SIT215 COMPUTATIONAL INTELLIGENCE

Assignment 1: Search for Agent Navigation

DANIEL MATAR, NEEL VISHALBHAI LODHIYA, AMAN BHARDWAJ, AND SAMUEL BOROUGH KAMAU

Contents

Introduction

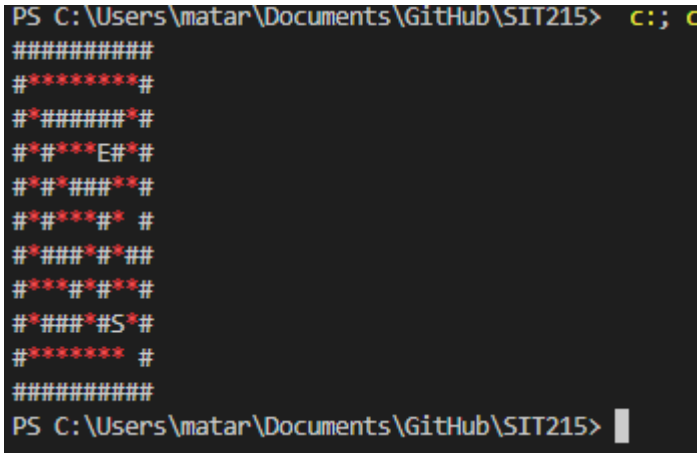
This is the first assignment task of SIT215 Computational Intelligence. This report details our findings in completing this task, what problems we faced, our results, and lessons learned.

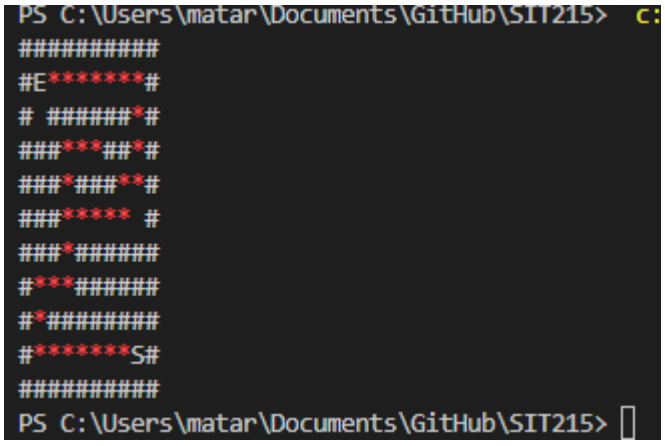
Problem Description

In this project, we will implement the Depth-First Search (DFS) algorithm to help a nonplayable character (NPC) navigate a maze. The maze is represented as a 2D grid, with walls represented as blocks and open paths represented as empty spaces. The NPC is represented by an ASCII character in the game, and can move in four directions (up, down, left, and right). Our task is to implement the DFS algorithm to find a path for the NPC to reach the endpoint in the maze. The NPC must navigate around walls to reach the endpoint. We will also implement a visualization of the NPC's movement in the maze.

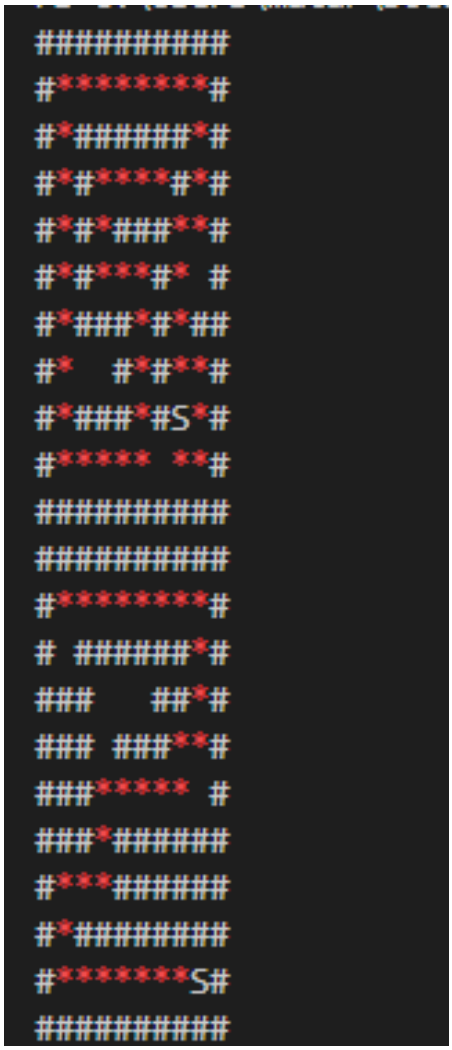
Results

Results of our solution show that DFS finds the path (Though it does not remove vertices that are not part of the solution afterwards). But it does not always find the shortest path to the endpoint.



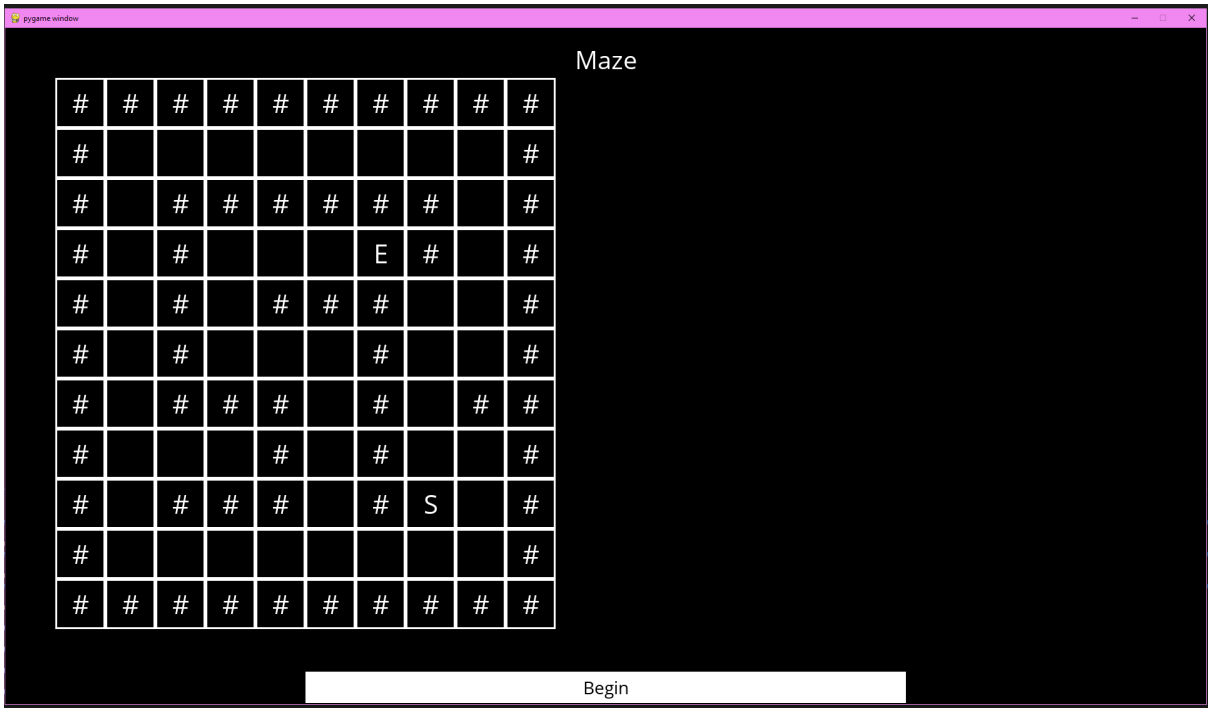


Though as mentioned our solution does not remove the vertices not a part of the path, which makes things hard to see which is the correct path. To remedy this, we simply checked to see if the next vertex with DFS returns true or false and set the vertex back to its original character if it was false.

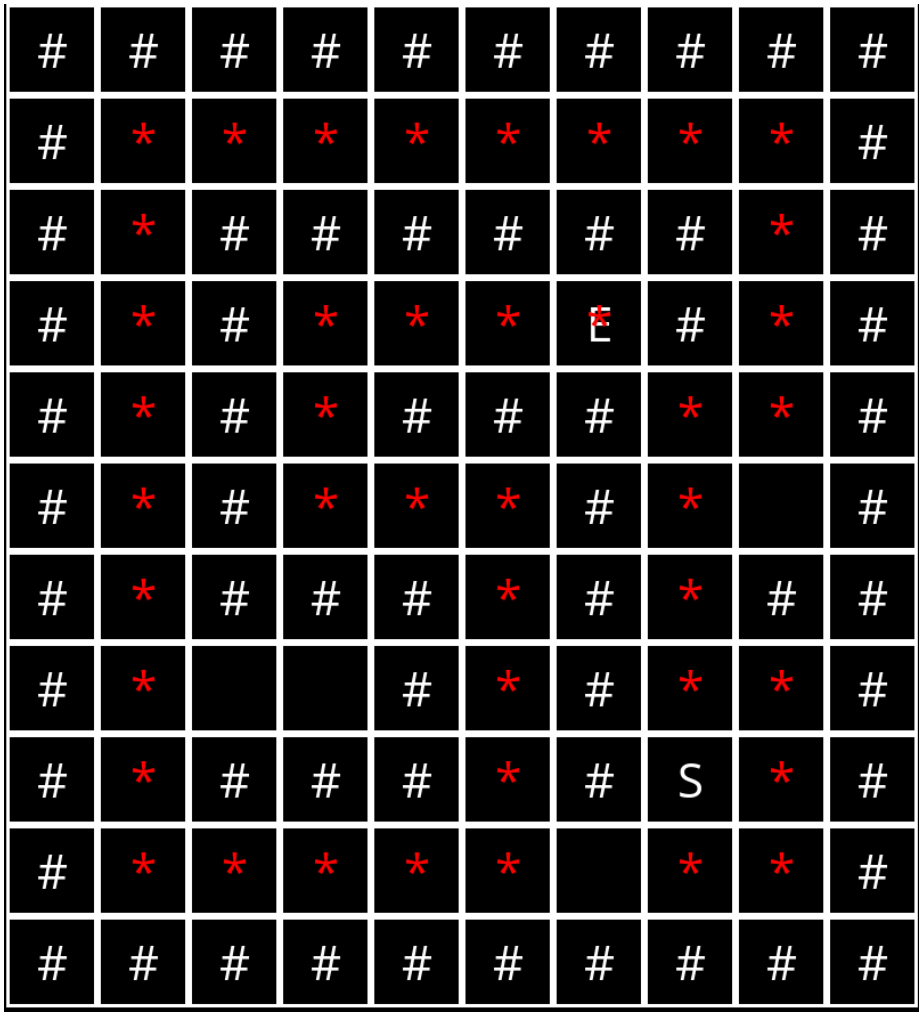


After doing so, the path is much clearer, and we can see the DFS algorithm at work. We also see that with different types of mazes the better the path chosen is.

Next is outputting the path in a visual manner by moving an 2d object through the path. As shown below.



And after it is done it looks like this:



Evaluation of our algorithm in terms of time taken and nodes explored

The bigger and more complex the longer it takes having a complexity of O(V+E), where V is the vertices and E is the edges.

Analysis

DFS is not picky in which way it chooses to go. If the vertex has not been visited and is in the bounds of the maze it will traverse as deep until it finds the end point. This is a problem for if we want the shortest path and computation wise as well.

In terms of speed, DFS is fast with a complexity of O(V+E), where V is vertices and E is edges. The same for BFS. But where DFS falls short is infinite loops, while BFS does not have this issue.

Our solution finds the path quite quickly and uses less memory but if as our maze becomes more intricate so becomes the possibility of loops that could confuse our algorithm.

Conclusions & Lessons Learned

DFS is quite simple and if we wish to improve on our solution we would be better off using BFS which is much better at finding the best/shortest path for these types of problems. We were able to find a path to the endpoint from different starting points, visualised this by showing the path in a different colour compared to the walls. Our visual display goes backwards (from the endpoint) because it finds the path before displaying it. This is something that in future if we had more time, we could implement a fix for but for demonstrations purpose we felt it was acceptable.

Acknowledgement of External Assistance (If Applicable)

We have used other unit material to get some information on DFS and BFS from SIT320 Advanced Algorithms (Full document included in GitHub), that was written by one of the students in this group (Daniel Matar) which is based off that unit's material. Also used PBL Task 2 Knights Tour as reference for our algorithm in this assignment. I had taken SIT215 before it was reworked and re-used some of my material.

References

Blades, A. (2020, March 08). *Solving Mazes with Depth-First Search*. Retrieved from Medium: https://medium.com/swlh/solving-mazes-with-depth-first-search-e315771317ae

Field, B. (n.d.). Retrieved April 14, 2021, from https://bradfieldcs.com/algos/graphs/knights-tour/

Geeks, G. F. (2020). *Geeks For Geeks*. Retrieved April 14, 2021, from https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/

Sam, S. (2018). *Tutorials Point*. Retrieved April 14, 2021, from https://www.tutorialspoint.com/The-Knight-s-tour-problem

Zaidi, N. (n.d.). *Deakin Sync Module 3a Graphs Part 1 - DFS.* Retrieved July 29, 2021, from https://d2l.deakin.edu.au/d2l/le/content/1031061/viewContent/5742250/View