# Project II Report

Firstly, we have defined data type Called Cell and is structured like that: (Int, Int).

Secondly, we have defined data type Called MyState it represents the robot position.

Thirdly, we have defined the up the function takes as input a state and returns the state resulting from moving up from the input state.

Fourthly, we have defined the down the function takes as input a state and returns the state resulting from moving down from the input state.

Fifthly, we have defined the left the function takes as input a state and returns the state resulting from moving left from the input state.

Sixthly, we have defined the right the function takes as input a state and returns the state resulting from moving right from the input state.

Seventhly, we have defined the collect the function takes as input a state and returns the state resulting from collecting from the input state.

Eighthly, we have defined the nextMyStates the function takes as input a state and returns the set of states resulting from applying up, down, left, right, and collect from the input state.

Ninthly, we have defined the isGoal the function takes as input a state, returns true if the input state has no more mines to collect and false otherwise.

Then we have the getDistances (Cell, [Cell]) function, which is the first of many helper functions used to construct the solution. This function takes as inputs a cell (for example called relativeCell) and a list of cells and it gets the Manhattan distance between the relativeCell and each cell in the list then places them in a list and returns that list.

After that we have the help ([Cell], [Int], Cell, Int) function. This is another helper function that takes as input a list of cells, a list of Ints containing the Manhattan distances produced by the preceding function, an accumulator cell, and an accumulator Manhattan distance. This function looks for the cell with the smallest Manhattan distance and returns it by traversing both the list of cells and the list of Ints and compare each length with the accumulator length. If it is indeed less than the accumulator cell then the function recures on itself with a new accumulator cell and length.

Then we have getClosestCell ([Cell], [Int]) this is another helper function. Basically, the function returns the target cell with the smallest distance relative to the relativeCell in the getDistances function, by calling the help function to allow for accumulation.

After that, we have defined function search the function takes as input a list of states. It checks if the head of the input list is a goal state, if it is a goal, it returns the head. Otherwise, it gets the next states from the state at head of the input list and calls itself recursively with the result of concatenating the tail of the input list with the resulting next states.

Then, we have defined function constructSolution the function takes as input a state and returns a set of strings representing actions that the robot can follow to reach the input state from the initial state.

Then we have the getPossibilities (MyState) function which takes as input a MyState ms and returns a list of strings that represent the way to which the robot can reach the target cells in the state. This is done by calling the calling the search function on the state and putting the result in the construct solution to get the list of strings.

Then we have the solveHelp (Cell, [Cell], Cell) function the last helper method. It takes as inputs an initial cell, an array of cells, and a target cell. It calls the getPossibilities on a cell containing only one cell, which is the target cell, then it concatenates the result with the result of recurring on itself with the initial cell changed to the target cell, the original list minus the target cell, and a new target cell from the original list minus the target cell, by calling the getClosestCell and the getDistances functions.

Finally, we have defined function solve the function takes as input a cell representing the starting position of the robot, a set of cells representing the positions of the mines, and returns a set of strings representing actions that the robot can follow to reach a goal state from the initial state.

**Bonus Implementation:** For the bonus implementation all of the above remain the same with the exception that we created an additional function called checkOpposites and used it to remove any unnecessary states from the search function to provide a slightly more efficient implementation. (i. e. we basically check if the string on the state and its preceding state, if they are opposites, like up and down, then we remove it because we already obtained the states of the preceding state.).

In the Next Page you Will find a gallery containing 2 different runs

And The additional bonus run

# Gallery

(The First Run)

```
*Main> solve (3, 0) [(0, 0), (2, 2), (3, 3), (1, 2), (3, 2), (2, 1), (0, 1), (2, 3), (2, 0), (1, 3)]
["up","collect","right","collect","right","collect","right","collect","up","collect","left","collect","up","left","collect","left","collect","down","down","down","right","right","collect","right","collect"]
```

(The Second Run)

```
*Main> solve (0, 0) [(1, 1), (2, 2), (2, 3), (3, 3), (3, 1), (1, 2)]
["down","right","collect","right","collect","down","collect","right","collect","down","collect","left","left","collect"]
```

(The Bonus Run)

```
*Main> solve (3, 0) [(0, 0), (2, 2), (3, 3), (3, 6), (1, 7), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]
["right","right","right","collect","up","left","collect","up","up","left","left","collect","down","right","right","right","right","right","right","collect","down","down","left","collect","down","down","down","collect","down","right","collect","down","right","collect","down","right","collect","up","up","up","up","left","left","left","left","collect"]
```

Report Prepared Ziad Elsabbagh.