Robot Mazes

Project I Checkpoint - Artificial Inteligence

Grupo 21
up201906086@up.pt Marcelo Henriques Couto
up201907361@up.pt Francisco Pinto de Oliveira

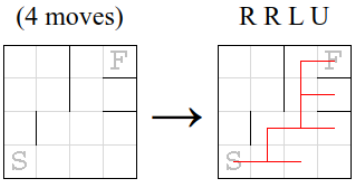
Faculdade de Engenharia da Universidade do Porto

April 25, 2022

Problem Description

Implement an Artificial Inteligence agent based on heuristic search methods to solve Robot Mazes puzzle.

- **Grid:** The puzzle is made up of a 5x5 grid, with a start and finish positions. Between each spot in the grid, there can be a wall.
- **Robot and Movement:** The robot is placed at the starting spot and can make a limited number of movements (Left, Right, Up or Down), depending on the puzzle. When there are no moves left, he will repeat the sequence chosen until he gets stuck or reaches the ending spot.



- **Objective:** The objective is to, for a given puzzle, decide the correct sequence of movements for the robot to loop from start to finish.

Problem Formulation as a Search Problem

State Representation: With N being the number of moves allowed, state S is an array of size N of the movements chosen.
Initial State: A random state. E.g. S = [Right,Up,Left,Down]
Objective Test: The cycle of movements of the given state, repeated indefinitely, always changes position after an iteration and eventually reaches the goal spot.
Operators:

Operator	Pre-Condition	Effect	Cost
ChangeMov(i,m)	$i >= 0 \wedge i < N \wedge S[i] \neq m$	$S[i] = m$	1

Table: Operators

With this approach, we consider every solution to be equally optimal, being the speed of the algorithm the desirable aspect to optimize.

Implementation

Language: Python
Environment: Anaconda
Dependencies: pygame, pygame_menu
Data Structures:

1. Map representing the positions of the walls in the maze - python *dictionary*
2. List representing the movements of the robot in sequence (state) - python *list*

Algorithms: Simulation of a given sequence of movements (state)

Approach

Heuristics

- **Compare with shortest path**
  1. Calculate shortest path from start to finish
  2. Simulate current solution
  3. Compare the two paths. The closest to the shortest path the better the solution
  4. **Evaluation functions:** given SP the shortest path from start to finish and P the path made by the simulation:
     - $length(SP) - length(SP.count(P))$, where $SP.count$ is the number of positions from SP that are in P
- **Manhattan Distance**
  1. Simulate the current solution
  2. Calculate Manhattan Distance from where the robot got stuck to the goal
  3. **Evaluation functions:** given f the function that gives the Manhattan Distance, S the state and M the maze
     - **MAN1** - $f(S)$ - non admissible
     - **MAN2** - $f(S)/(M.size \times 2)$ - admissible

Approach

Heuristics

- **Maximum Axis Distance**
  1. Simulate the current solution
  2. Calculate Maximum Axis Distance from where the robot got stuck to the goal ($max(distx_, dist_y)$)
  3. **Evaluation functions:** given f the function that gives the Maximum Axis Distance, S the state and M the maze
     - **MAD1** - $f(S)$ - non admissible
     - **MAD2** - $f(S)/M.size$ - admissible

The way we approached and interpreted the problem lead to a non-existance in comparison of solutions, as there is a fixed number of movements for each problem. For this matter, we chose to compare each solution simply through the speed of its attainance. Therefore, and because the closest solution to the root is not the only valid one, non admissible heuristics are acceptable.

Algorithms Implemented

- **Breadth-First Search (BFS)**
- **Depth-First Search (DFS)**
- **Iterative Deepening Search (IDS)**
- **Greedy Search**
- **A* algorithm**

As the cost of our operators is 1, **Uniform Cost Search** is equivalent to **Breadth-First Search**
We did not give any limit in depth for either **Depth-First Search** or **Iterative Deepening Search**, as there is always a solution and we do not allow for repeated nodes/states.

Results

		Maze 2 (4 moves)			Maze 8 (6 moves)		
Algorithm	**Heuristics**	**Time (ms)**	**Iterations**	**Depth**	**Time (ms)**	**Iterations**	**Depth**
BFS	-	27.48	143	4	937.86	3992	7
DFS	-	20.15	92	92	585.51	1868	1868
IDS	-	48.43	379	4	11006.86	80064	9
A*	MAN1	20.44	12	5	79.09	44	8
A*	MAN2	38.53	68	4	34.51	68	4
A*	Shortest Path	36.38	48	5	37.71	48	5
A*	MAD1	15.06	16	5	16.89	16	5
A*	MAD2	35.50	68	4	36.15	68	4
Greedy	MAN1	11.96	8	5	43.38	21	10
Greedy	Shortest Path	41.03	40	9	40.98	40	9
Greedy	MAD1	9.86	8	5	9.02	8	5

Note: Greedy algorithm was not worth running with admissible heuristics; Heuristics names are defined in the slides before.

Conclusions

From the different approaches analyzed we were able to draw some conclusions:

- Despite our heuristic functions not being, at our eyes, closely related to the problem and meaningful, they proved to greatly improve our algorithms' performance.
- The results show that our admissible heuristics made the **A* algorithm** find the 'ideal solution', if that would be the solution closest to the root node. However, the non admissible heuristics improved its speed greatly.
- In our interpretation of the problem, any valid solution is acceptable and the best solution is the one simply achieved the fastest. In this case, the **Greedy algorithm** was the best, as it yielded a solution in the least time and iterations.
- **IDS** was a particularly bad approach, as the optimal was not necessary.
- **DFS** as shown the same number of iterations as depth, as, without any limits, because a solution can always be reached through a certain branch, it will not backtrack ever, being effectively a brute-force approach.

Conclusions

The project was a success as we were able to

- create a representation of the problem as a search problem
- build algorithms capable of solving it and heuristics to aid them
- build an environment to test and showcase the algorithms
- analyze the results and understand the differences

Implementing the algorithms necessary to solve this problem has allowed us to deepen our knowledge and experience on search problems and algorithms.

References

- **Artificial Intelligence: A Modern Approach 3rd Edition:** Search algorithm implementation
- **Escape the Maze with A* Search Algorithm:** Help with heuristic discovery
- **Robot Mazes by Erich Friedman**: problem
- **Pygame**
- **Pygame menu**
- **Anaconda**
- Others:
  - **https://en.wikipedia.org/wiki/Admissible_heuristic**
  - **https://docs.python.org/3/library/heapq.html**
  - **https://wiki.python.org/moin/TimeComplexity**