CS 403 project 2

Type Cell=(Int,Int) to define a new type consisting of 2 Integers

data MyState = Null | S Cell [Cell] String MyState deriving(Show,Eq):new data type with 2 definitions either null or S Cell [Cell] String MyState. In addition, deriving show to be able to print values of a defined type and deriving eq to able to check for inequality

```
up:: MyState -> MyState

up (S (b,c) m s t)= if b>0  then (S ((b-1),c) m "up" (S (b,c) m s t))
                            else Null
```

up takes in a MyState and returns a MyState with the robot position shifted up one cell if possible. Otherwise Null is returned.

```
down:: MyState -> MyState

down (S (b,c) m s t)= if b<3  then (S ((b+1),c) m "down" (S (b,c) m s t))
                              else Null
```

down takes in a MyState and returns a MyState with the robot position shifted down one cell if possible. Otherwise Null is returned.

```
left:: MyState -> MyState

left (S (b,c) m s t)= if c>0  then (S (b,(c-1)) m "left" (S (b,c) m s t))
                              else Null
```

left takes in a MyState and returns a MyState with the robot position shifted left one cell if possible. Otherwise Null is returned.

```
right:: MyState -> MyState

right (S (b,c) m s t)= if c<3  then (S (b,(c+1)) m "right" (S (b,c) m s t))
                               else Null
```

right takes in a MyState and returns a MyState with the robot position shifted right one cell if possible. Otherwise Null is returned.

```
collect:: MyState -> MyState

collect (S (b,c) m s t)|(length m)==0=Null

                | (length m)==1 && elem (b,c) m  = (S (b,c) [] "collect" (S (b,c) m s t))

                                | (length m)==2 && elem (b,c) m && (m!!1)==(b,c) =
(S (b,c) (init m)  "collect" (S (b,c) m s t))

                                        |(length m)==2 && elem (b,c) m &&
(m!!0)==(b,c) = (S (b,c) [m!!1]  "collect" (S (b,c) m s t))

                                        | otherwise=Null
```

Collect checks the list of mines and if the robot happens to be at the same position as a mine,the mine cell is removed from the list.This happens by checking the length of the list first and it the length is zero, null is returned. Otherwise,if the length is 1 and robot is at the same position, a new Mystate with an empty list is returned. Moreover,if the length is 2,we first check which mine should be removed and then a new Mystate with a new list is returned.Finally,if the robot position does not match any of mines cells null is returned.

```
nextMyStates::MyState->[MyState]

nextMyStates v = removeItem ([up v] ++ [down v] ++ [left v] ++ [right v] ++
[collect v])

areTheySame ::  MyState-> [MyState]

areTheySame  y | y == Null = []

            | otherwise = [y]

removeItem :: [MyState]-> [MyState]

removeItem []=[]

removeItem (y:ys) = areTheySame y ++ removeItem ys
```

nextMyStates is responsible to get all the possible MyStates of the robots current MyState.removeItem is responsible to remove all the Null elements in a list of MyStates.

```
isGoal::MyState->Bool

isGoal(S (b,c) m s t)=if (length m)==0 then True

                      else False
```

isGoal checks if the current MyState of the robot has an empty list of mines which indicates **the** robot has reached its goal and true is returned. Otherwise, false is returned.

```
search::[MyState]->MyState

search []=Null

search (h:t)=if isGoal h then h

             else search (t ++ nextMyStates h)
```

search takes in a list of MyStates and returns the Mystate with the reached goal.This happens by checking whether the head of the MyStates list isGoal,if true the head is returned. Otherwise, a recursive call is done on the tail of the list plus the nextMystates of the discarded head.

```
constructSolution:: MyState ->[String]

constructSolution (S (b,c) m "" t) = []

constructSolution (S (b,c) m s t)=constructSolution t ++ [s]
```

ConstructSolution takes in a MyStates and returns a string with the actions taken from the initial state to the current MyState.

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

solve c l =constructSolution (search (nextMyStates (S c l "" Null)))
```

First,solve creates the intial Mystate of the robot with the robot cell and list of mines.Then,it finds the possible nextStates of the robot's position and and calls search on the next states to find which state will allow the robot to reach the goal.Finally constructSolution is called to find the set of actions to reach the goal.

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

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

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

Bonus Part

Type Cell=(Int,Int) to define a new type consisting of 2 Integers

data MyState = Null | S Cell [Cell] String MyState deriving(Show,Eq):new data type with 2 definitions either null or S Cell [Cell] String MyState. In addition, deriving show to be able to print values of a defined type and deriving eq to able to check for inequality

```
up:: MyState -> MyState

up (S (b,c) m s t)= if b>0  then (S ((b-1),c) m "up" (S (b,c) m s t))
                            else Null
```

up takes in a MyState and returns a MyState with the robot position shifted up one cell if possible. Otherwise Null is returned.

```
down:: MyState -> MyState

down (S (b,c) m s t)= if b<10 then (S ((b+1),c) m "down" (S (b,c) m s t))
                              else Null
```

down takes in a MyState and returns a MyState with the robot position shifted down one cell if possible. Otherwise Null is returned.

```
left:: MyState -> MyState

left (S (b,c) m s t)= if c>0  then (S (b,(c-1)) m "left" (S (b,c) m s t))
                              else Null
```

left takes in a MyState and returns a MyState with the robot position shifted left one cell if possible. Otherwise Null is returned.

```
right:: MyState -> MyState

right (S (b,c) m s t)= if c<10  then (S (b,(c+1)) m "right" (S (b,c) m s t))
                              else Null
```

right takes in a MyState and returns a MyState with the robot position shifted right one cell if possible. Otherwise Null is returned.

```
collect:: MyState -> MyState

areTheySame1 :: Cell -> Cell-> [Cell]


areTheySame1 (a,b) (c,d) | (a == c && b==d) = []
            | otherwise = [(c,d)]


removeItem1 :: Cell -> [Cell] -> [Cell]

removeItem1 a []=[]


removeItem1 (a,b) ((c,d):ys) = areTheySame1 (a,b) (c,d)  ++ removeItem1 (a,b) ys


collect (S (b,c) m s t1 ) =
                if(elem (b,c) m)
                    then (S (b,c) (removeItem1 (b,c) m ) "collect" (S (b,c) m s t1))
                    else  Null
```

Collect checks the list of mines and if the robot happens to be at the same position as a mine,the mine cell is removed from the list.This happens by (areTheySame1) method which removes the cell of the mine from the list if it has the same location as the robot and it keeps checking on the rest of the list till it find the mine that has the same location and remove it.But if it didn't find a mine with the same location an unchanged list will be returned.

```
nextMyStates::MyState->[MyState]

nextMyStates v = removeItem ([up v] ++ [down v] ++ [left v] ++ [right v] ++ [collect v])
```

```
areTheySame ::  MyState-> [MyState]
areTheySame  y | y == Null = []
               | otherwise = [y]
removeItem :: [MyState]-> [MyState]
removeItem []=[]
removeItem (y:ys) = areTheySame y ++ removeItem ys
```

nextMyStates is responsible to get all the possible MyStates of the robots current MyState.removeItem is responsible to remove all the Null elements in a list of MyStates.

```
isGoal::MyState->Bool
isGoal(S (b,c) m s t)=if (length m)==0 then True
                      else False
```

isGoal checks if the current MyState of the robot has an empty list of mines which indicates **the** robot has reached its goal and true is returned. Otherwise, false is returned.

```
search::[MyState]->MyState
search []=Null
search (h:t)=if isGoal h then h
             else search (t ++ nextMyStates h)
```

search takes in a list of MyStates and returns the Mystate with the reached goal.This happens by checking whether the head of the MyStates list isGoal,if true the head is returned. Otherwise, a recursive call is done on the tail of the list plus the nextMystates of the discarded head.

```
constructSolution:: MyState ->[String]
constructSolution (S (b,c) m "" t) = []
constructSolution (S (b,c) m s t)=constructSolution t ++ [s]
```

ConstructSolution takes in a MyStates and returns a string with the actions taken from the initial state to the current MyState.

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

solve c l =constructSolution (search (nextMyStates (S c l "" Null)))

First,solve creates the intial Mystate of the robot with the robot cell and list of mines.Then,it finds the possible nextStates of the robot's position and and calls search on the next states to find which state will allow the robot to reach the goal.Finally constructSolution is called to find the set of actions to reach the goal.

We chose a 11x11 grid of cells and three mines

```
Main> solve (5,0) [(5,2),(5,1),(5,3)]
["right","collect","right","collect","right","collect"]

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