilizado en esta aplicación para poder comunicar el backend con el frontend gracias a la instalación de plugins como “JSLT”, “mysql connector java” o “javax servlet”.

---

<sup>1</sup> Fuente: <https://es.wikipedia.org/wiki/Tomcat>

<sup>2</sup> Fuente: [https://es.wikipedia.org/wiki/Java\\_Servlet](https://es.wikipedia.org/wiki/Java_Servlet)

<sup>3</sup> Fuente: <https://es.wikipedia.org/wiki/Maven>

## 2.5 Docker

Por último está docker, el desplegador de contenedores, en el cual se ha desplegado el contenedor que contiene el servidor apache que ejecutará la aplicación, la base de datos MySQL y el phpMyAdmin.

## 2.6 MySQL

Es uno de los sistemas gestores de datos más populares y usados. Se utiliza principalmente en aplicaciones web y puede usarse desde consola o haciendo uso de herramientas para facilitar su uso de forma más visual, como “phpMyAdmin” o “MySQL Workbench”.<sup>4</sup>

## 2.7 Phpmyadmin

Herramienta creada usando php para facilitar la administración de bases de datos MySQL a través de páginas web. Permite la creación y borrado de bases de datos y la creación, borrado y edición de tablas y campos. En resumen, se puede ejecutar cualquier sentencia SQL de forma más visual.<sup>5</sup>

## 2.8 Navegador

Utilizados para interpretar código html y ejecutar javascript. Algunos de los más famosos son Google Chrome, Firefox, Safari, Opera o Edge.

En esta práctica se ha utilizado poder comprobar el uso de la aplicación además de interactuar con ella y verificar su correcto funcionamiento.

## 2.9 GitHub

Una de las plataformas de desarrollo corporativo más famosas y utilizadas. Utiliza el sistema de control de versiones GIT. Herramienta muy útil para poder modificar código sin miedo a solapar cosas que funcionaban anteriormente. También permite trabajar la misma aplicación desde diferentes ubicaciones.

---

<sup>4</sup> <https://es.wikipedia.org/wiki/MySQL>

<sup>5</sup> <https://es.wikipedia.org/wiki/PhpMyAdmin>

### 3. Instrucciones

El objetivo del juego es, de forma resumida, llegar a la meta/salida en el menor tiempo posible. Para esto el jugador debe moverse por las distintas habitaciones que forman la mazmorra a través de puertas, las cuales en ocasiones deberá abrir ya que pueden estar cerradas. Para abrir las puertas cerradas debe conseguir monedas, las cuales le permitirán coger las llaves que abren las puertas.

Con esto dicho se va a explicar paso a paso al usuario como manejar la aplicación.

#### 3.1 Selector de nivel

Ventana inicial del juego. A través de aquí se puede seleccionar qué nivel se va a jugar. En la aplicación hay dos niveles:

- El tutorial: nivel sencillo que permite al jugador ver las mecánicas básicas del juego, las cuales son moverse, coger objetos y abrir puertas.
- Juego: nivel más complejo en el que el jugador tendrá que desenvolverse algo mejor para superar el nivel.

Simplemente selecciona el nivel que quieres jugar y presiona el botón “Jugar” para empezar tu partida.

#### 3.2 Movimiento del jugador

El personaje manejado por el jugador se encuentra siempre en el medio de la habitación. Para mover al personaje entre habitaciones el jugador dispone de 4 flechas en la esquina inferior derecha de la pantalla. Cada flecha apunta a una dirección diferente (arriba, izquierda, abajo y derecha) y, por lo tanto, mueve al personaje en dicha dirección.



---

<sup>6</sup> Imagen de las flechas en el juego.

El éxito de este movimiento depende de lo que haya en la dirección a la que se mueve el jugador. El movimiento no abre las puertas cerradas aunque tengas la llave. Para abrir las puertas debes interactuar directamente con ellas y, si tienes la llave de esa puerta, esta se abrirá y podrás moverte a través de ella. Esta interacción se explicará más en detalle más adelante.

En la siguiente imagen podemos ver un ejemplo de muro arriba y abajo (en negro), un ejemplo de puerta cerrada (a la izquierda, en rojo) y uno de puerta abierta (a la derecha en gris):

### 3.2.1 Muros

Si el jugador se mueve hacia un muro volverá al centro de la misma habitación sin haber cambiado de sala, ya que no es posible atravesar paredes (al menos en este juego).

### 3.2.2 Puertas

Si el jugador se mueve hacia una puerta pueden pasar dos cosas:

- Si la puerta está cerrada pasará lo mismo que al desplazarse hacia un muro, volviendo al centro de la misma habitación sin haber logrado movernos.
- Si la puerta está abierta el personaje se desplazará a la habitación conectada por esa puerta, apareciendo en el centro de esa nueva sala.

## 3.3 Coger objetos

En las habitaciones puede que haya objetos. Para intentar cogerlos clicas sobre ellos y tu personaje los intentará coger automáticamente. Existen dos tipos de objetos que pueden aparecer en este juego con sus respectivas interacciones, usos y posibilidades:



### 3.3.1 Monedas

Las monedas no tienen ningún requisito para cogerse, simplemente clicas y se añadirán a tu inventario. Necesitas monedas para poder coger algunas llaves.

---

<sup>7</sup> Ejemplo de llave y moneda



### 3.3.2 Llaves

Para coger las llaves tendrás que pagar un determinado número de monedas. Si no tienes las monedas suficientes no podrás coger la llave. Si las tienes perderás las monedas que cuesta la llave y esta se añadirá a tu inventario.

### 3.3.3 Inventario:

El usuario cuenta con un inventario, el cual se muestra en todo momento en pantalla:



## 3.4 Interacción con lados

El jugador puede clicar en el centro de los lados de las habitaciones para interactuar con ellos. En función del tipo de lado el jugador recibirá una respuesta u otra.

### 3.4.1 Muros

Cuando un jugador clicca sobre un muro se le informa de que no hay ninguna interacción con los muros, no pueden abrirse ni cruzarse.

### 3.4.2 Puertas

Las puertas pueden estar abiertas o cerradas.

- Abiertas: una puerta abierta no tiene más interacción que ser cruzada, por lo que al clicar se informará al jugador de que la puerta ya está abierta y puede pasar por ella.

<sup>8</sup> Ejemplo de inventario en el que el usuario está en la habitación 4 y tiene 1 llave y 2 monedas.

- Cerradas: si una puerta está cerrada cuando el jugador clique sobre ella se intentará abrir. Para esto se comprobará si el jugador tiene la llave. En caso de tenerla la puerta pasará a estar abierta. En caso de estar cerrada se informará al jugador de que no tiene la llave y que por tanto la puerta permanecerá cerrada.



9

### 3.5 Cómo ganar

Un jugador gana en el momento en el que llega al final de la mazmorra y ve la siguiente mensaje:



10

<sup>9</sup> Ejemplo de tipos de pared de una habitación

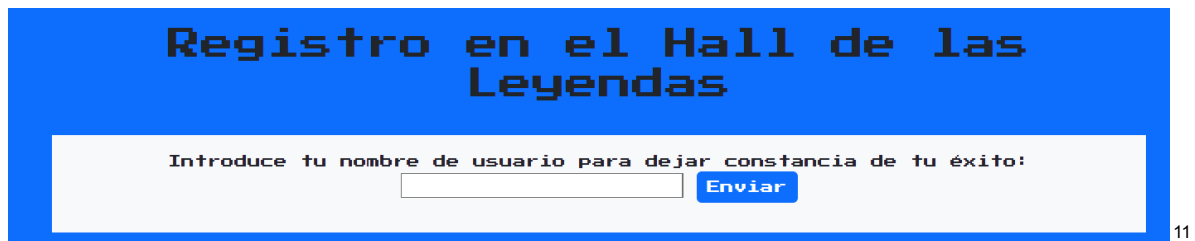
<sup>10</sup> Texto de victoria

### 3.5.1 Meta

El jugador no tiene manera de saber qué habitación es la meta hasta que entra en ella. Una vez este entra en la habitación designada como meta se muestra el mensaje visto antes y se bloquea el juego, haciendo que al clicar en cualquier lado (ya sea flecha, puerta o espacio en blanco) se redirija al usuario a la ventana de registro de puntuación. Si intentas reiniciar la partida no podrás.

### 3.5.2 Registrar puntuación

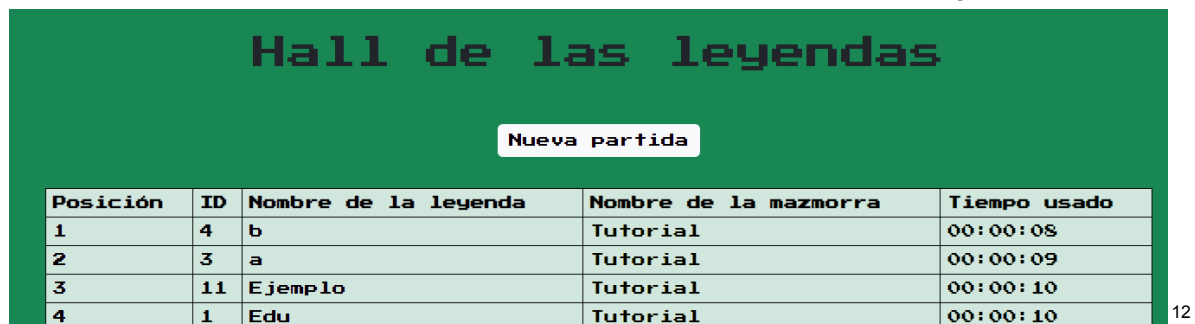
Al ganar una partida se redirigirá al jugador a la siguiente ventana:



11

El jugador debe introducir su nombre (no se permite introducir el nombre en blanco).

Una vez introducido se mostrará la tabla de puntuaciones con el nuevo registro añadido:



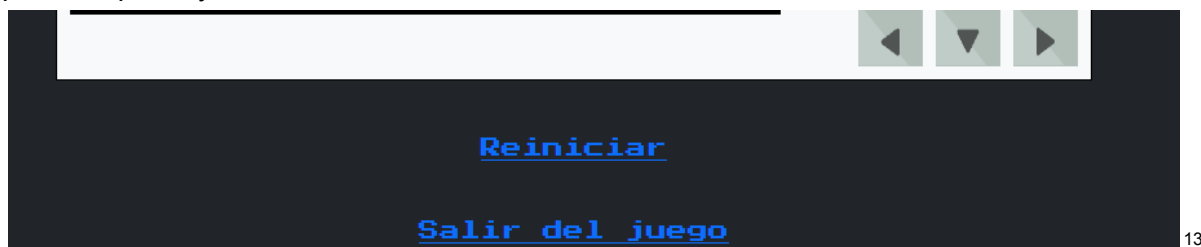
Posición	ID	Nombre de la leyenda	Nombre de la mazmorra	Tiempo usado
1	4	b	Tutorial	00:00:08
2	3	a	Tutorial	00:00:09
3	11	Ejemplo	Tutorial	00:00:10
4	1	Edu	Tutorial	00:00:10

12

## 3.6 Funciones extra.

### 3.6.1 Reiniciar

El juego cuenta con un botón que permite reiniciar la partida actual en el mismo mapa. Esta acción reinicia tanto al personaje y su inventario como a la mazmorra y los objetos y puertas que hay en ella.



<sup>11</sup> Registro de personaje al superar una mazmorra

<sup>12</sup> Ranking de jugadores que han superado la mazmorra ordenados por tiempo.

<sup>13</sup> Botón de reinicio del juego y Salir del juego

### 3.6.2 Nueva partida

Existen dos formas de acceder al menú de inicio y empezar una nueva partida desde el menú de selección:

- El botón de “Salir del juego” que se encuentra en la pantalla de juego (ver foto anterior) redirige al jugador a la pantalla de selección.
- De igual manera, el botón “Nueva partida” en la hace lo mismo (ver foto del [3.5.2 Registrar puntuación](#)).

Para concluir, lo siguiente es una muestra de pantalla de juego, en la cual se pueden ver la mayoría de las características definidas anteriormente:

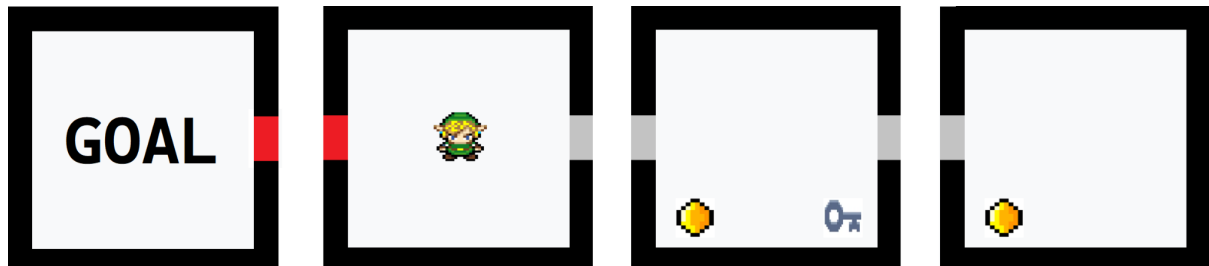


<sup>14</sup> Display completo del juego.

## 4. Los niveles:

### 4.1 Tutorial

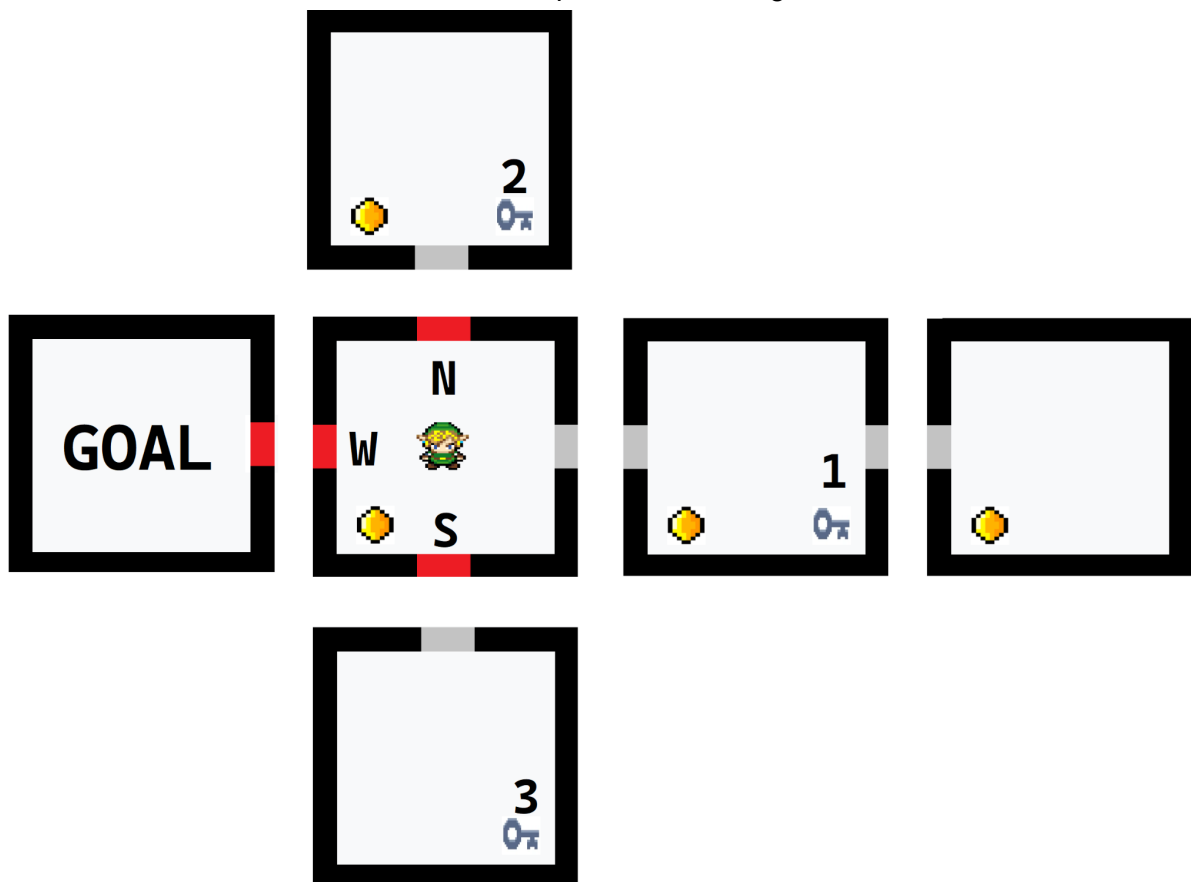
A continuación se muestra el diseño de mapa del nivel "Tutorial":



Para obtener la única llave que hay será necesario pagar dos monedas. Esta llave abrirá la única puerta cerrada que existe en el nivel y que lleva a la meta, dando por completada esta mazmorra.

### 4.2 Juego

A continuación se muestra el diseño de mapa del nivel "Juego":



La llave 1 cuesta 1 moneda y abre la puerta "N". La llave 2 cuesta 1 moneda y abre la puerta "S". La llave 3 cuesta 2 monedas y abre la puerta "W", la cual lleva a la meta y finaliza la partida.

## 5. Estructura teórica de la aplicación

En este apartado se explica cómo se organiza la aplicación en todos sus niveles y cómo funciona a nivel teórico. A pesar de que JSP no se considera *FrontEnd* o nivel cliente, en esta explicación será tratado como tal, ya que son los archivos que generarán los HTML con los que el cliente interactuará. Así mismo, la base de datos se considera *backend* o servidor, pero será tratada de forma independiente.

### 5.1 Nivel Cliente

Este nivel comprende la parte también conocida como “*Frontend*”. Es la parte con la que el usuario podrá interactuar con la aplicación, pudiendo enviar y recibir información (recoger monedas, moverse... y mostrar la habitación resultante de esta acción). Está compuesto por una combinación de JSP, CSS y JavaScript.

El uso de CSS se usa para, en combinación con Bootstrap, mejorar la estética de la web.

JavaScript por su parte se encarga de gestionar el canvas a nivel usuario, recogiendo las acciones del usuario y gestionando qué interacciones envía al servidor y de qué manera. También se encarga de gestionar el JSON enviado por el servidor y mostrar la información en forma de sala, inventario...

Por último está JSP, el cual se encarga de generar los HTML que se visualizan en la página web y de gestionar la información dada por el usuario y por el servidor. En esta parte es, por ejemplo, donde se encuentra el selector de nivel enviado al servidor para que este gestione la información, cree una nueva partida y envíe al cliente la habitación y estado del jugador a mostrar.

### 5.2 Nivel Servidor

Este apartado está completamente desarrollado en Java y es el encargado de gestionar la información dada por el cliente. Aquí es donde se inician las partidas y se crean, modifican y destruyen tanto los jugadores como las mazmorras con sus respectivas habitaciones y lados.

Para la realización de este proyecto se ha seguido la siguiente estructura:

- **Controllers:** aquí se indica que harán las páginas cuando se realicen las funciones de *Post* o *Get* en ellas. No interactúan con la base de datos directamente, sino que lo hacen a través de las clases del *package* “*services*”.
- **Services:** en este *package* se definen las clases encargada de interactuar con la información recibida por parte del controller, permitiendo recoger la información de los primeros para gestionarla en los segundos y devolverla cuando sea necesario a los controladores.
- **Model:** las clases que dan forma a los elementos que componen la aplicación. En este caso aquí se declaran las clases e interfaces *Coin*, *Door*, *DoorKey*, *Game*, *Item*, *MazeMap*, *Player*, *Room*, *Side*, *Wall* y *Winner*, así como sus atributos, constructores, *getters* y *setters*.
- **DAO:** las clases que gestionan directamente la información de la base de datos. Al igual que en “*model*”, suele haber una por cada elemento que se gestiona en la base de datos. En este caso lo único que se almacena en la base de datos son los ganadores (*Winners*).

- **Filters:** estas clases se encargan de evitar el acceso a algunos controladores. Evitan el acceso a la navegación entre salas o la obtención de objetos cuando todavía no se ha creado una partida o el acceso al formulario de victoria cuando todavía no se ha ganado la partida.
- **Utils:** aquí se encuentran las clases que tienen funcionalidades sin una definición exacta, encargadas principalmente de dar soporte al resto de clases. Aquí se encuentra el constructor de mazmorras, usado como un motor muy simple de creación de niveles, teniendo las herramientas necesarias para la creación de las habitaciones, puertas, objetos...

## 5.3 Nivel Base de datos

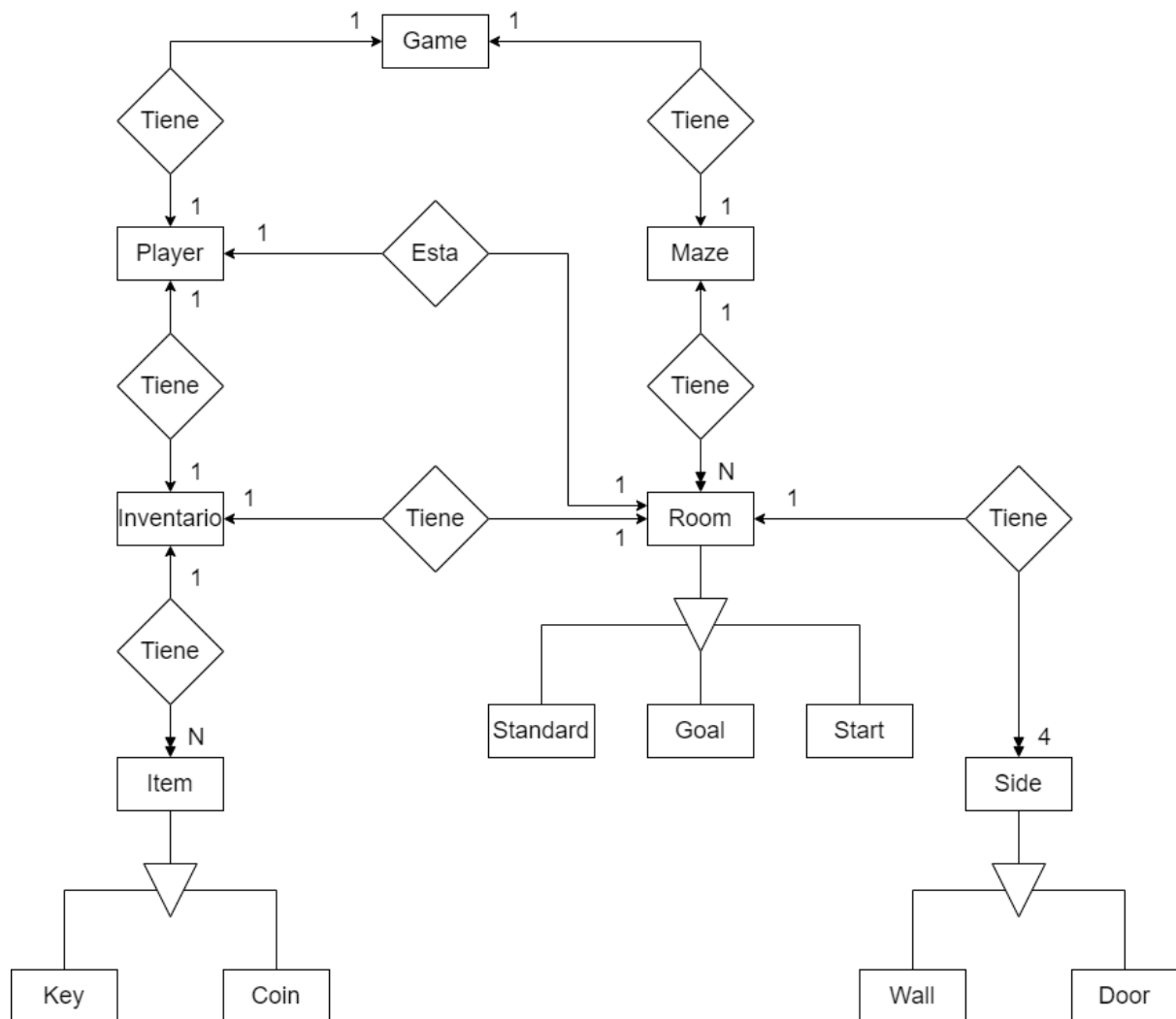
Por último está la base de datos. Implementada con MySQL y gestionada gracias a phpMyAdmin. Esta es la parte encargada de la persistencia de datos, permitiendo guardarlos aún cuando se apaga el servidor.

En esta aplicación la persistencia de datos se aplica únicamente al registro de los jugadores que han superado una mazmorra, quedando registrado su nombre, el del nivel superado y el tiempo logrado (además del id) haciendo referencia al registro de puntuaciones usado antiguamente en las máquinas “arcade”.

## 6. La aplicación

En este apartado se analiza en detalle y a nivel práctico como se ha realizado la práctica. Al igual que en el apartado anterior, se considerará en esta explicación a los archivos JSP como nivel cliente a pesar de formar parte del *BackEnd* ya que son los encargados de generar los HTML con los que el usuario interactuará. Nuevamente, el nivel de bases de datos también será tratado de forma aislada.

A continuación muestro el diagrama entidad-relación entre los diferentes elementos de la aplicación:



Este esquema se respeta prácticamente a lo largo de toda la aplicación con las siguientes excepciones:

- La herencia de habitación se ha implementado con dos variables booleanas (*isGoal* e *isStart*).
- El inventario del usuario es "ilimitado" (el límite de llaves y monedas que puede tener el jugador lo limita el tipo de mazmorra) pero el de las habitaciones nunca será superior a dos ya que en una habitación solo puede haber UNA llave y UNA moneda como máximo.

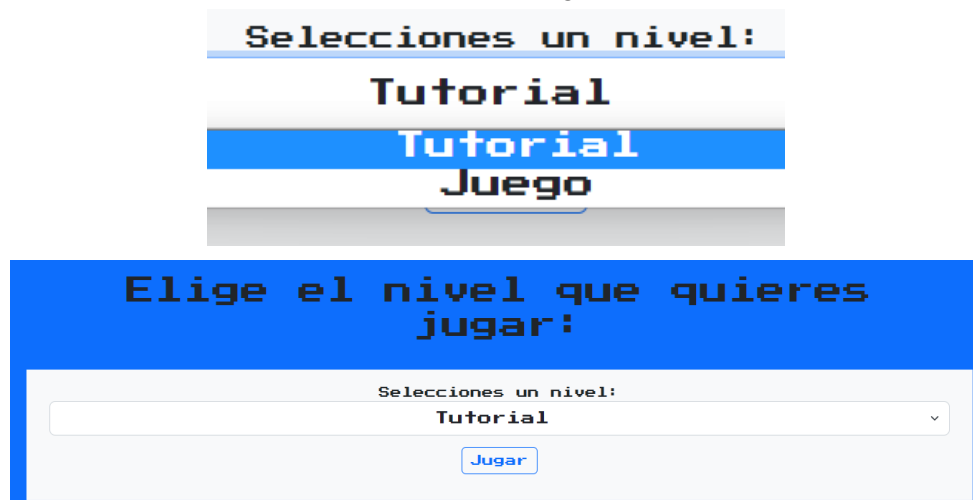


A continuación se explicarán todos los elementos pertenecientes al apartado “*BackEnd*” de este diagrama. Ya que realmente no se ha redactado ningún HTML y como se ha dicho anteriormente, se denominará “nivel cliente” en esta explicación a los archivos JSP, ya que estos son los encargados de generar los HTML. A pesar de esto, los JSP no pertenecen al “*FrontEnd*”.

## 6.1 Nivel cliente

La aplicación cuenta con un total de 5 archivos JSP diferentes que generan 5 HTML con los que el usuario interactúa, 4 que definiremos como principales y 1 de error.

- **start.jsp**: en esta pestaña el usuario debe seleccionar qué nivel quiere jugar, información que se enviará al servidor para generar una nueva partida:



Selecciones un nivel:

Tutorial

Tutorial

Juego

Elige el nivel que quieres jugar:

Selecciones un nivel:  
Tutorial

Jugar

- **navigation.jsp**: apartado encargado de mostrar el canvas que, con el apoyo de javascript, mostrará la habitación actual, sus objetos, el inventario del jugador y las flechas de dirección. También aquí se encuentran el botón de reinicio y de salir del juego, los cuales hacen redirect a /reset y /start respectivamente:



- **endform.jsp:** aquí se mostrará el formulario en el que el usuario debe introducir su nombre para registrar su partida en la tabla de puntuaciones, información que se enviará al servidor para que éste lo gestione con la base de datos.

## Registro en el Hall de las Leyendas

Introduce tu nombre de usuario para dejar constancia de tu éxito:

- **winners.jsp:** este jsp genera la tabla con todos los ganadores registrados en la base de datos, información que recibe del servidor.

## Hall de las leyendas

Posición	ID	Nombre de la leyenda	Nombre de la mazmorra	Tiempo usado
1	4	b	Tutorial	00:00:08
2	3	a	Tutorial	00:00:09
3	11	Ejemplo	Tutorial	00:00:10
4	1	Edu	Tutorial	00:00:10

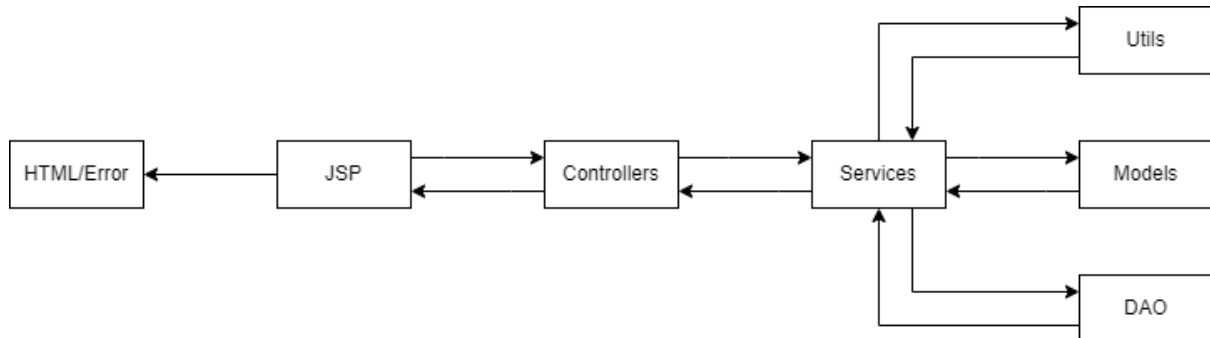
- **error.jsp:** este jsp tiene la función de informar al usuario de los errores que puedan suceder, ya sean por su culpa (intentar acceder a funciones de forma directa, como coger una moneda cuando no quedan en la habitación) o no (que la base de datos haya caído y no se pueda registrar una puntuación en ella). A continuación muestro un ejemplo de error al estar la base de datos apagada:

## Error

No ha sido posible guardar el usuario en la base de datos.

## 6.2 Nivel Servidor

En este apartado se explicarán uno por uno todos los componentes del programa con los que el usuario no tiene ningún tipo de interacción directa (ni generan los HTML que sí que la tendrán). El esquema de comunicación que siguen los diferentes componentes del *backend* es uno basado en el siguiente diagrama:



Los Controllers reciben datos de los HTML generados por los JSP, los cuales la envían a los Services, el cual los procesa, tanto con métodos propios como con el apoyo de las clases de tipo Model, DAO y Utils. Una vez los datos han sido procesados por los Services se devuelven al controlador que solicitó el servicio, el cual bien generará un nuevo HTML con en base a la nueva información en caso de éxito o enviará un mensaje al usuario informando de que, ya sea culpa suya o no, ha habido un error.

### Controllers:

Se han creado un total de 8 controllers para habilitar la comunicación entre el *frontend* y el *backend*:

- **StartController("/start")**: el GET de este controlador se encarga de borrar toda partida que pudiera estar empezada, ya que cuando el usuario accede aquí pierde el progreso de su partida actual. Por su parte el POST se encarga de recoger el input de usuario y generar un nuevo objeto "*Game*" en función del mapa seleccionado. Por último envía al usuario a */nav* para que comience a jugar.
- **NavController("/nav")**: este controlador usa únicamente el método GET. Este método se encarga de enviar la dirección en la que el jugador ha decidido moverse (la cual utiliza el formato */nav?dir=X* donde *x* es la dirección). Una vez el jugador ha intentado moverse (se haya movido o no) envía el JSON con la nueva información al *frontend*. Existe la posibilidad de que este controlador genere un error que será recogido y redirigirá al usuario a la ventana de error además de notificarlo.
- **OpenDoorController("/open")**: este controlador usa únicamente el método GET. Este método se encarga de enviar la dirección del lado con el que el jugador ha interactuado (la cual utiliza el formato */open?dir=X* donde *x* es la dirección). Una vez el jugador ha intentado abrir el lado (sea muro o puerta, con o sin éxito) se redirige a */nav* (sin dirección) para que se envíe un JSON actualizado al cliente. Existe la posibilidad de que este controlador genere un error que será recogido y redirigirá al usuario a la ventana de error además de notificarlo.
- **GetCoinController("/getcoin")**: este controlador usa únicamente el método GET. Este método intenta coger una moneda y dársela al jugador y al igual que en el caso de */open* redirige a */nav* para enviar un JSON actualizado al cliente. En caso de que

no haya moneda en la habitación este controlador generará un error que será recogido y redirigirá al usuario a la ventana de error además de notificarlo.

- **GetKeyController("/getkey")**: este controlador usa únicamente el método GET. Este método intenta coger una llave y dársela al jugador. Tenga o no monedas suficientes redirige a /nav para enviar un JSON actualizado al cliente. En caso de que no haya llave en la habitación este controlador generará un error que será recogido y redirigirá al usuario a la ventana de error además de notificarlo.
- **ResetController("/reset")**: este controlador usa únicamente el método GET. Este método crea un nuevo objeto "Game" en función al mapa que ya se había seleccionando y sustituye en "Game" actual por el nuevo, reiniciando así la partida.
- **EndFormController("/endform")**: el GET de este controller simplemente ejecuta endform.jsp para generar el html. El POST por su parte comprueba que se haya introducido un nombre (que no se haya enviado nada o un espacio en blanco). Si lo enviado se considera válido el servidor intenta guardar el registro en la base de datos. Existe la posibilidad de que este controlador genere un error en caso de que la base de datos no esté operativa o haya ocurrido algún problema a la hora de añadir el registro a ésta, el cual será recogido y redirigirá al usuario a la ventana de error además de notificarlo.
- **WinnersController("/winners")**: el POST de este controlador al igual que en el /nav elimina la sesión activa. A continuación pide al servidor el registro de todos los ganadores y lo envía al cliente para que lo muestre.

## Services:

Los servicios son los encargados de gestionar y manipular la información recibida de los controllers para devolvérsela a estos. Esta aplicación cuenta con los siguientes services:

- **DoorService**: este servicio cuenta con dos funciones, una para, dada una puerta de entrada y otra de salida, dar una puerta que las conecte y otra encargada de intentar abrir una puerta. Esta última función puede devolver un error si la dirección no es válida, informar de que se está intentando abrir un muro, informar de que la puerta ya está abierta o si hay una puerta, del resultado de intentar abrirla (si Player tiene la llave abrirá la puerta, sino no).
- **GameService**: aquí se encuentran las funciones encargadas de crear una nueva partida, la cual crea un nuevo Game con un Player asignado y un MazeMap en función del valor pasado por parámetro. También está la función encargada de definir un Game como ganado. Por último tiene los métodos encargados de generar el JSON que usa la aplicación para comunicarse con el cliente.
- **ItemService**: tiene dos funciones encargadas de contar respectivamente la cantidad de monedas o de llaves que hay en un inventario. También cuenta con funciones encargadas de añadir objetos que heredan de la interfaz Item en general a los inventarios, de quitar objetos de la clase Coin y de poner o quitar objetos Key de los inventarios ya sea arbitrariamente o de forma específica.
- **MazeMapService**: esta clase tiene los métodos que, con el uso de los métodos de la clase [MazeMapBuilder](#), crea los dos mapas de este juego, [Tutorial](#) y [Mazmorra](#).
- **PlayerService**: la primera función de esta clase es la de, una vez creado un Mazemap, crear un nuevo Player y colocarlo en la Room de salida. La siguiente clase es la encargada de intentar mover a los jugadores, siendo posible que se devuelva un error si la dirección no es válida. Por último cuenta con la función

encargada de introducir un usuario que ha ganado una partida en la base de datos de ganadores.

- **RoomService:** la primera función crea una nueva habitación. La segunda función intenta dar una llave a un jugador. Para esto primero comprueba que haya llave en la habitación y, si la hay, la saca y comprueba que el usuario pueda pagarla. Si no puede pagarla dejará la llave en la habitación. Si la puede pagar la añadirá al inventario del jugador. La última función de esta clase intenta dar una moneda al jugador. En este caso simplemente comprueba si hay una moneda en la habitación. Si la hay la quita y la añade al inventario del jugador. Tanto en este método como en el anterior si no hay el objeto solicitado en la habitación se devolverá un error.
- **SideService:** en primer lugar esta clase implementa una función que intentará que , debido al movimiento del jugador, este intente entrar por uno de los lados de la habitación, informando al jugador de que se está intentando mover hacia un muro o una puerta cerrada y por último moviendo al jugador en caso de que se esté moviendo en dirección a una puerta abierta. El siguiente método es uno de apoyo al anterior, que permite saber qué habitación es la de salida y cuál la de entrada cuando un jugador usa una puerta. Por último tiene una función que convierte un String de entrada en una dirección (NORTH, SOUTH, EAST o WEST).
- **WinnerService:** la primera función de esta última clase se encarga de crear objetos de la clase Winner, definiendo el nombre del jugador, el nombre del mapa que estaba jugando y el tiempo que ha tardado. La siguiente función obtiene de la base de datos la lista de todos los ganadores. A continuación y usando otras dos funciones de esta clase, ordena a los jugadores de menor a mayor tiempo y da formato dicho tiempo en modo HH:MM:SS.

## Model:

En el apartado de modelo se definen como clases la información en forma de entidad con la que se quiere interactuar en la aplicación. En esta práctica se cuenta con las siguientes clases o interfaces: **Item** (interfaz de la que heredan las clases **Coin** y **DoorKey**), **Side** (interfaz de la que heredan las clases **Door** y **Wall**), **Game**, **Mazemap**, **Player**, **Winner** y **Room**. Todas estas clases cuentan con getters y setters. Además, Winner tiene implementada la interfaz Comparable para facilitar la ordenación de la lista de ganadores de menor a mayor tiempo usado.

## DAO:

El paquete DAO cuenta con un total de 2 clases y 1 interfaz.

- **MySqlDatabase:** esta clase es la encargada de crear la conexión con MySql y mantener una única conexión activa.
- **WinnersDao:** interfaz que declara los métodos addToWinners y getWinners, necesarios para el correcto funcionamiento de la función ranking del juego.
- **WinnersDaoImpl:** clase que implementa los metodos declarado en la interfaz anterior, la cual implementa. Realiza las query necesarias para añadir Winners a la tabla Winners y para obtener todos los elementos de dicha tabla en formato lista.

## Filter:

Esta aplicación cuenta con 2 filtros:

- **GameFilter:** filtro encargado de controlar los accesos a /nav, /getcoin, /getkey, /open y /reset. En resumen evita el acceso a cualquier función del juego mientras no haya una partida empezada. Adicionalmente, en caso de que el jugador haya ganado se le redirige a /endform para que no se olvide de registrar la victoria.
- **EndFormFilter:** este filtro es el encargado de controlar cuando se puede acceder a /endform. Solo se podrá acceder cuando haya una partida activa y además dicha partida se haya ganado. En el caso de no haber partida se redirigirá a /start para empezar, mientras que si hay partida pero no se ha ganado se volverá a /nav para continuar con la partida.

## Utils:

Este apartado cuenta con una clase y una interfaz:

- **MazeMapBuilder:** esta interfaz contiene todos los métodos necesarios para construir todos y cada uno de los apartados de una mazmorra, desde construir las puertas, asignarles llave y puerta, construir estas puertas o asignar qué habitación es la meta, entre otras cosas.
- **MazeMapBuilderImpl:** esta clase implementa los métodos creados en la anterior interfaz aplicados a esta aplicación. Es una suerte de motor que nos permite crear fácilmente todo tipo de mazmorras para este juego de una forma rápida y sencilla.

## Excepciones:

Esta aplicación detecta 3 tipos de errores:

- **DAOException:** excepción que salta cuando hay algún problema a la hora de comunicarse con la base de datos, ya sea para enviarle o solicitarle información.
- **NoItemException:** excepción que salta cuando se intenta coger un objeto de una habitación que no tiene objetos del tipo solicitado. Esta excepción evita que un usuario haga trampas intentando añadir mediante url objetos a su inventario
- **NoValidDirException:** excepción que salta cuando se intenta, tanto mediante /nav como /open, acceder a una dirección no definida (el programa solo acepta N, S, W, E como direcciones válidas, todo lo demás hará saltar el error).