CSEN403 Project

Project 2 Report

Minesweeper Robot

Team 15
Sara Wasfy 52-0356
Farah Ashraf 52-3205
Mayar Sherif 52-0354
Mayar Bahnacy 52-7251

Defining the data types:

To begin, we defined the type of Cell as a point of two integer values, which indicates a position on the grid. After that, we created the Mystate structure, which reflects the robot's current state. It was either Null or S, then a cell representing the robot's position, a list of cells representing the positions of the mines to be collected, a text reflecting the last operation performed to reach this state, and the parent state. Also, MyState was deriving (Show, Eq) to be able to run the tests.

up:: Mystate -> Mystate

The function accepts a state as an input and returns the state that results from moving up from it. Null should be returned if continuing up will cause you to leave the grid's limits. We added two conditions for the function. If the x-coordinate of the robot’s location is 0, it would return null. If not, it would return a new state with a decremented x-coordinate.

down:: Mystate -> Mystate

The function accepts a state as an input and returns the state obtained by going down from that state. If travelling down will cause you to leave the grid's limits, Null should be returned. We added two conditions for the function. If the x-coordinate of the robot’s location is 3, it would return null. If not, it would return a new state with an incremented x-coordinate.

left:: MyState -> MyState

The function accepts a state as an input and returns the state obtained by travelling left from that state. Null should be returned if left will result in moving outside the grid's borders. We added two conditions for the function. If the y-coordinate of the robot’s location is 0, it would return null. If not, it would return a new state with a decremented y-coordinate.

right:: MyState -> MyState

The function accepts a state as an input and returns the state obtained by moving right from it. If moving right will take you outside the grid's bounds, Null should be returned. We added two conditions for the function. If the y-coordinate of the robot’s location is 3, it would return null. If not, it would return a new state with an incremented y-coordinate.

collect:: MyState -> MyState

The function takes a state as an input and returns the state that is created by gathering data from the input state. Collecting should not modify the robot's position, but it should remove the mine from the list of mines to be collected. Null should be returned if the robot is not in the same position as one of the mines. First, we constructed a helper called collect2:: Cell -> [Cell] -> [Cell]. This helper takes the location of the robot and the locations of the mines to compare them together using recursion. If the robot’s location matched with the location of one of the mines, it returns the location of that mine. If it didn’t, it would return an empty list. If collect2 returned an empty list, collect function would return Null. As for the second case, we constructed a helper called cleanUp:: Cell -> [Cell] -> [Cell]. This helper takes the location of the robot and the locations of the mine if the robot’s location matched with the location of one of the mines to remove the location of the mine collected.

nextMyStates::MyState->[MyState]

The function accepts a state as an argument and returns the set of states that arise from applying up, down, left, right, and collect to it. There should be no Null states in the output set of states. First, we appended all the states together in one list. Then, we constructed a helper called nextMyStates2:: [MyState] -> [MyState]. This helper removes any Null states from the list

isGoal::MyState->Bool

The function takes a state as an input and returns true if there are no more mines to collect in the input state (the list of mines is empty), and false otherwise. We added one condition that if the list of locations of the mines is empty, the function should return true.

search::[MyState]->MyState

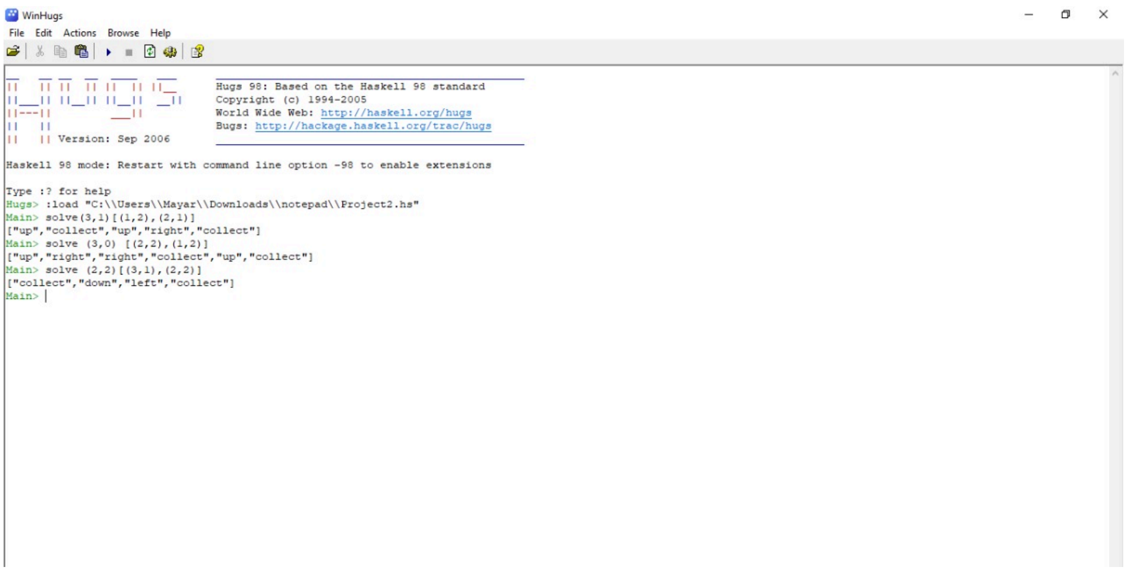
The function takes a list of states as input. It verifies if the input list's head is a goal state, and if it is, it returns the head. Otherwise, it retrieves the next states from the first state in the input list and calls itself recursively, concatenating the tail of the input list with the resulting next states. First, we called the isGoal function to check if the list of locations of the mines is empty, it should return the head state. Otherwise, it would return the tail appended with the result of calling nextMyStates on the head state.

constructSolution:: MyState ->[String]

The function accepts a state as an input and outputs a sequence of strings that indicate actions that the robot can take to get from the initial state to the input state. Only "up," "down," "left," "right," and "collect" are possible strings in the output list of strings. It takes the state and appends each string that indicate an action to a list until it reaches the parent state with the empty string.

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

The function takes as input a cell representing the robot's beginning location and a collection of cells indicating the mines' placements, and returns a set of strings describing actions that the robot can take to get from the original state to the goal state. We called the function nextMyStates on an initial state to apply all the available actions, then the function search on the result of nextMyStates to check if each resulting state is a goal state, then the function constructSolution on the result of search to get the list of actions for the robot to follow to reach a goal state.



```
__   __ __  __  ____   ___         _________________________________________
||   || ||  || ||  || ||__         Hugs 98: Based on the Haskell 98 standard
||___|| ||__|| ||__||  __||        Copyright (c) 1994-2005
||---||         ___||              World Wide Web: http://haskell.org/hugs
||   ||                            Bugs: http://hackage.haskell.org/trac/hugs
||   || Version: Sep 2006          _________________________________________

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
Hugs> :load "C:\\Users\\Mayar\\Downloads\\notepad\\Project2.hs"
Main> solve(3,1)[(1,2),(2,1)]
["up","collect","up","right","collect"]
Main> solve (3,0) [(2,2),(1,2)]
["up","right","right","collect","up","collect"]
Main> solve (2,2)[(3,1),(2,2)]
["collect","down","left","collect"]
Main> |
```

Bonus Implementation

Defining the data types

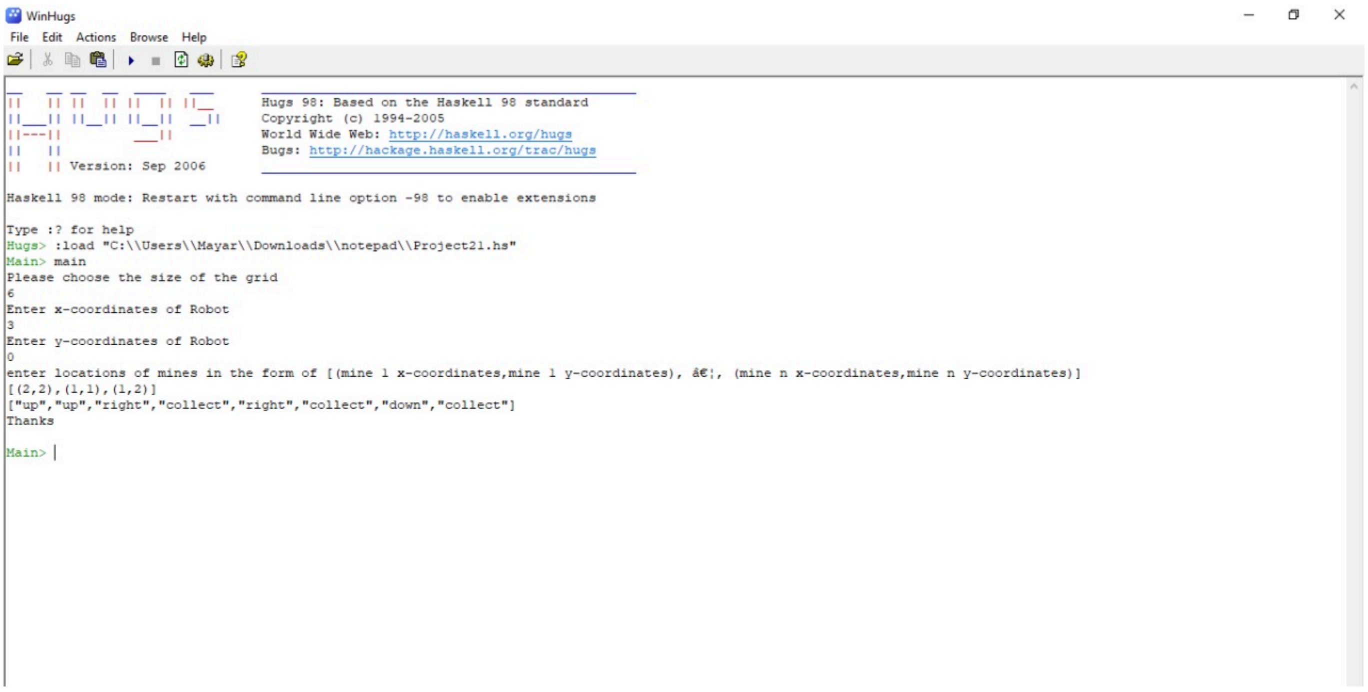
First, we defined a new Cell as a point of three integer values, which represents a position on the grid and the grid size (n). As for Cell of the normal 4x4 implementation, we changed its name to Celll. After that, we changed all the cells in the functions to be (x,y,n).

convertCelltoCell:: [Celll] -> Int -> [Cell]

In order not to change the data type of solve, we constructed this helper function which takes each Cell of two points only to create a cell of three points. To elaborate, it adds the grid size (n) to any position on the grid.

main:: IO ()

This function prompts the user to enter the grid size he/she want to play on, and assigns this input to the variable n. Then, it prompts the user to enter the initial x and y location of the robot, assigns it to variables x and y. After that, it prompts the user to enter a list of the locations of the mines and assigns it to the list l. Then, the function creates a Cell p of the x and y location of the robot plus the grid size (n). Finally, the function solve is called and given the location of the robot and the result of convertCelltoCell on the list of locations of the mines.



```
__   __ __  __  ____   ___         _________________________________________
||   || ||  || ||  || ||__         Hugs 98: Based on the Haskell 98 standard
||___|| ||__|| ||__||  __||        Copyright (c) 1994-2005
||---||         ___||              World Wide Web: http://haskell.org/hugs
||   ||                            Bugs: http://hackage.haskell.org/trac/hugs
||   || Version: Sep 2006          _________________________________________

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
Hugs> :load "C:\\Users\\Mayar\\Downloads\\notepad\\Project21.hs"
Main> main
Please choose the size of the grid
6
Enter x-coordinates of Robot
3
Enter y-coordinates of Robot
0
enter locations of mines in the form of [(mine 1 x-coordinates,mine 1 y-coordinates), â€¦, (mine n x-coordinates,mine n y-coordinates)]
[(2,2),(1,1),(1,2)]
["up","up","right","collect","right","collect","down","collect"]
Thanks

Main> |
```