The Maze Problem (project 4)

Project #4 - Due: April 20th, 2009, at the start of class. (Grading: Graded for 100% credit on 5/20, 80% credit on 5/22, after that date – no credit)

Purpose: To give the student experience working with recursion and various STL classes - vector and so forth.

Introduction

This program will find a treasure within any given maze using recursion with exhaustive searching with backtracking.

The program reads in the description of a maze from a text file. This is what an example maze file (text file) looks like:

```
### ##########################################################
### ### ## #      ### ############ ###### #######    ###
### ### ## ###### ###                              ## ###
### ### ## ###### ### ################ ########## #   ##
### ###    ###### ###               ## ############ # ##
### ############# ########## #########           X# # ##
###              ##       # ######################## # ##
#### # ############ ##### #                   ###### # ##
#### #               ##### ######################### #  #
#### ################################################## ####
####                                                     ###
##########################################################
```

The spaces represent open passages through the maze. The #'s represent walls. The X represents the treasure.

Some rules regarding the maze:

1. There is only one entrance and it can be on either of the four outer walls.
2. There may be no treasure in the maze, but if there is one - it will be the only one.

Your program:

1. Asks the user to input the name of the maze file.
2. Loads the contents of the file into a two-dimension array (using a vector of vectors).
3. Next your program will display the maze using the format seen above.
4. Then your program searches the outer walls for an entrance.
5. Finally, your program will find a path to the treasure (if it exists).
   a. Calls a recursive method to find the treasure. The job of each iteration of the recursive method is to find the next cell in the path to a solution.

Some hints about the design of the recursive method:

1. The method receives two int parameters that are the row and column of the

current position. (Obviously, your method must also be able to work with the maze - I am assuming your method is public to the maze object.)

2. The method returns a boolean indicating whether or not it found a treasure.
3. As you traverse to a particular square, mark it as part of a possible solution by changing it's contents to an asterisk (*). If it turns out not to be part of the solution, change the asterisk back to a zero.
4. There are two base cases that stop the recursion.
   a. "This square I'm on is the treasure." (Return true.)
   b. "This square I'm on is a dead end." (Return false.)
5. If neither base case is appropriate, check all the squares immediately adjacent to the current square to see if you can move into one or more of them. If you can, call the method recursively with the row and column of that square.

When the recursion ends, you should have a marked path to the treasure. Display the resulting maze on the console screen. For example, the final state of the above maze could be:

```
TREASURE FOUND
###*#########################################################
###*### ## #      ### ############ ###### #######****###
###*### ## ###### ###                  ************##*###
###*### ## ###### ### ################*########## #** ##
###*###    ###### ###               ##*#############*# ##
###*############## ########## ########************X##*# ##
###**             ##        # #######################*# ##
####*# ############# ##### #                  ######*# ##
####*#               ##### ########################*#  #
####*#######################################################*####
####************************************************** ###
#########################################################
```

Create for yourself one or two sample maze files and test possible cases. Your program must follow the <u>best object-oriented design principles</u>.

Also, use one of the other STL classes - lists, stack, queue, deque, and so forth to store directions to the treasure. The directions will contain all the necessary steps based on directions (N, S, E, and W).

Here's an example of the directions:

```
DIRECTIONS TO THE TREASURE
 S S S S S S E S S S S E E E E E E E E E E E E E E E E E E E E E E E E E E E
 E E E E E E E E E E E E E E E E E N N N N N N N E N N W W W S W W W W W W W W
 W W W S S S E E E E E E E E E E E
```

Testing:
On the Moses server - /gc/cs212/p2 - you will find an executable example of this program called "maze". Also, there are several example mazes - mza.txt, mzb.txt, mzc.txt, mzd.txt, mze.txt, mzz.txt.

You can execute the program by typing in /gc/cs212/p2/maze and then entering a maze file - like “/gc/cs212/p2/mzz.txt”

Also, you can copy all or any of the file to your directory for testing or inspiration.