Haskell Project

Team Number 52

Mohamed Mahmoud Ismaeil -> 52-5931

Nour Hesham Alansary -> 49-4236

Abdelrahman Amr Kamel -> 52-4555

Abdelrahman Essam Mohamed Elgharib -> 52-9298

Main Code:

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


delete x []=[]
delete x (h:t) = if(x==h) then t
                 else (h:delete x t)

up (S (x,y) lcell string oldstate)= if(x>0) then ((S ((x-1),y) lcell "up" (S (x,y) lcell string oldstate)))
                                    else Null

down (S (x,y) lcell string oldstate)= if(x<4) then (S ((x+1),y) lcell "down" (S (x,y) lcell string oldstate))
                                      else Null

left (S (x,y) lcell string oldstate)= if(y>0) then (S (x,(y-1)) lcell "left" (S (x,y) lcell string oldstate))
                                      else Null

right (S (x,y) lcell string oldstate)= if(y<4) then (S (x,(y+1)) lcell "right" (S (x,y) lcell string oldstate))
                                       else Null

collect (S (x,y) lcell string oldstate)= if(elem (x,y) lcell) then (S (x,y) (delete (x,y) lcell) "collect" (S (x,y) lcell string oldstate))
                                         else Null

removeNulls []=[]
removeNulls (h:t)= if(h==Null) then removeNulls t
                   else (h:removeNulls t)

nextMyStates state= removeNulls [up state,down state,left state,right state,collect state]

isGoal (S (x,y) lcell string oldstate)= if(length lcell==0) then True
                                        else False
search []=Null
search (h:t)= if(isGoal h) then h
              else search (t ++ nextMyStates h)

constructSolution (S (x,y) lcell string Null)=[]
constructSolution (S (x,y) lcell string oldstate)= constructSolution oldstate ++ [string]

solve pos mine= constructSolution (search [(S pos mine "" Null)])
```

Up -> it takes a state and checks if moving up will move the robot out of the gird since the origin is at the top right corner we check if the row position is greater than 0 then we can move by returning a new state where the row position is decremented and has string representing the direction of motion.

Down -> it takes a state and checks if moving down will move the robot out of the gird since the origin is at the top right corner we check if the row position is less than max grid height then we can move by returning a new state where the row position is incremented and has string representing the direction of motion.

left -> it takes a state and checks if moving left will move the robot out of the gird since the origin is at the top right corner we check if the column position is greater than 0 then we can move by returning a new state where the column position is decremented and has string representing the direction of motion.

right-> it takes a state and checks if moving down will move the robot out of the gird since the origin is at the top right corner we check if the column position is less than max grid width then we can move by returning a new state where the column position is incremented and has string representing the direction of motion.

Collect-> it takes a state and check if the current robot position is an element within the list of mines locations, if yes then we will remove that position from the mine list and then we will update the current state accordingly adding a new string “collect”

nextMyState -> it takes a state and returns a list of the four actions that the robot could perform at a given time and if any of them returns a Null they will automatically removed using the removeNull function.

isGoal -> it takes a state and checks if the current state has an empty mines list, if yes then returns true else false

search -> it takes a list of states and returns the state itself if it is a goal, otherwise it will keep on generating every possible action for every element in the list and will concatenate them to end of the list.

constructSolution-> it takes a successful state and will concatenate the head at the end of each iteration while calling itself on all of the states and just before hitting a Null state, it will stop.

Solve-> it takes an initial robot location and a list containing the locations of the mines and will call constructSolution from what ever succeeds from the search function which will take a list of only one state, which is the initial state.

Two Runs:

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

Bonus Runs:

I am sorry for the small font but the runs takes the width of the monitor and it isn’t enough.