German University in Cairo
الجامعة الألمانية بالقاهرة

CSEN 403: Concepts of Programming Languages

16th of June, 2022

Minesweeper Robot: Bonus Implementation

Authors

Team #99

Omar Hesham, 52-8724, T10

Nardy Mechael, 52-8695, T08

Omar Tamer, 52-11870, T15

Mohammed Khaled, 52-21246, T05

Introduction

This report is a continuation on the previous report of the main program implementation. This report will serve to highlight changes from the original implementation and explain the new algorithm developed in order to solve boards larger than 4x4 within reasonable resource and time constraints.

It is therefore advisable (although not entirely necessary to a large extent) that you read the main program report before reading this one.

Explanation

In order to be able to solve larger grids, A substantially more efficient algorithm had to be developed, which works as follows:

The idea is to obtain the nearest cell to the robot's current position, generate the sequence of moves required to reach that cell, and then recursively apply this technique until all cells have been visited.

The program will obtain all cells on the board and then apply the merge sort algorithm to sort them in ascending order of distance to the robot's location, that is, from closest to the robot's location to furthest.

The program will then apply the collect function (the implementation of which has been modified from the original) in order to visit all cells in increasing order of distance.

Important Modification

Due to the difference in algorithm used to find a solution this time, the MyState data type has been made into a non-recursive type, with its new definition being:

```
data MyState = S Cell [Cell] deriving (Show, Eq)
```

Implementation

Merge Sort

The following functions are the implementation of the Merge Sort algorithm required for the functionality of the program (divide, sort, and merge). We will not go into details about how merge sort works, as you have certainly become experts on the topic after correcting the midterm exam ;)

The aim of merge sort here is to sort the cells containing mines in ascending order of distance to the robot's cell.

manhattan :: Cell -> Cell -> Int

Takes as an input two cells and obtains the distance between them using the Manhattan distance algorithm. Any sorting within the program is done based on the distance obtained using this function.

get1stHalf :: [Cell] -> [Cell]

Takes as an input a list and returns the first half of that list.

get2ndHalf :: [Cell] -> [Cell]

Takes as an input a list and returns the second half of that list.

merge :: Cell -> [Cell] -> [Cell] -> [Cell]

The function takes as an input a cell representing the robot's location as well as the two sorted lists to be merged (which represent lists of cells containing mines.) The function then compares each mine cell's distance to the robot's cell and performs the merging on that basis.

mergeSort :: Cell -> [Cell] -> [Cell]

Applies the merge sort algorithm on the input list using the functions defined above. Similar to the merge function above, this function also takes a Cell as an input representing the robot's cell as it sorts the mine cells based on the distance to that cell.

Main Functions

The following functions are the functions concerned with obtaining the solution.

sortMines :: Cell -> [Cell] -> [Cell]

Takes as an input a cell representing the current position of the robot and a list of cells representing the positions of the mines and returns a sorted list of mines in ascending order of their proximity to the robot.

collect :: MyState -> [String]

Takes as an input a state S curr [mines] and generates the sequence of moves required to visit and collect all mines in order from closest to furthest from curr

getMyWay :: Cell -> Cell -> [String]

Takes as an input two cells the first representing the robot's cell and the second representing a cell containing a mine and returns the sequence of moves required to go from the robot's cell to the mine cell and collect it.

solve :: Cell -> [Cell] -> [String]

Takes as an input a cell representing the starting position of the robot, a list of cells representing the positions of the mines and returns the solution to the board in the form of a list of moves to be done by the robot.

The function's specific implementation is the following:

```
solve curr mines = collect (S curr (sortMines curr mines))
```

and the way it works is the description of the Explanation section.

Demonstration

The following section demonstrates two runs of the program with larger grid sizes.

Due to the very large (although not the largest possible this program can work with!) board sizes for both runs, the output cannot be captured in a single screenshot. Therefore, keep in mind that the solutions displayed in the images here are incomplete.

If you would like to see the full solution, the test cases for both runs are provided in the following pages so that you can run them on the program yourself.

Run 1 (1000x1000 board, 96 mines):

```
solve (0,0) [(892,879), (129,967), (144,612), (127,287), (809,153),
(886,931), (144,449), (794,594), (889,34), (649,57), (712,280),
(305,303), (160,701), (222,575), (688,946), (975,668), (889,582),
(366,456), (966,423), (544,272), (412,646), (618,904), (312,636),
(971,736), (793,551), (764,885), (305,424), (150,57), (27,520),
(241,270), (379,17), (534,212), (733,215), (796,422), (678,492),
(760,313), (933,201), (902,888), (301,695), (243,255), (97,149),
(753,152), (547,496), (826,232), (159,934), (771,594), (85,235),
(932,899), (533,462), (433,581), (364,549), (861,276), (850,586),
(123,118), (408,922), (118,637), (674,12), (130,2), (413,283),
(390,588), (5,543), (140,757), (88,838), (351,489), (900,963),
(591,538), (701,653), (892,193), (685,33), (420,735), (500,582),
(401,468), (828,431), (508,386), (931,469), (67,183), (624,262),
(246,917), (379,86), (81,922), (113,689), (78,754), (774,777),
(442,241), (398,41), (669,631), (630,145), (571,348), (480,875),
(825,699),(999,1000)]
```

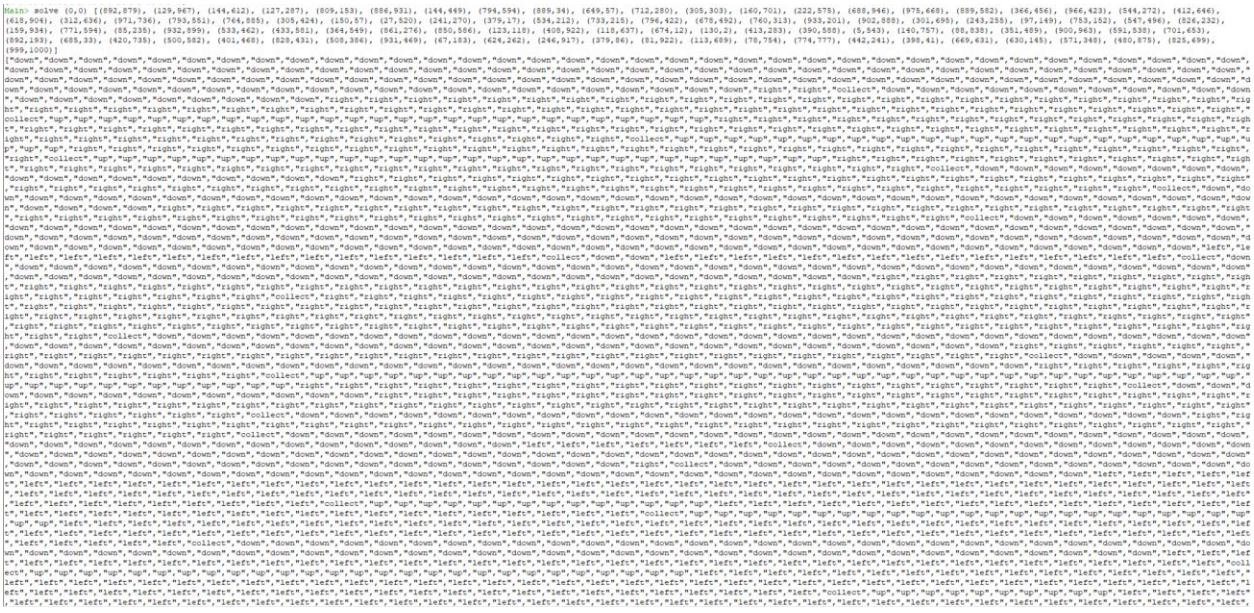

Figure (1): Run 1

Run 2 (10000x10000 board, 75 mines):

```
solve (0,0) [(1128,8719), (2500,5478), (9908,1137), (7472,4001),
(5118,1008), (3519,4563), (5744,2571), (3687,7249), (1013,111),
(1444,1788), (2200,6323), (3515,1142), (3084,8341), (7569,5853),
(5422,8781), (8612,4525), (7663,9316), (3610,4551), (1580,8289),
(9082,8037), (5416,2417), (9754,6408), (899,7528), (6432,6919),
(8744,5159), (7287,8825), (535,9166), (4847,5335), (9333,964),
(8791,7158), (2231,1361), (105,1013), (8792,6698), (5464,8852),
(1948,7208), (7353,5184), (2114,6581), (5980,871), (9977,7746),
(3594,1211), (9222,4233), (2378,2008), (5470,613), (5971,4365),
(8016,7156), (1386,3974), (7051,6669), (637,3161), (9906,6544),
(2804,2015), (1382,3628), (8010,3723), (5587,9246), (8528,8436),
(257,7977), (788,1996), (1498,7409), (4789,6670), (1187,7809),
(1704,7042), (8694,8700), (7861,5791), (1311,7246), (9746,1054),
(6076,1310), (8137,8725), (9496,275), (5552,2775), (6324,7168),
(228,1823), (5128,3258), (8384,6423), (1562,4266), (4892,7874),
(525,9660)]
```

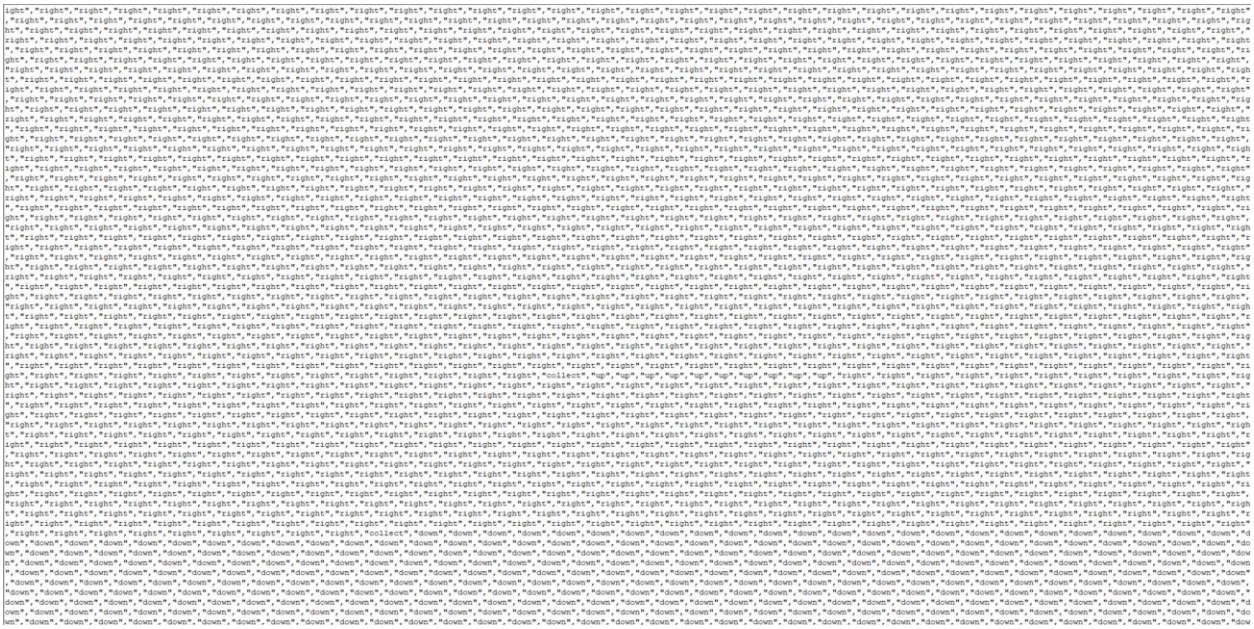

Figure (2): Run 2