Project 1 Maze Navigation

## Sudo code:

Int index\_to\_the\_right(index):

If Maze[index] != ‘s’,’e’,’p’ do

Return index to the right

Else

Return -1

Int index\_to\_the\_top(index):

If Maze[index] != ‘s’,’e’,’p’ do

Return index to the top

Else

Return -1

Int index\_to\_the\_bottom(index):

If Maze[index] != ‘s’,’e’,’p’ do

Return index to the bottom

Else

Return -1

Int index\_to\_the\_left(index):

If Maze[index] != ‘s’,’e’,’p’ do

Return index to the left

Else

Return -1

Int[] getXY(n)

Int getDistance(n, t)

n1 = getXY(n)

n2 = getXY(t)

return squreRoot((n1[0]-n2[0])^2+(n1[1]-n2[1])^2)

//A\* solution

//find all paths that are available

//select the closest path based on distance formula getDistance()

//mark it with p

//if there’s a dead end backtrack or loop (the end is the path itself) backtrack

//and fill in a wall every time backtracked

//backtrack until an empty path is found

//Then proceed

//repeat process until end is found

end <- index of end point

step <- 0

solution <- mutable list

**While** solution[step] != end **do**

nextStep <- solution[step]

availablePaths <- mutable list

//append all the available elements to available paths

**if**(index to the right != -1)**and** (element to the right != ‘s’) **do**

append index to the right to available paths

**if**(index to the left != -1)**and**(element to the right != ‘s’) **do**

append index to the left to available paths

**if**(index to the top != -1)**and**(element to the right != ‘s’) **do**

append index to the top to available paths

**if**(index to the bottom != -1)**and**(element to the right != ‘s’) **do**

append index to the bottom to available paths

//get shortest path from availablePaths and add it to nextStep

**If** availablePaths != null **do**

Least <- getDistance(availablePaths[0], end)

nextStep <- availablePaths[0]

**for** i in availablePaths **do**

**if** least > getDistance(i, end) **do**

nextStep <- i

**else**

nextStep<- solution[step]

**if** nextStep == start **do**

break

**if** nextStep == solution[step] **do**

maze[nextStep] = ’\*’

solution.pop()

step—

**else**

solution.append(nextStep)

**if** maze[nextStep] != ‘e’ **do**

maze[nextStep] = ‘p’

Step++

# Proof of efficiency class

Base Case = number of times a place holding character is written to the Maze array

There are two if statements in the while loop that could write characters in the Maze array

So the equation for t(n) = sum(n,0)2 = n-0+2 = n+2. g(n) = n

1/2n <= n+2 <= 2n if n = 3

Therefore t(n) = Θ(n).