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

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

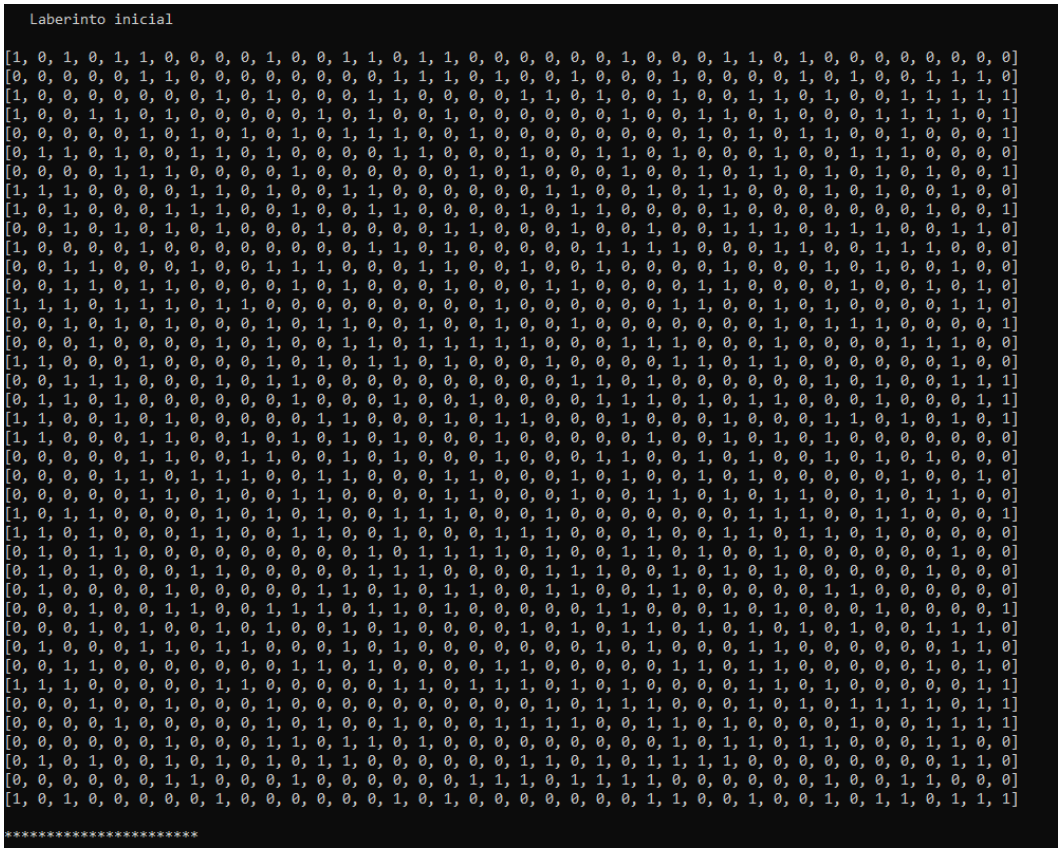
Primer programa

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.



2. 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.
3. 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.

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

5. 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]

6. 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.

7. De esta forma finalmente obtenemos un segundo camino.

```
  Laberinto de vuelta

[69, 68, 67, 66, 65, 64, 63, 62, 61, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 0, 0, 1, 1, 0, 60, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 59, 58, 57, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 56, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 55, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 54, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 53, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 52, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 51, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 50, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 49, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 48, 47, 46, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 45, 44, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 43, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 42, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 40, 41, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 38, 39, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 0, 35, 36, 37, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 34, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]
[1, 1, 0, 0, 1, 0, 1, 0, 0, 33, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 1, 0, 0, 32, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 31, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 30, 29, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 28, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 27, 26, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 25, 24, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 22, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 20, 19, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 18, 17, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 16, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 15, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 14, 13, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 11, 12, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 8, 9, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 6, 7, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0]
[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 5, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 4, 3, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1]

**********************
```

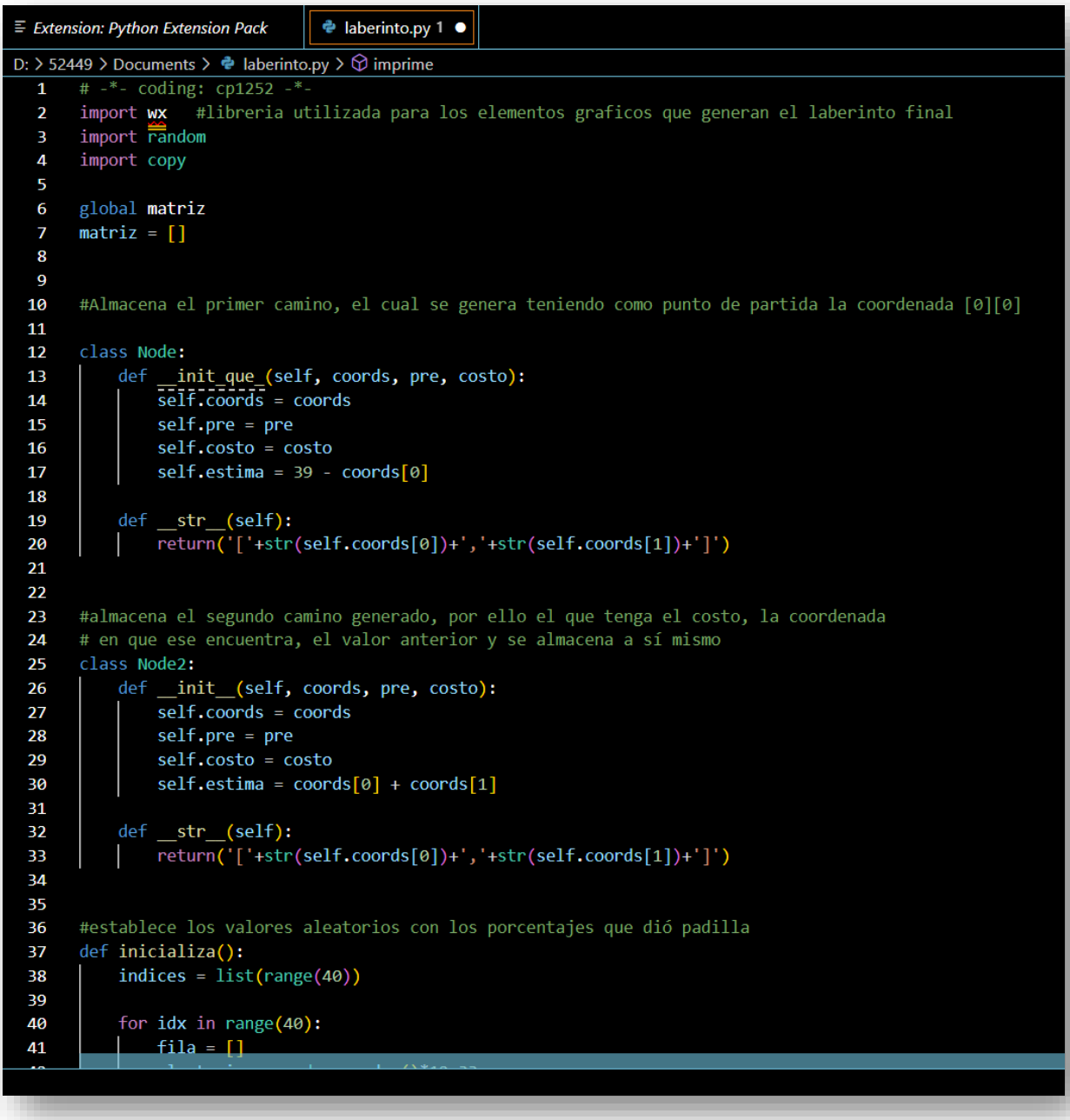
8. Finalmente volvemos a emplear el A* para determinar el mejor camino encontrado e imprimirlo.

```
Camino optimo

[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 1, 1, 0, 10, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 11, 12, 13, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 14, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 15, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 16, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 17, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 18, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 19, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 20, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 22, 23, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 24, 25, 26, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 27, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 28, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 30, 29, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 33, 32, 31, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 0, 35, 34, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 36, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]
[1, 1, 0, 0, 1, 0, 1, 0, 0, 37, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 1, 0, 0, 38, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 39, 40, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 41, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 42, 43, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 44, 45, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 46, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 47, 48, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 49, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 50, 51, 52, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 53, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 54, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 55, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 56, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 57, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 58, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 60, 59, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 62, 61, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 63, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 64, 65, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 66, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1]
Camino
```

9. De esta forma ya tenemos un laberinto con dos caminos y un camino seleccionado como el mejor.

10. Finalmente, solo imprimimos el laberinto resultante de manera grafica para poder identificar todo con mayor claridad.

Capturas de código documentado de Python



```
41          fila = []
42          aleatorio = random.random()*12+33
43          unos = int(40*aleatorio*0.01)
44          lugares = random.sample(indices, unos)
45
46          for jdx in range(40):
47
48              if jdx in lugares:
49                  fila.append(1)
50              else:
51                  fila.append(0)
52
53          matriz.append(fila)
54
55      # Forzar [0][0] a ser 1
56      if matriz[0][0]==0:
57          matriz[0][0]=1
58
59          for idx in range(39,1,-1):
60
61              if matriz[0][idx] == 1:
62                  matriz[0][idx] = 0
63                  break
64
65      actual = Node((0,0),None,0)
66
67      return actual
68
69  def imprime(actual):
70      copia = copy.deepcopy(matriz)
71      caminito = []               #almacena las coordenadas de los caminos que fueron generados para posteriormente
72                                  #comparar y evitar que se generen dos caminos iguales
73      recorrer = actual
74
75      while(recorrer.pre):
76          caminito = [recorrer.coords] + caminito
77          recorrer = recorrer.pre
78
79      for n,coords in enumerate(caminito):
80          copia[coords[0]][coords[1]] = n + 2
81
```

```
80              copia[coords[0]][coords[1]] = n + 2
81
82          for fila in copia:
83              print(fila)
84
85      #busca los adyacentes y ver si genera conexiones
86      def buscar(actual):
87          recorrer = actual
88          mayor = 0
89          regresar = actual
90
91          while(recorrer.pre):
92
93              if recorrer.coords[0] > mayor:
94                  regresar = recorrer
95                  mayor = recorrer.coords[0]
96
97              recorrer = recorrer.pre
98
99          while(actual != regresar):
100             actual = actual.pre
101
102         return actual
103
104
105     def buscarInv(actual):
106         recorrer = actual
107         menor = 1000
108         regresar = actual
109
110         while(recorrer.pre):
111
112             if recorrer.coords[0] < menor:
113                 regresar = recorrer
114                 menor = recorrer.coords[0]
115
116             recorrer = recorrer.pre
117
118         while(actual != regresar):
119             actual = actual.pre
120
```

D: > 52449 > Documents > laberinto.py > inicializa

```
        actual = actual.pre

    return actual


#compensa los valores y crea el camino final
def mutar(actual, pasados, candidatos):
    actual = buscar(actual)

    while actual.coords[0]<39 and matriz[actual.coords[0]+1][actual.coords[1]] == 1:
        nuevo = Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1)
        actual = nuevo
    candis = []

    if actual.coords[0]<39:
        if matriz[actual.coords[0]+1][actual.coords[1]] == 0:
            candis.append(Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1))

    if actual.coords[1]<39:
        if matriz[actual.coords[0]][actual.coords[1]+1] == 0:
            candis.append(Node((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))

    if candis:
        nuevo = random.choice(candis)
        compensar = []

        for idx in range(40):
            if matriz[nuevo.coords[0]][idx] == 1:
                repe = False
                for pasado in pasados:
                    if pasado.coords == (nuevo.coords[0],idx):
                        repe = True
                        break
                if not repe:
                    compensar.append(idx)
        if compensar:
            cero = random.choice(compensar)
            matriz[nuevo.coords[0]][cero] = 0

        matriz[nuevo.coords[0]][nuevo.coords[1]] = 1

```

```
157
158             matriz[nuevo.coords[0]][nuevo.coords[1]] = 1
159
160             actual = nuevo
161             pasados = pasados.union({actual})
162             candidatos = {actual}
163
164         return actual, pasados, candidatos
165
166     def mutarInv(actual, pasados, candidatos):
167         actual = buscarInv(actual)
168
169         while actual.coords[0]>0 and matriz[actual.coords[0]-1][actual.coords[1]] == 1:
170             nuevo = Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1)
171             actual = nuevo
172         candis = []
173
174         if actual.coords[0]>0:
175             if matriz[actual.coords[0]-1][actual.coords[1]] == 0:
176                 candis.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
177
178         if actual.coords[1]>0:
179             if matriz[actual.coords[0]][actual.coords[1]-1] == 0:
180                 candis.append(Node2((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
181
182         if candis:
183             nuevo = random.choice(candis)
184             compensar = []
185
186             for idx in range(40):
187                 if matriz[nuevo.coords[0]][idx] == 1:
188                     repe = False
189                     for pasado in pasados:
190                         if pasado.coords == (nuevo.coords[0],idx):
191                             repe = True
192                             break
193                     if not repe:
194                         compensar.append(idx)
195             if compensar:
196                 cero = random.choice(compensar)
197                 if nuevo.coords[0] > 5 or cero > 0:
198                     matriz[nuevo.coords[0]][cero] = 0
```

```
            if nuevo.coords[0] > 5 or cero > 0:
                matriz[nuevo.coords[0]][cero] = 0

        matriz[nuevo.coords[0]][nuevo.coords[1]] = 1

        actual = nuevo
        pasados = pasados.union({actual})
        candidatos = {actual}

    return actual, pasados, candidatos


#solo busca el mejor
def estrella(actual, candidatos, pasados):
    count = 0

    while candidatos and actual.coords[0] < 39 and count < 10000:
        count += 1
        hijos = []

        # Analizando vecinos de actual
        # Agregamos a hijos los vecinos con 1

        if actual.coords[1]>0:
            if matriz[actual.coords[0]][actual.coords[1]-1] == 1:
                hijos.append(Node((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
        if actual.coords[1]<39:
            if matriz[actual.coords[0]][actual.coords[1]+1] == 1:
                hijos.append(Node((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))
        if actual.coords[0]>0:
            if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
                hijos.append(Node((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
        if actual.coords[0]<39:
            if matriz[actual.coords[0]+1][actual.coords[1]] == 1:
                hijos.append(Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1))

        # Actual ya no es candidado
        candidatos = candidatos - {actual}

        # Si hubo hijos, vemos que no sean repetidos
        for hijo in hijos:
```

```
        # Si hubo hijos, vemos que no sean repetidos
        for hijo in hijos:
            repe = False
            for pasado in pasados:
                if pasado.coords == hijo.coords:
                    repe = True
                    break
            if not repe:
                candidatos = candidatos.union({hijo})

        # Si hay candidatos, buscamos el mejor
        if candidatos:
            menor = 10000
            for candi in candidatos:
                if candi != actual.pre and candi.costo + candi.estima < menor:
                    menor = candi.costo + candi.estima
                    actual = candi

            pasados = pasados.union({actual})

    return actual, candidatos, pasados

def estrellaInv(actual, candidatos, pasados):
    count = 0

    while candidatos and (actual.coords[0] > 0 or actual.coords[1] > 0) and count < 100000:
        count += 1
        hijos = []

        # Analizando vecinos de actual
        # Agregamos a hijos los vecinos con 1

        if actual.coords[1]>0:
            if matriz[actual.coords[0]][actual.coords[1]-1] == 1:
                hijos.append(Node2((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
        if actual.coords[1]<39:
            if matriz[actual.coords[0]][actual.coords[1]+1] == 1:
                hijos.append(Node2((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))
        if actual.coords[0]>0:
            if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
                hijos.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
```

```
275             if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
276                 hijos.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
277         if actual.coords[0]<39:
278             if matriz[actual.coords[0]+1][actual.coords[1]] == 1:
279                 hijos.append(Node2((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1))
280 
281         # Actual ya no es candidado
282         candidatos = candidatos - {actual}
283 
284         # Si hubo hijos, vemos que no sean repetidos
285         for hijo in hijos:
286             repe = False
287             for pasado in pasados:
288                 if pasado.coords == hijo.coords:
289                     repe = True
290                     break
291             if not repe:
292                 candidatos = candidatos.union({hijo})
293 
294         # Si hay candidatos, buscamos el mejor
295         if candidatos:
296             menor = 10000
297             for candi in candidatos:
298                 if candi != actual.pre and candi.costo + 39 - candi.estima < menor:
299                     menor = candi.costo + candi.estima
300                     actual = candi
301 
302             pasados = pasados.union({actual})
303 
304     return actual, candidatos, pasados
305 #se encarga de forzar el camino en caso de que no haya generado ya un camino en el paso inicial
306 def creaCamino(actual, candidatos, pasados):
307     count = 0
308 
309     while actual.coords[0] < 39 and count < 10000:
310         count += 1
311         actual, candidatos, pasados = estrella(actual, candidatos, pasados)
312 
313         # Si no hemos terminado, hacemos nuevo camino
314         if actual.coords[0] < 39:
315             actual, pasados, candidatos = mutar(actual, pasados, candidatos)
```

```
314             if actual.coords[0] < 39:
315                 actual, pasados, candidatos = mutar(actual, pasados, candidatos)
316
317         return actual, candidatos, pasados
318
319     def creaSegundo(actual, candidatos, pasados):
320         count = 0
321
322         while (actual.coords[0] > 0 or actual.coords[1] > 0) and count < 100000:
323             count += 1
324             actual, candidatos, pasados = estrellaInv(actual, candidatos, pasados)
325
326             # Si no hemos terminado, hacemos nuevo camino
327             if actual.coords[0] > 0 or actual.coords[1] > 0:
328                 actual, pasados, candidatos = mutarInv(actual, pasados, candidatos)
329
330         return actual, candidatos, pasados
331
332
333     #genera los gráficos del laberinto final
334     class LabeGraph(wx.Frame):
335
336         def __init__(self, parent, title, style=wx.SYSTEM_MENU | wx.CLOSE_BOX | wx.CAPTION):
337             super(LabeGraph, self).__init__(parent, title=title)
338
339             self.InitUI()
340             self.Centre()
341
342
343         def InitUI(self):
344
345             vbox = wx.BoxSizer(wx.VERTICAL)
346             gs = wx.GridSizer(40, 40, 1, 1)
347             for idx in range(40):
348                 for jdx in range(40):
349                     text = wx.StaticText(self, -1, "11")
350                     if matriz[idx][jdx]:
351                         text.SetBackgroundColour('white')
352                         text.SetForegroundColour('white')
353                     else:
354                         text.SetBackgroundColour('black')
355                         text.SetForegroundColour('black')
```

```
                    text.SetBackgroundColour('black')
                    text.SetForegroundColour('black')
                gs.Add(text)


        vbox.Add(gs, proportion=1, flag=wx.EXPAND)
        self.SetSizer(vbox)


def main():

    actual = inicializa()

    print("\n**********************\n")
    print("   Laberinto inicial\n")
    imprime(actual)

    candidatos = {actual}
    pasados = {actual}

    actual, candidatos, pasados = creaCamino(actual, candidatos, pasados)

    print("\n**********************\n")
    print("   Laberinto de ida \n")
    imprime(actual)

    actual = Node((0,0),None,0)
    candidatos = {actual}
    pasados = {actual}

    actual, candidatos, pasados = estrella(actual, candidatos, pasados)
    print("\n**********************\n")
    print("   Camino optimo\n")
    imprime(actual)

    caminito = []
    recorrer = actual

    while(recorrer.pre):
        caminito = [recorrer] + caminito
        recorrer = recorrer.pre
```

```
393         caminito = [recorrer] + caminito
394         recorrer = recorrer.pre
395 
396     print ("Caminito")
397     for ele in caminito:
398       print (ele)
399 
400     # # # Va pa atras
401     pasados = set()
402     for nodo in caminito:
403         if nodo.coords[0]>5 or nodo.coords[1]>5:
404             pasados = pasados.union({Node2(nodo.coords,None,0)})
405 
406     candidatos = set()
407     for idx in range(40):
408         if idx != actual.coords[1] and  matriz[39][idx] == 1:
409             candidatos = candidatos.union({Node2((39,idx),None,0)})
410 
411     pasados = pasados.union(candidatos)
412     actual = random.choice(list(candidatos))
413 
414     actual, candidatos, pasados = creaSegundo(actual, candidatos, pasados)
415 
416     print("Reversa")
417     print(actual.coords, actual.costo, actual.estima, actual.pre.coords)
418     print("\n**********************\n")
419     print("   Laberinto de vuelta\n")
420     imprime(actual)
421 
422     actual = Node((0,0),None,0)
423     candidatos = {actual}
424     pasados = {actual}
425 
426     actual, candidatos, pasados = estrella(actual, candidatos, pasados)
427     print("\n**********************\n")
428     print("   Camino optimo\n")
429     imprime(actual)
430 
431     caminito = []
432     recorrer = actual
433 
```

```
431         caminito = []
432         recorrer = actual
433 
434         while(recorrer.pre):
435             caminito = [recorrer] + caminito
436             recorrer = recorrer.pre
437 
438         print ("Caminito")
439         for ele in caminito:
440             print (ele)
441 
442         app = wx.App()
443         ex = LabeGraph(None, title='Laberinto')
444         ex.Show()
445         app.MainLoop()
446 
447 
448 
449     if __name__ == '__main__':
450         main()
451 
```

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

```
for(int i=0;i<N;i++){
    /*
        se calcula el numero de obstaculos por fila
        el 33% de 40 es 13 y el 42% de 40 es 16, por lo tanto
        la cantidad de obstaculos debe rondar de 13 a 16
    */
    int n_ob = 13 + rand()%4;
    while(0<n_ob){
        int x = rand()%N;
        if(casilla[x][i].simbolo == '.'){
            casilla[x][i].simbolo = '#';
            n_ob--;
        }//si no es un obstaculo

    }//fin while obstaculos
}//fin for i
```

```
    //casilla final
    casilla[fx][fy].simbolo= 'f';
    casilla[fx][fy].esCamino = true;
    //casilla de inicio
    casilla[0][0].simbolo = '.';
    casilla[0][0].esCamino = true;

    //imprime el laberinto lleno por rands
    cout<<"\t----------:LABERINTO PRIMATE:----------\n";
    mostrarLaberinto(casilla);
    system("pause");
    system("cls");
```

```
----------:LABERINTO PRIMATE:----------
.##.#..##..#..#.#..##...#.............
.....##......#......#.#..#..###...#####
#..........#.#..#....###..#.#..##..###.
#..#.........#....#####..##..#.#...##..
.#.#.#..#.......##.#.....##..##...##...#
#..#...#.##.#..#......#.....##.##.....#.
..#..#.#..##.####....#.###.#.#.##..#..#.
..#..#.#.##.#.#...#.#...#..#.....##..#.#
.##.......#.#..###.###.###.#####.###....
#...#.####...#...#.##.......#.....##...#
#...#.#.#.#..##..##.....#..##.#.#....###
.......###.....###..####...#.....#....#
#..#.#....##..##.##..#....#.#...###.#...
..#...#..#...####.....#..#.#.##...#.##.
#....#....#..#.#..#..##.##.###..........
.##....#.....#........#.#.#...##..#.##.
.#.#.#..#.#...#.....#....#.#..#.##.#.#..
.##.#....#......#........#..#..#.#..#...
.#.####.#...#..#....#.#....##.....####.
...#.#..###..####.###.##...#...#.#..#...
...##...#.###.....##..#..#.....#..##...#
..#...#..#.###..#...##....###.....#.#...
.##.#.##....#.##..##.....#.#...#####....
.#.........#....##.....##...#.#....#.#.
.#.#..####....#..##...#.............#.#
..##..##..#.#..##..##.##...##.#..#..##..
#...#..#...#.#...##..#...##..#..###..#..
.#....#.....#.......#...#....#..#..#..#
###.....#...#....##....##....#..#.....#
#..#.#.#..####..#....#..#.#.....##....#.
..##.#.....#...#...#.##.#....##.......##
....#.##.#.##.#....##............#..##.#
#..###..#.......###.#.#..##...##..#....#
#####.#..#....###.....#.###..#..#......#
.....#..#...##..#####..###....#.###.#..#.
....#.##.#####..####.##.#..#.....#....#
#..#....#..#.....#...####.###..#..##.#..
........##....#..##..#....#.#.#...###...
..#.#....#.#.#.#....#....##.##......#...
.#.##.f..#.#.....###.#.#........#.###.##
```

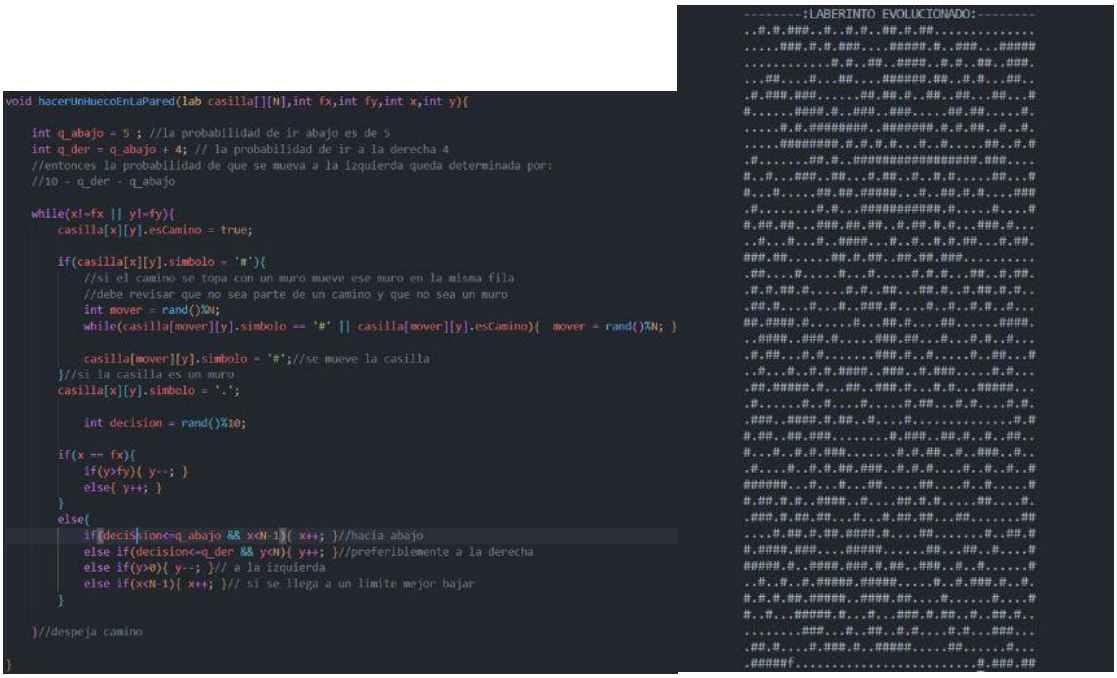
Después de hacer el llenado de la matriz, pasamos a la función "hacerunhuecoenlapared". Esta función nos permitirá obtener huecos para poder seguir avanzando en el laberinto. Esto se realizó usando mutación. Se hizo lo siguiente:

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.



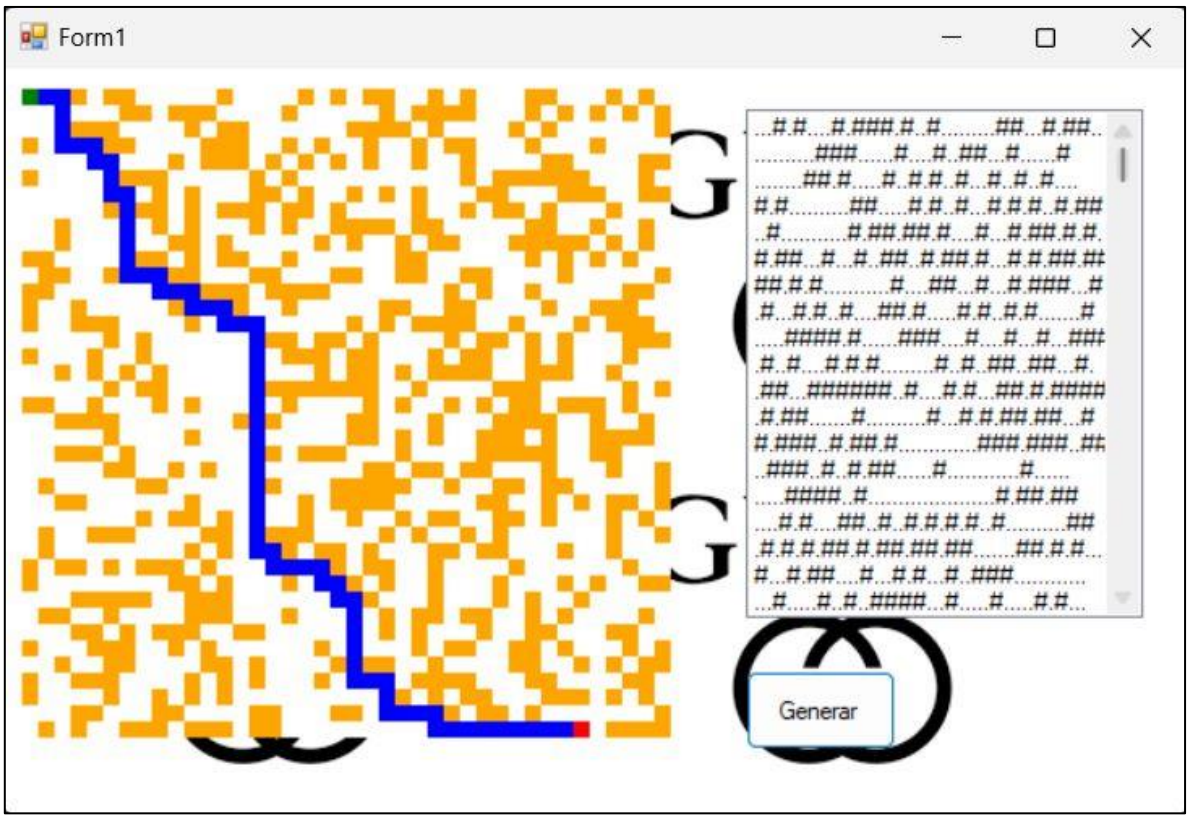
```
void hacerUnHuecoEnLaPared(lab casilla[][N],int fx,int fy,int x,int y){

    int q_abajo = 5 ; //la probabilidad de ir abajo es de 5
    int q_der = q_abajo + 4; // la probabilidad de ir a la derecha 4
    //entonces la probabilidad de que se mueva a la izquierda queda determinada por:
    //10 - q_der - q_abajo

    while(x!=fx || y!=fy){
        casilla[x][y].esCamino = true;

        if(casilla[x][y].simbolo = '#'){
            //si el camino se topa con un muro mueve ese muro en la misma fila
            //debe revisar que no sea parte de un camino y que no sea un muro
            int mover = rand()%N;
            while(casilla[mover][y].simbolo == '#' || casilla[mover][y].esCamino){  mover = rand()%N; }

            casilla[mover][y].simbolo = '#';//se mueve la casilla
        }//si la casilla es un muro
        casilla[x][y].simbolo = '.';

            int decision = rand()%10;

        if(x == fx){
            if(y>fy){ y--; }
            else{ y++; }
        }
        else{
            if(decision<=q_abajo && x<N-1){ x++; }//hacia abajo
            else if(decision<=q_der && y<N){ y++; }//preferiblemente a la derecha
            else if(y>0){ y--; }// a la izquierda
            else if(x<N-1){ x++; }// si se llega a un limite mejor bajar
        }

    }//despeja camino
}
```

```
--------:LABERINTO EVOLUCIONADO:--------
..#.#.###..#..#.#..##.#.##.............
.....###.#.#.###....#####.#..###...#####
...........#.#..##..####..#.#..##..###.
...##....#...##....######.##..#.#...##..
.#.###.###.....##.##.#..##..##...##...#
#.....####.#..###..###....##.##.....#.
.....#.#.########..#######.#.#.##..#..#.
.....########.#.#.#.#...#..#.....##..#.#
.#.......##.#..##################.###....
#..#...###..##...#.##..#..#.#.....##...#
#...#.....##.##.#####...#..##.#.#....###
.#.......#.#...##########.#.....#....#
#.##.##...###.##.##..#.##.#.#...###.#...
..#...#...#..####...#..#..#.#.##...#.##.
###.##......##.#.##..##.##.###..........
.##....#.....#...#.....#.#.#...##..#.##.
.#.#.##.#....#.#..##...##.#..#.#.##.#.#..
.##.#....#...#..###.#....#..#..#.#..#...
##.####.#.....#...##.#....##.....####.
..####..###.#.....###.##...#...#.#..#...
.#.##...#.#.....###.#..#.....#..##...#
..#...#..#.#.####..###..#.###.....#.#...
.##.#####.#...##..###.#...#.#.....######...
.#.....#..#....#.....#.##...#.#....#.#.
.###..####.#.##..#...#.............#.#
#.##..##.###.......#.###..##.#..#..##..
#...#..#.#.###.....#.#.##..#..###..#..
.#....#..#.#.##.###..#.#.#....#..#..#.#
######...#...#...##.....##....#..#.....#
#.##.#.#..####..#....##.#.#.....##....#.
.###.#.##.##...#...#.##.##...##......##
....#.##.#.##.####.#....##.......#..##.#
#.####.###....#####.....##...##..#....#
#####.#..####.###.#.##..###..#..#.....#
..#..#..#.#####.#####.....#..#.###.#..#.
#.#.#.##.#####..####.##....#.....#....#
#..#...#####.#...#..###.#.##..#..##.#..
.......###...#..##..#.#....#.#...###...
.##.#....#.###.#..#####.....##.....#...
.#####f.......................#.###.##
```

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.

```
----------:LABERINTO SOLUCION:----------
..#.#.###..#..#.#..##.#.##.............
a....###.#.#.###....#####.#..###...#####
a...........#.#..##..####..#.#..##..###.
aaa##....#...##....######.##..#.#...##..
.#a###.###......##.##.#..##..##...##...#
#.a....####.#..###..###.....##.##.....#.
..a..#.#.########..#######.#.#.##..#..#.
..a..########.#.#.#.#...#..#.....##..#.#
.#a......##.#..###################.###....
#.a#...###..##...#.##..#..#.#.....##...#
#.a.#.....##.##.#####...#..##.#.#....###
.#aaaaaa..#.#...###########.#.....#....#
#.##.##a..###.##.##..#.##.#.#...###.#...
..#...#a..#..####...#..#..#.#.##...#.##.
###.##.aa...##.#.##..##.##.###..........
.##....#aa...#...#.....#.#.#...##..#.##.
.#.#.##.#aa...#.#..##...##.#..#.##.#.#..
.##.#....#a..#..###.#....#..#..#.#..#...
##.####.#.aa...#...##.#....##......####.
..####..###a#.....###.##...#...#.#..#...
.#.##...#.#aa.....###.#..#.....#..##...#
..#...#..#.#a####..###..#.###.....#.#...
.##.#####.#.a.##..###.#...#.#...#####...
.#......#..#a...#.....#.##...#.#....#.#.
.###..####.#a##..#....#.............#.#
#.##..##.###aaa.....#.###..##.#..#..##..
#...#..#.#.###aaaaaa.#.#.##..#..###..#..
.#....#..#.#.##.###a.#.#.#....#..#..#..#
######...#...#...##a....##....#..#.....#
#.##.#.#..####..#..aa##.#.#.....##.....#.
.###.#.##.##...#...#a##.##...##.......##
....#.##.#.##.####.#a...##.......#..##.#
#.####.###....#####.aaa..##...##..#....#
#####.#..####.###.#.##a.###..#..#......#
..#..#..#.#####.#####.aa..#..#.###.#..#.
#.#.#.##.#####..####.##aa..#.....#....#
#..#...#####.#...#...###a#.##..#..##.#..
........###...#..##..#.#a...#.#...###...
.##.#....#.###.#..#####.a...##......#...
.#####faaaaaaaaaaaaaaaaa.....#.###.##
```

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