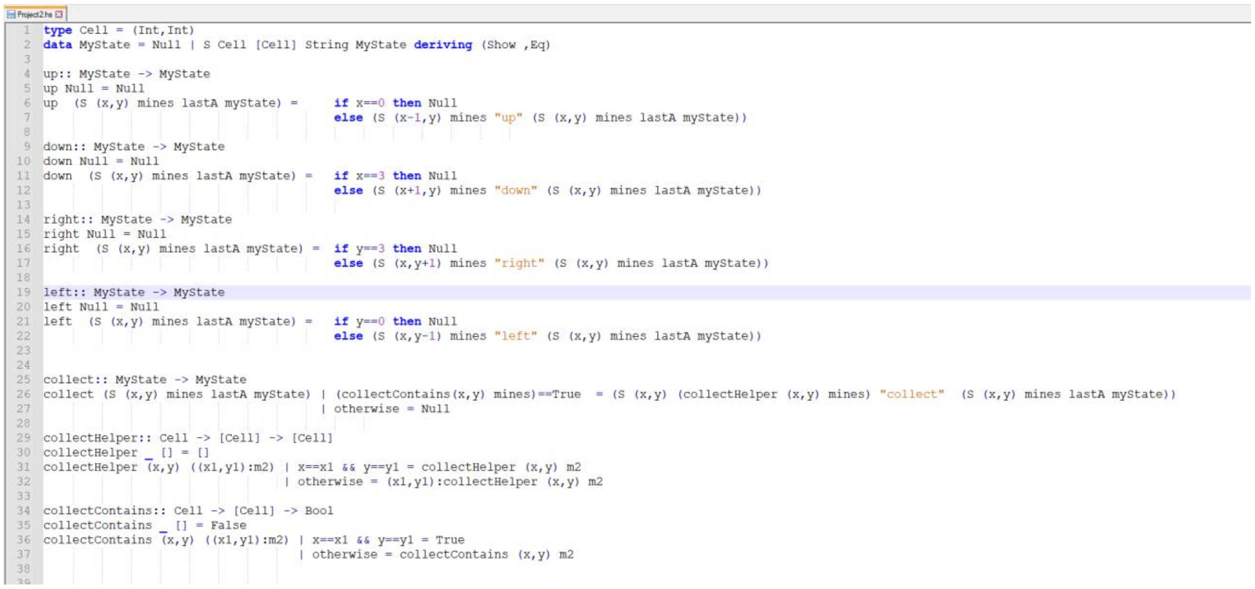
Concepts of Programming Languages

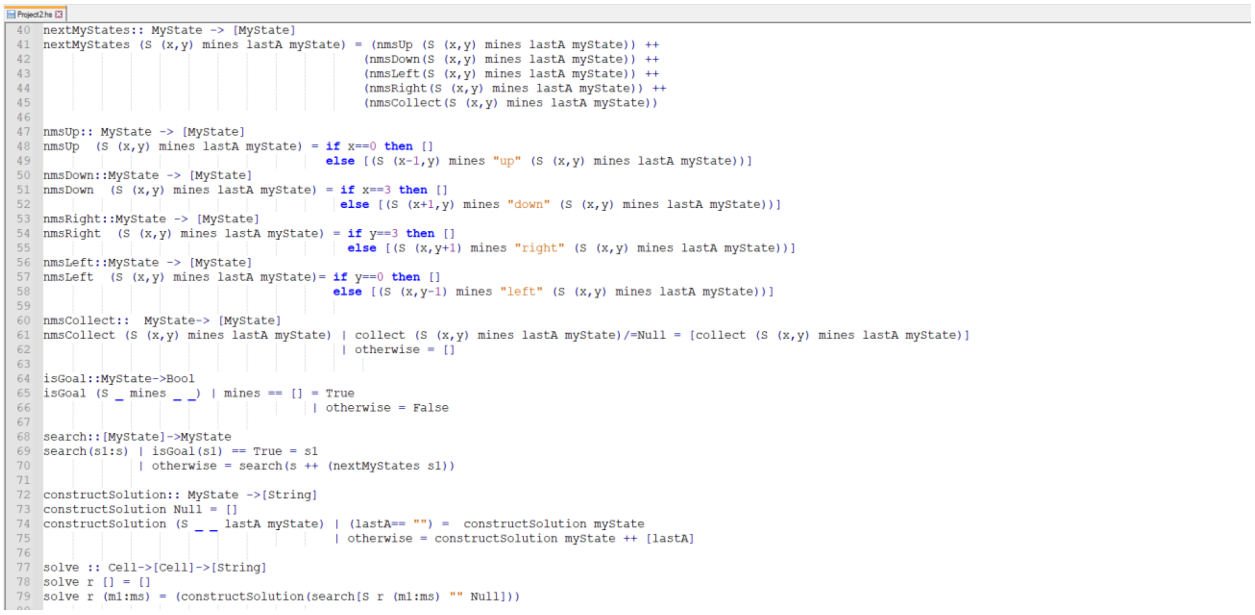
Project 2: Minesweeper Robot

Team 56

Implementation



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

up:: MyState -> MyState
up Null = Null
up  (S (x,y) mines lastA myState) =     if x==0 then Null
                                        else (S (x-1,y) mines "up" (S (x,y) mines lastA myState))

down:: MyState -> MyState
down Null = Null
down  (S (x,y) mines lastA myState) =   if x==3 then Null
                                        else (S (x+1,y) mines "down" (S (x,y) mines lastA myState))

right:: MyState -> MyState
right Null = Null
right  (S (x,y) mines lastA myState) =  if y==3 then Null
                                        else (S (x,y+1) mines "right" (S (x,y) mines lastA myState))

left:: MyState -> MyState
left Null = Null
left  (S (x,y) mines lastA myState) =   if y==0 then Null
                                        else (S (x,y-1) mines "left" (S (x,y) mines lastA myState))


collect:: MyState -> MyState
collect (S (x,y) mines lastA myState) | (collectContains(x,y) mines)==True  = (S (x,y) (collectHelper (x,y) mines) "collect"  (S (x,y) mines lastA myState))
                                      | otherwise = Null

collectHelper:: Cell -> [Cell] -> [Cell]
collectHelper _ [] = []
collectHelper (x,y) ((x1,y1):m2) | x==x1 && y==y1 = collectHelper (x,y) m2
                                 | otherwise = (x1,y1):collectHelper (x,y) m2

collectContains:: Cell -> [Cell] -> Bool
collectContains _ [] = False
collectContains (x,y) ((x1,y1):m2) | x==x1 && y==y1 = True
                                   | otherwise = collectContains (x,y) m2

```



```haskell
nextMyStates:: MyState -> [MyState]
nextMyStates (S (x,y) mines lastA myState) = (nmsUp (S (x,y) mines lastA myState)) ++
                                            (nmsDown(S (x,y) mines lastA myState)) ++
                                            (nmsLeft(S (x,y) mines lastA myState)) ++
                                            (nmsRight(S (x,y) mines lastA myState)) ++
                                            (nmsCollect(S (x,y) mines lastA myState))

nmsUp:: MyState -> [MyState]
nmsUp  (S (x,y) mines lastA myState) = if x==0 then []
                                       else [(S (x-1,y) mines "up" (S (x,y) mines lastA myState))]
nmsDown::MyState -> [MyState]
nmsDown  (S (x,y) mines lastA myState) = if x==3 then []
                                         else [(S (x+1,y) mines "down" (S (x,y) mines lastA myState))]
nmsRight::MyState -> [MyState]
nmsRight  (S (x,y) mines lastA myState) = if y==3 then []
                                          else [(S (x,y+1) mines "right" (S (x,y) mines lastA myState))]
nmsLeft::MyState -> [MyState]
nmsLeft  (S (x,y) mines lastA myState)= if y==0 then []
                                        else [(S (x,y-1) mines "left" (S (x,y) mines lastA myState))]

nmsCollect::  MyState-> [MyState]
nmsCollect (S (x,y) mines lastA myState) | collect (S (x,y) mines lastA myState)/=Null = [collect (S (x,y) mines lastA myState)]
                                         | otherwise = []

isGoal::MyState->Bool
isGoal (S _ mines _ _) | mines == [] = True
                                     | otherwise = False

search::[MyState]->MyState
search(s1:s) | isGoal(s1) == True = s1
             | otherwise = search(s ++ (nextMyStates s1))

constructSolution:: MyState ->[String]
constructSolution Null = []
constructSolution (S _ _ lastA myState) | (lastA== "") =  constructSolution myState
                                        | otherwise = constructSolution myState ++ [lastA]

solve :: Cell->[Cell]->[String]
solve r [] = []
solve r (m1:ms) = (constructSolution(search[S r (m1:ms) "" Null]))
```

Implemented Functions

Defined the type of Cell and Defined a constructor for MyState.

The functions (up,down,left,right) takes as input a state and returns the state resulting from moving (up,down,left,right) from the input state .

The function collect takes as input a state and returns the state resulting from collecting from the input state by using the 2 helper methods Below.

The function collectHelper takes as input robot cell and list of mines cells and returns a list of mines cells without the collected mine.

The function collectContains takes as input robot cell and list of mines cells and returns true if the robot is standing one of the mines and false otherwise.

The function nextMyStates takes as input a state and returns the set of states resulting from applying up, down, left, right, and collect from the input state by concatenating all of the below helper methods in a list of states.

The function nmsUp takes as takes as input a state and returns the set of states resulting from applying up from the input state

The function nmsDown takes as takes as input a state and returns the set of states resulting from applying down from the input state

The function nmsLeft takes as takes as input a state and returns the set of states resulting from applying left from the input state

The function nmsRight takes as takes as input a state and returns the set of states resulting from applying right from the input state

The function nmsCollect takes as takes as input a state and returns the set of states resulting from applying collect from the input state

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

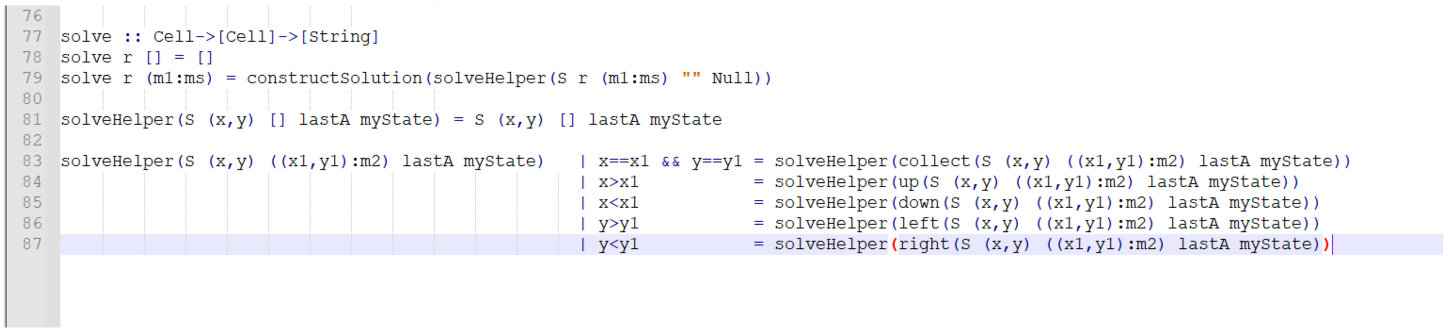
The function search takes as input a list of states. It checks if the head of the input list is a goal state by using isGoal function, if it is a goal, it returns the head. Otherwise, it gets the next states from the state at head of the input list and concatenate it to the tail.

The function constructSolution 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.

The function solve takes as input a cell representing the starting position of the robot, a set of cells representing the positions of the mines, and uses search and constructSolution to return a set of strings representing actions that the robot can follow to reach a goal state from the initial state.

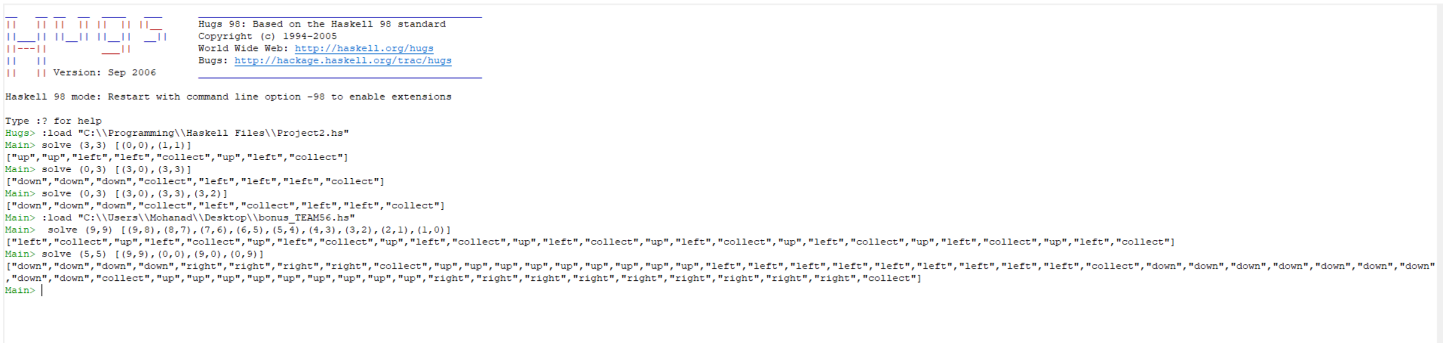
Bonus Implementation

I changed the method solve by using a method solveHelper instead of search that works as follows, the method compares the position of the robot with the first mine and moves the robot by calling the method recursively until he collects it so it gets removed from the mines list and keep on calling itself recursively until the mines list be empty and returns it.



```
solve :: Cell->[Cell]->[String]
solve r [] = []
solve r (m1:ms) = constructSolution(solveHelper(S r (m1:ms) "" Null))

solveHelper(S (x,y) [] lastA myState) = S (x,y) [] lastA myState

solveHelper(S (x,y) ((x1,y1):m2) lastA myState)    | x==x1 && y==y1 = solveHelper(collect(S (x,y) ((x1,y1):m2) lastA myState))
                                                   | x>x1           = solveHelper(up(S (x,y) ((x1,y1):m2) lastA myState))
                                                   | x<x1           = solveHelper(down(S (x,y) ((x1,y1):m2) lastA myState))
                                                   | y>y1           = solveHelper(left(S (x,y) ((x1,y1):m2) lastA myState))
                                                   | y<y1           = solveHelper(right(S (x,y) ((x1,y1):m2) lastA myState))
```

Runs / Grid Configurations



```
__   __ __  __  ____   ___      _________________________________________
||   || ||  || ||  || ||__      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:\\Programming\\Haskell Files\\Project2.hs"
Main> solve (3,3) [(0,0),(1,1)]
["up","up","left","left","collect","up","left","collect"]
Main> solve (0,3) [(3,0),(3,3)]
["down","down","down","collect","left","left","left","collect"]
Main> solve (0,3) [(3,0),(3,3),(3,2)]
["down","down","down","collect","left","collect","left","left","collect"]
Main> :load "C:\\Users\\Mohanad\\Desktop\\bonus_TEAM56.hs"
Main>  solve (9,9) [(9,8),(8,7),(7,6),(6,5),(5,4),(4,3),(3,2),(2,1),(1,0)]
["left","collect","up","left","collect","up","left","collect","up","left","collect","up","left","collect","up","left","collect","up","left","collect","up","left","collect","up","left","collect"]
Main> solve (5,5) [(9,9),(0,0),(9,0),(0,9)]
["down","down","down","down","right","right","right","right","collect","up","up","up","up","up","up","up","up","up","left","left","left","left","left","left","left","left","left","collect","down","down","down","down","down","down","down"
,"down","down","collect","up","up","up","up","up","up","up","up","up","right","right","right","right","right","right","right","right","right","collect"]
Main> |
```

- The first 3 configurations is for a 4x4 Grid on the Team56.hs file

1. The robot R starts at (3,3) while the mines are at (0,0) and (1,1).

   Actions: `["up","up","left","left","collect","up","left","collect"]`

2. The robot R starts at (0,3) while the mines are at (3,0) and (3,3).

   Actions: `["down","down","down","collect","left","left","left","collect"]`

3. The robot R starts at (0,3) while the mines are at (3,0),(3,3) and (3,2).

   Actions: `["down","down","down","collect","left","collect","left","left","collect"]`

- For the bonus we had two runs on a 10x10 Grid on the bonus_TEAM56.hs file

1. The robot R starts at (9,9) while the mines are at (9,8),(8,7),(7,6),(6,5),(5,4),(4,3),(3,2),(2,1),(1,0).

Actions:
```
["left","collect","up","left","collect","up","left","collect","up","left","collect","up",
"left","collect","up","left","collect","up","left","collect","up","left","collect","up","
left","collect"]
```

2. The robot R starts at (5,5) while the mines are at (9,9),(0,0) ,(9,0) ,(0,9)

Actions:
```
["down","down","down","down","right","right","right","right","collect","up","up","up","up
","up","up","up","up","up","left","left","left","left","left","left","left","left","left"
,"collect","down","down","down","down","down","down","down","down","down","collect","up",
"up","up","up","up","up","up","up","up","right","right","right","right","right","right","
right","right","right","collect"]
```