
GENERACIÓN DE LABERINTO CON ALGORITMO DE WILSON

REPOSITORIO: https://github.com/alexra99/SI_C1-8.git

1. ELECCIÓN DE LENGUAJE DE PROGRAMACIÓN

Para la resolución de este problema hemos optado por usar el lenguaje de programación Python. Tras haber estado investigando y barajar otras opciones como Java, definitivamente escogimos Python por las siguientes razones:

- **Sencillez:** se puede conseguir lo mismo que en otros lenguajes como Java en muchas menos líneas de código.
- **Documentación:** hemos encontrado mucha más documentación sobre este tema y ejemplos en los que apoyarnos escritos en Python.
- **Nuevas funciones:** nunca habíamos trabajado con ficheros json, sin embargo hemos encontrado una forma muy fácil y sencilla de trabajar con ellos usando Python.
- **Ampliar conocimientos:** hasta ahora, la mayoría de las prácticas de las demás asignaturas las habíamos realizado en Java, excepto en redes 2 dónde ya empezamos a dar los primeros en lenguaje Python y nos dio muy buenas sensaciones. Por ello queremos seguir ampliando conocimientos en este lenguaje y consideramos que el hecho de realizar esta práctica en Python nos ayudará a conseguirlo.

2. ESTRUCTURA GENERAL DEL PROGRAMA

Nuestro programa consta de tres clases:

- **Cell.py:** en esta clase definimos las propiedades y métodos relacionados con las celdas. Por ejemplo, métodos para saber si una celda esta unida a otra, crear uniones entre celdas, deshacerlas, etc.
- **Grid.py:** en esta clase se tratan aspectos relacionados con el tablero del laberinto, como por ejemplo: generar el tablero, recorrerlo, seleccionar posiciones aleatorias en él. Además, en esta clase tenemos los métodos para pintar y guardar la imagen del laberinto, y para generar el fichero .json.
- **Wilson.py:** en esta clase se realizan las operaciones, es decir, haciendo uso de los métodos y propiedades de las clases cell y grid se realiza el algoritmo de Wilson para generar el laberinto.

3. ASPECTOS IMPORTANTES RELACIONADOS CON EL CÓDIGO

La implementación del algoritmo de Wilson no ha sido muy complicado, sin embargo, la parte relacionado con pintar el laberinto, y generar el json nos ha costado un poco mas ya que nunca había trabajado con algo parecido.

PINTAR EL LABERINTO Y GENERAR IMAGEN

Para ello hemos utilizado la librería de Python PIL (Python Imaging Library) en concreto los módulos image e imageDraw. La idea general del algoritmo es recorrer el tablero indicar donde se empieza a dibujar en el método draw.line e ir comprobando si la celda actual esta unida a otra, es decir, si hay pared o no.

```
from PIL import Image, ImageDraw
from json import dump
```

```
def save_image(self):
    """Dibujar el laberinto"""
    SIZE_LINE = 20
    im = Image.new('RGBA', ((self.columns + 4) * SIZE_LINE, (self.rows + 4) * SIZE_LINE), (255, 255, 255, 255))
    draw = ImageDraw.Draw(im)

    for r in range(self.rows):
        for column in range(self.columns):
            """Va comprobando donde tiene que empezar a dibujar y si tiene pared en alguna coordenada(N,S,E,O)"""
            startx, starty = (column + 2) * SIZE_LINE, (r + 2) * SIZE_LINE
            if not self.grid[r][column].isLinked(self.grid[r][column].cellNorth):
                draw.line((startx, starty, startx + SIZE_LINE, starty), fill=128)
            if not self.grid[r][column].isLinked(self.grid[r][column].cellEast):
                draw.line((startx + SIZE_LINE, starty, startx + SIZE_LINE, starty + SIZE_LINE), fill=128)
            if not self.grid[r][column].isLinked(self.grid[r][column].cellSouth):
                draw.line((startx, starty + SIZE_LINE, startx + SIZE_LINE, starty + SIZE_LINE), fill=128)
            if not self.grid[r][column].isLinked(self.grid[r][column].cellWest):
                draw.line((startx, starty, startx, starty + SIZE_LINE), fill=128)

    im.save(f'Lab_{self.rows}_{self.columns}.png')
```

GENERAR JSON

Para generar el json hemos utilizado al librería json de Python y le hemos dado el formato que se pedía. Para saber los vecinos de cada celda recorreremos el tablero y en cada celda con el método islinked() vemos si tienen celda unida en el norte, sur, este y oeste, esto indicará si hay vecino en esa coordenada o no.

```
def save_json(self):
    """Generar el json con el formato dado"""
    output = {
        "rows": 4,
        "cols": 4,
        "max_n": 4,
        "mov": [[-1, 0], [0, 1], [1, 0], [0, -1]],
        "id_mov": ["N", "E", "S", "O"],
        "cells": {}
    }

    for r in range(self.rows):
        for column in range(self.columns):
            key_cell = f'({r}, {column})'
            output["cells"][key_cell] = {
                "value": 0,
                "neighbors": [self.grid[r][column].isLinked(self.grid[r][column].cellNorth),
                             self.grid[r][column].isLinked(self.grid[r][column].cellEast),
                             self.grid[r][column].isLinked(self.grid[r][column].cellSouth),
                             self.grid[r][column].isLinked(self.grid[r][column].cellWest)]
            }

    with open(f'Lab_{self.rows}_{self.columns}.json', 'w') as outfile:
        dump(output, outfile)
```

LEER JSON

Para leer el json en el constructor del *grid* hemos puesto un *else* que, en caso de introducir un fichero se crea el *grid* correspondiente con los datos correspondientes. Para crear el resto del laberinto se van leyendo los vecinos que pueda tener una celda y se crean las uniones con el método `link()`

```
else:
    with open(filename) as fdata:
        jsondata = load(fdata)
    self.rows = jsondata['rows']
    self.columns = jsondata['cols']
    self.grid = self.prepare_grid()
    for row in self.each_row():
        for cell in row:
            key = f'{cell}'
            row, col = cell.row, cell.column
            cell.cellNorth = self[row - 1, col]
            cell.cellSouth = self[row + 1, col]
            cell.cellWest = self[row, col - 1]
            cell.cellEast = self[row, col + 1]

            if jsondata['cells'][key]['neighbors'][0]:
                cell.link(self[row - 1, col])
            if jsondata['cells'][key]['neighbors'][1]:
                cell.link(self[row, col + 1])
            if jsondata['cells'][key]['neighbors'][2]:
                cell.link(self[row + 1, col])
            if jsondata['cells'][key]['neighbors'][3]:
                cell.link(self[row, col - 1])
```

Get_sucessor_node

Nos devuelve una lista con los nodos generados y con todos sus atributos asignados. Para ellos se recorre la lista de sucesores y se le van asignando los valores de sus atributos correspondientes dependiendo de la estrategia elegida ya que por ejemplo, el "value" de cada nodo varía dependiendo de esta. Una vez que creados se añaden a lista para poder usarlos en la solución final.

```
def get_sucessor_nodes(current_node, g, list_sucessors, all_sucessor, strategy, end):
    list_nodes = []
    value = 0
    for suceso in list_sucessors:
        all_sucessor.append(suceso)
        ID = len(all_sucessor)
        cost = current_node.cost + g[suceso[1].get_tuple()].value + 1
        state = suceso[1]
        father_id = current_node.ID
        depth = current_node.depth + 1
        action = suceso[0]
        h = abs(suceso[1].row - end[0]) + abs(suceso[1].column - end[1])
        node = Node(ID, cost, state, father_id, action, depth, h, value)
        if strategy == 1:
            value = depth
        elif strategy == 2:
            value = 1.0/(depth+1)
        elif strategy == 3:
            value = cost
        elif strategy == 4:
            value = abs(suceso[1].row - end[0]) + abs(suceso[1].column - end[1])
        elif strategy == 5:
            value = cost + h
        node.value = value
        #output = f'[{node.ID}]:({node.cost},{node.state},{node.father_id},{node.action},{node.depth},{node.h},{Decimal(node.value)})'
        #print(f"* {output}")
        list_nodes.append(node)
    return list_nodes
```

Expandir

Es el método que se usa para expandir la frontera, va recorriendo la lista de sucesores buscando nodos sucesores sin visitar, los añade a la frontera y los marca como visitados.

```
def expandir(current_node, g, visited, expanded, frontier, all_sucessor, strategy, end):
    list_sucessores = get_sucessor_nodes(current_node, g, current_node.state.neighbors_format(), all_sucessor, strategy, end)
    if strategy == 2:
        for suceso in list_sucessores:
            if not visited[f"{suceso.state}"]:
                push_frontier(frontier, suceso)
                visited[f"{suceso.state}"] = True
            elif not expanded[f"{suceso.state}"]:
                frontier.remove_by_pos(suceso.state.get_tuple())
                push_frontier(frontier, suceso)
                visited[f"{suceso.state}"] = True
    else:
        for suceso in list_sucessores:
            if not visited[f"{suceso.state}"]:
                push_frontier(frontier, suceso)
                visited[f"{suceso.state}"] = True
```

Push_frontier

Con este método controlamos los nodos que entran en la frontera y los ordenamos para a la hora de sacar la solución esta venga dada en el orden correcto. Para la ordenación nos servimos de cuatro criterios. El primero es comparar el campo "value". Si este campo es igual el siguiente criterio es mirar el número de fila. A continuación en el caso de coincidir se mira el número de columna. Por último si este criterio también coincidiera se ordenaría por menor ID.

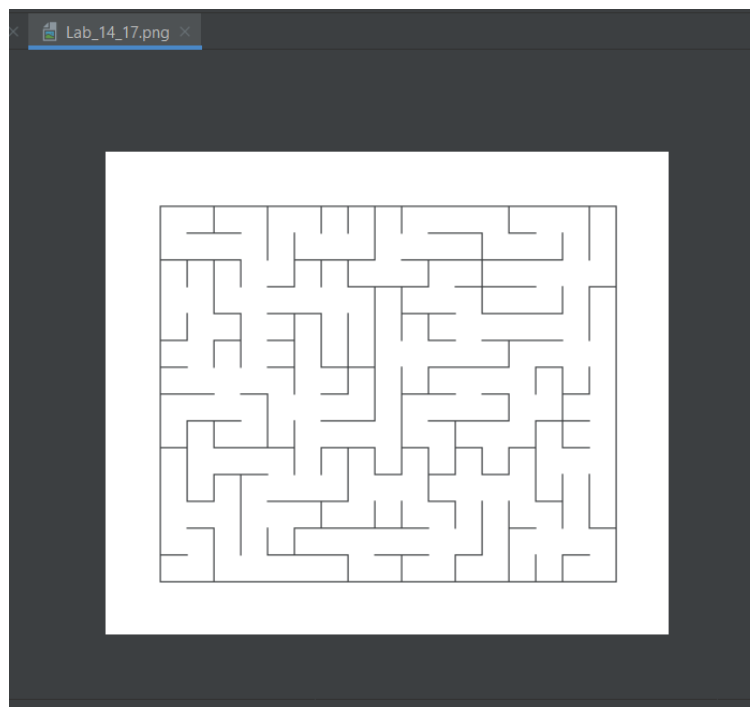
```
def push_frontier(frontier, node):
    if (frontier.is_empty()):
        frontier.push(node)
    else:
        x = frontier.get_size() - 1
        pos = frontier.items[x].state.get_tuple()[0]
        while x >= 0:
            if ((frontier.get_value(x) < node.value)):
                frontier.insert(x+1, node)
                break
            elif(frontier.get_value(x) == node.value):
                if(frontier.items[x].state.get_tuple()[0] < node.state.get_tuple()[0]):
                    frontier.insert(x+1, node)
                    break
                else:
                    if(frontier.items[x].state.get_tuple()[0] == node.state.get_tuple()[0]):
                        if(frontier.items[x].state.get_tuple()[1] < node.state.get_tuple()[1]):
                            frontier.insert(x+1, node)
                            break
                    elif(frontier.items[x].state.get_tuple()[1] == node.state.get_tuple()[1]):
                        if (frontier.items(x).state.ID < node.ID):
                            frontier.insert(x+1, node)
                            break
            if (x==0):
                frontier.insert(0, node)
        x = x-1
```

4. PRUEBAS

Hemos realizado varias pruebas para comprobar si el resultado era el esperado, al principio tuvimos problemas a la hora de centrar el laberinto en la hoja, ya que cuando por ejemplo un laberinto era muy ancho no se mostraba por completo en la hoja. Este problema lo resolvimos haciendo que la hoja se orientara según el número de filas y columnas.

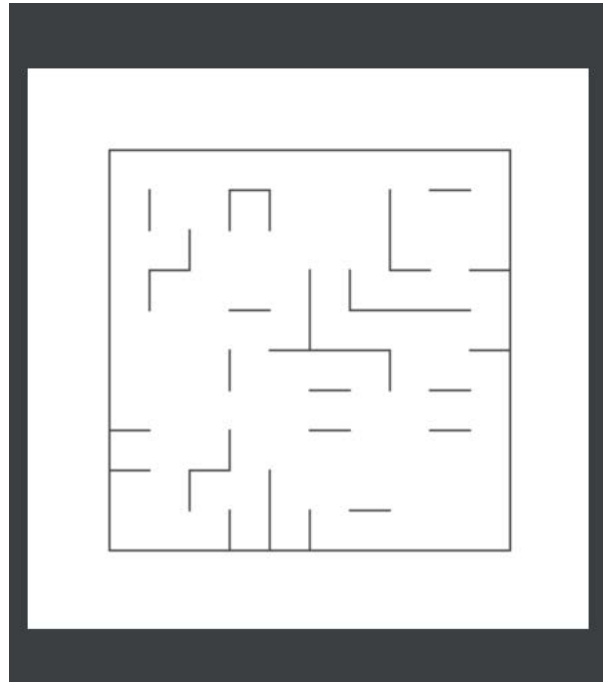
```
SIZE_LINE = 20
im = Image.new('RGBA', ((self.columns + 4) * SIZE_LINE, (self.rows + 4) * SIZE_LINE), (255, 255, 255, 255))
draw = ImageDraw.Draw(im)
```

En cuanto al json no tuvimos ningún problema considerable y actualmente podemos generar imágenes y archivos json como los que se pedían, aquí un ejemplo realizado con nuestro programa:



```
{
  "rows": 4, "cols": 4, "max_n": 4, "mov": [[-1, 0], [0, 1], [1, 0], [0, -1]], "id_mov": ["N", "E", "S", "O"],
  "cells": {
    "(0, 0)": {"value": 0, "neighbors": [false, true, true, false]},
    "(0, 1)": {"value": 0, "neighbors": [false, true, false, false]},
    "(0, 2)": {"value": 0, "neighbors": [false, true, false, false]},
    "(0, 3)": {"value": 0, "neighbors": [false, false, true, true]},
    "(0, 4)": {"value": 0, "neighbors": [false, true, true, false]},
    "(0, 5)": {"value": 0, "neighbors": [false, false, true, true]},
    "(0, 6)": {"value": 0, "neighbors": [false, true, true, false]},
    "(0, 7)": {"value": 0, "neighbors": [false, false, true, false]},
    "(0, 8)": {"value": 0, "neighbors": [false, false, true, false]},
    "(0, 9)": {"value": 0, "neighbors": [false, true, false, true]},
    "(0, 10)": {"value": 0, "neighbors": [false, true, false, true]},
    "(0, 11)": {"value": 0, "neighbors": [false, true, false, true]},
    "(0, 12)": {"value": 0, "neighbors": [false, false, true, true]},
    "(0, 13)": {"value": 0, "neighbors": [false, true, false, false]},
    "(0, 14)": {"value": 0, "neighbors": [false, true, true, true]},
    "(0, 15)": {"value": 0, "neighbors": [false, false, true, true]},
    "(0, 16)": {"value": 0, "neighbors": [false, true, false, false]},
    "(1, 0)": {"value": 0, "neighbors": [true, true, false, true]},
    "(1, 1)": {"value": 0, "neighbors": [false, true, false, true]},
    "(1, 2)": {"value": 0, "neighbors": [false, true, false, true]},
    "(1, 3)": {"value": 0, "neighbors": [true, true, true, true]},
    "(1, 4)": {"value": 0, "neighbors": [true, false, true, false]},
    "(1, 5)": {"value": 0, "neighbors": [true, true, false, false]},
    "(1, 6)": {"value": 0, "neighbors": [true, true, false, true]},
    "(1, 7)": {"value": 0, "neighbors": [true, false, false, true]},
    "(1, 8)": {"value": 0, "neighbors": [true, true, true, false]},
    "(1, 9)": {"value": 0, "neighbors": [true, true, false, true]},
    "(1, 10)": {"value": 0, "neighbors": [false, true, false, true]},
    "(1, 11)": {"value": 0, "neighbors": [false, false, false, true]},
    "(1, 12)": {"value": 0, "neighbors": [true, true, false, false]},
    "(1, 13)": {"value": 0, "neighbors": [false, true, false, true]},
    "(1, 14)": {"value": 0, "neighbors": [true, false, true, false]},
    "(1, 15)": {"value": 0, "neighbors": [true, false, true, false]},
    "(1, 16)": {"value": 0, "neighbors": [true, false, true, false]},
    "(2, 0)": {"value": 0, "neighbors": [false, false, true, false]},
    "(2, 1)": {"value": 0, "neighbors": [false, false, true, false]},
    "(2, 2)": {"value": 0, "neighbors": [false, false, true, false]},
    "(2, 3)": {"value": 0, "neighbors": [true, true, true, false]},
    "(2, 4)": {"value": 0, "neighbors": [true, false, false, true]},
    "(2, 5)": {"value": 0, "neighbors": [false, false, true, false]},
    "(2, 6)": {"value": 0, "neighbors": [false, false, true, false]},
    "(2, 7)": {"value": 0, "neighbors": [false, true, false, true]},
    "(2, 8)": {"value": 0, "neighbors": [true, true, false, true]},
    "(2, 9)": {"value": 0, "neighbors": [false, true, false, true]},
    "(2, 10)": {"value": 0, "neighbors": [false, false, true, true]},
    "(2, 11)": {"value": 0, "neighbors": [false, false, true, true]},
    "(2, 12)": {"value": 0, "neighbors": [false, true, false, true]},
    "(2, 13)": {"value": 0, "neighbors": [false, true, false, true]},
    "(2, 14)": {"value": 0, "neighbors": [true, true, true, true]},
    "(2, 15)": {"value": 0, "neighbors": [true, true, true, true]},
    "(3, 0)": {"value": 0, "neighbors": [true, true, true, true]},
    "(3, 1)": {"value": 0, "neighbors": [true, false, true, true]},
    "(3, 2)": {"value": 0, "neighbors": [true, false, true, true]}
  }
}
```

Una vez implantado el metodo de leer los json hicimos pruebas para ver si se generaban correctamente y nos dimos cuenta de que no estaba del todo bien hecho ya que los laberintos salían a medio pintar. Es decir, al compararlos con lo que de verdad tenía que salir nos dimos cuenta de que eran lo mismo pero el nuestro estaba incompleto. Este es un ejemplo con "puzzle_10x10.json":



Se solucionó al darnos cuenta que faltaban unos parámetros al llamar al constructor.

Conforme a la comprobación de la funcionalidad del programa se han hecho una cantidad elevada de pruebas ya que hemos ido comprobando uno por uno que todos los resultados coincidiesen con los proporcionados por los profesores. Esto ha sido un trabajo costoso debido en principio a nuestra falta de conocimientos, cosa que se ha ido solucionando a lo largo del curso. Aún así tuvimos bastantes problemas a la hora de que todo saliera como se esperaba. Por ejemplo, la frontera y lo referente a su ordenación ha sido modificado con bastante frecuencia. Muchas veces también se llegó al resultado pero por caminos que no eran los optimos. Con todo ello al final se llegó a los resultados esperados como se muestra en la siguiente imagen donde se muestra la comparación de la salida de nuestro programa con un archivo dado por los profesores en una de las estrategias propuestas.

Introduzca opción en el rango [1-5]

3

```
[id][cost,state,father_id,action,depth,h,value]
[0](0,(0, 0),None,None,0,8,0)
[1](3,(1, 0),0,S,1,7,3)
[4](4,(2, 0),1,S,2,6,4)
[6](5,(2, 1),4,E,3,5,5)
[13](6,(2, 2),6,E,4,4,6)
[20](7,(2, 3),13,E,5,3,7)
[27](9,(3, 3),20,S,6,2,9)
[47](10,(4, 3),27,S,7,1,10)
[60](14,(4, 4),47,E,8,0,14)
```

sol_5x5_UNIFORM.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
[id][cost,state,father_id,action,depth,h,value]
[0](0,(0, 0),None,None,0,8,0)
[1](3,(1, 0),0,S,1,7,3)
[4](4,(2, 0),1,S,2,6,4)
[6](5,(2, 1),4,E,3,5,5)
[13](6,(2, 2),6,E,4,4,6)
[20](7,(2, 3),13,E,5,3,7)
[27](9,(3, 3),20,S,6,2,9)
[47](10,(4, 3),27,S,7,1,10)
[60](14,(4, 4),47,E,8,0,14)
```