TEC | Tecnológico de Costa Rica

Technological Institute of Costa Rica

IC 3002
*Algorithm analysis*

I Project

*(Labyrinth - The Desert of Confusion)*

Professor:

Eng. José Angel Campos Aguilar

Students

Elder León Pérez *2023166120*

Gerny Díaz Hall 2022172980

Delivery date:

30/04/2025

I Semester, 2025

## Introduction

This document will detail everything done by the students to reach the solution of the project, both their implementation methods, as well as the problems they had and how they solved it.

The main objective of this project is the implementation of a solution that uses the backtracking algorithm approach to solve the classic maze game.

The developed program allows the generation of random mazes of different sizes, finding the exit from a starting point defined or selected by the user, and saving and loading previously generated solutions. The resolution of the maze is visualized graphically, showing the path followed by the algorithm and the cells visited.

## Description of the problem

The central problem addressed by this project is the implementation of a solution for the classic maze game, where the challenge is to find a valid path from a defined starting point to an exit point through a network of paths and blockages. This problem is solved using the backtracking algorithms approach.

Key aspects of the problem to be solved include:

1. **Labyrinth Generation:** The need to create mazes dynamically and randomly, with different sizes and ensuring the existence of at least one valid path between the start and the exit.

2. **Labyrinth Resolution:** Develop a function capable of finding a solution to the labyrinth from a specific starting point or from any point selected by the user, applying the backtracking algorithm.

3. **Process Visualization:** Graphically show the path taken by the backtracking algorithm while searching for the solution, indicating the cells visited and highlighting the final path found.

4. **Solution Management:** Allow the user to save the solution found for a maze and be able to load it later, offering the possibility of visualizing how the algorithm solves the same maze from a different starting point than the original.

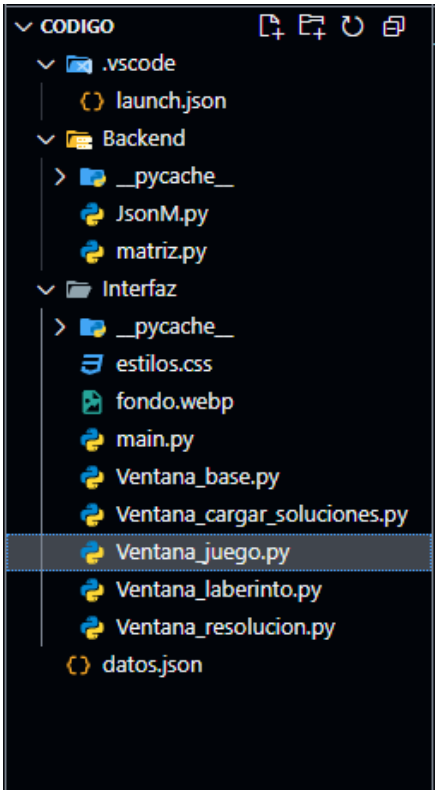
**Environments and downloadables:**

The project was developed in the python programming language (although for experimental reasons it was tried to do it in golang), in which PySide 6 was used

Before executing the project we need to run the following in the root folder of the project:

pip install pyside6

With this we would already have all the necessary files to execute the project (graphically)




Libraries

- random,
- time
- json
- pyside6

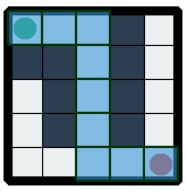
| **Objectives achieved** | State | Reason |
|---|---|---|

| Objectives | | |
|---|---|---|
| Generate mazes | Accomplished | |
| Solving laberintos | Accomplished | |
| Solve from any point | Accomplished | |
| Save Labyrinth | Accomplished | |
| Load Maze | Accomplished | |
| Show Paths | Accomplished | Not shown with animation |
| Backtrakig visualization | No Accomplished | Failed to connect backend logic to interface |
| Cells and colors | Accomplished | |
| Securing the Best Case | No Accomplished | Occasionally the algorithm does not find the best case because of the limitation to 20 paths |

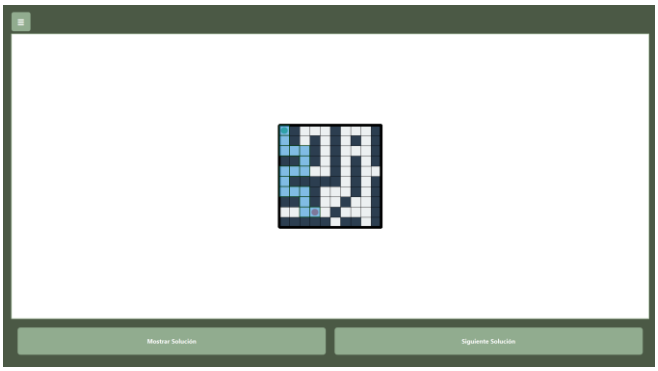
**Functionality Testing**

The solution for each maze size is then visualized

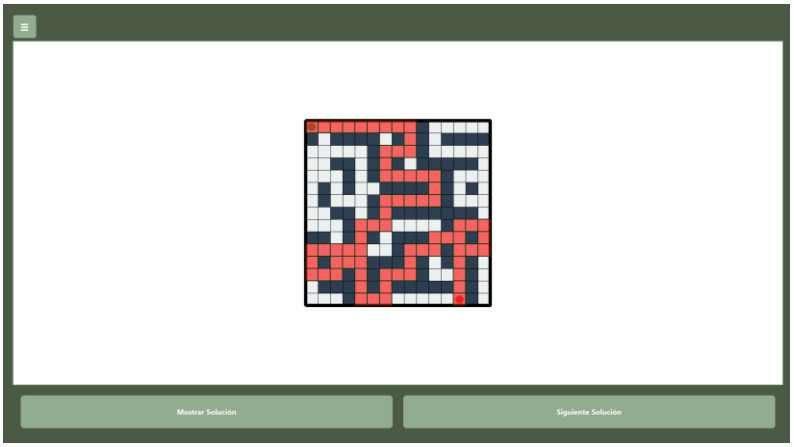
5*5



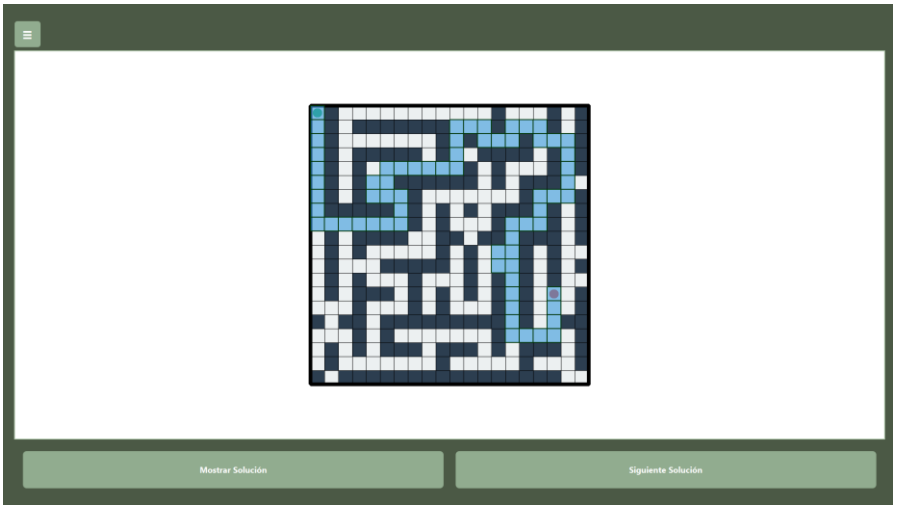
10*10



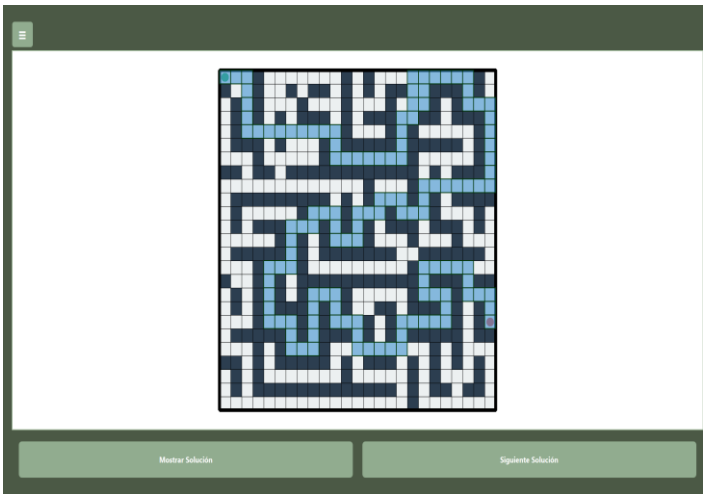

15*15 worst case



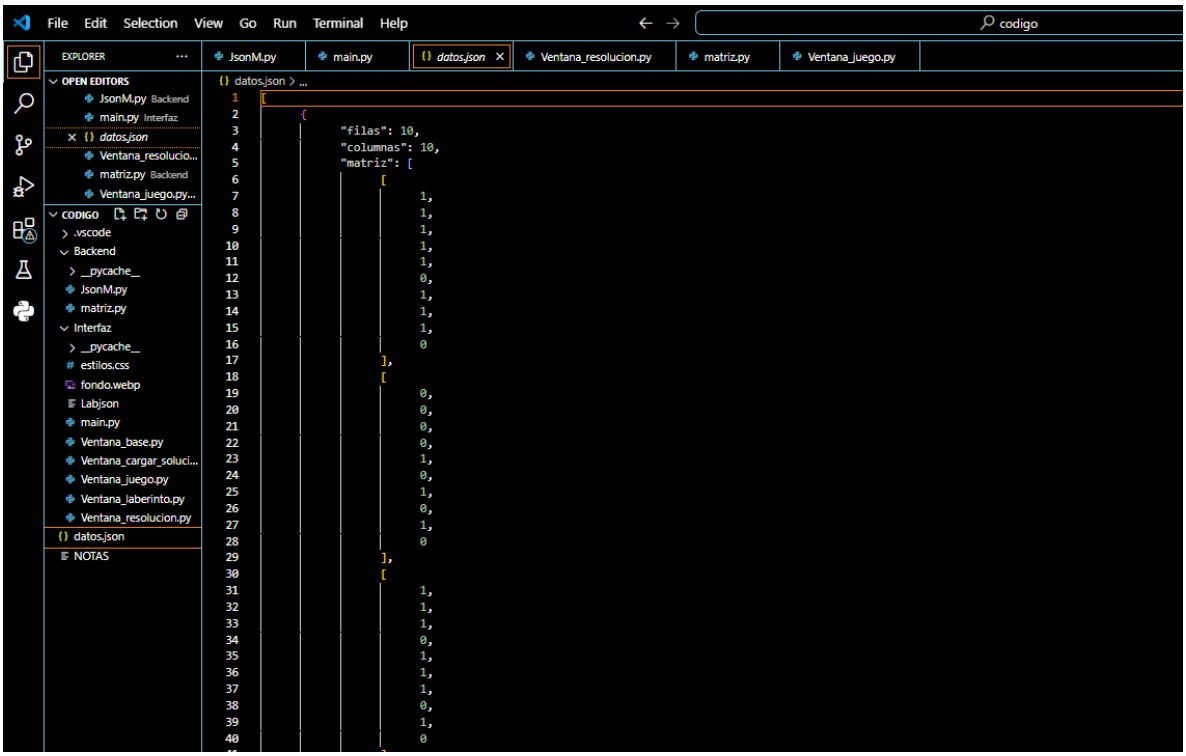

20*20



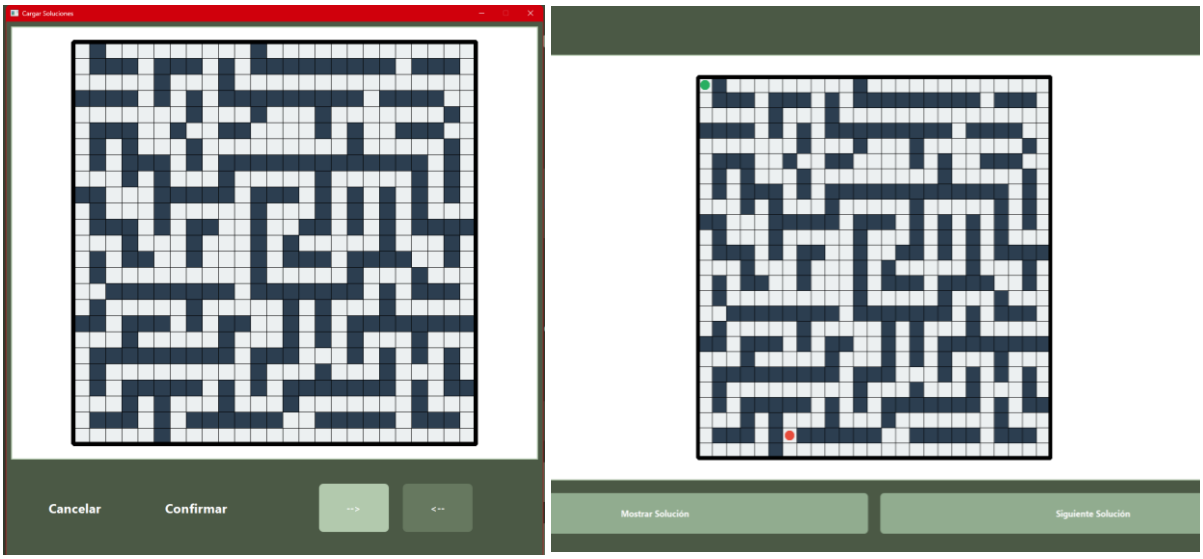

25*25




Saving Maps




Load saved maps




## Backtracking explained

The algorithm receives two tuples of positions, one initial and one final, it has two stops by force, one in a time limit and another by limit of path calculations, it makes a call to the second part of the code with parameters of start, the path and the points visited recursively travels the labyrinth exploring all the solutions without prioritizing any until the current point is equal at the end makes an inclusion From the path to the list of total paths like this until you travel all possible positions or reach the limits by force, quantity and time before finishing Arrange from lowest to largest by the length of the row the row of solutions for ease in interface

```
def solucionarMatriz(self, posicion0, posFinal0):
    fila_inicio, col_inicio = posicion0
    fila_final, col_final = posFinal0
    todos_los_caminos = []
    Time=20
    Inicio=time.time()
    if not (0 <= fila_inicio < self.filas and 0 <= col_inicio < self.columnas and
        0 <= fila_final < self.filas and 0 <= col_final < self.columnas):

        return False

    if self.datos[fila_inicio][col_inicio] != 1 or self.datos[fila_final][col_final] != 1:

        return False
```

```
""" Resolves the maze using dfs
Args:
    fila_inicio (int): starting row.
    col_inicio (int): starting column.
    camino (List[Tuple[int, int]]): current path.
    visitado (Set[Tuple[int, int]]): visited positions.

Returns:
   true if a solution is found, false otherwise.
Raises:
    false if the positions are ivalid or exceeds margin matrix
"""
def solucionarMatriz2(fila_inicio ,col_inicio,camino,visitado):
    if len(todos_los_caminos) >=20:
        return
    fila, columna = fila_inicio, col_inicio
    if time.time()-Inicio  > Time:
        return

    if (fila, columna) in visitado:
        return
    camino.append((fila, columna))
    visitado.add((fila, columna))
    if [fila, columna] == [fila_final, col_final]:
        todos_los_caminos.append(camino.copy())
    else:
        if columna + 1 < self.columnas and self.datos[fila][columna + 1] == 1 :
            solucionarMatriz2(fila,columna+1, camino, visitado)
        if columna - 1 >= 0 and self.datos[fila][columna - 1] == 1 :
            solucionarMatriz2(fila,columna-1, camino, visitado)
        if fila + 1 < self.filas and self.datos[fila + 1][columna] == 1   :
            solucionarMatriz2(fila+1,columna, camino, visitado)
        if fila - 1 >= 0 and self.datos[fila - 1][columna] == 1 :
            solucionarMatriz2(fila-1,columna, camino, visitado)


    camino.pop()
    visitado.remove((fila, columna))
solucionarMatriz2(fila_inicio, col_inicio, [], set())
if  len (todos_los_caminos)>0:
        self.soluciones = todos_los_caminos
        self.soluciones.sort(key=len)
        return True
else:

        return False
```