Project 2

CSEN 403: Concepts of Programming Languages

MINESWEEPER Robot

Team #123

Abdelrhaman Mohamed 52-4615

Ahmad Hoseiny 52-4094

Ali Hussein 52-6565

Mazen Mohamed 52-2735

Supervised by:

Dr Nourhan Ehab

Eng Hadeel Adel

Description

1- Higher Order Functions :

1] Direction

Take as an input a String “str” and a User defined data type “MyState”

Check whether if

- “up” → Decrement X by 1
- “down” → Increment X by 1
- “left” → Decrement Y by 1
- “right” → Increment Y by 1

Check whether if

- Up && x == 0 → Null
- Down && x == 3 → Null
- Left && y == 0 → Null
- Right && y == 3 → Null

Return back the New State

```haskell
-- Direction [ up , down , left , right ]

-- Type

direction :: String -> MyState -> MyState

-- Body

direction str (S (x,y) l ls state) | str == "up" && x == 0 = Null
                                   | str == "down" && x == 3 = Null
                                   | str == "left" && y == 0 = Null
                                   | str == "right" && y == 3 = Null

direction str (S (x,y) l ls state) | str == "up"   = S (x-1,y) l ("up")    (S (x,y) l ls state)
                                   | str == "down" = S (x+1,y) l ("down")  (S (x,y) l ls state)
                                   | str == "left" = S (x,y-1) l ("left")  (S (x,y) l ls state)
                                   | str == "right"= S (x,y+1) l ("right") (S (x,y) l ls state)
```

2- Helper Functions :

1] Remove

Takes two inputs which is an element “e” and a list of same type and remove all elements in the List which is equal to “e”.

```
-- remove [remove element from a list]

-- Type

remove :: Eq a => a -> [a] -> [a]

-- Body

remove e (h:t) = [ x | x <- (h:t) , e /= x ]
```

2] Search Helper

Iterate over List of State and return first State that is Goal

```
-- Type

searchH :: [MyState] -> MyState

-- Body
searchH [] = Null
searchH ((S (x,y) l ls state):t) = if isGoal (S (x,y) l ls state)
                                   then (S (x,y) l ls state)
                                   else searchH  t
---------------------------
```

3- Required Functions

1- Up

Using Defined High Order Function “direction” , we are going to insert as an input String “up” to ensure moving up and pass input in “up” function to the “direction” funtion and return new State.

```
-- Type

up :: MyState -> MyState

-- Body

up state = direction "up" state
```

2- Down

Using Defined High Order Function “direction” , we are going to insert as an input String “down” to ensure moving down and pass input in “down” function to the “direction” funtion and return new State.

```
-- Type
down :: MyState -> MyState
-- Body
down state = direction "down" state
```

3- Left

Using Defined High Order Function “direction” , we are going to insert as an input String “left” to ensure moving left and pass input in “left” function to the “direction” funtion and return new State.

```
-- Type
left :: MyState -> MyState
-- Body
left state = direction "left" state
```

4- Right

Using Defined High Order Function “direction” , we are going to insert as an input String “right” to ensure moving right and pass input in “right” function to the “direction” funtion and return new State.

```
--Type
right :: MyState -> MyState
-- Body
right state = direction "right" state
```

5- Collect

Check if List of Cells of Mines contain a Cell which is Same as Robot's Cell aka position using predefend function "elem"

IF True → remove it using "remove" helper function to remove Cell

from List of Cells of Mines in MyState

and return new State.

IF False → return Null

```
-- Type

collect :: MyState -> MyState

-- Body

collect (S (x,y) l ls state) |  elem (x,y) l =  (S (x,y) (remove (x,y) l) "collect" (S (x,y) l ls state))
                             |  otherwise = Null
```

6- Next MyStates

Generate all possible Next States

Up
Down
Left
Right
Collect

And insert them in a list and remove any Null States which indicate that action is not achievable.

```
-- Type

nextMyStates:: MyState -> [MyState]

-- Body

nextMyStates state = [ x | x <- list , x /= Null]
       where {

        nextUp      = up    state;
        nextDown    = down  state;
        nextLeft    = left  state;
        nextRight   = right state;
        nextCollect = collect state;
        list        = [nextUp,nextDown,nextRight,nextLeft,nextCollect];

       };
```

7- Goal

Check if List of Cells of Mines in MyState is Empty

IF True → return True (indicating that all mines are collected)

IF False → return False (indicating that there is still mines not collected)

```
-- Type

isGoal :: MyState -> Bool

-- Body

isGoal (S (x,y) l ls state) | l == [] = True
                            | otherwise = False
```

8- Search

Check if head of List of MyState is a Solution then return it back

Otherwise, generate all possible next States of Current State using

"nextMyStates" of head of list and insert them at back of list so to get

Shortest possible solution.

Note : we used Helper function so that check if any of States in List ist a Goal before Concatinating next State of head.

```
-- Type

nextMyStates:: MyState -> [MyState]

-- Body

nextMyStates state = [ x | x <- list , x /= Null]
       where {

        nextUp      = up    state;
        nextDown    = down  state;
        nextLeft    = left  state;
        nextRight   = right state;
        nextCollect = collect state;
        list        = [nextUp,nextDown,nextRight,nextLeft,nextCollect];

       };
```

9- Construct Solution

Take as an input a User defined data type MyState and take String that represent last action performend and generate a list of Strings showing all action made to reach current State

```
-- Type

constructSolution:: MyState ->[String]

-- Body

constructSolution (S (x,y) l ls state) | flag = constructSolution state ++ [ls]
                                       | otherwise = []
        where {

          flag = ls == "collect" || ls == "up" || ls == "down" || ls == "left" || ls == "right";

        };
```

10- Solve

- Take as an input a Cell which indicate current position of Robot and list of Cells indicating postion's of mines
- Set initial `initialState = (S cell (h:t) "" Null);` condition to as no action is still made
- Search for shortest possible Solution using "search" function
- Return back required actions for Solution to achieve as a List of String using "constructSolution" function.

```
-- Type

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

-- Body

solve cell (h:t) = solution
      where{

        initialState = (S cell (h:t) "" Null);
        goalState    = search [initialState];
        solution     = constructSolution goalState;

      };
```

First Grid

/	0	1	2	3
0	X			R
1				
2		X		
3				

Robot Postion: (0,3)

Mine’s Position : (0,0) , (2,1)

Answer

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

Second Grid

/	0	1	2	3
0	X			
1				
2		R		
3		X		

Robot Postion: (2,1)

Mine's Position : (0,0) , (3,1)

Answer

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

Bonus First Grid

/	0	1	2	3	4	5
0	X					
1				X		
2		R			X	
3						
4				X		X
5	X				X	

Robot Position: (2,1)

Mine's Position: (0,0),(5,0),(1,3),(2,4),(4,3),(5,4),(4,5)

```
Main> solve (2,1) [(0,0),(5,0),(1,3),(2,4),(4,3),(5,4),(4,5)]
["right","right","down","down","collect","down","right","collect","right","up","collect","up","up","left","colle
ct","up","left","collect","up","left","left","left","collect","down","down","down","down","down","collect"]
```

Bonus Second Grid

/	0	1	2	3	4	5	6	7	8	9
0										
1				X			X			
2		R			X					
3								X		
4				X		X				
5										
6										
7					X		X			
8										
9				X			X			

Robot Position: (2,1)

Mine's Position: (1,3),(1,6),(2,4),(3,7),(4,2),(4,5),(7,4),(7,6),(9,2),(9,6)

```
Main> solve (2,1) [(1,3),(1,6),(2,4),(3,7),(4,2),(4,5),(7,4),(7,6),(9,2),(9,6)]
["down","down","right","collect","down","down","down","down","down","collect","right","right","right","right","collect","up","u
p","collect","left","left","collect","right","up","up","up","collect","right","right","up","collect","up","up","left","collect"
,"left","left","down","collect","up","left","collect"]
```