**Manual Técnico**

**INTRODUCCION:**

MazeSolver es una aplicación diseñada para resolver cualquier tipo de laberinto que se le pueda presentar, sin importar su tamaño o dificultad. MazeSolver es perfecta para mostrar todas las soluciones distintas que puede tener un solo laberinto y entre esas escogerá el camino más corto para mostrárselo al usuario, si este lo desea.

**DESARROLLO:**

Esta aplicación fue desarrollada principalmente en Java con el IDE NetBeans, con el fin de separar mejor cada método que usamos para que la aplicación funcionara correctamente, creamos 5 clases llamadas **Program, Maze, Mice, Path, Point**. Para un mejor entendimiento vamos a definir cada una:

* **Clase Program:** es la clase principal en la que nos encontramos con una pequeña interfaz que se ejecuta en consola para saber cuales son las acciones que desea realizar el usuario, pero lo mas importante de esta clase es que al momento de ejecutar busca el archivo que contiene el laberinto el cual el usuario desea resolver y lo envía a la clase laberinto para poder ser usado.
* **Clase Maze:** esta clase recibe el laberinto que leímos en la clase programa, creando con este archivo una matriz de orden mxn en la cual podamos ejecutar todas las acciones del ratón hasta llegar al final. También esta es la que muestra en pantalla las soluciones del laberinto.
* **Clase Mice:** en la clase Mice se crea el objeto ratón el cual es el que vamos a posicionar en la entrada del laberinto, luego escogeremos un lado por el cual pueda ir el ratón de manera aleatoria, evaluaremos si se puede mover a esa dirección y si lo puede hacer se cambiará la posición del ratón y luego apilaremos su ubicación como coordenadas. De lo contrario, el ratón se devolverá a la última posición en la que estaba, desapilando las coordenadas y cerrando el lugar por el que acaba de pasar para que no lo vuelva a hacer.
* **Clase Path:** al final el recorrido por los caminos y encontrar la solución, era necesario guardar las coordenadas de todo el camino por el cual el ratón caminó, por lo que creamos la clase Path que guarda pilas diferentes con caminos, en cada pila se apilan las coordenadas (objetos de la clase Point).
* **Clase Point:** con el fin de guardar de una manera mas ordenada y coherente las coordenadas que tiene cada punto en el laberinto (la matriz laberinto), creamos la clase Point, la cual guarda un objeto con dos atributos, “x” y “y” y los devuelve en el momento que sea necesario.

Para la manipulación del laberinto, primero leímos el archivo que el usuario le proporciona con el laberinto, para esto, pasamos cada renglón (eliminando los espacios) a una pila, luego cada carácter de esa pila lo desapilamos y por último lo añadíamos a la matriz para que fuera el laberinto, esto lo repetimos hasta que la pila estuviera vacía.

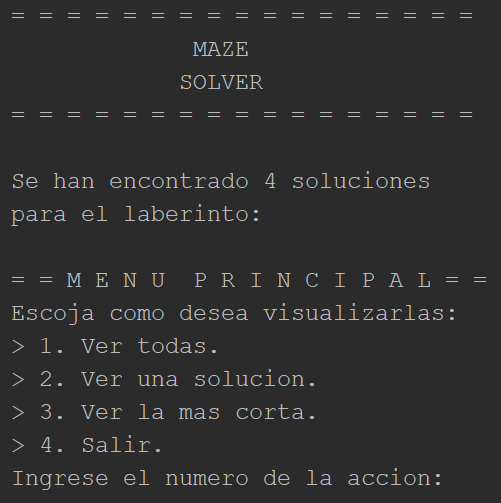
Con el fin de mostrar las soluciones correctas y bien explicadas al usuario, nos centramos en crear un numero aleatorio cada que el ratón se mueve para que así encuentre todas las soluciones posibles y que no siga un patrón especifico.

Para guardar los caminos y que pudieran ser usados después ya sea para verificar que no hayan dos caminos iguales o para mostrárselos al usuario, creamos pilas que guardaran objetos de la clase Point

(con coordenadas), estas pilas se iban llenando cada que el ratón hacía un movimiento y las guardamos en otra pila que tuviera los arreglos totales de caminos posibles para resolver el laberinto.

Como el laberinto se resolvía de manera aleatoria nos encontramos con que podían aparecer caminos iguales, por lo que, en el momento de terminación, cuando no encontrara mas caminos diferentes iba a comparar cada camino de la pila de caminos totales y simplemente iba a eliminar los que fueran copia de uno, para dejar siempre solo caminos diferentes.

**FUNCIONAMIENTO EN CONSOLA:**

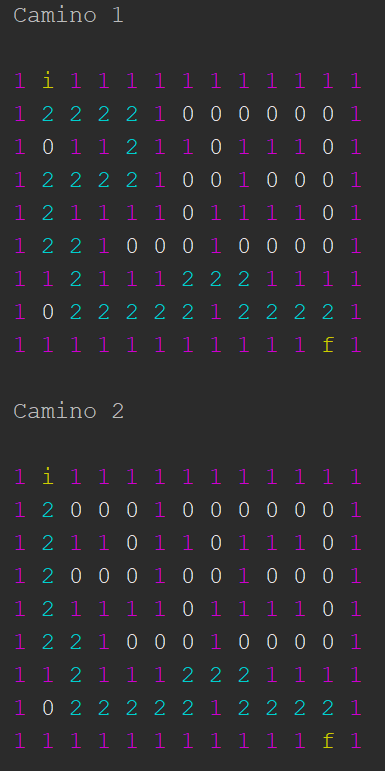
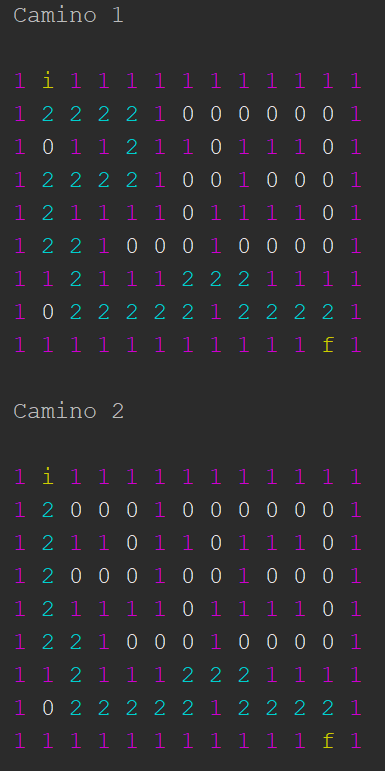
****Al ejecutar el programa tenemos esta pequeña interfaz de usuario, primero muestra el numero total de caminos por los cuales se puede resolver el laberinto, luego le da diferentes opciones al usuario para que interactúe con este.

Puede escoger entre “Ver todas” que muestra todas las soluciones separadas.

“Ver una solución” si se escoge esta el usuario pondrá el numero de la solución que sea ver y este se la mostrará.

Y por último “Ver la mas corta” que, como su nombre lo dice, muestra la solución mas corta por la que se resuelve el laberinto.

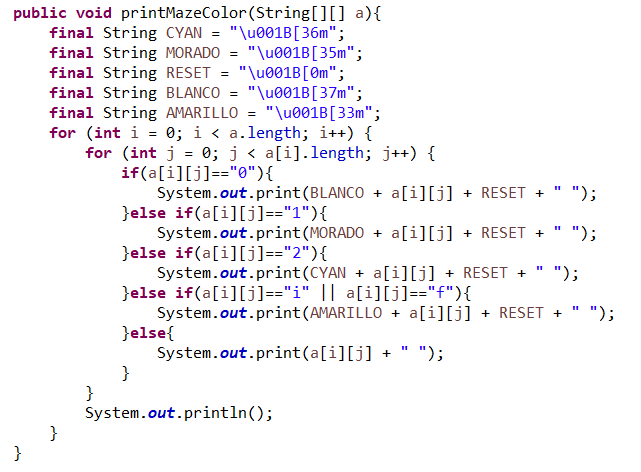
Así se muestran las soluciones, los unos son las paredes por las cuales no se puede cruzar, los ceros son los caminos vacíos por los cuales se puede pasar y los dos son el camino que la aplicación encontró como respuesta, la i representa el inicio y la f el final.

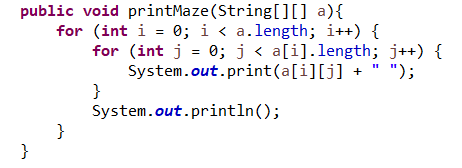
**NOTA IMPORTANTE:**

Para que se vean mejor las soluciones del laberinto, usamos NetBeans, que nos permite pintar en consola de una manera fácil y rápida, sin embargo, esto no funciona para cualquier IDE, por lo que es necesario tener en cuenta esto:

Si la aplicación se abre en NetBeans, el método que imprime el laberinto es este

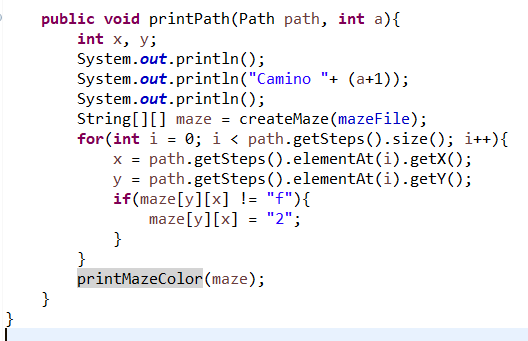


Al momento de imprimir cada objeto de la matriz, revisa cual es el color adecuado y lo pinta, sin embargo, esta forma no funciona bien en otros IDEs por lo que agregamos un método para imprimir el laberinto sin color, el cual es el siguiente:



**¿cómo cambiar de uno a otro?** Para las personas que usen un IDE diferente a NetBeans y tengan problemas en la impresión del laberinto, pueden cambiarlo de una forma muy sencilla, simplemente van a la clase **Maze** y buscan el método printPath, al final de este verán que se ejecuta el método printMazeColor(maze); , esta línea de código la deben cambiar por printMaze(maze); y listo, ahora funcionara correctamente pero sin color, por lo que será más difícil ver la solución en pantalla.

Antes de hacer este cambio se ve así:



Después del cambio, el método queda de esta forma:

