Snowplow Problem

Yulun Zeng, Yuyuan Liu, Lahray Rivera, Sydney McCormick

Prerequisites & Inputs

- Create grid size by asking for inputs:
  - Input X is columns
  - Input Y is rows
  - Each input must be >2
- User will pick a method
  - 1: Bottom left to top right
  - 2: Clears rows then column
  - 3: Random

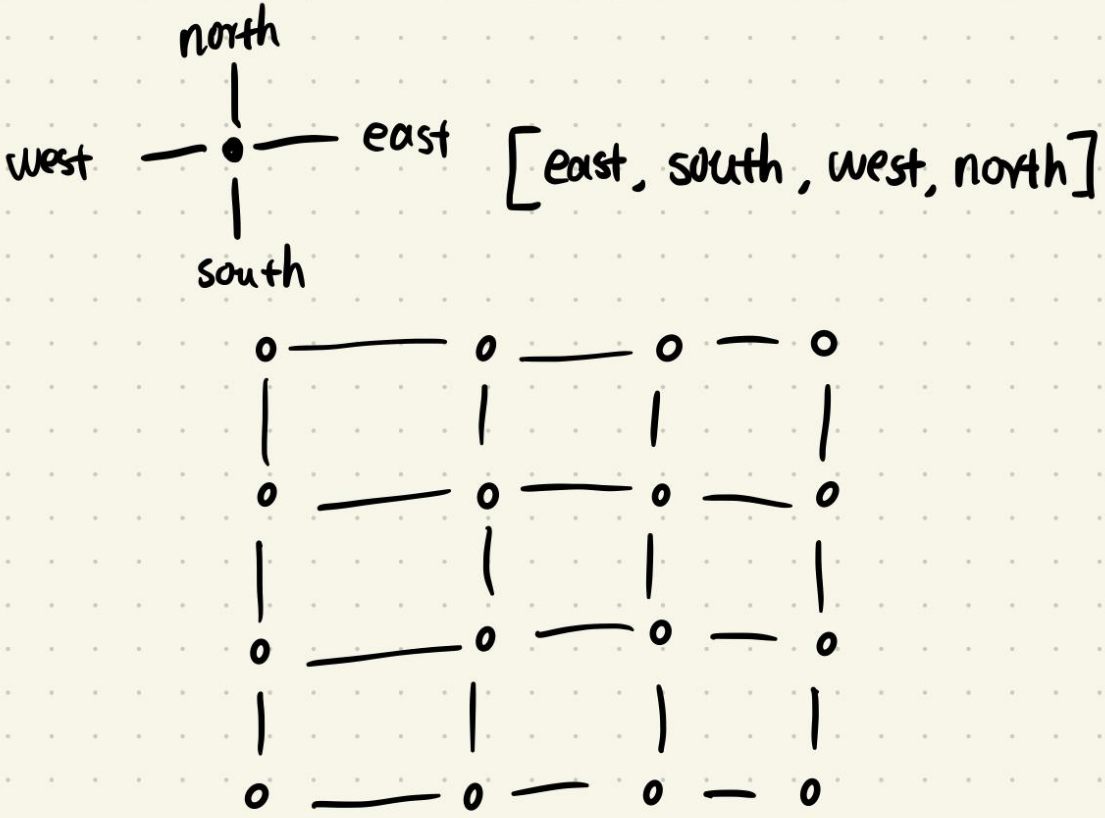
```
1   import random
2   # Ask for customized road grid size
3   print("Precondition: Row/column should be greater than 2")
4   x=(int)(input("How many columns?"))
5   y=(int)(input("How many rows?"))
6   # If the dimension of the grid is less than 2, then one of the method won't work, exit
7   if (x<=2 or y<=2):
8           exit()

13  choice=(int)(input("Which method would you like to choose? 1 or 2 or 3? (Just choose a number)"))
```

createNode() Function

- Node is the point connecting the road in 4 directions
- North, South, East, West
- x-y coordinate of the node is the same as the index in the 2D list.

```
16  def createNode():
17          global node
18          node=[]
19          for i in range (x):
20                  node.append([])
21                  for j in range (y):
22                          node[i].append([0,0,0,0])
```


north
west
east
south
[east, south, west, north]

createRoad() Function

- Creates the amount of roads in the grid
- Each road/edge will have a unique number

```
25  def createRoad():
26          global road
27          road=[]
28          for i in range (2*x*y-x-y):
29                  road.append(i+1);
```

createMap() Function

- We will call this function later to generate a map
- The map will have a customized size and labeled roads
- This function uses the createRoad() and createNode() functions
- The inner and outer roads are labeled using loops

```
32  def createMap():
33          global node, x, y, road,timer, moveX,moveY
34          createRoad()
35          createNode()
36          timer=0
37          moveX=0
38          moveY=0
```

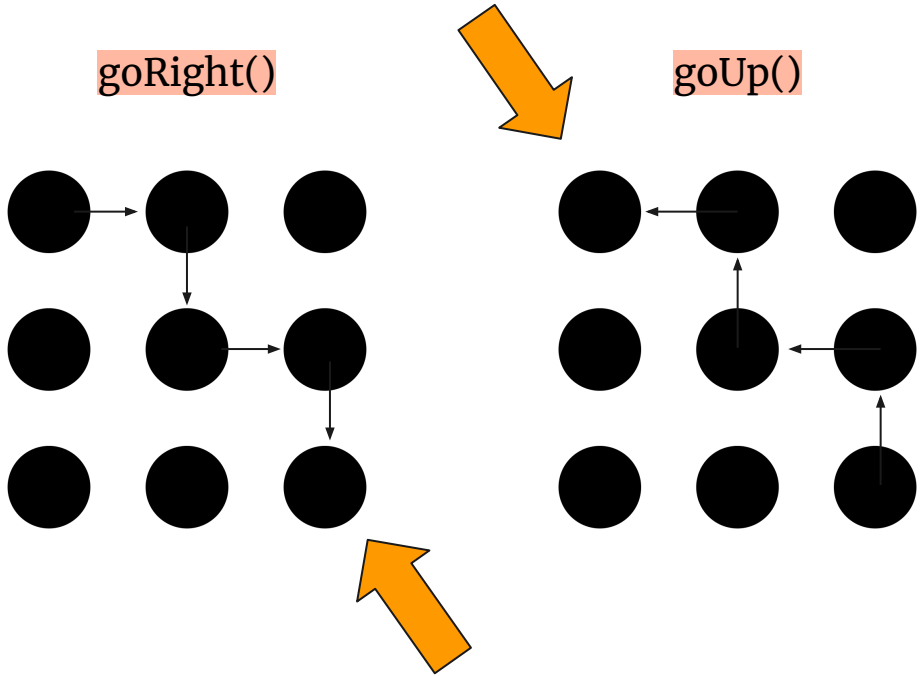
move(direction) Function

- Uses global variables to move the robot
- Each time the robot moves, it:
  - Updates location
  - Counts time
  - Marks cleared paths

```
83  def move(direction):
84          global timer,moveX,moveY,road,node
85          if(direction=="left"):
86                  print("(",moveX,",",moveY,") to (",moveX-1,",",moveY,")")
87                  road.append(node[moveX][moveY][2])
88                  node[moveX][moveY][2]=0
89                  node[moveX-1][moveY][0]=0
90                  moveX-=1
```

goRight() and goUp() Functions

- These functions are used in method 1
- goRight() moves robot to lower right of map
- goUp() moves robot to upper left of map



1

```
if (choice==1):
        move("right")
        goUp()
        while(True):
                if(moveX==0):
                        if(node[moveX][moveY][1]!=0 and node[moveX][moveY][3]!=0):
                                move("down")
                                move("up")
                                move("up")
                                goRight()
```

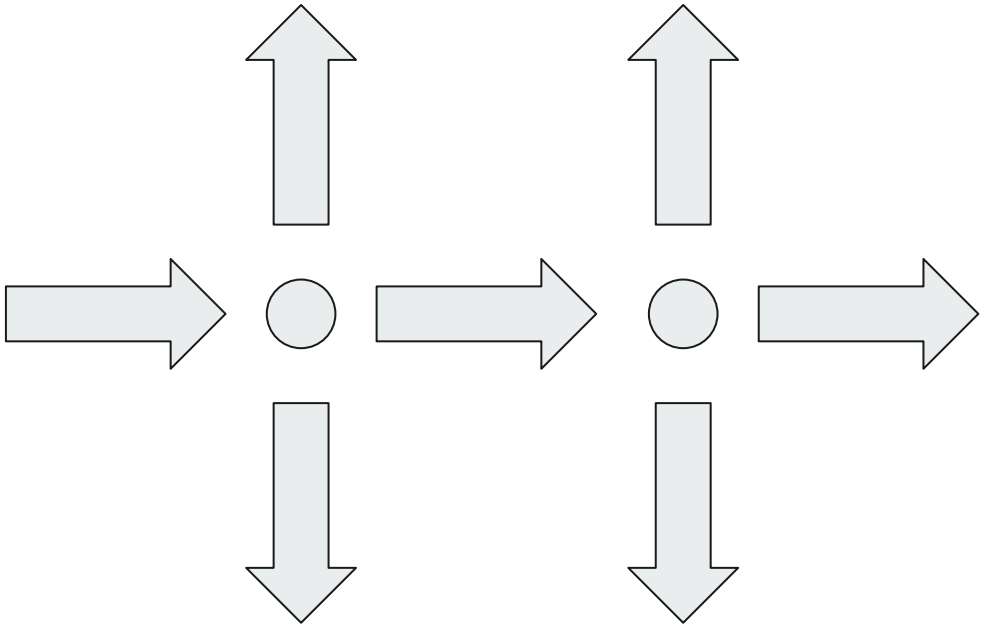
2

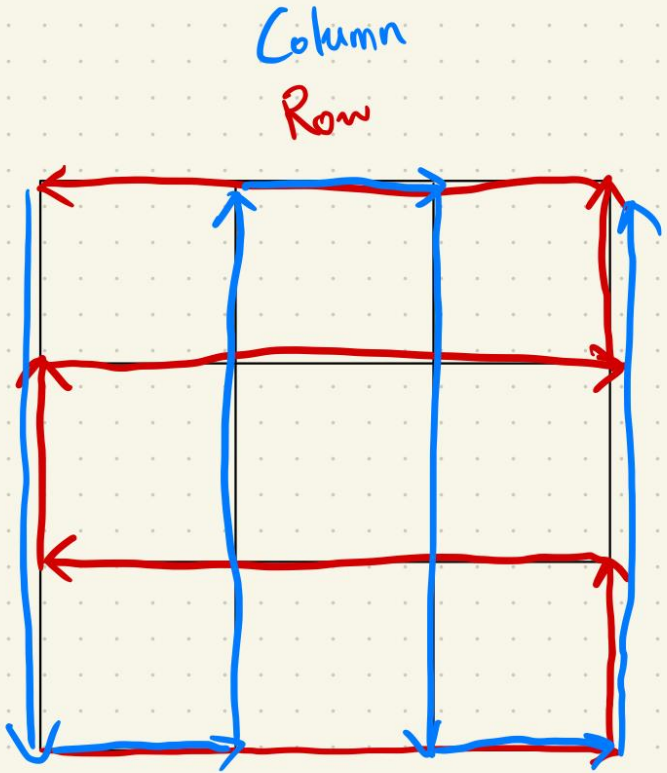
```
elif (choice==2):
        while(True):
                for i in range (x-1):
                        move("right")
                if(moveY!=y-1):
                        move("up")
                for i in range (x-1):
                        move("left")
                if(moveY!=y-1):
                        move("up")
                else:
                        break
```

3

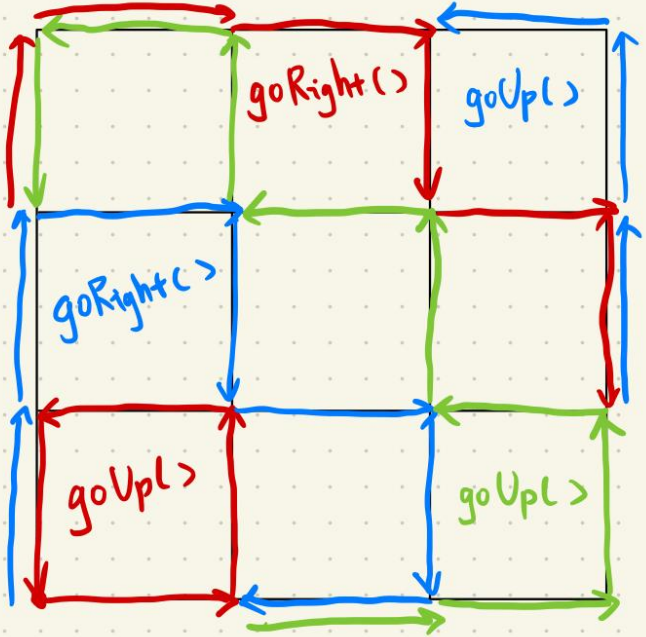
```
elif(choice==3):
        trials=[]
        #Do 50 trials
        for tri in range (50):
                print("Trial ",tri+1)
                createMap()
                while(True):
                        countStop=0
                        for i in range(x):
                                for j in range (y):
                                        if(node[i][j]==[0,0,0,0]):
                                                countStop+=1
```

Default:





Option 2:



Option 1:

Creating the Methods

- First, we call the createMap() function
- The program will run 1 of 3 options, chosen by the user

Option 1:

Moves through the grid by moving in a “step” pattern from the bottom left to the upper right

Option 2:

Cleans the rows first then finishes the columns of the grid

Option 3:

Default movement: Chooses a random path of reachable nodes connected by snowy edges

Finishing

- The program records the time used in each trial
- The program prints the time it took
- The program prints a grid of 0's, showing the roads are cleared

```
                #Record the time used in each trial
                trials.append(timer)
        #Set get the fastest time
        timer=min(trials)
        #Prints the trial with the fastest time
        print("Trial", trials.index(timer)+1," took the least time")
#Indicates the total/best time used in specific case.
print("Took ",timer," minutes to clear.")
#This should show a list of lists of lists of 0, which means all the roads are cleared
for k in node:
        print(k)
```

Conclusion

- We concluded that the default method, or option three, that randomly plowed the grid, was the least effective of the three methods, but running it for enough trials has a chance to produce the best result.
- As the size becomes larger option 3 becomes less efficient
- In smaller grid sizes option 3 can be faster since there are less roads to cover