Intro to AI Assignment 1 Report

Daniyar

October 2022

1 Algorithm Flow

To solve the shortest path finding problem, I used 2 algorithms: A* and Backtracking.
The detailed explanation of each line of code of the algorithms is in the .java file, so here I will only explain the main optimizations:

1) A*

Keep the list of open and closed cells, and store the parents of each cell to traverse the path once we have found it.

2) Backtracking

To optimize the backtracking algorithm, I introduced the current best path for each cell on the map, and once the path of the current cell is greater than path of the neighbour algorithm is going to visit, we do not make a call to the next recursion. Also, I added the max length of path it could be on the map based on tests.

2 Statistical Analysis

Based on 100000 maps generated, the following statistics were retrieved

---

```
A* with scenario 1:
Mean execution time: 0.021465 ms
Mode execution time: 0.02 ms
Median execution time: 0.017100 ms
Standard deviation execution time: 0.028738 ms
Wins: 936
Loses: 64
Wins percentage: 93.60
Loses percentage: 6.40

Backtracking with scenario 1:
Mean execution time: 26.657463 ms
Mode execution time: 0.00 ms
Median execution time: 25.659850 ms
```

```
Standard deviation execution time: 12.698561 ms
Wins: 936
Loses: 64
Wins percentage: 93.60
Loses percentage: 6.40


A* with scenario 2:
Mean execution time: 0.045881 ms
Mode execution time: 0.02 ms
Median execution time: 0.023400 ms
Standard deviation execution time: 0.563272 ms
Wins: 936
Loses: 64
Wins percentage: 93.60
Loses percentage: 6.40

Backtracking with scenario 2:
Mean execution time: 31.452412 ms
Mode execution time: 0.02 ms
Median execution time: 27.129570 ms
Standard deviation execution time: 13.128411 ms
Wins: 936
Loses: 64
Wins percentage: 93.60
Loses percentage: 6.40
```

As it could be seen, perception of scenario does not impact significantly on performance of the algorithms.
However, there is one significant problem I want to tell about.
In the statistics above, mean execution time of A* is around 0.03 ms and 30 ms for Backtracking. However, by analysing maps not in a loop, usually it is around 0.5-2 ms of time for A* and 100-200 ms for Backtracking.
I do not understand what might be the reason of this issue, because code for computing execution time, as well all other statistics methods is correct (you can check the Analyser class in my submission and make sure of this).
The only thing that comes to my mind is that Java does some runtime optimizations, or the garbage collector here is the case.
Also, you can always run those tests by yourself, because I left the code for their generation in the file.

3 PEAS Description

1) Agent Type - Jack Sparrow
2) Performance measure - The path to the Dead Man's Chest was found
3) Environment - Sea
4) Actuators - Take Rum casks, fire Rum casks to destroy The Kraken
5) Sensors - Perception of Jack

Properties of the environemnt - Partially Observable, Single Agent, Determenistic, Sequential, Static, Discrete, Known

4 Highlights

It was asked to include Graphical representation of maps that were impossible to solve. However, I did not find any unsolvable maps.
Instead, I will show some interesting (mostly because of its difficulty) maps that I want to highlight

1) Map 1

In this map, Jack has to check all enemies that might exist on the map.

```
Input: [0,0] [5,2] [2,2] [0,2] [0,3] [8,8]
Output:
Win
16
[0,0] [1,1] [2,0] [3,1] [4,0] [5,0] [6,0] [7,1] [7,2] [7,3] [6,4] [5,5] [4,4]
    [3,4] [2,5] [1,4] [0,3]
 ---------------------
|   0 1 2 3 4 5 6 7 8 |
| 0 * - R * - - - - - |
| 1 - * $ - * - - - - |
| 2 * $ K $ - * - - - |
| 3 - * $ - * - - - - |
| 4 * $ $ $ * - - - - |
| 5 * $ D $ - * - - - |
| 6 * $ $ $ * - - - - |
| 7 - * * * - - - - - |
| 8 - - - - - - - - T |
 ---------------------
0.9846999999999999 ms
```

2) Map 2)

This is the map with the largest best path among all maps I saw (this test was made by my

classmate)

```
Input: [0,0] [6,6] [7,1] [8,5] [8,0] [8,6]
Output:
Win
24
[0,0] [1,1] [2,2] [3,3] [4,4] [4,5] [4,6] [4,7] [5,8] [6,8] [7,8] [8,7] [8,6]
    [8,7] [7,8] [6,8] [5,8] [4,7] [3,6] [3,5] [4,4] [5,3] [6,2] [7,1] [8,0]
 ---------------------
|   0 1 2 3 4 5 6 7 8 |
| 0 * - - - - - - - - |
| 1 - * - - - - - - - |
| 2 - - * - - - - - - |
| 3 - - - * - * * - - |
| 4 - - - - * * * * - |
| 5 - - - * - $ $ $ * |
| 6 - $ * - - $ D $ * |
| 7 $ * $ - - $ $ $ * |
| 8 * $ - - - R * * - |
 ---------------------
1.9058 ms
```