Report Team 41

Implemented Functions :-

First to clarify , we treat the robot position as (x,y)

(1) up

Takes as an input a State , decrements the x coordinate of the robot only if it wont get out of the board else it will return Null

(2) down

Takes as an input a State , increments the x coordinate of the robot only if it wont get out of the board else it will return Null

(3) left

Takes as an input a State , decrements the y coordinate of the robot only if it wont get out of the board else it will return Null

(4) right

Takes as an input a State , increments the y coordinate of the robot only if it wont get out of the board else it will return Null

(5) collect

Takes as an input a State , if the robot is currently standing above a mine , then it will collect it (the mine will be removed from the array of mines to be collected) else it will return Null

(6) nextMyStates

Takes as an input a State , applies on it up , down , right , left , collect functions , then puts the results of all those functions into an array only if , the result of the function isn't Null

(7) isGoal

Takes as an input a State , checks if its array of mines to be collected is empty or not , if empty return True else return False

(8) search

Takes as an input a list of State , check if the 1st state in the list is a goal state (apply isGoal function on the 1st element) , if True then return this element

Else , apply search again on the appending of the (rest of the list) with (applying nextMyStates on this element)

(9) constructSolution

Takes as an input a State , returns a list of strings , the strings represent the name of the functions that need to be applied to go from the initial state to the input state

(10) solve

Takes as an input a cell and a list of cells , the cell represent the stating position of the robot while the list of cells represent the positions of the mines , it returns a list of strings , the strings represent the name of the functions that need to be applied to win the game (collect all mines)

Sample runs :-

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

Now to make the game work on bigger grids :-

We need to change the down and right functions , instead of being bounded to the board limits , it will be bounded to the x,y of the first mine in the list of mines to be collected .

Sample runs of bigger grids :-

```
Main> solve (5,0) [(2,2),(1,2)]
["up","up","up","right","right","collect","up","collect"]
Main> solve (5,0) [(4,2),(3,1)]
["up","up","right","collect","down","right","collect"]
Main>  solve (6,0) [(4,5),(3,1)]
ERROR - C stack overflow
Main>  solve (6,1) [(4,4),(3,1)]
ERROR - C stack overflow
Main> solve (6,1) [(4,4),(3,1)]
ERROR - C stack overflow
Main> solve (5,5) [(4,4),(3,1)]
ERROR - C stack overflow
Main> solve (3,0) [(5,2),(1,6)]
ERROR - C stack overflow
Main> solve (3,0) [(5,2),(1,2)]
ERROR - C stack overflow
Main> solve (3,0) [(4,2),(1,2)]
["down","right","right","collect","up","up","up","collect"]
Main>
```

As one can see , it gives stack overflow in some cases , Why ?

Limitations :-

The mines are saved in an array , it takes O(n) to search . A better implementation will be using hash tables instead of array as hash table search in O(1).