

**Centro de Ciencias Básicas**

**Inteligencia Artificial**

*Laberinto Inteligente*

Ingeniería en Computación Inteligente

Semestre 3° A

agosto-diciembre 2022

|  |  |
| --- | --- |
| **Integrantes:** | **ID** |
| Durón Láriz Miguel Ángel | 331992 |
| Elías del Hoyo César Eduardo | 262045 |
| López Flores Kandy Fabiola | 326912 |
| Ortiz Quiroz Ángel David | 261481 |
| Rivera Delgadillo Ximena | 261261 |
| Sandoval Pérez José Luis | 261731 |
| Saucedo Ortega Diego Emanuel | 261230 |
| Torres Macías Carlos Daniel | 244543 |
|  |  |

Docente: Doc. Alejandro Padilla Díaz

Fecha de entrega: 3 de diciembre del 2022

Contenido

[Introducción 2](#_Toc121173204)

[Desarrollo 4](#_Toc121173205)

[Conclusión 28](#_Toc121173206)

[Bibliografía 29](#_Toc121173207)

En este documento debemos poner **portada, desarrollo (imágenes y código), conclusiones particulares y de equipo**.

# Introducción

Durante este proyecto, realizamos un laberinto inteligente el cual tenía como base el ejercicio del algoritmo genético. El objetivo de dicho problema era crear un algoritmo que encontrara un camino o dos, dentro de una matriz de 40 \* 40, que permitiera tener una salida al final de este. Las indicaciones del proyecto eran las siguientes:

* Creación de una matriz de 40\*40, la cual contendrá de 33 a 45% de 1s
* El laberinto será generado de manera aleatoria, valiéndose además de técnicas como la mutación, donde el único valor fijo será el punto de partida en la entrada [0,0] de la matriz
* Generar mínimo dos caminos que lleven a la meta, para determinar el mejor de ambos, hacer uso de alguno de los métodos de búsqueda analizados en el curso.

Previo a la realización del proyecto, tuvimos que investigar un poco sobre los algoritmos genéticos (realizado en segundo parcial de este semestre) y la investigación de laberintos inteligentes.

Los algoritmos inteligentes son un tipo de programa informático que permite a los ordenadores realizar tareas que normalmente requieren inteligencia humana. Estos programas permiten que los ordenadores aprendan y se adapten a su entorno, reconozcan objetos y sonidos y procesen la información del mismo modo que lo hacen los seres humanos.

El laberinto inteligente permite encontrar una solución óptima mediante métodos de búsqueda y apoyándose con el algoritmo genético, el cual nos permitirá realizar mutaciones para encontrar un camino a la salida.

Para ello realizamos 2 algoritmos que nos permitieron comparar las diferencias que encontramos entre ambos programas y, a su vez, obtener una solución correcta.

# Desarrollo

**Primer programa**

Este ejercicio lo realizamos en el lenguaje de Python. La realización de este algoritmo fue creando

Pasos para la realización del código:

1. El primer paso del programa es generar una matriz de bits aleatorios de 40\*40 a la cual vamos a forzar para que su primer valor es decir el valor [0][0] de la matriz siempre sea.

Es decir que nuestro laberinto siempre inicie de la esquina superior izquierda.

Patrón de fondo

Descripción generada automáticamente

1. Después de generar la matriz aleatoria con nuestra puerta utilizamos el método A\* para buscar si en este ya se encuentra un camino que una la primera fila con la última. En otras palabras, que recorra de nuestra puerta a alguna salida (1) de la última fila. De ser que no continuamos y mutamos.
2. Mutamos para ir generando un camino desde nuestra puerta hasta la última fila, para esto localizamos el último punto donde se genera un camino y a partir de esto mutamos un valor aleatorio inferior o derecho, para que de esta forma forcemos a que el camino se acerque a la última fila.

Guardamos las posiciones de los (0) mutados para compensar en la fila según el porcentaje que se haya seleccionado.

Patrón de fondo

Descripción generada automáticamente

1. Posteriormente volvemos a aplicar A\* para localizar que nuestro camino sea el mejor y obtener al mismo tiempo el peso de este.

Patrón de fondo

Descripción generada automáticamente

1. Imprimimos las coordenadas del camino desarrollado:

Caminito

[1,0]

[2,0]

[3,0]

[3,1]

[3,2]

[3,3]

[3,4]

[3,5]

[3,6]

[3,7]

[4,7]

[5,7]

[5,8]

[5,9]

[6,9]

[6,10]

[6,11]

[7,11]

[8,11]

[9,11]

[9,12]

[9,13]

[10,13]

[10,14]

[10,15]

[10,16]

[10,17]

[11,17]

[12,17]

[13,17]

[14,17]

[15,17]

[15,18]

[15,19]

[15,20]

[16,20]

[16,21]

[17,21]

[17,22]

[17,23]

[18,23]

[18,24]

[18,25]

[18,26]

[18,27]

[19,27]

[19,28]

[20,28]

[20,29]

[20,30]

[21,30]

[22,30]

[23,30]

[24,30]

[24,31]

[25,31]

[25,32]

[25,33]

[25,34]

[25,35]

[26,35]

[26,36]

[27,36]

[27,37]

[27,38]

[28,38]

[29,38]

[30,38]

[31,38]

[32,38]

[33,38]

[34,38]

[35,38]

[35,37]

[35,36]

[34,36]

[34,35]

[34,34]

[35,34]

[36,34]

[37,34]

[37,35]

[38,35]

[39,35]

1. Guardamos las coordenadas de la salida y repetimos el proceso de manera inversa con la condicional de que la salida de este nuevo camino sea la entrada [0][0] de la matriz.
2. De esta forma finalmente obtenemos un segundo camino.

Patrón de fondo

Descripción generada automáticamente

1. Finalmente volvemos a emplear el A\* para determinar el mejor camino encontrado e imprimirlo.

Patrón de fondo

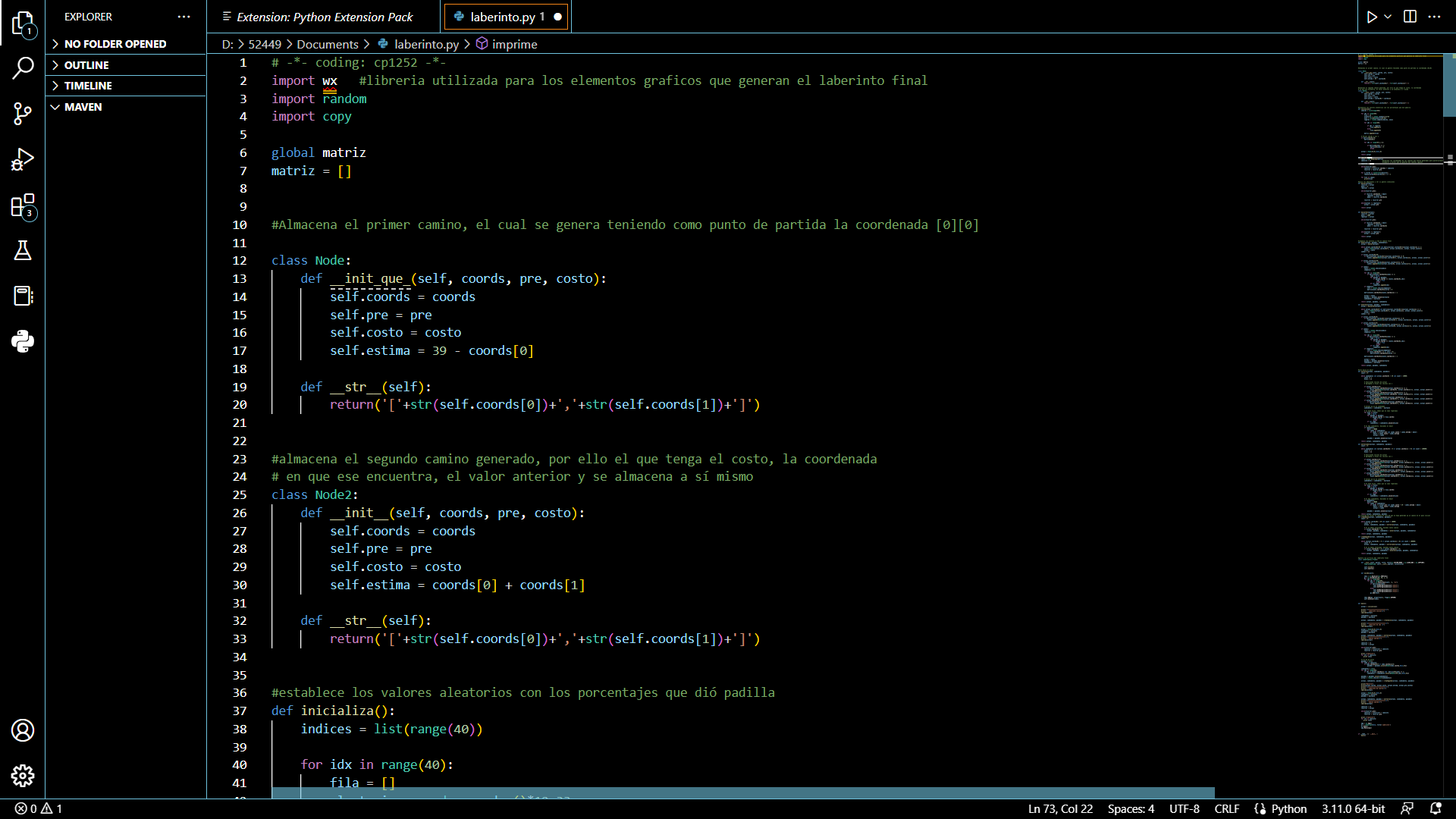
Descripción generada automáticamente

1. De esta forma ya tenemos un laberinto con dos caminos y un camino seleccionado como el mejor.
2. Finalmente, solo imprimimos el laberinto resultante de manera grafica para poder identificar todo con mayor claridad.

Código QR

Descripción generada automáticamente

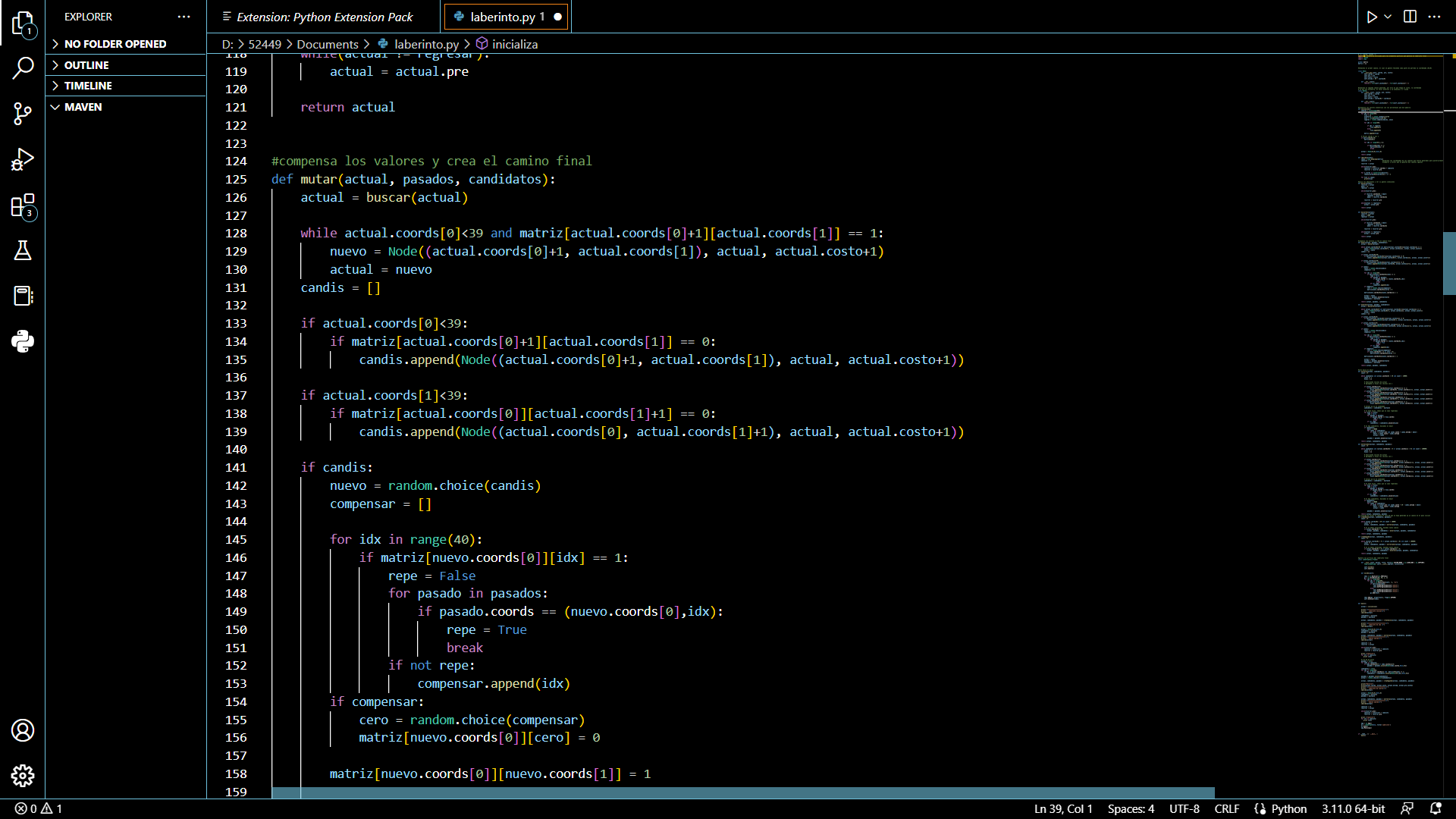
Capturas de código documentado de Python





Texto

Descripción generada automáticamente





Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Pantalla de computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

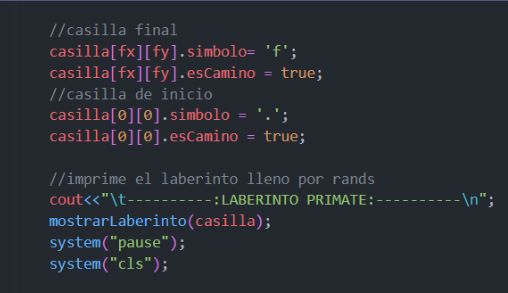
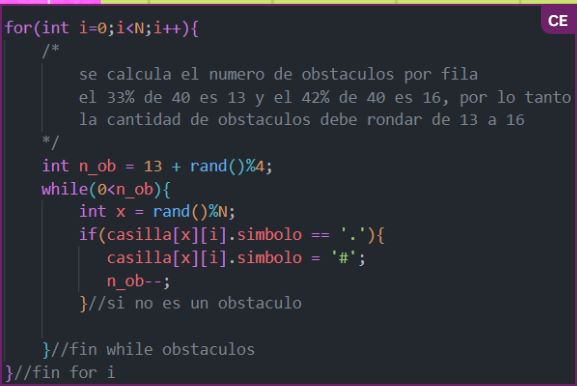
Texto

Descripción generada automáticamente

**Segundo programa**

Para este segundo programa, empleamos el lenguaje de C++ y C#, pues el primer modelo fue en C++, pero para una mejor presentación y diseño del proyecto, usamos C# en Visual Studio

Este programa se basa en la creación de un algoritmo que permita mover barreras de forma que permita abrir huecos y encontrar un camino a la salida de laberinto. Tenemos una matriz aleatoria de 40 x 40 la cual se llenará con ceros y unos (en nuestro caso, usamos “.” para los huecos y “#” para las barreras). Estos espacios se llenarán con un porcentaje determinado obtenido al azar entre el 33% y 42%. Siempre se tendrá el espacio [0,0] con un hueco, para empezar el camino. “f” será el final del camino, situado siempre en la última fila



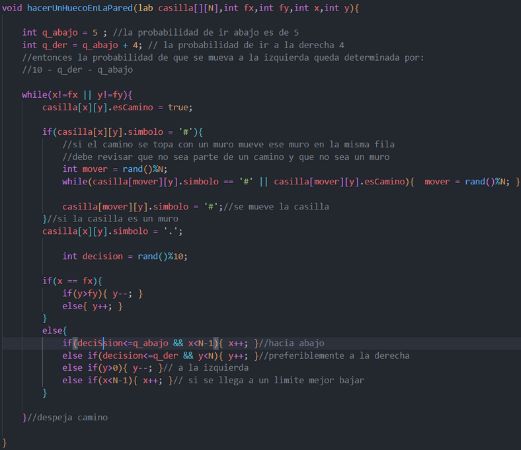
Después de hacer el llenado de la matriz, pasamos a la función “hacerunhuecoenlapared”. Esta función nos permitirá obtener huecos para poder seguir avanzando en el laberinto. Esto se realizó usando mutación. Se hizo lo siguiente:

Se obtuvo un porcentaje para tomar en cuenta la posición del hueco a realizar con los siguientes parámetros:

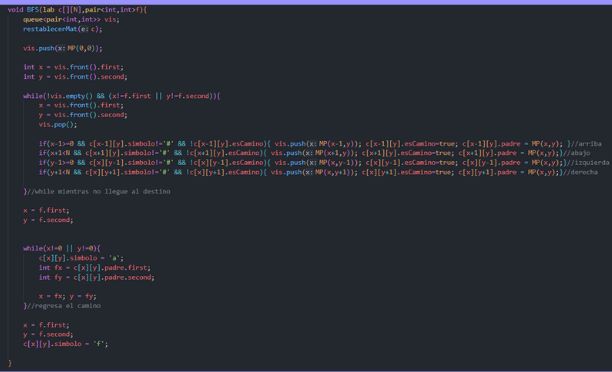
* 50% de probabilidad para mutar en la casilla de abajo
* 40% de probabilidad para mutar en la casilla de la derecha
* 10% de probabilidad para mutar en la casilla de la izquierda

Esto con el fin de obtener de manera aleatoria, mediante las probabilidades, un camino que no sea muy monótono y siempre en dirección hacia la derecha abajo. Sin embargo, no se usó un porcentaje de hacer hueco arriba, porque entonces el laberinto se haría tedioso y en ocasiones tendría a tardarse más.

Al avanzar, si la casilla es un hueco, simplemente se desplazará hacia ese espacio. Pero si tenemos un muro, dicho muro será movido a un espacio situado en la misma fila, pero sin que este afecte al camino ya establecido hasta el momento, para entonces crear un hueco y seguir avanzando. Este procedimiento seguirá hasta llegar a la letra f, el final del laberinto. En cierta ocasión donde lleguemos al borde de la matriz y aún no encontremos la solución, se realizará en forma recta, desde el borde hasta el fin del laberinto para concluir más rápido la salida. Hasta aquí se ha encontrado la solución en base a los huecos realizados.

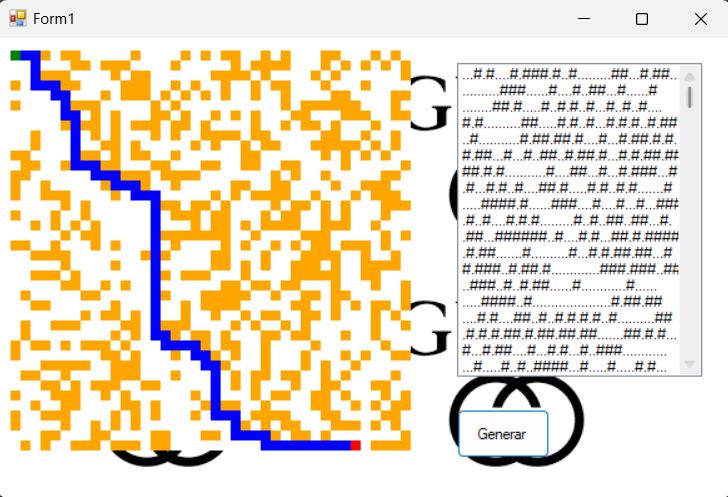


Para finalizar, ser realiza la función usando el método BFS (*Breadth First Search*). Este método de búsqueda en anchura nos permitirá encontrar la solución al camino, de forma que explorará cada uno de los nodos (espacios de la matriz, en este caso) vecinos, encontrando una solución óptima. BFS permite localizar todos los huecos disponibles que hay en el laberinto, identificando cada uno de ellos y marcándolos, para obtener la mejor solución. Este algoritmo lo que permitirá es encontrar el camino y este se cambiará por la letra “a”, de forma que se observe el resultado final.

****

****

Para una mejor presentación del resultado final, realizamos el mismo código en C#, usando Visual Studio, de forma que pudimos darle un diseño más llamativo y que muestre el camino con colores, cambiando el camino de “a” con un color distinto al de los huecos y barreras

****

# Conclusión

Con este proyecto podemos concluir que los algoritmos de búsqueda tienen un sinfín de aplicaciones, en este proyecto se abordó darle solución a un laberinto, cuyo objetivo era encontrar la salida. Nos pareció interesante como es que cada algoritmo de búsqueda trabaja de una manera distinta, al hacer las comparaciones de ambos programas nos dimos dar cuenta que ambos son opuestos. Uno obtiene la solución óptima y otro obtiene la solución más rápida. Sin duda un proyecto muy divertido de realizar y gracias a la ayuda de los entornos gráficos todo fue más fácil y digerible.

Conclusión personal

Los laberintos siempre me han parecido interesantes, crear un laberinto fue de las cosas que más rescato de este semestre, me sorprendió como es que logre desarrollar mis capacidades para poder desarrollar un código que determinara la solución de un laberinto. Sin duda de mis proyectos favoritos.

# Bibliografía

*Difference between Breadth Search (BFS) and Deep Search (DFS)*. (2020, 25 mayo). Encora. https://www.encora.com/es/blog/dfs-vs-bfs

Jain, S. (2021, 5 marzo). *The Insider’s Guide to A\* Algorithm in Python*. Python Pool. https://www.pythonpool.com/a-star-algorithm-python/

*Miembros de clase en C++ Variables y Métodos - Clases y Objetos en C++ (Práctica 1)*. (s. f.). CodinGame. https://www.codingame.com/playgrounds/50557/clases-y-objetos-en-c-practica-1/miembros-de-clase-en-c-variables-y-metodos