Project 2: Minesweeper Robot

Code:

```
type Cell = (Int,Int)


getX :: Cell -> Int

getX (x , y) = x


getY :: Cell -> Int

getY (x , y) = y


data MyState = Null | S Cell [Cell] String MyState deriving (Show,Eq)



up :: MyState -> MyState

up Null = Null

up (S (x,y) cell string myState) =if x == 0 then Null else S (x - 1 , y) cell
"up" (S (x,y) cell string myState)



down :: MyState -> MyState

down Null = Null
down (S (x,y) cell string myState) = if x == 3 then Null else S (x+1,y) cell
"down" (S (x,y) cell string myState)



left :: MyState -> MyState

left Null = Null

left (S (x,y) cell string myState) =if y == 0 then Null else S (x,y-1) cell
"left" (S (x,y) cell string myState)



right :: MyState -> MyState

right Null = Null

right (S (x,y) cell string myState) = if y == 3 then Null else S (x,y+1) cell
"right" (S (x,y) cell string myState)



collect :: MyState -> MyState


collect (S (x, y) (h : t) string myState)

  | x == getX h && y == getY h = S (x , y) t "collect" (S (x,y) (h : t) string
myState)

  | t /= [] = collect (S (x,y) t string myState)

  | otherwise = Null
```

```
nextMyStates :: MyState -> [MyState]

nextMyStates (S (x,y) cell string myState) = helpercollect (S (x,y) cell string myState) ++ helperRight (S (x,y) cell string myState) ++ helperLeft (S (x,y) cell string myState) ++ helperup (S (x,y) cell string myState) ++ helperdown (S (x,y) cell string myState)


helperRight (S (x,y) cell string myState) = [right (S (x,y) cell string myState) | right (S (x,y) cell string myState) /= Null]

helperLeft (S (x,y) cell string myState) = [left (S (x,y) cell string myState) | left (S (x,y) cell string myState) /= Null]

helperup (S (x,y) cell string myState) = [up (S (x,y) cell string myState) | up (S (x,y) cell string myState) /= Null]

helperdown (S (x,y) cell string myState) = [down (S (x,y) cell string myState) | down (S (x,y) cell string myState) /= Null]

helpercollect (S (x,y) cell string myState) = [collect (S (x,y) cell string myState) | collect (S (x,y) cell string myState) /= Null]



isGoal :: MyState -> Bool

isGoal (S (x,y) cell string myState) | null cell = True

                                     | otherwise  = False
```

```
search :: [MyState] -> MyState

search [] = Null

search (h:t) = if isGoal h then h else search (t ++ nextMyStates h)
```

```
constructSolution:: MyState ->[String]

constructSolution (S (x,y) cell string myState) = if myState == Null then
[] else constructSolution myState ++ [string]
```

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

solve (curx,cury) (h:t) = constructSolution (search (nextMyStates(S
(curx,cury) (h:t) "" Null)))
```

Description:

Up: In this method as long as the robot is not in the 0 Column, the robot is allowed to move up and the return state is the same state with the difference that the X-coordinate is decreased by 1, else it returns null.

Down: In this method as long as the robot is not in the 3rd Column, the robot is allowed to move down and the return state is the same state with the difference that the X-coordinate is increased by 1, else it returns null.

Right: In this method as long as the robot is not in the 3rd row, the robot is allowed to move right and the return state is the same state with the difference that the Y-coordinate is increased by 1, else it returns null.

Left: In this method as long as the robot is not in the 0 row, the robot is allowed to move left and the return state is the same state with the difference that the Y-coordinate is decreased by 1, else it returns null.

Collect: In this method the program checks if the robot is standing on one of the mines which means if the coordinates of the robot is same as one of the mines, then we will return the same state with the difference that we remove one of the mines, else null.

nextMyStates: In this method we used four helper methods to check the if the right cell is eligible to move on or not and the same for the left, down and up else returns an empty list.

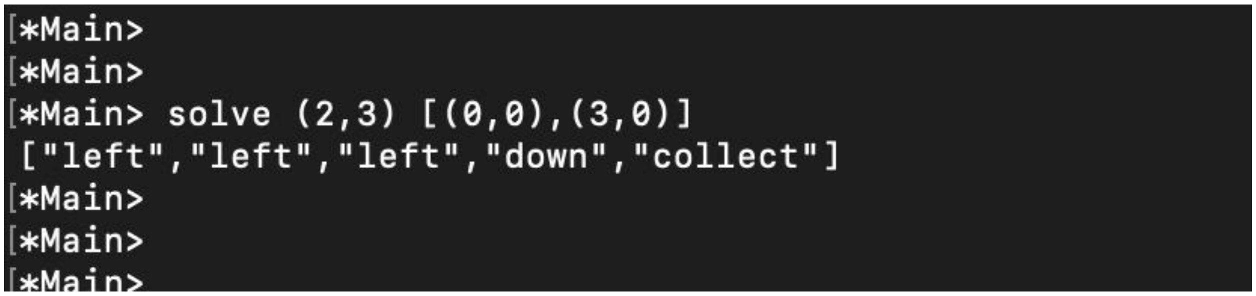
IsGoal: in this method it checks whether there are any left mines in the game or not. If there are no mines it will return true otherwise false.

Search: If the list of states is empty it will return null, else 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

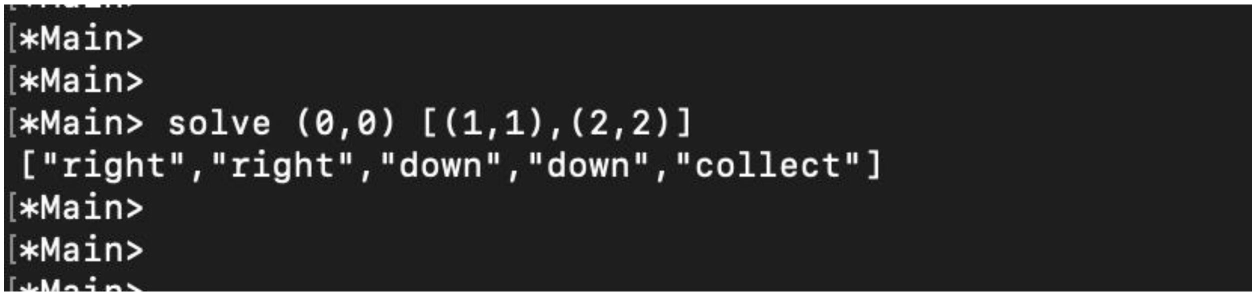
constructSolution: returns an empty list of strings if the parent state == null else calls itself recursively and concatenating the states to each other as a list of strings.

Solve: In this method we call constructSolution to concatenate a list of strings which holds the all the states the robot needs to follow to reach for the goal state for the initial state by using search.

Screenshots:



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



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