Costa Rica Institute of Technology

Algorithm Analysis

PROJECT #1

Professor:

Eng. Jose Angel Campos Aguilar

Students:

Brasly Villarebia Morales *2023105915*

Sidney Salazar Jimenez 2023282075

Jeanpooll Domenech Cerdas 2019255017

Submission Date:

April 30, 2025

Group 60

First Semester, 2025

Index

Introduction

Maze games represent an interesting mental challenge for the player, who must find the most efficient path from an entry point to an exit point, evaluating various possible alternatives. These mazes are not only interesting from a recreational perspective but also from a programming perspective, since solving them involves designing algorithms capable of finding solutions under certain conditions. In this way, maze games become an excellent opportunity to apply and reinforce basic programming knowledge by implementing algorithms that are fundamental to the development of more complex systems.

This project will address the decisions made by the team regarding the management of the matrix that represents the maze, the algorithms implemented for its validation and solution, as well as the choice of the graphical interface used to visually represent the path.

General Objective

Develop an application capable of generating, storing, validating, and solving mazes of different sizes using automated algorithms, implementing backtracking techniques and always guaranteeing the existence of a valid path between a starting point and an exit point, as well as its graphical display.

Specific Objectives

Design and implement a maze-generating algorithm that creates matrices of sizes between 5x5 and 25x25, always ensuring the existence of a valid and accessible path between the starting and exit points.

Develop a storage system that allows generating mazes to be saved and loaded in text files or JSON format, facilitating their reuse and subsequent analysis.

Implement a backtracking algorithm that automatically solves the generated mazes, allowing for a clear view of the path from start to exit.

Versions of Libraries and Compilers use

Python 3.13.2 64-bit and Anaconda

PyQt 5.15.11

User Manual

Instructions to Run the Program

1. **Install Python 3.13.2 (64-bit)**
   Before running the program, you need to have Python installed on your computer. Make sure you have Python 3.13.2 (64-bit) installed.
2. **Install PyQt 5.15.11**
   The project uses PyQt 5.15.11 for the graphical user interface, so you need to install this version of PyQt.
   - In a terminal or command prompt on your computer, run the command: "pip --version" to make sure you have pip (the Python package manager) installed. If you don't have it, follow the installation instructions on the official Python Website.
   - To install PyQt 5.15.11, open a terminal or command prompt and run the command: "pip install PyQt5==5.15.11". This will install the correct version of PyQt, and if you have another version, it will be updated to the required one.
3. **Clone the Repository**
   You must clone the repository where the project code is located. To do this, you need to have Git installed on your system. You can install it from Git. After installing it, you can run the command: "git --version" to confirm that the installation was successful.
   For cloning the repository, you should:
   - Open a terminal or command prompt in the location where you want to save the project and then run the command: "git clone https://github.com/Braslyvm/PY1_Analisis.git"
4. **Run the Program**
   To run the program, you need to:
   - Go to the "Code" folder and open the file "FrontEnd.py" in your integrated development environment (IDE).
   - Once the file is open, you can run it by pressing the “F5” key or by selecting the "Run" option from the menu.

Instructions to Play or Use the Program

At the start of the game, the player will be meet with log screen where they'll can choose two option:

Play: when this option is chosen, the player can choose the sizes of the labyrinth (5x5, 10x10, 15x15, 20x20, 25x25) will be offer two game modes:

Create Laberint Automatic:

In this game mode, the player will see a labyrinth generated by the system in the window, close to it will be three bottoms that allow the player to release the next actions:

Save labyrinth: This method will save the generated labyrinth so the player can access it later on in the option "Load labyrinth". The name of the labyrinth can be personalized.

Resolve labyrinth: In this mode, the labyrinth will be resolved automatically, next to the solution there are two bottoms that show all possible paths from the entry to the exit. All solutions are correct, but some of them are better than the others in terms of the shortest path. The system will first show the "best" solution, then the "medium" solution and lastly the "worst" solution, clicking the right-handed bottom will show the "medium" solutions until it gets to the "worst" solution, the left bottom will go back to the past solution.

Play: Given the current Labyrinth, the player will be able to move up, sides and down in the Labyrinth, however the player can't stand in a wall and will not be accessible to them. The goal of the game is to find the shortest path from the entry to the exit.

Create Labyrinth Personalized:

Change Entry: When the window pops up, it will ask the player to choose a starting point for the character. When chosen, the game will show two buttons that will let the player change the entry or validate the entry chosen, if the player chooses a wall as an entry the game will take it as not valid and will let the player choose another one.

View Solution: In this mode, the labyrinth will be resolved automatically, next to the solution there are two bottoms that show all possible paths from the entry to the exit. All solutions are correct, but some of them are better than the others in terms of the shortest path. The system will first show the "best" solution, then the "medium" solution and lastly the "worst" solution, clicking the right-handed bottom will show the "medium" solutions until it gets to the "worst" solution, the left bottom will go back to the past solution.

Save Labyrinth Personalized: This method will save the generated labyrinth so the player can access it later on in the option “Load labyrinth”. The name of the labyrinth can be personalized.

Backtracking: In this mode, the player will be able to see how the character uses the algorithm to find the closest path from the entry to the exit

Load labyrinth:

In this mode, the player can see in at the side of the screen, A list of all saved labyrinth, when clicked, the player will see a preview of the labyrinth and will be able to choose if they want to load it at: the Automatic Labyrinth, the Customised Labyrinth or Delete the labyrinth.

Comparison between a Brute-Force method and a optimization method

Both methods can arrive at the same result depending on the context, so they share commonalities. A brute-force method will find a way to reach a solution without considering efficiency or elegance, while an optimization method seeks the same solution, or even a better one, but using fewer resources and guided by a more intelligent approach

Comparative	Brute force	Optimization
Algorithms used on the project	- Backtracking and DFS	Quicksort
What does it do	- Try every cell possible from the starting position - Saves in a list every path that leads to the goal	Sorts paths by length to display to the player the worst, average, and best path
Execution speed	- As see in the project, the execution time of it was considerably slow due to this methodology	Very fast execution
Precisión	- Can find all possible routes, including the worst and longest ones	Return a list that has sorted in a accurate way the paths to the exit from the shortest to the largest
Computational cost	- High	Low

Conclusion

The project has successfully met the requested objectives by developing a functional application that generates, solves, and saves mazes of different sizes. The system allows the user to play in both automatic and custom modes, visualize the execution of the backtracking algorithm in real-time, and manage solutions effectively.

However, some areas need improvement: the backtracking algorithm sometimes struggles to find the best solution, the maze wall generation can result in mazes with too many paths and too few walls, which looks unnatural, and the order in which solutions are saved could be adjusted to ensure the best solution appears first.

Despite these minor issues, the project has successfully implemented the required features, and with some improvements, it can offer a more optimized experience.