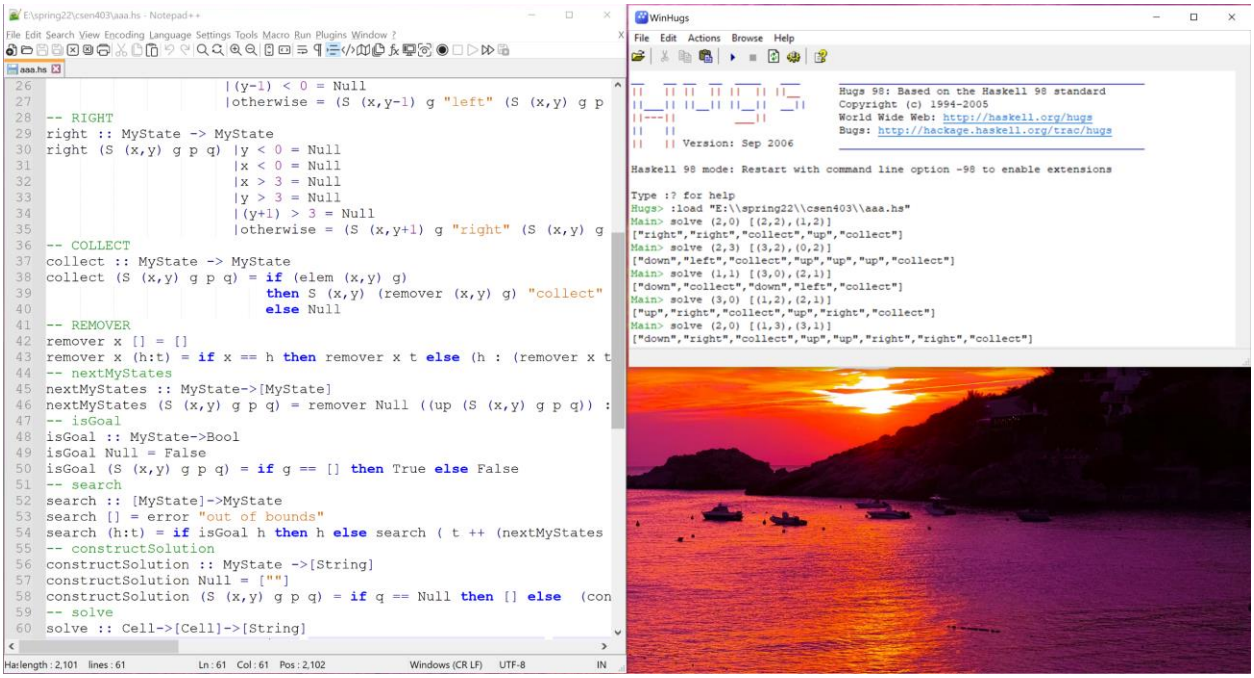
Description of functions:

1- Up: the function takes a state and returns the result state after the minesweeper robot has moved up, note that that if the coordinates exceeds 3 or fall short of 0 Null is returned to keep the minesweeper robot inside the gird.
2- Down: the function takes a state and returns the result state after the minesweeper robot has moved down, note that that if the coordinates exceeds 3 or fall short of 0 Null is returned to keep the minesweeper robot inside the gird.
3- Left: the function takes a state and returns the result state after the minesweeper robot has moved left, note that that if the coordinates exceeds 3 or fall short of 0 Null is returned to keep the minesweeper robot inside the gird.
4- Right: the function takes a state and returns the result state after the minesweeper robot has moved right, note that that if the coordinates exceeds 3 or fall short of 0 Null is returned to keep the minesweeper robot inside the gird.
5- Collect: the function takes a state as input and preforms the action “collect” which, if the minesweeper robot is in the same cell as a mine, removes the mine from the list of mines and returns the resulting state, else the functions return Null.
6- Remover: the function the function takes an element x and a list and return a list without x elements
7- nextMyStates: the function returns a list of states after preforming the following operations: Up, Down , Left, Right and Collect on the state taken as input.
8- isGoal: the function checks whether the state taken as input has any mines left to collect, if there are still mines it returns false, otherwise it returns true.
9- Search: the function takes as an input a list of states and apply “isGoal”[7] function on the head of the list and returns the head if “isGoal”[7] is true otherwise, it applies “nextMyStates”[6] on the head and then append the result to the tail of the input list with the tail now being at the front.
10- constructSolution: the function returns a list of strings that acts as instructions for the robot to move from its current state to the input state.
11- Solve: the function takes as input the current cell of the robot and the cells of the mines and returns a list of strings that is basically the plan of action of the minesweeper robot to fully eradicate all the mines on the grid.

Screenshot of multiple different grid configurations with the returned solutions:



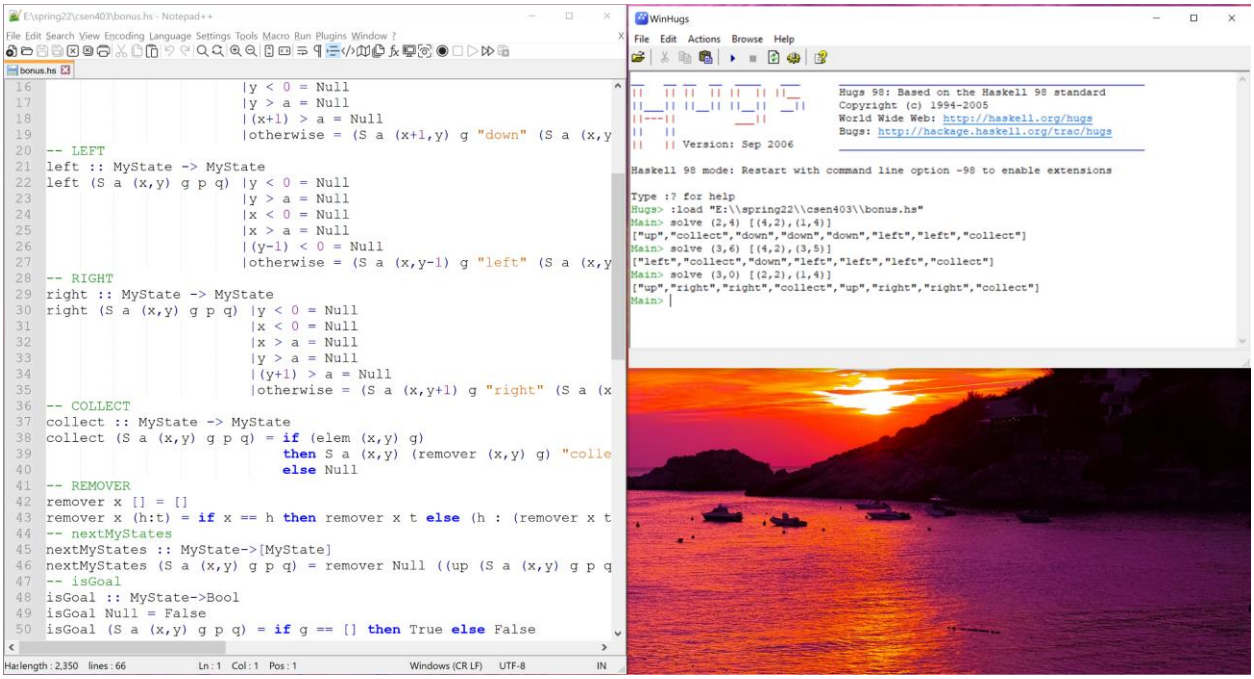
```
                    |(y-1) < 0 = Null
                    |otherwise = (S (x,y-1) g "left" (S (x,y) g p
-- RIGHT
right :: MyState -> MyState
right (S (x,y) g p q) |y < 0 = Null
                      |x < 0 = Null
                      |x > 3 = Null
                      |y > 3 = Null
                      |(y+1) > 3 = Null
                      |otherwise = (S (x,y+1) g "right" (S (x,y) g
-- COLLECT
collect :: MyState -> MyState
collect (S (x,y) g p q) = if (elem (x,y) g)
                          then S (x,y) (remover (x,y) g) "collect"
                          else Null
-- REMOVER
remover x [] = []
remover x (h:t) = if x == h then remover x t else (h : (remover x t
-- nextMyStates
nextMyStates :: MyState->[MyState]
nextMyStates (S (x,y) g p q) = remover Null ((up (S (x,y) g p q)) :
-- isGoal
isGoal :: MyState->Bool
isGoal Null = False
isGoal (S (x,y) g p q) = if g == [] then True else False
-- search
search :: [MyState]->MyState
search [] = error "out of bounds"
search (h:t) = if isGoal h then h else search ( t ++ (nextMyStates
-- constructSolution
constructSolution :: MyState ->[String]
constructSolution Null = [""]
constructSolution (S (x,y) g p q) = if q == Null then [] else  (con
-- solve
solve :: Cell->[Cell]->[String]
```

```
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 "E:\\spring22\\csen403\\aaa.hs"
Main> solve (2,0) [(2,2),(1,2)]
["right","right","collect","up","collect"]
Main> solve (2,3) [(3,2),(0,2)]
["down","left","collect","up","up","up","collect"]
Main> solve (1,1) [(3,0),(2,1)]
["down","collect","down","left","collect"]
Main> solve (3,0) [(1,2),(2,1)]
["up","right","collect","up","right","collect"]
Main> solve (2,0) [(1,3),(3,1)]
["down","right","collect","up","up","right","right","collect"]
```

(Bonus) Screenshot of multiple different grid configurations with the returned solutions:



```
                    |y < 0 = Null
                    |y > a = Null
                    |(x+1) > a = Null
                    |otherwise = (S a (x+1,y) g "down" (S a (x,y
-- LEFT
left :: MyState -> MyState
left (S a (x,y) g p q) |y < 0 = Null
                       |y > a = Null
                       |x < 0 = Null
                       |x > a = Null
                       |(y-1) < 0 = Null
                       |otherwise = (S a (x,y-1) g "left" (S a (x,y
-- RIGHT
right :: MyState -> MyState
right (S a (x,y) g p q) |y < 0 = Null
                        |x < 0 = Null
                        |x > a = Null
                        |y > a = Null
                        |(y+1) > a = Null
                        |otherwise = (S a (x,y+1) g "right" (S a (x
-- COLLECT
collect :: MyState -> MyState
collect (S a (x,y) g p q) = if (elem (x,y) g)
                          then S a (x,y) (remover (x,y) g) "colle
                          else Null
-- REMOVER
remover x [] = []
remover x (h:t) = if x == h then remover x t else (h : (remover x t
-- nextMyStates
nextMyStates :: MyState->[MyState]
nextMyStates (S a (x,y) g p q) = remover Null ((up (S a (x,y) g p q
-- isGoal
isGoal :: MyState->Bool
isGoal Null = False
isGoal (S a (x,y) g p q) = if g == [] then True else False
```

```
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 "E:\\spring22\\csen403\\bonus.hs"
Main> solve (2,4) [(4,2),(1,4)]
["up","collect","down","down","down","left","left","collect"]
Main> solve (3,6) [(4,2),(3,5)]
["left","collect","down","left","left","left","collect"]
Main> solve (3,0) [(2,2),(1,4)]
["up","right","right","collect","up","right","right","collect"]
Main>
```