**The Maze Game Report**

Programming with C++ 2023

Tilburg School of Humanities and Digital Sciences

Michael Pavlik Snr:2070814

30.11.2023

**Introduction**

Maze-solving algorithms are the baseline for any programming language, demonstrating problem-solving and decision-making skills. What makes the task interesting is the fact that it is computationally interesting, and it has plenty of real-world applications like GPS navigation, video games, intelligent traffic control, or robotics. In this project, I created a maze-solving agent in C++, that solves the maze without user input. With a predefined maze structure with a starting point, end point, hurdles and boosters in the maze using depth-first search, the agent finds a way out of the maze. The problem can be described as classic navigation and spatial orientation in the maze.

           In the following sections, I will describe general information about the project, code structure, used algorithm, and last but not least challenges, breakthroughs, and conclusion.

**Project Description**

The project consists of designing and implementing an autonomous agent that escapes a maze in C++ programming language. The maze is predefined, which means the layout of the starting point, end point, and walls is defined before the agent runs. If the goal or end point cannot be reached, the agent will go back to starting point. The agent is supposed to run and solve the maze without human intervention.

           There are several rules, which influence an agent’s behavior and environment. The vision field of the agent is 1 block to each side. The agent is not supposed to see behind the walls or corners. To make the game more interesting, hurdles and boosters can be found in the maze. “Goggles” or “speed potion” are boosters and “fog” or “slowpoke potion” are the hurdles. “Goggles” increase the vision field of the agent by 1 and “speed potion” increases the number of blocks in the agent’s step to a maximum of 3 per move. On the other hand, “fog” reduces the vision field by 1 to a minimum of 1, and “slowpoke potion” reduces the agent’s width of the step by 1 to a minimum of 1.

           Lastly, the code should display the game interface to the user. Indicating what the maze looks like, where are the walls, empty spaces, hurdles, boosters, and finally the position of the agent.

**Code Overview**

**The code I created for the task is used for an agent to solve the maze using a depth-first search algorithm, according to the project description in Visual Studio. This part of the report aims to inform a reader about the code's core components, structure, and design.**

**1.Libraries and Enum**

**Firstly, the code starts with the inclusion of standard C++ libraries, namely: “<iostream>”, “<vector>”, “<stack>” and “<fstream>”. This lays the groundwork for essential operations of the program together with the setting of the used namespace, in this case, std namespace. Secondly, the enumeration declarations follow. Declaring cell types which are displayed in the maze, specifically: “EMPTY”, “WALL”, “START”, “GOAL”, “GOGGLES”, “SPEED\_POTION”, “FOG”, “SLOWPOKE\_POTION” and “AGENT”. Next, declare all the possible directions in which the agent can move in the maze: “UP”, “DOWN”, “LEFT” and “RIGHT”.**

**2.Struct definition**

**Introducing a “Position” struct as a fundamental data structure that stores the row and column coordinates. This provides an abstract position of the elements within the maze.**

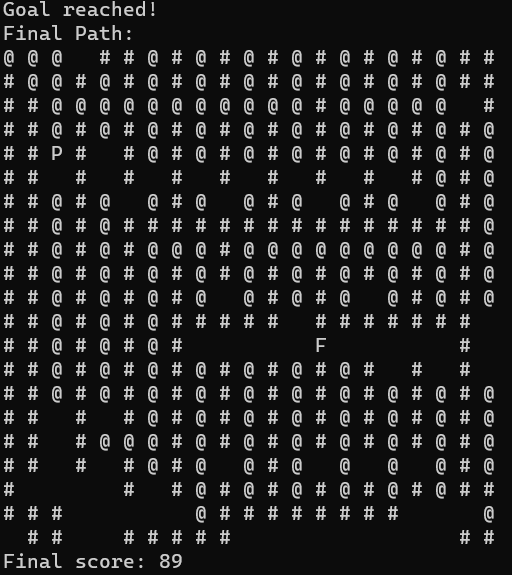
**3.The Maze Class**

**In the middle of the code is located a “Maze” class, which stores a maze-solving logic. Starting with defining the maze grid, visited cells, starting and end position, stack that keeps track of the path that the agent took, the initial perspective field of the agent, step width of the agent, direction, and score. Next follow the functions to check if the move is valid, to mark the spot as visited, and to display a maze and agent.**

**4. Boosters, Hurdles and Search Algorithm**

**Afterwards, the code contains functions to display the final path that was taken by the agent and handling special cases like encountering boosters and hurdles. Followed by handling error cases i.e. missing starting point or end point. In the second half of the code the depth-first search is defined with capability of updates and handling cases of boosters and hurdles.**

**5. Main Function**

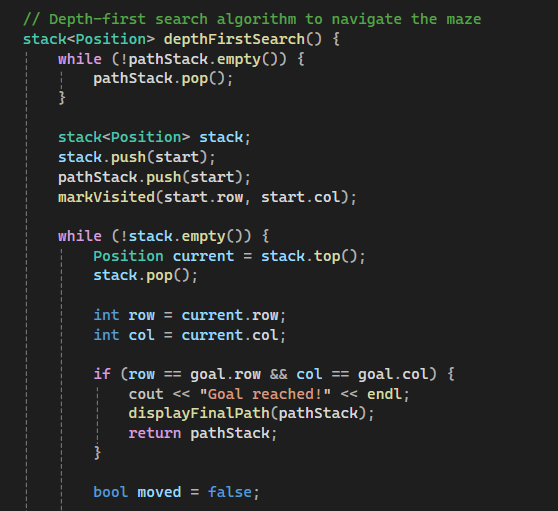
**Las but not least, the main function is called, where the maze display is initiated, and depth-first search of the given maze is executed. Lastly, the score and final path are displayed for examination.**

***Picture 1*** Final path of the agent displayed*.*

*(“#” represents walls and “@” represents agent)*

**Algorithm**

**As mentioned before, the search algorithm I chose to use is depth-first search. It is truly not the best choice for the maze solving task, but I was curious to find out about the performance of this search in the task.**

**Depth-first search algorithm as the name implies, travels deep into the graph in its search. The algorithm starts at a root node and from there it explores each branch in depth before backtracking. It uses stack to keep track of visited nodes and branches. Its ability to traverse deep into a graph structure allows for the exploration of intricate relationships, providing insights into complex systems (Sedgewick & Wayne, 2011). DFS can be applied to multiple tasks such as topological sorting, network routing etc.**

***Picture 2*** Implementation of the Depth-first search algorithm.

**Challenges, Breakthroughs and Conclusion**

When it comes to challenges faced during development of this project, I would like to first mention the right choice of the program. My first choice was program Eclipse for C++, which later resulted in frustration and demotivation because of meaningless errors flagged on comments etc. After I switched to Visual Studio the whole process seemed simpler and clearer. However, it is possible that it was my fault making mistakes in a set-up of the program in the beginning or simply Eclipse not being a right fit for me. Some challenges also occurred during the development of the code itself. Implementing a boosters and hurdles in a way that they actually affect the behavior of the agent was very challenging task for me. Furthermore, loading a maze from a separate text file raised some difficulties in a process. After a class discussion I have found out that the agent jumps walls while affected by booster. I tried to solve the problem but unfortunately, I failed.

All in all, during the project development I learned a lot about the language itself, the importance of choosing right tools for work and having people around that are willing to help in case problems occur. Moreover, I learned to prioritize efficiently since we have 4 projects with deadline in November and it was not easy to plan all the projects and working on my internship at the same time.

**References**

Sedgewick, R., Wayne, K. (2011). *Algorithms, 4th Edition.*. Addison-Wesley. ISBN: 978-0-321-57351-3

Special thanks to Zhivko Parapanov for helping me with my project.

**Appendix**

1. Obsah obrázku vzor, Symetrie, design, látka

   Popis byl vytvořen automatickyMaze structure

***Picture 3*** Design of the maze *where 0=empty space, 1=wall, 2=start, 3=goal,*

***4=goggles, 5=speed potion, 6=fog and 7=slowpoke potion.***

1. Github link for the code

https://github.com/CptC0de/Programming-with-Cpp-Project.git