

Maze-Solver 1.1.0

Entornos de desarrollo - DAW

Versión 1.0. mayo del 2024

Redactado por:
Adrián Peñalver Fernández
Firma
12 de mayo del 2024



	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Tabla de contenido

Introducción.	2
1. Planteamiento inicial y algoritmo empleado.....	3
2. Definición del funcionamiento básico.	7
2.1 Ejecución completa del programa.	7
2.2 Situaciones excepcionales.....	9
3. Definición en UML del programa.	11
3.1 Casos de uso (Diagramas y fichas).	11
3.2 Diagramas de clases.	19
4. Definición individual de cada clase del programa.	21
4.1 Maze.....	21
4.2 Input	23
4.3 Config	24
4.4 Main.....	26
4.5 Session.....	26
4.6 Coordinate	27
4.7 DAO.....	28
4.8 User	29
4.9 Utils	30
5. Desarrollo de la implementación de cada funcionalidad del programa.	31
6. Desarrollo de la implementación del algoritmo del primer camino.	41
7. Desarrollo de la implementación del camino más corto.....	44
8. Conclusiones.....	47
9. Bibliografía	49

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Introducción.

El software desarrollado consiste en un programa básico en java mediante el cual se puede resolver un problema clásico de buscar un camino entre dos puntos.

Se trata pues de un problema típico de teoría de grafos en el cual se deberá aplicar un algoritmo de búsqueda en profundidad para poder encontrar una posible salida y para encontrar la solución más corta.

Además, el programa presentará una interfaz básica por consola y hará uso del paradigma de Programación Orientada a Objetos, haciendo uso de clases que permitirán crear sesiones de usuario, carga de laberintos, conexiones a base de datos, etc.

Los laberintos, con sus sinuosos caminos y callejones sin salida, han cautivado la imaginación humana durante siglos. Representan un desafío, un enigma, un viaje hacia lo desconocido. Desde el antiguo laberinto del minotauro en Creta hasta los extraños y simbólicos jardines de la Quinta da Regaleira en Sintra, estos espacios intrigantes nos invitan a explorar, a perdernos y, en última instancia, a encontrar la salida.

Ya sea a través de mitos, como el famoso hilo de Ariadna o como sistemas metódicos y complejos de optimización como la teoría de grafos, el hombre siempre ha visto la resolución de laberintos como un reto e incluso un pasatiempo.

El software desarrollado consiste en un programa básico en java mediante el cual se puede resolver un problema clásico de buscar un camino entre dos puntos.

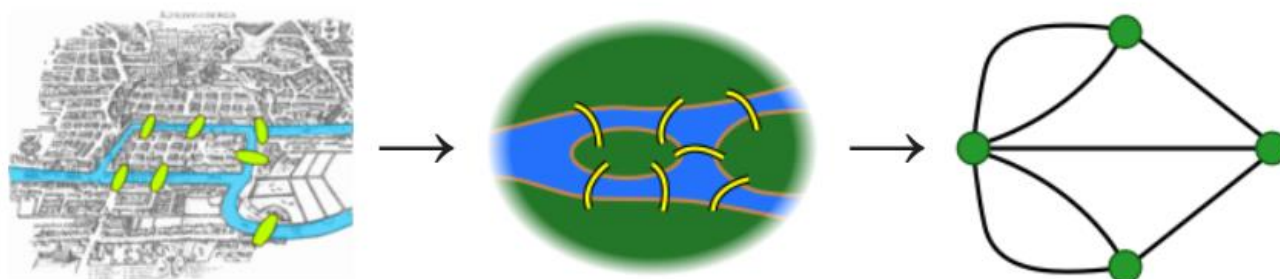
Se trata pues de un problema típico de teoría de grafos en el cual se deberá aplicar un algoritmo de búsqueda en profundidad para poder encontrar una posible salida y para encontrar la solución más corta.

Además, el programa presentará una interfaz básica por consola y hará uso del paradigma de Programación Orientada a Objetos, haciendo uso de clases que permitirán crear sesiones de usuario, carga de laberintos, conexiones a base de datos, etc.

Existe una gran variedad de algoritmos para la resolución de este tipo de problemas, así como una gran cantidad de variantes para cada uno de estos algoritmos, que añaden factores extra a considerar, como podrían ser el esfuerzo para cada camino, el tráfico, tiempos medios para recorrer los caminos, etc.

Uno de los algoritmos para la resolución de este tipo de problemas es el algoritmo de Dijkstra, también conocido como algoritmo de caminos mínimos. Este tipo de algoritmo es usado por una gran cantidad de aplicaciones y utilidades que usamos a diario como puede ser Google Maps, aplicaciones para explorar las opciones de movilidad del transporte público, así como la famosa página web de Rome2Rio.

Uno de los problemas más famosos de la aplicación de búsqueda de caminos es el de los puentes de Königsberg, el cuál trata sobre como recorrer toda una serie de puentes sin repetir ninguno de ellos, de forma que se vuelva al punto de inicio. Este problema, así como su solución representan la aparición de la teoría de grafos en las matemáticas. Además, el problema crea una relación directa entre el entorno urbano y el grafo, pudiéndose también extrapolar a un laberinto.



Analogía entre una ciudad y un grafo.

Finalmente, el problema fue resuelto por el matemático suizo Euler. Además de solucionarlo propuso varias reglas y características clave para poder resolver diferentes variantes de este tipo de problemas.

En este trabajo, expondremos la forma de tomar un laberinto como un grafo y buscaremos la solución más corta, la primera que encuentre y también nos indicará si el problema no tiene una solución que se ajuste a la física del laberinto y las restricciones impuestas.

1. Planteamiento inicial y algoritmo empleado.

Para este programa el laberinto a resolver estará representado por un fichero .txt que contendrá caracteres de tipo '#' almohadilla que representarán paredes y/o extremos del laberinto. Por otro parte, los espacios en blanco representan los pasillos o nodos del laberinto.

Para poder entender bien cómo funciona el algoritmo, así como para determinar las limitaciones de este, es necesario especificar los siguientes datos:

- El laberinto presenta a lo largo de todo su perímetro casillas de tipo pared, es decir, representadas por el carácter almohadilla, de forma que queda totalmente acotado, en caso de que el laberinto no quede totalmente acotado, el algoritmo puede no funcionar correctamente.
- Al analizar los espacios en blanco, solo se tendrán en cuenta las casillas que se encuentran arriba, abajo, a la izquierda y a la derecha. Es decir, no se tendrán en cuenta casillas que se encuentren en la diagonal de dicha casilla.
- Los espacios en blanco pueden representar 3 tipos diferentes de casillas: pasillos, nodos y finales.
- Los nodos son todas aquellas casillas en blanco que se encuentran rodeadas de 3 o más espacios en blanco. Tal como se ha especificado, estas casillas serán las que no son diagonales. Es decir, un nodo estará rodeado de 3 o 4 espacios en blanco.
- Los pasillos son las casillas en blanco rodeadas por 2 espacios en blanco.
- Los finales son casillas que están rodeadas por una única casilla en blanco.
- Además de todo lo expuesto, existen dos tipos de casillas de tipo espacio/blanco que tendrán

características especiales. Estas serán las casillas de entrada y salida.

- Casilla de entrada, será una casilla en blanco que será marcada con el caracter S mayúscula. Representa el punto de llegada, el que debe ser encontrado por el algoritmo.
- La casilla entrada, será una casilla en blanco que será marcada con el caracter E en mayúscula. Representa el punto de partida, desde el cual se buscará la salida.

Un ejemplo de un laberinto podría ser el siguiente:

		0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3					
												0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	
0	-	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	
1	-	#	E	#								#																				#	
2	-	#		#		#	#	#		#		#				#	#	#		#	#	#	#	#	#	#	#	#	#	#	#	#	
3	-	#				#				#		#								#	#	#	#	#	#	#	#	#	#	#	#	#	
4	-	#		#		#	#	#		#	#	#				#	#	#		#	#	#	#	#	#	#	#	#	#	#	#	#	
5	-	#		#						#		#					#			#	#	#	#	#	#	#	#	#	#	#	#	#	
6	-	#		#	#	#		#	#	#	#	#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	
7	-	#		#												#				#	#	#	#	#	#	#	#	#	#	#	#	#	
8	-	#		#		#	#	#	#	#		#	#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	
9	-	#		#			#	#	#	#		#	#							#	#	#	#	#	#	#	#	#	#	#	#	#	
10	-	#		#	#	#	#			#		#	#	#	#					#	#	#	#	#	#	#	#	#	#	#	#	#	
11	-	#		#	#		#			#		#				#	#	#		#	#	#	#	#	#	#	#	#	#	#	#	#	
12	-	#		#	#			#	#	#	#	#				#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	
13	-	#								#		#					#			#	#	#	#	#	#	#	#	#	#	#	#	#	
14	-	#	#	#		#	#	#		#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	
15	-	#	#			#										#				#	#	#	#	#	#	#	#	#	#	#	#	#	
16	-	#	#	#		#	#	#	#	#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	
17	-	#				#				#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
18	-	#		#		#		#	#	#	#	#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	
19	-	#	#	#			#			#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
20	-	#	#	#	#	#		#	#	#	#	#				#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	
21	-	#	#				#			#		#							#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
22	-	#	#	#	#		#	#	#	#		#				#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
23	-	#	#	#	#	#				#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
24	-	#		#	#	#	#			#	#	#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
25	-	#			#	#	#			#	#					#				#	#	#	#	#	#	#	#	#	#	#	#	#	#
26	-	#		#	#	#	#			#		#				#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
27	-	#			#											#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
28	-	#		#			#	#	#	#		#				#			#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
29	-	#	#	#				#	#	#	#	#							#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
30	-	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

En el ejemplo podemos ver las casillas de entrada y salida marcadas con S y E en color rojo, también se ven todas las casillas finales.

Es importante indicar que, si una casilla en blanco es final y se marca como entrada o salida, esta deja de ser considerada como final.

A la hora de explicar y entender el algoritmo, serán de gran importancia las casillas finales y como están van cambiando a lo largo de la ejecución del algoritmo, por ello, he las he marcado en amarillo, para posteriormente seguir desarrollando el ejemplo.

A la hora de buscar una posible solución al laberinto, es necesario especificar que en caso de que no sea posible encontrar una solución se debe indicar esta situación al usuario. También se debe considerar que una solución está compuesta por una combinación de n casillas donde cada una de ellas es única para cada una de las m soluciones. Esto es que ninguna casilla puede estar repetida, el algoritmo no debe hacer bucles.

Tras exponer las principales características del entorno, definiciones básicas y reglas mínimas, se procede a exponer el tipo de algoritmo utilizado, dejando para el apartado 7 y 8 el análisis técnico en profundidad.

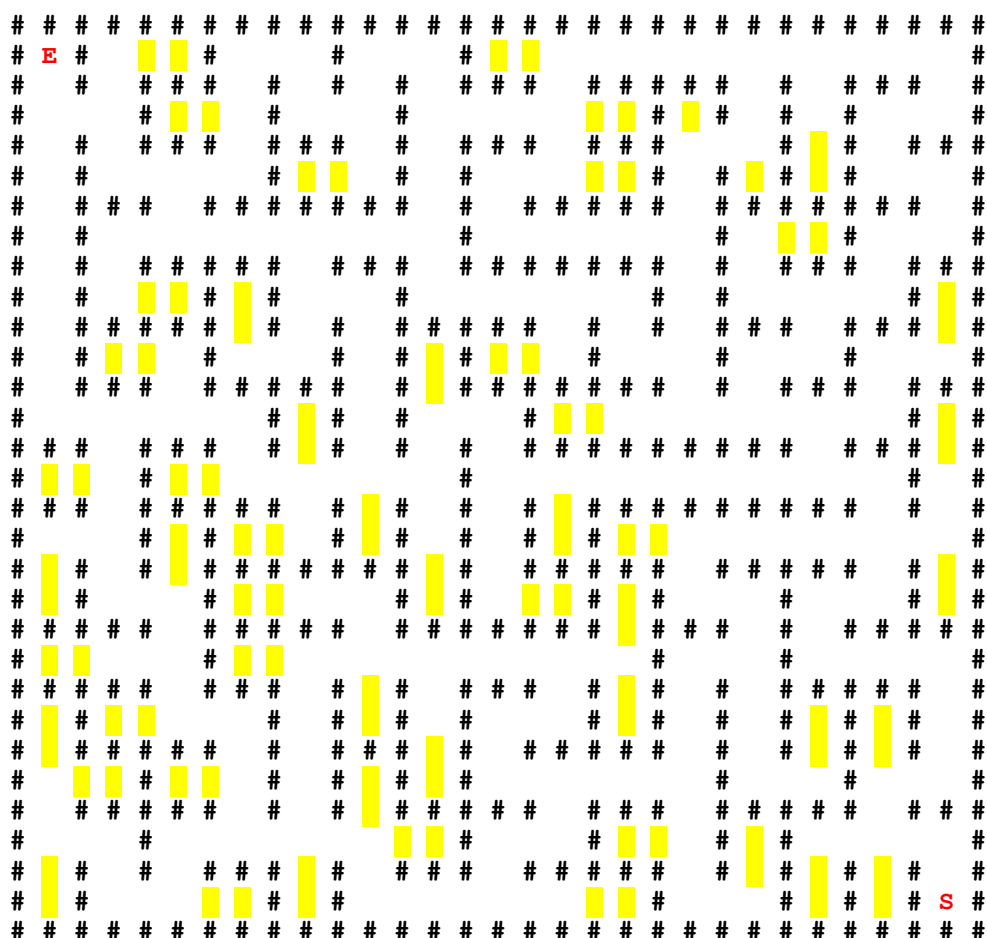
El problema ha sido resuelto mediante dos formas. Para los laberintos de una única solución se ha utilizado un algoritmo simplificador del laberinto, mientras que, para los problemas más complejos, se hace uso de un algoritmo de back-tracking o vuelta-atrás.


ALGORITMO SIMPLIFICACION:

Independientemente del tipo de laberinto, el programa empieza con una simplificación progresiva del laberinto, recorriendo todas las filas y las columnas de la matriz, en caso de determinar que la casilla que está siendo analizada es una sin salida, entonces se marca automáticamente como pared.

El recorrido de la matriz se hace de arriba hacia abajo y de izquierda a derecha.

Al final de cada recorrido, se comprueba si la cantidad de casillas en blanco es la misma que en la iteración anterior. Mientras que el número de casillas en blanco se siga reduciendo, entonces se seguirá realizando este proceso.



	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

En cada iteración, el laberinto irá simplificándose, de forma que los pasillos se van reduciendo y progresivamente algunos nodos pasan a ser pasillos y otras desaparecen, tal como se puede ver en el laberinto anterior. Donde las casillas de amarillo representarían las nuevas “paredes” del laberinto tras 2 iteraciones.

Finalmente, cuando ya no pueda ser simplificado más, se procede a utilizar el algoritmo de back-tracking. Si el laberinto es de una sola posibilidad, el primer camino coincidirá con la primera coincidencia, en caso contrario no existirá una solución. (Es decir, se usa el back-tracking como comprobador de la solución, ya que el primer algoritmo ya debe haber resuelto el problema).

Sin embargo, si el laberinto es complejo y posee varias soluciones para un mismo planteamiento, es necesario detenerse en este punto y explicar de forma detenida este segundo algoritmo de back-tracking.

ALGORITMO BACK-TRACKING:

El algoritmo de back-tracking consiste un algoritmo que recorre todas las posibles combinaciones del laberinto hasta que da con una de las soluciones. Para este problema se han desarrollado dos variantes. La primera de ellas busca la primera opción disponible, indicando también si no se ha podido encontrar ninguna solución y, por otra parte, la otra variante busca de entre todas las posibles soluciones cual es la que representa el camino más corto.

El algoritmo hace uso de recursividad, por lo que permite escribir este en muy pocas líneas, pero hace que la complejidad a la hora de comprender su funcionamiento aumente considerablemente.

El proceso sería el siguiente:

Se comienza analizando una casilla y se comprueban varias condiciones: Si la casilla examinada es la salida, si la casilla examinada es una pared.

En caso de ser la casilla de salida devolverá un valor true.

En caso de tratarse de una pared, devolverá el valor false.


En caso de no cumplirse ninguna de las dos condiciones anteriores, entonces el programa añade la casilla actual a la cola de casillas de la solución y comienza a realizar cuatro llamadas recursivas a sí mismo. Cada una de estas llamadas especifican que la próxima casilla a analizar es la de arriba, derecha, abajo e izquierda respectivamente.

Si en algunas de estas llamadas se obtiene un true, el programa va retrocediendo hasta el inicio y devuelve el camino porque se ha encontrado una solución. En caso de que tras las 4 llamadas no se obtenga ningún true, entonces la casilla en análisis se elimina de la cola y se vuelve al punto al que se llamó a la función para que siga ejecutándose.

Este fragmento anterior funciona para el caso de buscar una solución, pero debe ser optimizado para buscar el camino más corto. La variante sería la siguiente:

Se analiza si la casilla en análisis es la casilla de salida, una pared o parte del camino actualmente recorrido.

Si la casilla es una pared o parte del camino ya recorrido se devuelve false. En caso de ser la casilla de salida, entonces se comprueba si el camino actual es menor que el último encontrado. En caso de ser así, se guarda la nueva solución más corta.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

En caso de no cumplirse ninguna de las condiciones se comprueba si el camino almacenado hasta el momento es más largo que el más corto almacenado, en caso de cumplirse la condición, se devuelve false para hacer más eficiente el algoritmo.

Tras analizar estas estas condiciones, se añade la casilla al camino actual, y se comienzan las llamadas recursivas hacia arriba, derecha, abajo e izquierda.

Finalmente, tras ejecutar las cuatro llamadas, se elimina la última casilla y se devuelve false, volviendo atrás hacia la anterior llamada recursiva.

De esta forma, se analizan todos los casos posibles y se evita buscar soluciones más largas a la más corta obtenida.

La condición de evitar analizar soluciones más largas a la solución más corta encontrada hasta el momento, junto con la simplificación previa, hace que el algoritmo sea más eficiente, aun así es posible mejorarlo aún más. Por ejemplo, creando un orden de llamadas recursivas en función a la distancia de cada camino hacia el siguiente nodo, es decir, tratando de empezar siempre por el camino más corto. Una solución algo similar al algoritmo Dijkstra.

Este algoritmo de back-tracking es bastante ineficiente debido a que se trata de un algoritmo que busca absolutamente todas las probabilidades, por lo que en casos complejos el algoritmo es muy lento, la búsqueda de un camino más corto se trata de un caso de fuerza bruta.

2. Definición del funcionamiento básico.

El funcionamiento básico consiste en un programa que se ejecuta desde la consola del S.O. o del IDE.

Se presenta inicialmente un menú para usuarios no logueados y posteriormente otro con opciones de usuarios logueados al sistema.


Desde cada uno de estos menús podremos realizar diferentes acciones en función de los privilegios de cada usuario, así como otras funciones diferentes en caso de no ser usuarios registrados, a continuación, se expone toda la ejecución de forma detallada.

Posteriormente en la sección 4, se expondrá de forma estandarizada y técnica mediante diagramas UML.

2.1 Ejecución completa del programa.

El programa se inicia con un menu para usuarios genéricos, en este menú se presentan tres opciones principales:

- **Login:** Esta opción permite al usuario registrarse con su nombre de usuario y contraseña. En caso de que el usuario esté registrado y el proceso de login sea correcto, entonces el usuario pasará automáticamente al menú de usuarios registrados, el cual veremos posteriormente. Si el usuario no está presente en la base de datos, entonces se avisará igualmente como si hubiera fallado a la hora de introducir los datos.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

- **Signup:** Esta opción permite a un usuario genérico poder registrarse, para ello este usuario deberá introducir varios datos en el programa, los cuales son: el nombre de usuario, nombre completo, contraseña, nif, email, dirección y fecha de nacimiento. Una vez introducidos estos datos, se comprueba si el nombre de usuario, nif y email son únicos en la base de datos, así como si estos son correctos según el formato especificado.

Si se cumplen ambas condiciones, entonces se comprueba el formato de los siguientes datos y si son correctos, se procede a añadir al usuario en la base de datos y se vuelve al menú principal de usuarios no logueados.

En caso de que algo falle, se indica el error y devuelve al usuario al menú principal.

- **Exit:** Esta opción permite al usuario cerrar el programa, si se selecciona esta opción se pedirá confirmación al usuario antes de ejecutar la orden.

Por otra parte, si el usuario existe y ha conseguido loguearse de forma correcta, este usuario será dirigido automáticamente al menú de usuarios logueados, en él podremos encontrar las siguientes acciones:


- **Cargar Laberinto:** Al seleccionar esta opción se mostrará un listado con todos los archivos de texto que contienen laberintos, el usuario deberá seleccionar una de las opciones mostradas para que este quede guardado en el programa en ejecución. Tras seleccionar un laberinto, se volverá el menú de usuarios logueados.

Si el usuario no desea cargar ninguno, también existe una opción que permite salir de este submenú sin realizar ninguna carga.

- **Ver laberinto:** Esta opción permite al usuario visualizar el laberinto seleccionado anteriormente. Solo podrá acceder a esta opción si previamente se ha cargado un laberinto. Dentro de esta opción el laberinto se podrá mostrar con o sin la entrada y salida marcadas, en función de si estas han sido establecidas previamente o no.
- **Establecer entrada y salida:** Esta opción permite al usuario establecer las casillas de entrada y salida respectivamente. Para ello el usuario podrá visualizar el laberinto con un indicador que facilita al usuario especificar la casilla haciendo uso de su fila y columna. Tras especificar la entrada, se debe hacer lo mismo con la salida. Solo se podrán especificar casillas de entrada y salidas válidas, en caso contrario no se permitirá y se informará al usuario del error, dando la opción de corregir la selección o volver al menú de usuarios logueados. Se especificará en situaciones excepcionales las situaciones concretas que pueden afectar a este apartado.

Al igual que en el caso anterior, esta opción solo será accesible si el usuario ha cargado previamente un laberinto.

- **Buscar caminos:** Esta opción permite al usuario acceder a un submenú donde se presentan tres acciones más, se explica en el siguiente apartado. Solo podrá accederse a este submenú tras la carga de uno de los laberintos.
- **Información usuario:** Esta opción muestra todos los datos referentes al usuario actualmente logueado, mostrando todos sus datos, incluidos la id que posee dentro de la base de datos, así como los años del usuario, también su rol, que por defecto se establece en user. Quedará excluida de esta sección la contraseña.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

- **Logout:** Esta opción permite al usuario cerrar su sesión, de forma que puede volver al menú de usuarios no registrados. Antes de ejecutarse se pide confirmación al usuario.
- **Exit:** Esta opción permite al usuario cerrar el programa, si se selecciona esta opción se pedirá confirmación al usuario antes de ejecutar la orden. Al ejecutarse, se cierra el programa directamente sin pasar por el menú inicial.

Por último, el submenú de buscar caminos se compone de tres opciones:

- **Camino más corto:** Este apartado permite ejecutar el algoritmo planteado para la resolución del laberinto sobre el laberinto cargado por el usuario, usando la variante más compleja de los dos algoritmos explicados.
- **Primer camino:** Al igual que el anterior, ejecuta el algoritmo en su versión simple para hallar el primer camino que se pueda encontrar.
- **Volver:** Esta opción permite al usuario volver al menú de usuarios registrados, de esta forma se sale de este submenú y se pueden seguir realizando otro tipo de acciones.

2.2 Situaciones excepcionales.

Si bien es cierto que el funcionamiento general del programa ha sido descrito anteriormente, también es necesario mencionar que algunas funciones pueden mostrar comportamientos específicos y menos usuales para ciertos tipos de entradas y comportamientos. A continuación, se recapitulan las diferentes acciones, esta vez haciendo énfasis en las situaciones excepcionales que pueden darse:

En el menu para usuarios genéricos:

- **Login:** La situación excepcional que puede llegar a darse es la de que el usuario trate de introducir un nombre de usuario y contraseña que no estén presentes en la base de datos o que sean erróneas. O bien dejando ambos campos en blanco.


Para todos estos casos nos indicará el mensaje: Nombre de usuario y/o contraseña incorrectos.

- **Signup:** Por la cantidad de datos a tratar, este metodo presenta una gran cantidad de posibles situaciones excepcionales, tales como introducir nombre de usuario, nif, email, contraseña nombre o fecha siguiendo un formato no valido.

En caso de darse un formato no valido, se indica mediante un mensaje del tipo: (Formato de nickname incorrecto. Por favor, corrija los campos indicados antes de continuar con el registro). Y se devuelve al usuario al menu principal.

Otra situación a tener en cuenta es cuando los datos son correctos pero los datos no son únicos en la base de datos, en este caso se indicará: Los datos no son únicos en la base de datos.


Por otra parte, en el menú de usuarios logueados, podremos encontrar las siguientes acciones:

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

- **Cargar Laberinto:** A la hora de cargar el laberinto puede darse la situación de que el archivo que está siendo cargado no cumpla con los requisitos expuestos en los primeros capítulos de este proyecto, ya sea porque se trate de un formato diferente, o se trate de un laberinto incompleto. En este caso se enviará un mensaje de error al usuario indicando un problema a la hora de cargar el archivo.
- **Establecer entrada y salida:** Principalmente nos encontramos con 3 situaciones excepcionales, la primera de ellas es establecer una casilla que quede fuera del tamaño de la matriz, donde se avisa del rango máximo que podemos utilizar. Otro caso sería si coincide una de las casillas con una pared, donde se indicará y por último que las casillas coincidan, donde el programa también nos dará un aviso al respecto.
- **Buscar caminos:** Esta opción permite al usuario acceder a un submenú donde se presentan tres acciones más, en el siguiente apartado se tratarán las situaciones excepcionales de estos casos.

Por último, el submenú de buscar caminos podemos mencionar los siguientes casos:

- **Camino más corto y primer camino:** La situación excepcional que podría llegar a darse, es que el laberinto no posea un camino que permita llegar de la casilla de salida a la de llegada, en este caso el programa avisa indicando que no existe una solución posible.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

3. Definición en UML del programa.

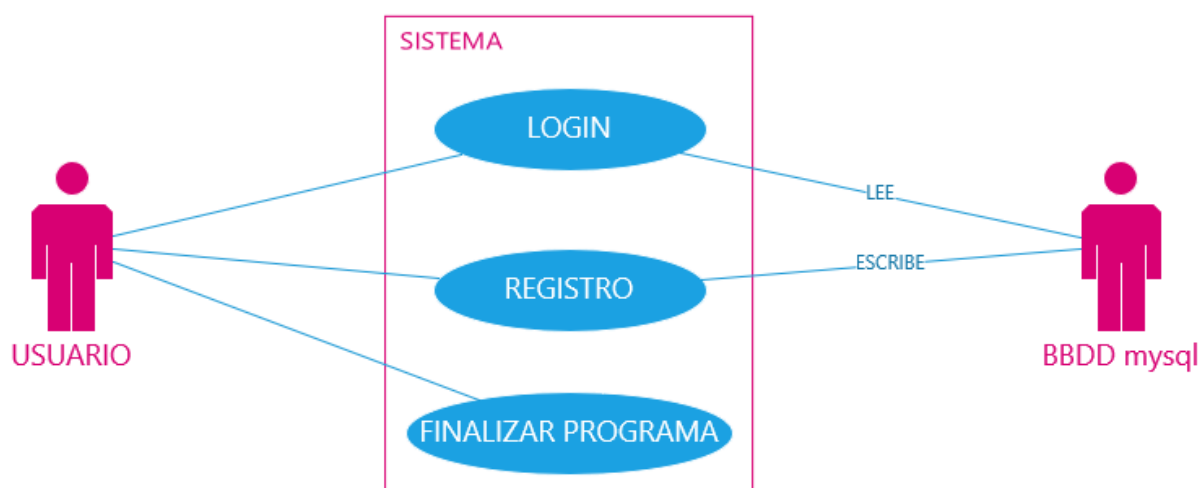
A continuación, se realizará un análisis detallado del programa, sus clases y casos de uso. Con este análisis se podrán conocer los detalles del proyecto, a la vez que se expondrá de forma esquemática y sintetizada toda la información expuesta en los apartados anteriores.

Se comenzará con diagramas de casos de uso, donde se adjuntará a demás a cada uno de los casos de uso, las fichas de cada uno de los métodos.


Posteriormente se mostrará el diagrama de clases, donde se podrá ver una vista simplificada de las clases, así como las relaciones entre ellas.

3.1 Casos de uso (Diagramas y fichas).


3.1.1 Caso 1: UNLOGGED OPTIONS



ID: CU1-01	LOGIN
Descripción	El caso de login describe el proceso mediante el cual un usuario autentica su identidad para acceder a las acciones avanzadas del programa, utilizando credenciales previamente registradas en una base de datos mysql.
Secuencia normal	<ol style="list-style-type: none"> 1. Se pide al usuario que introduzca un nombre de usuario y su contraseña. 2. Se encripta la contraseña siguiendo un patrón MD5. 3. Se hace una consulta a la base de datos para comprobar si existe un registro con coincidencias de ambos campos: contraseña y usuario. <ol style="list-style-type: none"> 3.1 En caso negativo, se devuelve un valor null que activa un mensaje de aviso al usuario indicando que no existe tal usuario y/o contraseña. 3.2 En caso de existir una coincidencia, se crea un nuevo usuario leyendo los valores almacenados en la base de datos y se da la bienvenida al usuario. Se pone el valor de logged a true para que el usuario pueda acceder al menu de opciones avanzadas.
Excepciones	Pueden darse las siguientes excepciones: <ol style="list-style-type: none"> 1. La base de datos no está accesible y se devuelve un valor null desde el DAO. 2. El usuario y/o contraseña no coinciden y se devuelve un null como usuario.

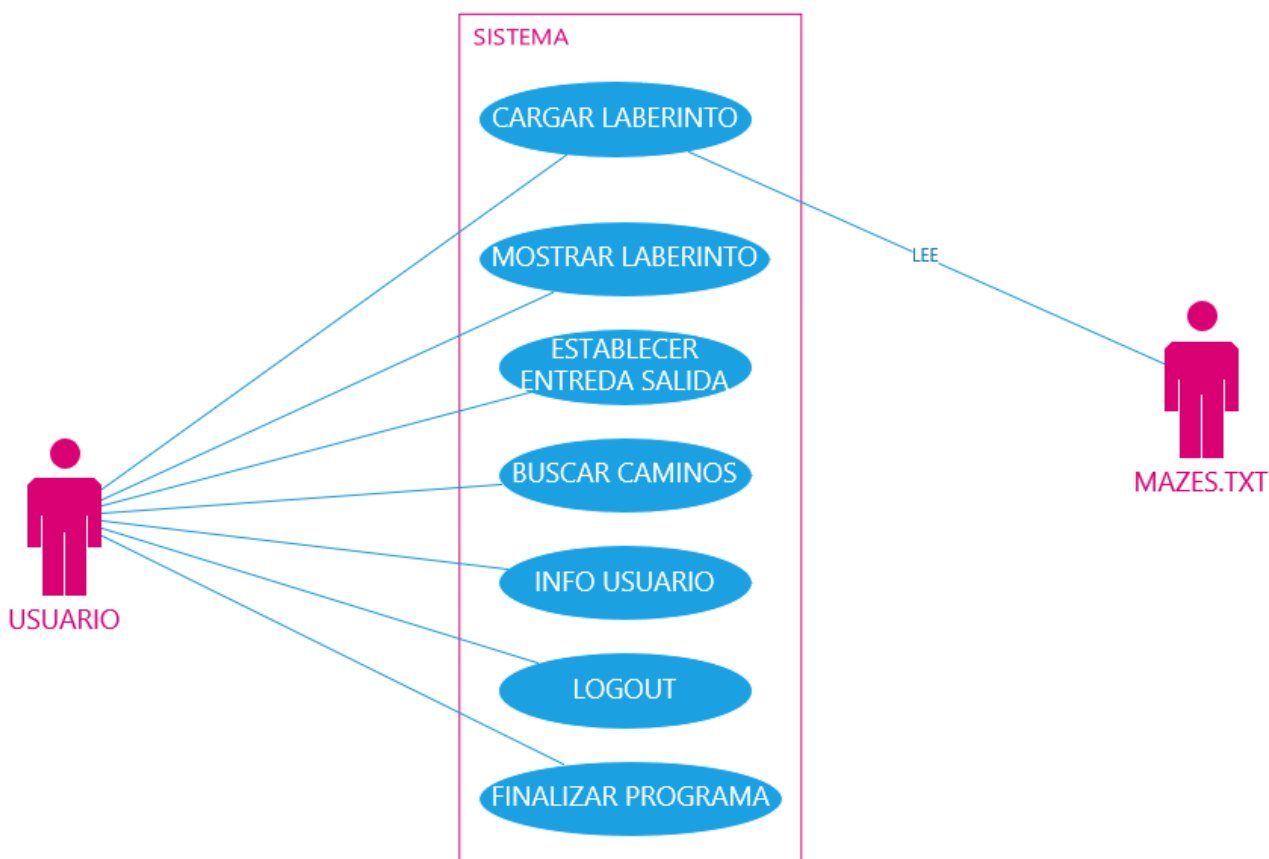
	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

ID: CU1-02	REGISTRO
Descripción	El caso de uso permite que un usuario genérico pueda crear un nuevo usuario desde el menu de opciones previo al login. Tras el registro, el nuevo usuario quedará almacenado en la base de datos.
Secuencia normal	<ol style="list-style-type: none"> Se muestra un banner que indica el inicio del registro. Tras el banner comienza el proceso de registro donde se irán pidiendo los siguientes datos: <ol style="list-style-type: none"> Nombre de usuario, siguiendo el patrón @ seguido del nombre deseado. Contraseña, siguiendo un formato de al menos una mayúscula, una minúscula, un número y un símbolo, la longitud debe ser de al menos 8 caracteres. Se vuelve a solicitar la contraseña para comprobar si coinciden. Se pide un NIF con formato español de 9 dígitos y una letra final. Se pide un Email con formato valido. Se realiza una comprobación para ver si los datos introducidos hasta el momento son únicos en la base de datos. Se siguen pidiendo los siguientes datos que ya no deben ser únicos en la base de datos: <ol style="list-style-type: none"> Nombre completo: Se debe introducir al menos un nombre y un apellido. El nombre y los apellidos empezarán en mayusculas. Dirección postal. Fecha de nacimiento en formato dd/mm/aaaa, además la fecha debe de ser valida, teniendo en cuenta años bisiestos. Se pasará la fecha a formato SQL Se mostrará un mensaje indicando que el usuario se está creando. En caso de que todos los campos introducidos sean validos en formato y sean únicos los valores clave. Se creará un nuevo usuario y se utilizará un metodo del DAO para mandar este usuario a la base de datos y ser almacenado. Si el usuario se guarda sin problema, se avisa al usuario de que el usuario se ha guardado correctamente. En otro caso se avisa del error al usuario y se vuelve al menú principal.
Excepciones	<p>A lo largo de la ejecución de este caso de uso pueden darse varias situaciones excepcionales, tales como:</p> <ol style="list-style-type: none"> Fallo a la hora de comprobar los datos por las siguientes razones: <ol style="list-style-type: none"> Nombre de usuario con formato incorrecto o nulo. Formato de contraseña no valido o contraseña nula, La comprobación de la segunda contraseña no coincide. Formato de nif incorrecto o nulo. Formato de email incorrecto o nulo. Fallo al conectar con la base de datos al comprobar si las primary key son únicas. Las primary key no son únicas en la base de datos. El formato de nombre completo no es correcto o es nulo. El formato de fecha no es válido, la fecha es nula o es una fecha que no existe. <p>En todos estos casos, parará la ejecución de este metodo para comprobar los datos y se avisará del tipo de error, por lo tanto, no se creará ningún usuario,</p> Fallo a la hora de guardar el usuario. Puede deberse a un fallo en la conexión


	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024
	con la base de datos, en este caso se avisa de la imposibilidad de conectarse con la base de datos y no se guardará ningún usuario.	

ID: CU1-03	FINALIZAR PROGRAMA
Descripción	Este caso se utiliza para abandonar la ejecución del programa desde el menú de usuarios logueados y no logueados.
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de salir. 2. Se muestra un mensaje de aviso indicando al usuario que va a salir del programa, se solicita que confirme esta acción pulsando la tecla S ya sea en mayúscula o en minúscula. <ol style="list-style-type: none"> 2.1 Si pulsa la tecla s, el programa termina mostrando un mensaje de despedida. 2.2 Si pulsa cualquier otra tecla, el programa vuelve al menú desde el cual fue llamado este caso de uso.
Excepciones	No se prevén excepciones.

3.1.2 Caso 2: LOGGED OPTIONS



ID: CU2-01	CARGAR LABERINTO
Descripción	Caso de uso que permite al usuario realizar una carga de un laberinto almacenado en un documento txt.
Secuencia	1. Se muestra al usuario un listado con todos los laberintos disponibles y sus


		Proyecto: Maze-Solver 1.1.0	
		Asignatura: Entornos de Desarrollo - DAW	
		Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024
normal	<p>nombres.</p> <p>2. El usuario debe seleccionar uno de los laberintos cargados o bien seleccionar la opción de salir.</p> <p>2.1 Si decide pulsar en salir, el usuario volverá al menú de opciones de usuarios logueados.</p> <p>3. Si el usuario ha seleccionado un laberinto, este será cargado desde un archivo txt y será almacenado en una variable de la clase maze. Y se mostrará la variable booleana de charged como true.</p> <p>3.1 Si ya existía un laberinto cargado previamente, el programa reiniciará todos los valores del anterior laberinto antes de cargar uno nuevo.</p> <p>4. Se mostrará al usuario un mensaje indicando que el laberinto se ha cargado exitosamente y se devolverá al usuario al menú de usuarios logueados.</p>		
Excepciones	<p>A lo largo de la ejecución de este caso de uso pueden darse varias situaciones excepcionales como las siguientes:</p> <p>Error al abrirse el fichero donde se almacenan los laberintos, esto puede ser debido a que se ha cambiado la ruta del fichero. En tal caso nos avisará con un mensaje y volverá al menú principal sin cargar ningún laberinto.</p> <p>Error al leer un laberinto, posibles problemas al modificar el archivo incorrectamente mientras está siendo manipulado.</p> <p>Error en caso de que el formato del laberinto esté corrupto, como podría ser que el laberinto no posee al mismo número de columnas en todas sus filas y viceversa.</p> <p>En ambos casos, se muestra un mensaje avisando al usuario y se vuelve al menú principal sin cargar ningún laberinto.</p>		


ID: CU2-02	MOSTRAR LABERINTO
Descripción	Este caso de uso permite visualizar por consola el laberinto cargado por el usuario.
Secuencia normal	<p>1. El usuario selecciona la opción de ver laberinto desde el menú de usuarios logueados.</p> <p>1.1 Si no se ha cargado ningún laberinto previamente se avisará y nos devolverá al menú principal.</p> <p>1.2 En caso de existir un laberinto cargado previamente, se continuará con la ejecución del código.</p> <p>2. Se muestra por pantalla el actual laberinto indicando nombre del laberinto, también se mostrarán numeradas las filas y las columnas.</p> <p>3. Si las casillas de entrada y salida ya han sido cargadas, se mostrarán marcadas en el laberinto.</p> <p>4. Finalmente se pide que el usuario pulse sobre la tecla intro para continuar con la ejecución del programa.</p> <p>5. Tras pulsar en intro, se devolverá al usuario al menú de usuarios logueados.</p>
Excepciones	No están previstas situaciones excepcionales puesto que se han manejado las excepciones durante la carga del laberinto.

ID: CU2-03	ESTABLECER ENTRADA Y SALIDA
Descripción	Este caso de uso permite al usuario establecer casillas de entrada y salida a un laberinto previamente cargado.

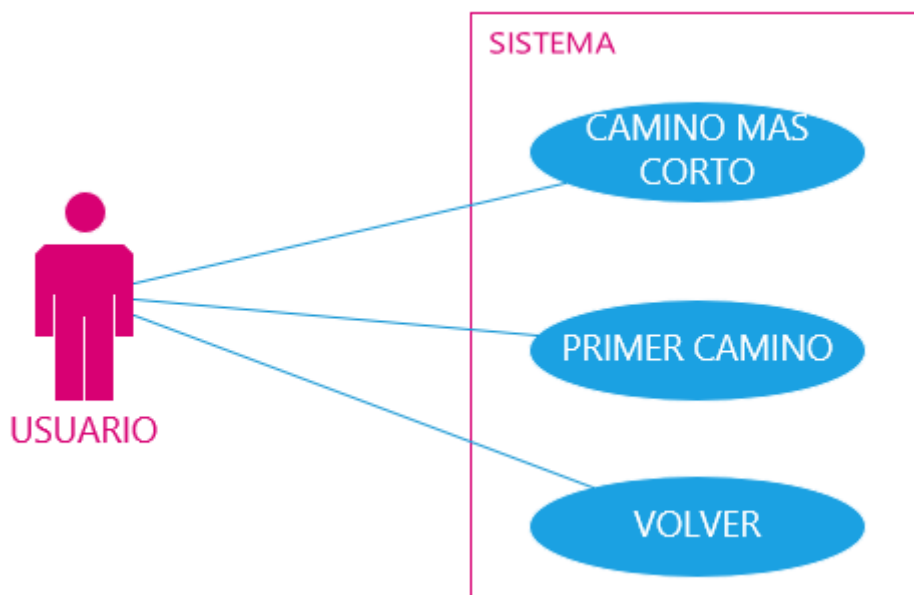
Secuencia normal	<ol style="list-style-type: none"> Tras la selección de la opción cargar casilla de entrada y salida, pueden darse las siguientes opciones: <ol style="list-style-type: none"> El laberinto no está cargado, se avisa al usuario y se devuelve a este al menu principal. Si el laberinto se encuentra cargado, se procederá con los siguientes pasos: Se resetean los valores del laberinto y se mostrará el laberinto cargado por consola. Se solicita la fila de la casilla de entrada. Se solicita la columna de la casilla de entrada. Se evalúan los valores: <ol style="list-style-type: none"> Si los valores son adecuados se prosigue con el código. Si los valores no son adecuados se solicita al usuario que confirme si desea o no volver a intentarlo. En caso de establecerse la casilla de entrada, se muestra un mensaje al usuario indicando que se ha cargado la casilla de entrada. Se muestra el laberinto con la casilla de entrada cargada. Se solicita la fila de la casilla de salida. Se solicita la columna de la casilla de salida. Se evalúan los valores: <ol style="list-style-type: none"> Si los valores son adecuados se prosigue con el código. Si los valores no son adecuados se solicita al usuario que confirme si desea o no volver a intentarlo. En caso de establecerse la casilla de salida, se muestra un mensaje al usuario indicando que se ha cargado la casilla de salida. Se muestra el laberinto con la casilla de entrada y salida cargadas. Si el usuario desiste de establecer la casilla de entrada o la de salida, automáticamente, se reinician las casillas establecidas y se vuelve al menu principal.
Excepciones	<p>Las excepciones que pueden tener lugar durante la ejecución de este caso de uso son las siguientes:</p> <p>Si se establece una fila o columna igual o menor que 0.</p> <p>Si se establece una fila de columna que sea igual o mayor al número de columnas o filas respectivamente.</p> <p>Si una de las casillas coincide con una pared.</p> <p>Si ambas casillas coinciden en las mismas coordenadas.</p> <p>En todos estos casos, se informa del tipo de error al usuario y se pedirá al usuario si desea volver a intentarlo.</p>

ID: CU2-04	BUSCAR CAMINOS
Descripción	Este caso de uso permite al usuario acceder a un menu secundario dentro del menu de usuarios registrados.
Secuencia normal	<ol style="list-style-type: none"> Tras seleccionar la opción pueden darse las siguientes situaciones: <ol style="list-style-type: none"> Si el laberinto no está cargado, se mostrará un mensaje y se devolverá al usuario al menu principal. Si el usuario no ha cargado las casillas de entrada y salida, se devolverá al usuario al menu principal. En otro caso, se continua la ejecución. Se abre el nuevo menu.

		Proyecto: Maze-Solver 1.1.0	
		Asignatura: Entornos de Desarrollo - DAW	
		Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024
	3. Los posibles casos de uso se analizarán a continuación en el apartado 4.1.3		
Excepciones	No se prevén excepciones.		
ID: CU2-05	INFO USUARIO		
Descripción	Caso de uso que permite al usuario visualizar toda la información referente a sus datos personales.		
Secuencia normal	1. El usuario selecciona la opción de visualizar sus datos. 2. Se hace una llamada al metodo info() del usuario actual. 3. Se muestra la información por pantalla. 4. El usuario deberá pulsar enter para continuar con la ejecución del programa y volver al menu de usuarios registrados.		
Excepciones	No se prevén excepciones.		
ID: CU2-06	LOGOUT		
Descripción	Este caso de uso permite al usuario salir de la sesión, poniendo su usuario a null y volviendo al menu de usuarios no logueados.		
Secuencia normal	1. El usuario selecciona la opción de logout y aparece un mensaje preguntando si desea cerrar sesión. 1.1 Si el usuario escribe algo diferente a una S mayúscula o minúscula, automáticamente se devuelve al usuario al menu de usuarios registrados. 1.2 Si el usuario confirma que desea cerrar sesión pulsando S, entonces se mostrará un mensaje despidiendo al usuario y avisando de que la sesión ha quedado cerrada. El usuario se pondrá a null y logged a false. Tras esto el usuario volverá al menu de usuarios no registrados.		
Excepciones	No se prevén excepciones.		
ID: CU2-07	FINALIZAR PROGRAMA		
Descripción	Este caso se utiliza para abandonar la ejecución del programa desde el menú de usuarios logueados y no logueados.		
Secuencia normal	1. El usuario selecciona la opción de salir. 2. Se muestra un mensaje de aviso indicando al usuario que va a salir del programa, se solicita que confirme esta acción pulsando la tecla S ya sea en mayúscula o en minúscula. 2.1 Si pulsa la tecla s, el programa termina mostrando un mensaje de despedida. 2.2 Si pulsa cualquier otra tecla, el programa vuelve al menú desde el cual fue llamado este caso de uso.		
Excepciones	No se prevén excepciones.		


	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

3.1.3 Caso 3: BUSCAR CAMINOS



ID: CU3-01	CAMINO MAS CORTO
Descripción	Este caso de uso permite al usuario poder visualizar el camino mas corto entre las casillas de entrada y salida seleccionadas. Siempre y cuando exista tal posibilidad. Además, mostrará la cantidad de casillas y el tiempo de cálculo.
Secuencia normal	<ol style="list-style-type: none"> 1. Tras seleccionar la opción se comienza a ejecutar el algoritmo: 2. Se comienza realizando una simplificación del laberinto eliminando caminos sin salida. 3. Tras la simplificación se comienza a cronometrar la ejecución y se ejecuta el algoritmo de backtracking avanzado, realizando una comprobación de todos los caminos posibles. 4. En caso de que el laberinto tenga una solución, el programa mostrará al usuario la solución mas corta. Primero mostrando las casillas y cantidad de pasos. También se mostrará el tiempo de ejecución en ms. Posteriormente se mostrará el laberinto con el camino mostrado sobre él. <ol style="list-style-type: none"> 4.1 Si no se encuentra ninguna solución, el programa mostrará un mensaje de aviso al usuario y devolverá a este al menu de maze. 5. Tras visualizar el laberinto el usuario deberá pulsar en enter para poder continuar con la ejecución del programa y volver así al menu de maze.
Excepciones	Se prevé una situación excepcional en caso de no existir una solución manejada mediante un mensaje al usuario, el resto de las excepciones ya quedan recogidas al establecer casillas de entrada y salida y en la carga del laberinto.

ID: CU3-02	PRIMER CAMINO
Descripción	Este caso de uso permite al usuario poder visualizar el primer camino que el algoritmo es capaz de encontrar entre las casillas de entrada y salida seleccionadas. Siempre y cuando exista tal posibilidad. Además, mostrará la cantidad de casillas y el tiempo de cálculo.
Secuencia normal	<ol style="list-style-type: none"> 1. Tras seleccionar la opción se comienza a ejecutar el algoritmo: 2. Se comienza realizando una simplificación del laberinto eliminando caminos sin salida. 3. Tras la simplificación se comienza a cronometrar la ejecución y se ejecuta el

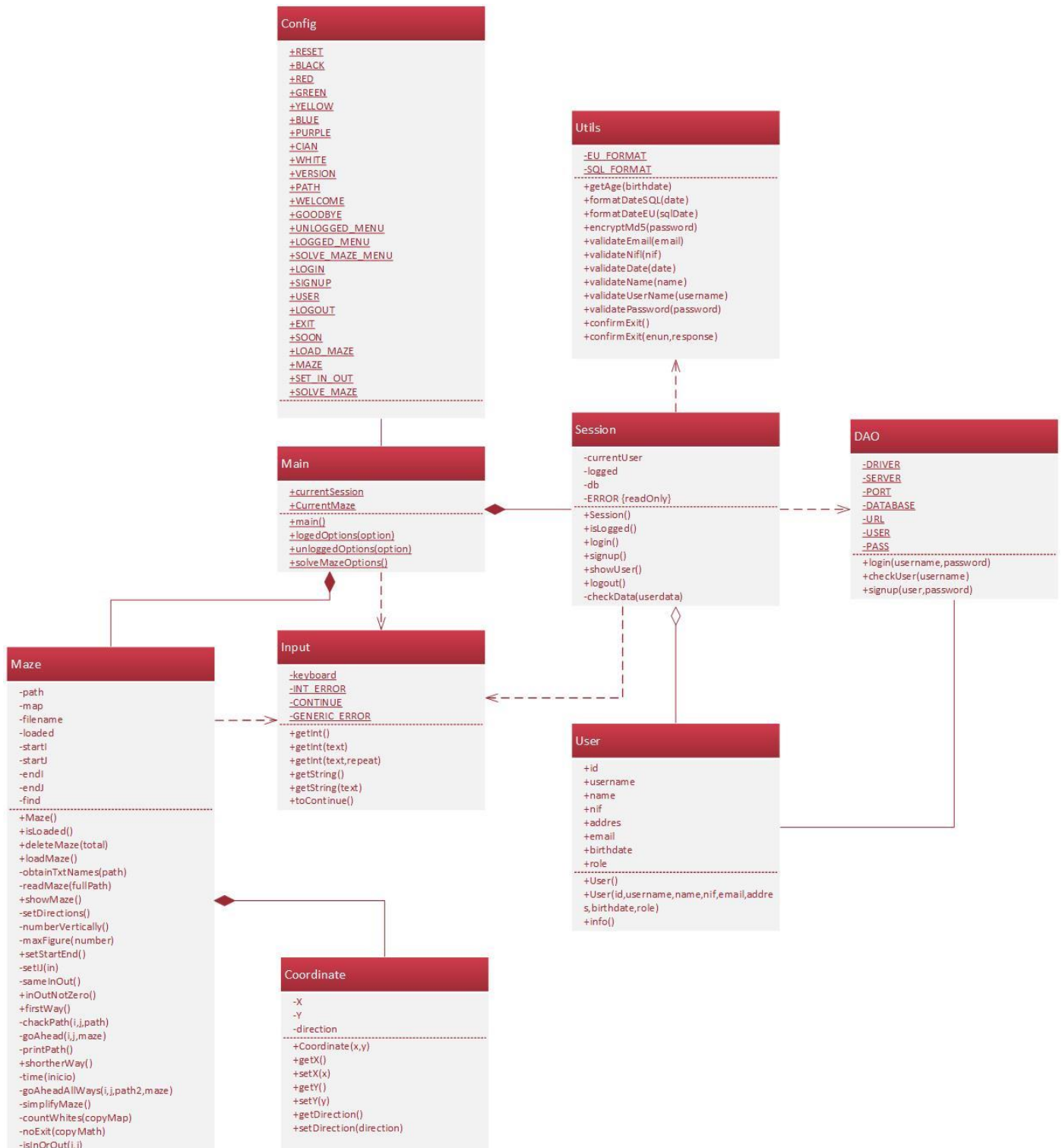
	Proyecto: Maze-Solver 1.1.0		
	Asignatura: Entornos de Desarrollo - DAW		
	Versión del documento: V1.0 Versión del proyecto: V1.1.0		Fecha: 12/05/2024
	<p>algoritmo de backtracking.</p> <p>4. En caso de que el laberinto tenga solución, el programa mostrará al usuario la primera solución que encuentre. Primero mostrando las casillas y cantidad de pasos. Posteriormente se mostrará el laberinto con el camino mostrado sobre él. También se mostrará el tiempo de ejecución en ms.</p> <p>4.1 Si no se encuentra ninguna solución, el programa mostrará un mensaje de aviso al usuario y devolverá a este al menu de maze.</p> <p>5. Tras visualizar el laberinto el usuario deberá pulsar en enter para poder continuar con la ejecución del programa y volver así al menu de maze.</p>		
Excepciones	Se prevé una situación excepcional en caso de no existir una solución manejada mediante un mensaje al usuario, el resto de las excepciones ya quedan recogidas al establecer casillas de entrada y salida y en la carga del laberinto.		


ID: CU3-03	VOLVER
Descripción	Este caso se utiliza para volver a un menú anterior. En este caso al menú de usuario logueados.
Secuencia normal	1. El usuario selecciona la opción de volver. 2. Se muestra un mensaje de aviso indicando al usuario que va a volver al menu anterior, se solicita que confirme esta acción pulsando la tecla S ya sea en mayúscula o en minúscula. 2.1 Si pulsa la tecla s, el programa termina vuelve al menú anterior. 2.2 Si pulsa cualquier otra tecla, el programa se queda en el menú actual desde el cual se ha llamado a este caso de uso.
Excepciones	No se prevén excepciones.

3.2 Diagramas de clases.

A continuación, se expone un diagrama de clases donde se pueden apreciar los métodos y atributos de cada una de estas clases, así como las relaciones que existen entre estas.

En la sección 4, se verá una definición mucho más detallada de cada una de estas clases.



	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

En el diagrama anterior se puede apreciar que la clase que presenta mayor número de dependencias con otras es la clase Session. Además, también cabe destacar la relación de agregación entre la clase User y Session. También dos relaciones de composición: La primera de ellas entre Coordinate y Maze, y la segunda entre Main, Session y Maze.

La clase Main es la que contiene el metodo main(), desde esta clase se lleva a cabo el control del flujo del programa, esta clase está compuesta por un objeto de la clase Maze y otro de la clase Session. Además, esta clase mantiene relación con la Clase Config, la cual presenta varios parametros que facilitan la usabilidad del programa al usuario.

La clase Maze contiene todos los métodos y atributos necesarios para cargar un laberinto, visualizarlo, establecer casillas de entrada y salida, así como solucionar el laberinto por medio de dos variantes. Para poder realizar estas acciones, la clase Maze hace uso de varios objetos de la clase Coordinate. Esta clase Coordinate representa casillas en el laberinto con sus coordenadas y dirección.


La clase Input es utilizada por varias clases, entre ellas: Maze, Main y Session. Esta clase contiene varios métodos que permiten obtener valores de teclado del usuario, ya sean enteros o strings.

La clase Session lleva a cabo las funciones de registro, acceso y logout entre otras opciones. De esta forma permite que un objeto de la clase User acceda a las funcionalidades del programa.

La clase User se compone de atributos que definen a un usuario y métodos para obtener los datos de estos.

La clase Utils contiene varios métodos estáticos que facilitan muchas acciones durante una sesión, como conversión de fechas, encriptación, verificación de strings, etc.

Por último, la clase Session también hace uso de la clase DAO, la cual es responsable de crear una conexión con base de datos y realizar varias funciones con ella, como la carga de usuarios y otro tipo de comprobaciones.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

4. Definición individual de cada clase del programa.


Se explica ms detalladamente cada metodo y atributos, se pueden incluir fragmentos de codigo

4.1 Maze

Maze
<pre> -path: Stack<Coordinate> -map: char[][] -filename: String -loaded: boolean -startI: int -startJ: int -endI: int -endJ: int -find: boolean ----- +Maze() +isLoading(): boolean -deleteMaze(total: boolean): void +loadMaze(): void -obtainTxtNames(path: String): ArrayList<String> -readMaze(fullPath: String): boolean +showMaze(): void -setDirections(): void -numberVertically(): void -maxFigure(number: int): int +setStartEnd(): void -setIJ(in: boolean): boolean -sameInOut(): boolean +inOutNotZero(): boolean +firstWay(): void -chackPath(i: int, j: int, path: Stack<Coordinate>): boolean -goAhead(i: int, j: int, maze: char[][]): boolean -printPath(): void +shorterWay(): void -time(inicio: long): void -goAheadAllWays(i: int, j: int, path2: Stack<Coordinate>, maze: char[][]): boolean -simplifyMaze(): char[][] -countWhites(copyMap: char[][]): int -noExit(copyMath: char[][]): char[][] -isInOrOut(i: int, j: int): boolean </pre>

Los atributos de esta clase son todos privados, la única forma de poder acceder a ellos desde otras clases será mediante getters.

- **-path: Stack<Coordinate>**


	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Path es una pila que se utiliza para almacenar objetos de la clase coordenada, será usado para almacenar la posible solución al laberinto.

- **-map: char[][]:** Map es un array bidimensional de caracteres, en el cual quedará almacenado el laberinto en tiempo de ejecución.
- **-filename: String:** Filename representa el nombre del laberinto que ha sido cargado.
- **-loaded: boolean:** Loaded es un booleano que permite saber si actualmente la clase tiene cargado un laberinto o no.
- **-startI: int:** StartI y StartJ representan las coordenadas en fila y columna dentro de map para la casilla de entrada del laberinto.
- **-startJ: int**
- **-endI: int:** endI y endJ representan las coordenadas en fila y columna dentro de map para la casilla de salida del laberinto.
- **-endJ: int**
- **-find: boolean:** Find es un booleano usado para determinar cuándo un laberinto ha obtenido una solución.

Respecto a los métodos, tenemos lo siguiente:

- **+Maze():** Se trata del constructor, el cual establece find y loaded a false.
- **+isLoading(): boolean:** devuelve el estado del campo loaded.
- **-deleteMaze(total: boolean): void:** Permite resetear todo los campos de la clase o bien únicamente las casillas de entrada y salida.
- **+loadMaze(): void:** Lista los laberintos disponibles y permite al usuario seleccionar uno de ellos.
- **-obtainTxtNames(path: String): ArrayList<String>:** Usado en loadMaze, permite obtener un listado con el nombre de todos los ficheros de laberintos disponibles.
- **-readMaze(fullPath: String): boolean:** Metodo que lee un laberinto seleccionado y lo carga en la variable map.
- **+showMaze(): void:** Permite visualizar el laberinto, irá mostrando datos de entrada, salida y camino en función de donde sea llamado el método.
- **-setDirections(): void:** permite asignar las direcciones de cada objeto coordenada.
- **-numberVertically(): void:** metodo que se encarga de mostrar los numeros de columnas en forma vertical.
- **-maxFigure(number: int): int:** metodo privado para calcular el número de cifras de un numero dado.
- **+setStartEnd(): void:** Metodo encargado de establecer las casillas de entrada y salida del laberinto, delega en el método setIJ.
- **-setIJ(in: boolean): boolean:** Este metodo pide los numeros de fila y columna para cada una de las casillas y maneja las situaciones excepcionales que puedan presentarse.
- **-sameInOut(): boolean:** Metodo para comprobar si las casillas de entrada y salida coinciden.
- **+inOutNotZero(): boolean:** Se encarga de comprobar que ninguna de las casillas se encuentra en la posición 0,0.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024


- **+firstWay(): void:** Metodo para calcular el primer camino posible de un laberinto, para ello hace uso de varios métodos privados que ejecutan el algoritmo, simplifican el laberinto, etc.
- **-checkPath(i: int, j: int, path: Stack<Coordinate>): boolean:** Comprueba las Casillas guardadas como partes del camino solución, para ver si una casilla dada, coincide con alguna de estas.
- **-goAhead(i: int, j: int, maze: char[][]): boolean:** Metodo que desarrolla el algoritmo de backtracking, en su versión simple.
- **-printPath(): void:** Imprime todas las casillas de la solución y la cantidad de estas.
- **+shorterWay(): void:** Metodo para calcular el camino más corto posible de un laberinto, para ello hace uso de varios métodos privados que ejecutan el algoritmo, simplifican el laberinto, etc.
- **-time(inicio: long): void:** Calcula el tiempo de ejecución de la resolución del laberinto.
- **-goAheadAllWays(i: int, j: int, path2: Stack<Coordinate>, maze: char[][]): boolean:** Método que desarrolla el algoritmo de backtracking, en su versión más compleja, haciendo comprobaciones de soluciones halladas previamente y buscando todas las posibilidades.
- **-simplifyMaze(): char[][]:** Metodo general que permite ir simplificando un laberinto dado tratando de quitar del laberinto secciones que no serán soluciones del laberinto y favorecerán que el algoritmo se ejecute de forma mas eficiente.
- **-countWhites(copyMap: char[][]): int:** Metodo privado que cuenta cuantas casillas en blanco quedan en el laberinto tras cada iteración.
- **-noExit(copyMap: char[][]): char[][]:** Comprueba si una casilla es un punto sin salida que debe ser eliminado.
- **-isInOrOut(i: int, j: int): boolean:** Comprueba si una casilla cualquiera es la de entrada o la de salida.

4.2 Input

Input
-keyboard: Scanner -INT_ERROR {readOnly}: String -CONTINUE {readOnly}: String -GENERIC_ERROR {readOnly}: String
+getInt(): int +getInt(text: String): int +getInt(text: String, repeat: String): int +getString(): String +getString(text: String): String +toContinue(): void

Los atributos de esta clase son todos privados, la única forma de poder acceder a ellos desde otras clases será mediante getters, además son atributos estáticos y tres de ellos son fijos.

- **-keyboard: Scanner:** Este objeto permite la lectura de entradas de teclado por parte del usuario.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024


- **-INT_ERROR {readOnly}: String:** Se trata de un valor fijo que muestra un mensaje de error para valores de enteros.
- **-CONTINUE {readOnly}: String:** Valor fijo que muestra un mensaje para el usuario indicando que acción debe realizar para continuar la ejecución.
- **-GENERIC_ERROR {readOnly}: String:** Mensaje de error genérico para ser usado en varios de los métodos de la clase.

Respecto a los métodos, tenemos lo siguiente:

- **+getInt(): int:** Metodo para obtener un entero por teclado.
- **+getInt(text: String): int:** Metodo para obtener un entero por teclado. En este caso, se mostrará un mensaje al usuario.
- **+getInt(text: String, repeat: boolean): int:** Metodo para obtener un entero por teclado. En este caso, se mostrará un mensaje al usuario y se podrá especificar que en caso de error insista hasta que se ingrese un valor adecuado.
- **+getString(): String:** Metodo para obtener un texto por teclado.
- **+getString(text: String): String:** Metodo para obtener un texto por teclado mostrando un mensaje previamente al usuario.
- **+toContinue(): void:** Metodo que muestra un mensaje al usuario y bloque la ejecución del programa hasta que se pulse una tecla.


4.3 Config

Config
<u>+RESET {readOnly}: String</u> <u>+BLACK {readOnly}: String</u> <u>+RED {readOnly}: String</u> <u>+GREEN {readOnly}: String</u> <u>+YELLOW {readOnly}: String</u> <u>+BLUE {readOnly}: String</u> <u>+PURPLE {readOnly}: String</u> <u>+CIAN {readOnly}: String</u> <u>+WHITE {readOnly}: String</u> <u>+VERSION {readOnly}: String</u> <u>+PATH {readOnly}: String</u> <u>+WELCOME {readOnly}: String</u> <u>+GOODBYE {readOnly}: String</u> <u>+UNLOGGED_MENU {readOnly}: String</u> <u>+LOGGED_MENU {readOnly}: String</u> <u>+SOLVE_MAZE_MENU {readOnly}: String</u> <u>+LOGIN {readOnly}: String</u> <u>+SIGNUP {readOnly}: String</u> <u>+USER {readOnly}: String</u> <u>+LOGOUT {readOnly}: String</u> <u>+EXIT {readOnly}: String</u> <u>+SOON {readOnly}: String</u> <u>+LOAD_MAZE {readOnly}: String</u> <u>+MAZE {readOnly}: String</u> <u>+SET_IN_OUT {readOnly}: String</u> <u>+SOLVE_MAZE {readOnly}: String</u>

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Esta clase esta compuesta exclusivamente por una serie de atributos públicos, estáticos y fijos:

- **+RESET {readOnly}: String:** Valor que debe ser concatenado al final de una cadena de texto para volver a establecer el color en blanco.
- **+BLACK {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en negro.
- **+RED {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en rojo.
- **+GREEN {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en verde.
- **+YELLOW {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en amarillo.
- **+BLUE {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en azul.
- **+PURPLE {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en morado.
- **+CIAN {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en cian.
- **+WHITE {readOnly}: String:** Valor que debe ser concatenado al inicio de una cadena de texto para volver a establecer el color en blanco.
- **+VERSION {readOnly}: String:** Muestra la versión actual del programa.
- **+PATH {readOnly}: String:** Indica la ruta de donde se encuentran almacenados los laberintos.
- **+WELCOME {readOnly}: String:** Banner para dar la bienvenida al usuario.
- **+GOODBYE {readOnly}: String:** Banner para despedir al usuario.
- **+UNLOGGED_MENU {readOnly}: String:** Menu con las opciones de usuario no logueado.
- **+LOGGED_MENU {readOnly}: String:** Menu con las opciones de usuario logueado.
- **+SOLVE_MAZE_MENU {readOnly}: String:** Menu de opciones para la resolución de laberintos.
- **+LOGIN {readOnly}: String:** Banner para la opcion login.
- **+SIGNUP {readOnly}: String:** Banner para la opcion signup.
- **+USER {readOnly}: String:** Banner para la opcion de ver usuario.
- **+LOGOUT {readOnly}: String:** Banner para dar la opcion de logout.
- **+EXIT {readOnly}: String:** Banner para la opcion de salir.
- **+SOON {readOnly}: String:** Banner para partes incompletas del programa.
- **+LOAD_MAZE {readOnly}: String:** Banner para la carga de laberintos.
- **+MAZE {readOnly}: String:** Banner para mostrar un laberinto.
- **+SET_IN_OUT {readOnly}: String:** Banner para establecer Casillas de entrada y salida.
- **+SOLVE_MAZE {readOnly}: String:** Banner para la resolución de laberintos.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

4.4 Main

Main
<u>+currentSession: Session</u> <u>+CurrentMaze: Maze</u> <hr/> <u>+main(args: String[]): void</u> <u>+loggedOptions(option: int): void</u> <u>+unloggedOptions(option: int): void</u> <u>+solveMazeOptions(): void</u>

Los atributos de esta clase son los siguientes, los cuales son públicos y estáticos:

- **+currentSession: Session:** Sesión actual del programa, inicialmente establecida como logged false. Solo se establecerá a logged true tras el proceso de login.
- **+CurrentMaze: Maze:** Representa el laberinto con el que se está trabajando, inicialmente se establece a null, posteriormente tras la carga podrá ir cambiando entre las diferentes opciones disponibles.


Los métodos son los siguientes:

- **+main(args: String[]): void:** Método principal, sobre el cual se mostrarán los diferentes menus por los que el usuario podrá navegar.
- **+loggedOptions(option: int): void:** Opciones que se muestran tras el login del usuario en el programa.
- **+unloggedOptions(option: int): void:** Opciones que se muestran a usuarios no logueados.
- **+solveMazeOptions(): void:** Opciones para la resolución de laberintos.

4.5 Session

Session
<u>-currentUser: User</u> <u>-logged: boolean</u> <u>-db: DAO</u> <u>-ERROR {readOnly}: String</u> <hr/> <u>+Session()</u> <u>+isLoggedIn(): boolean</u> <u>+login(): void</u> <u>+signup(): void</u> <u>+showUser(): void</u> <u>+logout(): void</u> <u>-checkData(userdata: String[])</u>

Los atributos de la clase Session son los siguientes, los cuales serán privados:

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

- **-currentUser: User:** Representa a un objeto de la clase User, hasta que no se realiza el login, se mantiene a null.
- **-logged: boolean:** este campo indica si la sesión actual ha realizado el login o no.
- **-db: DAO:** Objeto de la clase DAO que permite la interacción de la sesión actual con la base de datos maze.
- **-ERROR {readOnly}: String:** Este parámetro muestra un mensaje de error para los diferentes métodos que puedan hacer uso de él.

Los métodos son los siguientes:


- **+Session():** Constructor de la clase Session que establece el valor de logged a false y el usuario a null.
- **+isLogged(): boolean:** Permite saber si la sesión ha realizado o no el login.
- **+login(): void:** Método que permite realizar un login dentro de la aplicación, en caso de realizarse exitosamente, se cambia el valor de logged a true y se carga un usuario.
- **+signup(): void:** Metodo que permite crear un nuevo usuario, se comprobará que los valores introducidos coincidan con unas reglas básicas y que los valores clave sean únicos en la base de datos.
- **+showUser(): void:** Método que muestra la información del usuario que ha iniciado sesión.
- **+logout(): void:** Método para cerrar la sesión del usuario, poner el usuario actual a null y cambiar logged a false.
- **-checkData(userdata: String[]):** Metodo privado usado en signup para determinar si los datos introducidos son válidos.

4.6 Coordinate

Coordinate
-X: int -Y: int -direction: char
+Coordinate(x: int,y: int) +getX(): int +setX(x: int): void +getY(): int +setY(y: int): void +getDirection(): char +setDirection(direction): void

Los atributos privados de esta clase son los siguientes:

- **-X: int:** Indica la posición de la fila de la casilla.
- **-Y: int:** Indica la posición de la columna de la casilla.
- **-direction: char:** Indica la dirección hacia la que se dirige el camino que pasa por dicha casilla.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Los métodos son los siguientes:

- **+Coordinate(x: int ,y: int):** Constructor que crea objetos de la clase coordenada, iniciándolo con una fila y columna.
- **+getX(): int:** Permite obtener la fila de la casilla.
- **+setX(x: int): void:** Permite cambiar la fila de la casilla.
- **+getY(): int:** Permite obtener la fila de la columna.
- **+setY(y: int): void:** Permite cambiar la fila de la columna.
- **+getDirection(): char:** Permite obtener la dirección de la casilla.
- **+setDirection(direction): void:** Permite cambiar la dirección de la casilla.

4.7 DAO


DAO
-DRIVER {readOnly}: String -SERVER {readOnly}: String -PORT {readOnly}: String -DATABASE {readOnly}: String -URL {readOnly}: String -USER {readOnly}: String -PASS {readOnly}: String
+login(username: String,password: String): User +checkUser(user: User): boolean +signup(u: User,password: String): boolean

Atributos privados, estáticos y finales de la clase DAO:

- **-DRIVER {readOnly}: String:** Valor del controlador de SQL para Java.
- **-SERVER {readOnly}: String:** Servidor de la base de datos.
- **-PORT {readOnly}: String:** Puerto de la base de datos.
- **-DATABASE {readOnly}: String:** Nombre de la base de datos.
- **-URL {readOnly}: String:** URL complete de la base de datos.
- **-USER {readOnly}: String:** Usuario para acceder a la base de datos.
- **-PASS {readOnly}: String:** Contraseña para acceder a la base de datos.

Los métodos de la clase son los siguientes:

- **+login(username: String,password: String): User:** Permite comprobar si un nombre de usuario y contraseña dados pertenecen a la misma fila, en caso afirmativo se permite concluir satisfactoriamente el login.
- **+checkUser(user: User): boolean:** Comprueba si un usuario dado contiene datos clave únicos que se repitan en la base de datos.
- **+signup(u: User,password: String): boolean:** Añade un nuevo usuario a la base de datos, dado un usuario y una contraseña previamente encriptada.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

4.8 User


User
+id: int +username: String +name: String +nif: String +addres: String +email: String +birthdate: String +role: String
<hr/> +User() +User(id: String, username: String, name: String, nif: String, email: String, addres: String, birthdate: String, role: String) +info(): void +getUsername(): String +setUsername(username: String): void +getName(): String +setName(name: String): void +getNif(): String +setNif(nif: String): void +getEmail(): String +setEmail(email: String): void +getAddres(): String +setAddres(addres: String): void +getBirthdate(): String +setBirthdate(birthdate: String): void +getRole(): String +setRole(role: String): void +getId(): int

Los atributos de la clase son públicos y son los siguientes:

- **+id: int:** id del usuario en la base de datos.
- **+username: String:** nombre de usuario, Nick.
- **+name: String:** Nombre completo del usuario.
- **+nif: String:** DNI del usuario.
- **+addres: String:** Dirección postal.
- **+email: String:** Email del usuario.
- **+birthdate: String:** Fecha de nacimiento del usuario.
- **+role: String:** Rol del usuario, por defectos es user, existe también el rol de admin.

Los métodos son los siguientes:

- **+User():** Constructor vacío de la clase User.
- **+User(id: String, username: String, name: String, nif: String, email: String, addres: String, birthdate: String, role: String) :** Constructor de la clase User que inicializa todos sus campos.
- **+info(): void:** Muestra toda la información del usuario actual.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

- **+getUsername(): String:** Permite obtener el nombre de usuario.
- **+setUsername(username: String): void:** Permite establecer un nuevo nombre de usuario.
- **+getName(): String:** Permite obtener el nombre.
- **+setName(name: String): void:** Permite establecer un nuevo nombre.
- **+getNif(): String:** Permite obtener el DNI del usuario.
- **+setNif(nif: String): void:** Permite establecer un nuevo DNI.
- **+getEmail(): String:** Permite obtener el email del usuario.
- **+setEmail(email: String): void:** Permite establecer un nuevo Email.
- **+getAddres(): String:** Permite obtener la dirección del usuario.
- **+setAddres(addres: String): void:** Permite establecer una nueva dirección.
- **+getBirthdate(): String:** Permite obtener la fecha de nacimiento del usuario.
- **+setBirthdate(birthdate: String): void:** Permite establecer una nueva fecha de nacimiento.
- **+getRole(): String:** Permite obtener el rol del usuario.
- **+getId(): int:** Permite obtener el id del usuario.

4.9 Utils

Utils
<u>-EU_FORMAT {readOnly}: String</u> <u>-SQL_FORMAT {readOnly}: String</u> <hr/> <u>+getAge(birthdate: String): int</u> <u>+formatDateSQL(date: String): String</u> <u>+formatDateEU(sqlDate: String): String</u> <u>+encryptMd5(password: String): String</u> <u>+validateEmail(email: String): boolean</u> <u>+validateNif(nif: String): boolean</u> <u>+validateDate(date: String): boolean</u> <u>+validateName(name: String): boolean</u> <u>+validateUserName(username: String): boolean</u> <u>+validatePassword(password: String): boolean</u> <u>+confirmExit(): boolean</u> <u>+confirmExit(enun: String, response: String): boolean</u>

Los atributos de la clase son privados, estáticos y fijos:

- **-EU_FORMAT {readOnly}: String:** Formato estandarizado para las fechas en Europa.
- **-SQL_FORMAT {readOnly}: String:** Formato estandarizado para las fechas en SQL.

Los métodos son los siguientes:

- **+getAge(birthdate: String): int:** Permite obtener la edad de un usuario calculándola desde la fecha de nacimiento y teniendo en cuenta la fecha actual.
- **+formatDateSQL(date: String): String:** Permite cambiar el formato de fecha de EU a SQL.

- **+formatDateEU(sqlDate: String): String:** Permite cambiar el formato de fecha de SQL a EU.
- **+encryptMd5(password: String): String:** Permite encriptar una contraseña en MD5.
- **+validateEmail(email: String): boolean:** Comprueba el formato de un email.
- **+validateNif(nif: String): boolean:** Comprueba el formato de un nif.
- **+validateDate(date: String): boolean:** Comprueba el formato de una fecha.
- **+validateName(name: String): boolean:** Comprueba el formato de un nombre.
- **+validateUserName(username: String): boolean:** Comprueba el formato de un nick.
- **+validatePassword(password: String): boolean:** Comprueba el formato de una contraseña.
- **+confirmExit(): boolean:** Permite pedir una confirmación antes de realizar una salida.
- **+confirmExit(enun: String, response: String): boolean:** Permite pedir una confirmación para realizar alguna función, haciendo uso de un mensaje personalizado.

5. Desarrollo de la implementación de cada funcionalidad del programa.

A continuación, se realiza un análisis detallado de las principales funcionalidades del programa, las cuales son: login, logout, signup, cargar laberinto, ver laberinto y establecer casillas de entrada y salida. Posteriormente en los apartados 6 y 7 se realizará también un análisis detallado de los algoritmos de resolución del laberinto.

La intención de este apartado es realizar un análisis más en detalle del código del programa, desde una perspectiva de desarrollador.

• Login


Tal como Podemos ver en el código, la funcionalidad de login comienza haciendo uso de la clase estática Config, desde la cual se realiza una impresión en consola de un banner para la opción de login.

Posteriormente, se crea un array de String para almacenar el usuario y contraseñas que serán introducidos por el usuario haciendo uso de la clase estática Input. De esta clase se hace uso de la función getString() Con ella se muestra un mensaje por consola al usuario y posteriormente devuelve el String escrito por el usuario.

Tras obtener la contraseña, se procede a encriptarla con el método encryptMD5() de la clase Utils.

```
public void login() {
    System.out.println(Config.LOGIN);
    String[] userdata = new String[2]; // LEE USUARIO Y CONTRASEÑA
    userdata[0] = Input.getString("\n\tIntroduzca su nombre de usuario: ");
    userdata[1] = Input.getString("\tIntroduzca su contraseña: ");
    System.out.println();

    userdata[1] = Utils.encryptMd5(userdata[1]);
}
```


	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

La siguiente acción que realizar será obtener el usuario actual haciendo uso del metodo login de la clase DAO. Este método recibe el usuario y la contraseña. Devolverá un usuario si el login es correcto, por otra parte, devolverá un null en caso de no poder realizarse el login.

En caso de recibirse un null, en el condicional se ejecutará el else, que mostrará al usuario un mensaje en color rojo, avisando del error, también volverá a resetear el usuario.

Si por el contrario se consigue realizar el login, entonces se mostrará un mensaje en color verde, dando la bienvenida al usuario, y estableciendo el valor booleano logged del usuario actual en true. De esta forma el usuario podrá acceder al menú de opciones avanzadas.

```

this.currentUser = db.login(userdata[0], userdata[1]);

if (currentUser!=null) { // BUSCA UNA COINCIDENCIA EN LA BASE DE DATOS

    System.out.println(Config.GREEN+"\tLogin Correcto: Bienvenido
"+userdata[0]+"."+Config.RESET);
    this.logged=true;

} else {

    this.currentUser=new User();
    System.out.println(Config.RED+"\t\tNombre de usuario y/o contraseña
incorrectos."+Config.RESET);
}

}

```

- **Logout**

Tal como Podemos ver en el código, la funcionalidad de logout comienza haciendo uso de la clase estática Config, desde la cual se realiza una impresión en consola de un banner para la opción de logout.

Tras la impresión del banner se hace una llamada al método confirmExit() de la clase Utils, en este caso se hace uso del metodo que hace uso de dos strings, uno de ellos como enunciado para mostrar al usuario, por otra parte, el segundo string es usado como un carácter o texto para la confirmación del mensaje mostrado al usuario.

Para este caso, si el usuario pulsa S, ya sea mayúscula o minúscula, se confirmará la salida.

```


public void logOut() {

    System.out.println(Config.LOGOUT);

    if(Utils.confirmExit("\n\t¿Seguro que desea cerrar la sesión? SI - s ", "S")) {
        // PIDE CONFIRMACION ANTES DE CERRAR SESION
        System.out.println("\n\tHasta la proxima " + currentUser.username + ".");
        System.out.println("\tSESION CERRADA");
        logged=false; // SE PONE A FALSE PARA VOLVER A MENU INICIAL
        this.currentUser= new User(); // EL USUARIO SE PONE A NULL
    }

}

```

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Si el usuario termina confirmando la salida, entonces se mostrará un mensaje de despedida para el usuario y se informará de que la sesión está cerrada.

Se establecerá el valor booleano logged del usuario a false, para forzar la salida de este del menu de opciones avanzadas y así hacer que el usuario vuelva al menu principal de usuario no logueados.

Finalmente se establece el usuario como un usuario nuevo construido con un constructor vacío.

- **Signup**

Tal como Podemos ver en el código, la funcionalidad de signup comienza haciendo uso de la clase estática Config, desde la cual se realiza una impresión en consola de un banner para la opción de signup.

Posteriormente se crea un array para almacenar todos los campos del nuevo usuario.

En el primer if, se realiza una comprobación para ver si el metodo privado checkData() de la clase session ha devuelto un true o false.

Devolverá un true si todos los valores insertados son válidos según los formatos establecidos. En caso de que uno de los datos no sea válido, se devolverá automáticamente un valor false.

```
public void signup() {
    System.out.println(Config.SIGNUP);
    String[] userdata = new String[7];
    if(checkData(userdata)) { // COMPRUEBA QUE SEAN VALORES VALIDOS
        // resto del codigo
    }
}
```

En el metodo checkdata tenemos el siguiente código:

En él se irá asignado a cada uno de los espacios de memoria del array, uno de los campos del usuario. Primero se solicitarán los campos guardados en la base de datos como únicos y clave primaria, tras ser solicitados, se comprobará con sus respectivos métodos de la clase Utils si el formato es válido.

En caso de ser valido, se seguirán pidiendo más campos, en caso contrario, se avisa del error y automáticamente se devuelve un false.

```
private boolean checkData(String[] userdata) {
    userdata[0] = Input.getString("\n\tIntroduzca un nombre de usuario (@nombre): ");
    if(!Utils.validateUserName(userdata[0])) {
```

```

        System.out.println(Config.RED+"\t\tFormato de Nickname
incorrecto."+Config.RESET);
        System.out.println(ERROR);
        return false;
    }

    // codigo

```

Tras obtener los campos únicos de la base de datos, se hace uso del metodo checkUser de la clase DAO, con este metodo se comprueba si existe un usuario en BBDD que comparta alguno de estos datos únicos. Si se da ese caso, se devolverá un mensaje de aviso y el metodo devuelve un false.

En caso de ser datos únicos se seguirán pidiendo el resto de los campos del usuario.

```

        User test = new User();
        test.setUsername(userdata[0]);
        test.setNif(userdata[1]);
        test.setEmail(userdata[2]);

        if(db.checkUser(test)) { // COMPRUEBA QUE SEAN UNICAS LAS PK

            System.out.println(Config.RED+"\n\t\tLos datos no son unicos en la base de
datos."+Config.RESET);
            return false;
        }

        userdata[4] = Input.getString("\tIntroduzca su nombre completo (Nombre completo y
dos apellidos, comenzando en mayúsculas): ");
        if(!Utils.validateName(userdata[4])) {

            System.out.println(Config.RED+"\t\tFormato de nombre no
valido."+Config.RESET);
            System.out.println(ERROR);
            return false;
        }

        // codigo

```

Por último, si todos los campos son correctos y no se han repetido datos en la base de datos, entonces el metodo finaliza con true y se informa al usuario de que se está creando su usuario.


```

        System.out.println("\n\tCreando usuario, por favor espere...");
        System.out.println();
        return true;
    }

```

Tras comprobar los datos, si son correctos se empezará a ejecutar el código interno del condicional.

Primero Se informa al usuario de que todos los datos han sido correctos, posteriormente se crea un nuevo usuario con todos los campos, estableciéndose un id a 0 y el role como user.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Tras crear el usuario se hace uso del metodo signup() de la clase DAO, donde se enviará el objeto usuario creado y la contraseña de este encriptada.

Si este metodo funciona correctamente, entonces se informará al usuario de que se ha añadido de forma correcta y terminará la ejecución del metodo.

En caso contrario se avisa del error y al igual que en el caso anterior, termina la ejecución del metodo.

```

        if(checkData(userdata)) { // COMPRUEBA QUE SEAN VALORES VALIDOS

            /* EN CASO AFIRMATIVO */
            System.out.println(Config.GREEN+"\t\tTodos los datos son
correctos."+Config.RESET);

            User newUser = new
User("0",userdata[0],userdata[4],userdata[1],userdata[2],userdata[5],userdata[6],"user")
;

            if(db.signup(newUser, userdata[3])) {

                System.out.println(Config.GREEN+"\t\tUsuario añadido de forma
correcta."+Config.RESET);

            }else {

                System.out.println(Config.RED+"\t\tError al guardar
usuario."+Config.RESET);

            }

        }

    }
}

```

- **Cargar laberinto**

Para cargar un laberinto se comienza haciendo uso de un ArrayList que se inicializa con el metodo obtainTxtNames(), el cual carga los nombres de los diferentes laberintos disponibles para posteriormente mostrarlos en un menu:

```

public void loadMaze() {

    int num = 0;
    String file = "";

    ArrayList<String> mazeNames = obtainTxtNames(Config.PATH); // OBTIENE LOS
NOMBRES DE LOS LABERINTOS

    do { // LOS MUESTRA EN UN MENU

        // Codigo

    }while(true);
}

```

En el menú tendremos el siguiente flujo de trabajo:

```
do { // LOS MUESTRA EN UN MENU

    System.out.println("\n\t----- LABERINTOS DISPONIBLES -----");

    for(int i =0; i<mazeNames.size();i++) {

        System.out.println("\t["+(i+1)+"]" + " ---> " + mazeNames.get(i));

    }

    System.out.println("\t[0] ---> SALIR");

    num = Input.getInt("\n\tSeleccione una opcion [0-"+mazeNames.size()+"]: ", false);

    if(num==0) {
        System.out.println("\tVOLVIENDO AL MENU...");
        return;
    }else if(num>=1 && num<=mazeNames.size()) {
        file+=mazeNames.get(num-1);
        break;
    }else {
        System.out.println(Config.RED+"\tDebe Seleccionar una opcion entre [0-
"+mazeNames.size()+"]: "+Config.RESET);
    }
}while(true);
```

Primero se mostrarán todos los laberintos y el usuario deberá seleccionar uno de ellos, tras obtener el laberinto deseado, se pasa por unos condicionales para evaluar la selección.

Si la opcion es 0, se volverá al menu principal de opciones de usuario logueado. En caso de ser una opcion no contemplada, se envía un mensaje al usuario para que seleccione una opcion valida.

En caso de seleccionar una opción válida, entonces se almacenará en un String el nombre del fichero y se saldrá del menú.

Posteriormente se comprueba si ya existía un laberinto previamente cargado y en caso afirmativo se resetean todos los valores.

Se procede a la lectura del laberinto y se almacena en la variable map, en caso de ejecutarse sin problemas, se avisa al usuario, se establece el nombre del laberinto y se establece loaded a true.

```
if(isLoaded()) { // EN CASO DE SELECCIONAR OTRO LABERINTO

    deleteMaze(true); // SE RESETEAN LOS VALORES COMPLETOS

}

if(readMaze(Config.PATH+file)) { // SI EL LABERINTO SE LEE CORRECTAMENTE

    System.out.println(Config.GREEN+"\n\tEL ARCHIVO "+file+" HA SIDO CARGADO
EXITOSAMENTE."+Config.RESET);
```

```
this.fileName=file; // SE LE DA NOMBRE
this.loaded=true; // SE VUELVE A INDICAR QUE ESTÁ CARGADO

}else {
    System.out.println(Config.RED+"\n\tERROR AL LEER EL ARCHIVO "+file+Config.RESET);
}
```

- **Ver laberinto**

Para mostrar el laberinto, primero se muestra el nombre del laberinto, posteriormente se llama a un metodo llamado numberVertically() que se encarga de mostrar los numeros de columnas en posicion vertical.

```
public void showMaze() {

    System.out.println("\n\tLaberinto: " + fileName); // SE INDICA EL NOMBRE

    numberVertically(); // SE MUESTRAN LOS NUMEROS DE COLUMNA EN VERTICAL
```

Posteriormente se comienza a recorrer toda la matriz de caracteres. Primero revisando si la casilla en evaluación se trata de la casilla de entrada o la de salida. En caso afirmativa se imprimirá con E o S según sea salida o entrada. Se establecerá el valor booleano check como false para no realizar los siguientes pasos.

```
for (int i = 0; i < map.length; i++) {

    System.out.print("\t" + i + " - \t");

    for (int j = 0; j < map[0].length; j++) {

        boolean check = true;

        if ((i == startI && j == startJ) && (startI != 0 && startJ != 0)) {
            System.out.print(Config.RED+"I "+Config.RESET);
            check = false;
        } else if ((i == endI && j == endJ) && (endI != 0 && endJ != 0)) {
            System.out.print(Config.RED+"F "+Config.RESET);
            check = false;
        }

    }

}
```

En caso de no ser la casilla de entrada o salida analizara el path de casillas actual para ver si se debe pintar parte del camino o no. Si la casilla es parte del Path se pinta en el laberinto y el booleano camino se establece como true.

```
if (check) {

    boolean camino = false;

    for (int k = 0; k < path.size(); k++) {

        if(i==path.get(k).getX() && j==path.get(k).getY()) {
```

```

        System.out.print(Config.GREEN+path.get(k).getDirection()+" "+Config.RESET);
                                camino=true;
            }

        }

        if(!camino) {
            System.out.print(map[i][j]+" ");
        }

    }

    }

    System.out.println();

}

}

```

Finalmente, si la casilla no es ni entrada, ni salida, ni camino, entonces se pintará como un espacio vacío que representa un pasillo.

- **Establecer casillas de entrada y salida**

Se hace uso del siguiente método, el cual hace a su vez uso de otro metodo especializado para insertar las casillas:

```

public void setStartEnd() {

    deleteMaze(false); // SE RESETEAN LAS CASILLAS ANTERIORES
    showMaze();

    if(setIJ(true)) { // SI LA CASILLA DE ENTRADA SE ESTABLECE
        System.out.println(Config.GREEN+"\r\tCASILLA DE ENTRADA
FIJADA."+Config.RESET);
    }else { // EN OTRO CASO, SI EL USUARIO DESISTE SE TERMINA EL PROGRAMA.
        System.out.println(Config.RED+"\r\tNO SE HA PODIDO FIJAR LA CASILLA DE
ENTRADA."+Config.RESET);
        return;
    }

    showMaze();
    if(setIJ(false)) { // SI LA CASILLA DE SALIDA SE ESTABLECE
        System.out.println(Config.GREEN+"\r\tCASILLA DE SALIDA
FIJADA."+Config.RESET);
    }else { // EN OTRO CASO, SI EL USUARIO DESISTE SE TERMINA EL PROGRAMA.
        System.out.println(Config.RED+"\r\tNO SE HA PODIDO FIJAR LA CASILLA DE
SALIDA."+Config.RESET);
    }

    showMaze();

}

```

Primero se establece la casilla de entrada y posteriormente la de salida, además se muestra el laberinto para ver cómo van quedando establecidas. Si una de ellas no queda establecida, todas ellas quedan reseteadas.

Se hace uso del metodo SetIJ() para establecer las coordenadas de las casilla, se envía como parametro al metodo un booleano para determinar si se establece la de entrada o la de salida.

```
private boolean setIJ(boolean in) {
    int num = 0;
    String casilla = "";

    if(in) {
        casilla="entrada";
    }else {
        casilla="salida";
    }
}
```

Se comienza determinado si es la casilla de entrada o la de salida, posteriormente se entra en un bucle while donde se establecerán las coordenadas, a no ser que el usuario decida dejar de intentarlo.

```
do {
    do { // ESTABLECE Y COMPRUEBA LA FILA
        num=Input.getInt("\r\tIntroduzca la fila de "+casilla+": ", true);

        if(num<0 || num>map.length-1) {
            System.out.println(Config.RED+"\r\tEl número debe de ser mayor
o igual que 0 y menor que "+(map.length-1)+Config.RESET);
        }

    }while(num<0 || num>map.length-1);

    if(in) {
        startI=num;
    }else {
        endI=num;
    }
}
```

Tras establecer la coordenada en I, se procede a almacenarla en el laberinto, ya se la se salida o entrada. A continuación, se realiza el mismo proceso, pero esta vez para establecer la coordenada j:

```
do { // ESTABLECE Y COMPRUEBA LA COLUMNA
    num=Input.getInt("\tIntroduzca la columna de "+casilla+": ", true);

    if(num<0 || num>map[0].length-1) {
        System.out.println(Config.RED+"\r\tEl número debe de ser mayor
o igual que 0 y menor que "+(map[0].length-1)+Config.RESET);
    }

    }while(num<0 || num>map[0].length-1);
```



```

if(in) {
    startJ=num;
}else {
    endJ=num;
}

```

Una vez establecidas las coordenadas, se comprueba si coinciden las de entrada y salida. En caso afirmativo, se indicará al usuario si desea seguir intentando lo o salir.

```

if(sameInOut()) { // SI LAS CASILLAS DE ENTRADA Y SALIDA SON LAS MISMAS

    System.out.println(Config.RED+"\r\tLas casillas de entrada y salida coinciden."+Config.RESET);

    if(Utils.confirmExit("\r\t¿Desea ingresar otra casilla de "+casilla+"? SI-S NO-N ", "N")) {
        deleteMaze(false); // SE RESETEAN ENTRADA Y SALIDA
        return false;
    }
}

```

En caso de no ser las mismas, entonces se comprobarán las siguientes reglas para determinar si son válidas, como determinar si es una pared. Si todo es correcto, entonces se terminará la ejecución, si algo falla se preguntará al usuario si desea volver a intentarlo.

```

    } else {

        if (in && (map[startI][startJ] == ' ')) { // SI LA CASILLA ES VALIDA...
            return true;
        } else if (!in && (map[endI][endJ] == ' ')) { // SI LA CASILLA ES VALIDA...
            return true;
        } else { // EN OTRO CASO...

            System.out.println(Config.RED+"\r\tla casilla coincide con una pared."+Config.RESET);

            if (Utils.confirmExit("\r\t¿Desea ingresar otra casilla de " + casilla + "? SI-S NO-N ", "N")) {
                deleteMaze(false); // SE RESETEAN ENTRADA Y SALIDA
                return false;
            }
        }
    }

}

}while(true);
}

```

6. Desarrollo de la implementación del algoritmo del primer camino.

Tras la exposición de la implementación de los principales métodos y funciones del programa, procedo a desarrollar la implementación del algoritmo de resolución de laberinto, en este caso el del primer camino.

Para ello, desarrollará el proceso de simplificación, el cual es común al algoritmo del primer camino y para el algoritmo del camino más corto. Posteriormente se desarrollará el algoritmo en sí.

El método de simplificación del laberinto devuelve como resultado de su ejecución una matriz de caracteres que representa el laberinto cargado, pero ya simplificado, tal como se explicó en el apartado 1.

Primero se realiza una copia del laberinto para que el original quede inalterado. Mediante dos for se recorre la matriz y se va copiando en la nueva matriz llamada copyMap.

```
private char[][] simplifyMaze() {  
    char[][] copyMap = new char[map.length][map[0].length];  
    for (int i = 0; i < map.length; i++) {  
        for (int j = 0; j < map[i].length; j++) {  
            copyMap[i][j] = map[i][j];  
        }  
    }  
}
```

Una vez copiada la matriz se llama a un método llamado countWhites() que devuelve un entero indicando la cantidad de casillas en blanco que posee el laberinto.

```
int min=countWhites(copyMap);
```

Tras esta llamada se comienza un bucle, que tras terminar devolverá el laberinto simplificado. En el interior del bucle ocurre lo siguiente:

```
do {
```

Primero se hace una llamada a un método noExit() que asigna a la variable copyMap() un nuevo valor, para ello internamente el método analiza las casillas blancas del mapa que solo tienen una casilla blanca contigua. En ese caso, esa casilla se convierte automáticamente en una pared, siempre y cuando esa casilla no sea la de entrada o la de salida.

Posteriormente se vuelve a hacer una llamada a countWhites() para ver si el número de casillas en blanco se ha reducido.

```
copyMap=noExit(copyMap);
int whites = countWhites(copyMap);
```

Tras la ejecución de estos dos métodos se realiza un análisis en un condicional para ver si el número de casillas en blanco actualmente es menor que el numero de casillas en blanco en la iteracion anterior.

En caso afirmativo, se asigna que el numero de casillas en blanco actual es el número mínimo de casillas en blanco. En caso negativo, se determina que ya no puede simplificarse más el algoritmo y termina el bucle, devolviendo el laberinto ya simplificado.

```
        if(whites<min) {
            min=whites;
        }else {
            break;
        }

    }while(true);

    return copyMap;
}
```

Tras la simplificación del laberinto se continua con la ejecución del algoritmo de backtracking:

```
public void firstWay() {

    path.clear();

    char[][] maze = simplifyMaze();

    long inicio = System.currentTimeMillis();

    if (goAhead(startI, startJ,maze)) {

        printPath();
        showMaze();

    } else {


        System.out.println(Config.RED+"\n\tEl laberinto no tiene
solución."+Config.RESET);

    }

    time(inicio);
}
```

En este método se controla el flujo de ejecución del algoritmo y la solución. Primero se limpia el Path con las soluciones, en caso de que exista una solución calculada con anterioridad, luego se simplifica el laberinto.

Tras la simplificación del laberinto, se calculan los milisegundos al inicio de la ejecución.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Se llega a un condicional donde se analiza el resultado de un metodo llamado goAhead() el cual contiene el algoritmo en si y que devuelve un booleano. True si se ha alcanzado la salida y false en caso contrario.

Para el caso afirmativo se mostrarán las casillas y los pasos mediante el método printPath() y posteriormente el laberinto completo con la solución mediante el metodo showMaze()

En caso negativo se avisa al usuario de que el laberinto no tiene solución.

Finalmente se hace una llamada al metodo time() que recibe el tiempo en milisegundos calculado inicialmente y realiza un calculo para mostrar al usuario por pantalla el tiempo de ejecución del algoritmo.

A continuación, se desarrolla en profundidad el metodo goAhead() que contiene el algoritmo en sí:

Se tratade un algoritmo de backtracking que hace uso de recursividad por lo cual, si bien es cierto que es muy poco extenso en líneas de código, requiere un gran esfuerzo cognitivo para entender su comportamiento.

El método, ya se a en su primera ejecución como en cualquiera de sus llamadas recursivas, recibe como parametros unas coordenadas de casilla y la matriz del laberinto.

El proceso se inicia desde la casilla de entrada y por cada casilla que se recorra comprueba sucesivamente si se trata de la casilla de salida, en caso afirmativo, la llamada a este metodo concluiría devolviendo a la llamada anterior un valor de true y así sucesivamente hasta la primera llamada.

En caso negativo, se comprobará si la casilla se trata de una pared o si es una casilla que ya ha sido visitada y añadida al listado de casillas solución. En caso afirmativo, se devolverá un false. En caso negativo se continua con la ejecución.

Si tras estas primeras dos comprobaciones se comprueba que la casilla es un pasillo y que no ha sido ya añadida al conjunto de casillas solución, entonces se añade a esta provisionalmente al conjunto de soluciones.

```
private boolean goAhead(int i, int j, char[][] maze) {


    if (i == endI && j == endJ) {
        return true;
    }

    if (maze[i][j] == '#' || checkPath(i,j,path)) {
        return false;
    }

    path.push(new Coordinate(i,j));
}
```

Tras añadir la casilla actual al Path, se procede a hacer 4 llamadas recursivas sucesivas de este metodo. Estas 4 llamadas tienen como objetivo analizar las cuatro casillas colindantes de la casilla en análisis.

Lógicamente una de esas llamadas devolverá siempre un false al tratarse de la casilla por la que ya ha venido el camino recorriendo el laberinto.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

También es posible que, analizando el resto de las casillas, estas sean paredes o sean parte de la solución anterior, etc. Por lo tanto, si en ninguna de las llamadas recursivas se ha devuelto un true significa que no se ha alcanzado la casilla de salida y tras las 4 comprobaciones se elimina la casilla actual del listado Path.

Si estamos en una llamada recursiva intermedia, entonces volveremos a la llamada al metodo anterior y se continuará con la ejecución del resto de llamadas recursivas que quedan pendientes, así sucesivamente hasta que una de las llamadas devuelva un true.

En ese caso todas las llamadas recursivas devolverán un true y dejarán almacenadas todas las casillas recorridas hasta llegar a la salida.

En caso de que tras agotar todas las llamadas posibles y se realizan todas las posibles combinaciones, no se alcance la salida, significa que no se puede alcanzar y todas las llamadas recursivas terminarán borrando de Path su casilla y posteriormente devolviendo un false, pro lo que se devolverá un Path vacío y un booleano false.

```

    if (goAhead(i, j+1,maze)) {
        return true;
    }

    if (goAhead(i-1, j,maze)) {
        return true;
    }

    if (goAhead(i, j-1,maze)) {
        return true;
    }

    if (goAhead(i+1, j,maze)) {
        return true;
    }

    path.pop();
    return false;
}

```

7. Desarrollo de la implementación del camino más corto.

Tras desarrollar el algoritmo del primer camino, se procede con el desarrollo del algoritmo del camino más corto. En este caso, se trata del mismo algoritmo y flujo de ejecución, pero con algunos matices y comprobaciones que permiten optimizar la búsqueda. El proceso empieza con el siguiente método, el cual comienza de forma idéntica al metodo firstWay()

```

public void shorterWay() {

    path.clear();
    char[][] maze = simplifyMaze();

    long inicio = System.currentTimeMillis();

```

A continuación, se crea una pila donde se almacenarán las soluciones provisionales, inicialmente se establece con un tamaño superior al que puede tener en alguna de las soluciones. Mediante un for, se van añadiendo las coordenadas.

```
Stack<Coordinate> path2 = new Stack<>();

int size = (map.length)+(map[0].length)*10;

for (int i = 0; i < size; i++) {

    path.push(new Coordinate(0,i));

}
```

Tras iniciar la pila, se llama al metodo goAheadAllWays() el cual ejecuta el algoritmo y que se explicará a continuación.

Si tras ejecutarse, se establece que find es true, entonces significa que existe una solución optima y tras esto se imprime el camino, los pasos y el laberinto con la solución.

En caso negativo, se manda un mensaje al usuario. Tras esto se informa al usuario del tiempo de ejecución del algoritmo.

```
goAheadAllWays(startI, startJ, path2,maze);

if(this.find) {

    printPath();
    showMaze();

}else {
    System.out.println(Config.RED+"\n\tNo hay ninguna solución
posible."+Config.RESET);
}

time(inicio);

}
```

El metodo goAheadAllWays() es el siguiente, el cual recibe las coordenadas de casilla, la pila con la solución actual y el laberinto.

Comienza al igual que en el caso anterior analizando si la casilla en análisis es la de salida. En caso afirmativo, se comprueba si la solución actual es menor que la última almacenada.

En caso afirmativo se limpia la ultima almacenada y se copia la solución actual como la nueva solución más corta, también se especifica find como true, ya que se ha alcanzado una solución. Finalmente se devuelve un true para volver al punto de llamada al metodo desde la llamada recursiva anterior.

```
private boolean goAheadAllWays(int i, int j, Stack<Coordinate> path2, char[][] maze ) {

    if (i == endI && j == endJ) {
```

```
if(path2.size()<path.size()) {  
    path.clear();  
    path.addAll(path2);  
    find = true;  
}  
  
return true;  
}
```

Posteriormente si la casilla no es la de salida, se comprueba si se trata de una pared o de una casilla de la solución actual. Si es afirmativo se devuelve un false para volver a la llamada recursiva anterior.

```
if (maze[i][j] == '#' || checkPath(i,j,path2)) {  
    return false;  
}
```


En caso negativo se continua la ejecución y se analiza si la solución actual supera en longitud a la solución mas corta hasta el momento.

En caso afirmativo, se devuelve un false para optimizar el algoritmo y evitar alcanzar la salida por un camino que ya es mas largo que otro calculado con anterioridad.

```
if (path2.size()>path.size()) {  
    return false;  
}
```

En caso negativo, se añade la casilla a la solución actual del laberinto y se comienzan las llamadas recursivas al igual que en el caso anterior. La gran diferencia radica en que en este caso se realizarán todas las llamadas recursivas para garantizar que se han revisado todas las posibilidades, Finalmente se eliminan las casillas y se devuelve un false. Ya que la solución óptima queda almacenada siempre en la clase Maze.

```
path2.push(new Coordinate(i,j));  
  
goAheadAllWays(i, j+1, path2,maze);  
goAheadAllWays(i-1, j, path2,maze);  
goAheadAllWays(i, j-1, path2,maze);  
goAheadAllWays(i+1, j, path2,maze);  
  
path2.pop();  
return false;  
}
```


	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

8. Conclusiones.

Como conclusión se puede mencionar posibles mejoras a realizar en futuras versiones y limitaciones del algoritmo.

- Las mejoras que se pueden realizar en el proyecto pueden ser entre otras la implementación de un log para administradores del sistema que puedan comprobar las interacciones de los diferentes usuarios con la aplicación, de forma que se pueda obtener información que pueda ser de utilidad a la hora de realizar mejoras o solucionar problemas una vez desplegada esta.
- Se puede crear la posibilidad de un usuario con rol de administrador que pueda realizar funciones avanzadas, como puede ser consultar usuarios registrados, cambiar datos de estos, eliminar usuarios, añadir usuarios, etc.
- Añadir la opción para que cada usuario de forma individual pueda cambiar sus propios datos personales y que estos sean almacenados de nuevo en la base de datos.
- Como último paso se puede desarrollar toda la aplicación como una aplicación con interfaz gráfica de usuario GUI en vez de con una interfaz de línea de comandos. Esto mejora la vistosidad de la aplicación facilita y agiliza la experiencia de usuario y también permite un despliegue en otras plataformas de forma más ágil.

Las 4 mejoras propuestas aquí han sido desarrolladas a posteriori de este trabajo, encontrándose actualmente el software en la versión 2.4.0. Quedando desarrollada la interfaz gráfica y las mejoras anteriormente comentadas.

Para las mejoras que no aplican la GUI se han ido desarrollando progresivamente las mejoras, con las respectivas versiones 1.2.0, 1.3.0 y 1.4.0. Estas versiones continúan siendo versiones CLI.

Como propuesta personal considero que puede ser interesante que los usuarios puedan añadir sus propios laberintos desde la aplicación y que se pueda crear una interfaz que permita la comunicación asíncrona entre usuarios de forma que se de un toque más social a la aplicación.


Respecto al algoritmo, uno de los puntos fuertes es el uso de un proceso de simplificación previo que reduce considerablemente el tiempo de resolución del problema, además es más eficiente cuanto mayor y más complejo sea el laberinto a simplificar. Se trata de un proceso que puede ser utilizado en cualquier tipo de algoritmo, por lo que ayuda en general a que los algoritmos que lo usan sean más eficientes.

Respecto a limitaciones y puntos a mejorar del algoritmo destacaré los siguientes puntos:

Tras la simplificación del algoritmo, se podría realizar un paso previo a la aplicación del algoritmo que establezca pesos a las bifurcaciones de los diferentes nodos. Es decir, que en cada nodo se especifique las casillas que hay hasta el siguiente nodo y las coordenadas del siguiente nodo.

Esto puede ser utilizado para descartar recorrer caminos que se dirigen hacia el mismo nodo, tratando de seleccionar siempre el más corto. También se podría utilizar para establecer orden de prioridad a la hora de realizar llamadas recursivas.

De esta forma las llamadas recursivas comenzarían siempre por la casilla que presenta menos pasos hasta su siguiente nodo. Así el algoritmo sería mucho más rápido y eficiente.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Respecto a la ejecución en sí del algoritmo, se podrían realizar mejoras a la hora de evaluar precondiciones. Ya que en la resolución del laberinto solo se va a encontrar la salida al final del camino, es más eficiente realizar la comprobación de si la casilla actual es la salida tras comprobar si esta se trata de una pared o de una casilla ya recorrida.

Se ahorran así bastantes operaciones que van a resultar infructuosas y ralentizan el proceso.

Para el caso del camino más corto quedaría así la evaluación:

```
private boolean goAheadAllWays(int i, int j, Stack<Coordinate> path2, char[][] maze ) {
    if (maze[i][j] == '#' || checkPath(i,j,path2)) {
        return false;
    }

    if (i == endI && j == endJ) {
        if(path2.size()<path.size()) {
            path.clear();
            path.addAll(path2);
            find = true;
        }
        return true;
    }

    if (path2.size()>path.size()) {
        return false;
    }

    path2.push(new Coordinate(i,j));

    goAheadAllWays(i, j+1, path2,maze);
    goAheadAllWays(i-1, j, path2,maze);
    goAheadAllWays(i, j-1, path2,maze);
    goAheadAllWays(i+1, j, path2,maze);


    path2.pop();
    return false;
}
```

En la ejecución de la resolución de un laberinto de 80 * 82, antes del cambio propuesto el tiempo de ejecución es de aproximadamente 85.046 (s).

Tras la modificación el tiempo se reduce a unos 70.475 (s).

Si además se especifica como primera comprobación si el camino actual es superior al mas corto, entonces se reduce a 68.166 (s).


Por lo tanto, solamente con este cambio, se ha reducido el tiempo de ejecución en un 19,85%. Pudiendo ser mucho mejor si además se aplican las mejoras mencionadas previamente.

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024

Por último, se podría hacer uso de computación multihilo para las llamadas recursivas que reducirían de forma significativa el tiempo de ejecución.

9. Bibliografía

- Diseño y Análisis de Algoritmos (Backtracking) Universidad Rey Juan Carlos.
<https://www.cartagena99.com/recursos/visor.php?idrec=5733&fich=alumnos/temarios/Backtracking.pdf>
- Problema de los puentes de Königsberg. Wikipedia.
https://es.wikipedia.org/wiki/Problema_de_los_puentes_de_K%C3%B6nigsberg

	Proyecto: Maze-Solver 1.1.0	
	Asignatura: Entornos de Desarrollo - DAW	
	Versión del documento: V1.0 Versión del proyecto: V1.1.0	Fecha: 12/05/2024