CSEN 403: Concepts of Programming Languages

Project 2

Team 67

Mathew Hany	T16	52-21647
Mark Mahrous	T07	52-23533
Boles Waheed	T16	52-22774
Rafeek Bassem	T16	52-11911

Functions we implemented

Function Name	Short Description
`up :: MyState -> MyState`	Takes a state and returns a new state after moving the robot up or **Null** if the robot will be out of the grid.
`down :: MyState -> MyState`	Takes a state and returns a new state after moving the robot down or **Null** if the robot will be out of the grid.
`left :: MyState -> MyState`	Takes a state and returns a new state after moving the robot left or **Null** if the robot will be out of the grid.
`right :: MyState -> MyState`	Takes a state and returns a new state after moving the robot right or **Null** if the robot will be out of the grid.
`collect :: MyState -> MyState`	Takes a state and if the robot is on a mine cell, it will collect it and return the new state, otherwise it will return **Null**.
`nextMyStates :: MyState -> [MyState]`	Takes a state and returns the states after moving the robot up, down, left and right as well as collecting the mine at its location. It will not include any Null states.
`isGoal :: MyState -> Bool`	Checks the current state, if there are no mines left to collect it returns True, otherwise it returns False.
`search :: [MyState] -> MyState`	Takes a list of states, and takes the first state, if its a goal it returns it, otherwise it appends all the possible next states from this first state at the end of the list, and it calls itself recursively on the new list.
`constructSolution :: MyState -> [String]`	Takes a state and returns a list of all actions to take to reach this state.
`solve :: Cell -> [Cell] -> [String]`	Takes the location of the robot and a list of the mines' locations, and returns a list of the actions to take to win the game.
`elem :: (Foldable t, Eq a) => a -> t a -> Bool`	Takes an element and a list, and returns True if the element is a member of the list or False otherwise.

`delete :: Eq t => t -> [t] -> [t]`	Take an element and a list and return the list after removing the element from it.
`notNull::MyState -> Bool`	Takes a state and returns True if the state is not null, or False if the state is null.

Type Defined

`Name`	Definition	Description
`Cell`	(Int, Int)	A pair of integers
`MyState`	Null | S Cell [Cell] String MyState deriving (Show, Eq)	Null => The Initial State S is a constructor that takes the position of the robot, the list of mines, a string indicating the last action, and the previous state..

Screenshots

First Run

	0	1	2	3
0			X	
1				R
2	X			
3				

```
Type :? for help
Hugs> :load "F:\\GUC\\Semester4\\Courses\\CSEN403\\project.hs"
Main> solve (1,3) [(0, 2),(2, 0)]
["up","left","collect","down","down","left","left","collect"]
```

Second Run

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

```
Type :? for help
Hugs> :load "F:\\GUC\\Semester4\\Courses\\CSEN403\\project.hs"
Main> solve (0, 0) [(1, 0), (2, 1) , (3, 2)]
["down","collect","down","right","collect","down","right","collect"]
Main>
```

Bonus

Functions used in the Bonus

Function Name	**Short Description**
`distance :: Cell -> Cell -> Int`	Takes 2 cells and calculates the distance between them
`nearestCell :: Cell -> [Cell] -> Cell`	Takes a cell A and a list of cells and returns the nearest cell in the list to cell A.
`moveToX :: Cell -> Cell -> [String]`	Takes the start cell and the end cell, and returns a list of instructions in the vertical axis (up or down) to reach from the start cell vertical position to the end cell vertical position.
`moveToY :: Cell -> Cell -> [String]`	Takes the start cell and the end cell, and returns a list of instructions in the horizontal axis (left or right) to reach from the start cell horizontal position to the end cell horizontal position.
`solve :: Cell -> [Cell] -> [String]`	Takes the location of the robot and a list of the mines' locations, and returns a list of the actions to take to win the game.
`delete :: Eq t => t -> [t] -> [t]`	Take an element and a list and return the list after removing the element from it.

Type Defined

`Name`	Definition	Description
`Cell`	(Int, Int)	A pair of integers

Screenshots for the bonus

First Run

	0	1	2	3	4	5	6	7	8	9	10
0	X										
1											
2											
3											
4											
5			X								
6										X	
7											X
8											
9											X
10	X										R

```
Type :? for help
Hugs> :load "C:\\Code\\UniProjects\\haskell\\bonus2.hs"
Main> solve (10, 10) [(9, 10), (7, 10), (6, 9), (10, 0), (5, 2), (0, 0)]
["up","collect","up","up","collect","up","left","collect","up","left","left","le
ft","left","left","left","left","collect","up","up","up","up","up","left","left"
,"collect","down","down","down","down","down","down","down","down","down","down"
,"collect"]
Main>
```

Second Run

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

```
Type :? for help
Hugs> :load "C:\\Code\\UniProjects\\haskell\\bonus2.hs"
Main> solve (5, 6) [(6, 6), (6, 4), (5, 4), (0, 0), (2,
3), (4, 2)]
["down","collect","left","left","collect","up","collect"
,"up","left","left","collect","up","up","right","collect
","up","up","left","left","left","collect"]
```