CSCI 145 Week 5
Assignment 1

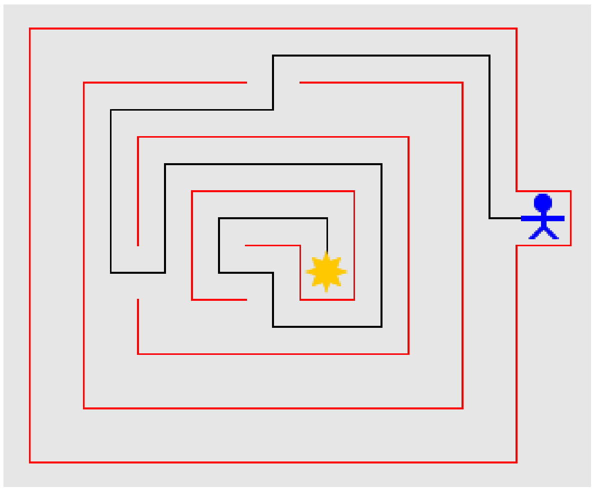
Assignment Description

In this assignment you will create a Java program to recursively solve mazes. Your program will not only solve the maze, but draw a picture of the solution. You must submit your program by 11:59 pm on Sunday, October 25th, 2015. This assignment will be worth 15% of your grade for the course.

Please read the entire assignment before beginning work on it. In particular, the order of development in “Suggested Steps” near the end, is not the same as the order of discussion of the features of your program.

Mazes

A maze consists of a grid of squares, which we will refer to as cells, a starting and ending cell. The goal is to find a path through the maze, from start to end. The maze will contain walls between cells that block movement. Shown below is an example maze. Walls are in red. The little blue figure is the starting location and the gold star is the ending location. The path from start to end is in black.



Solution Method

You will use recursion to solve the maze. The recursion works as follows, with the initial value for the current location being the starting location.

1. If the current location is the ending location, you are done. Return to the calling program reporting success. This is the base case of the recursion.
2. Mark that you have visited the current location.

3. From the current location, attempt to move in each of the four cardinal directions (up, down, left, and right).
   - You can move from any cell to an adjacent cell if there is no wall intervening and you have not yet visited that cell.
4. If you can move in one of the four directions, then recursively repeat starting at step 1 with the new current location. (This is the recursive step.)
   - If the recursive step returns reporting success, the maze has been solved. Return reporting success.
   - If the recursive step returns reporting failure, then repeat this step with a new direction. If all directions have been tried. Go to step 5.
5. If you have tried moving in all available directions and they have all failed, then
   a) unmark this cell has having been visited and
   b) return to the previous caller, reporting failure.

That's it. Using this method, a path will eventually be found if one exists. If none exists, this method will report failure

Requirements

In addition to solving the maze, your program must read command line arguments, read the description of the maze from an input file, and animate your search for a solution within the maze. Here are the details.

Input File

1. Your program must read a puzzle from a text file. The name of the file will be given as a command line argument. If the file cannot be opened for input (for example, if it cannot be found in the current directory), the program is to display an appropriate message and terminate.
2. Input file is organized as follows:
   a) The first line contains two numbers, h and w, the height and width of the puzzle. Both of these numbers will be between one and fifty, inclusive.
   b) The remaining lines are a picture of the maze. There will be 2h+1 lines in the picture. Each line will consist of 2w+1 characters, not including end-of-line characters.
   c) Even numbered lines (starting with line zero) represents the walls between rows of cells. The even numbered characters (again starting from zero) will be '+' characters indicating the corner of a cell. The odd numbered characters will be a minus sign ('-') or a space (' ') indicating a wall ('-') or no wall (' ').
   d) Odd numbered lines represent a row of cells. The even numbered characters will be a vertical bar ('|') or a space (' ') indicating a wall ('|') or no wall (' '). Odd numbered characters will be either a space (' ') or one of the letters ('S') for the start location or ('E') for the end location.

Here is an example of a simple input file:

```
2 3
+-+-+-+
|S|   |
+ + + +
|   |E|
+-+-+-+
```

You are allowed to assume that:

a) The file is correctly formatted as described above. You do not have to do any error checking on the input file.

b) There will be exactly one start and one end location in the maze description.

c) That all the "exterior" walls are in place. That is, there will always be walls that block movement that exits the maze.

You are not allowed to assume that:

a) There is a path from the start to the end. Your search for a path can fail.

b) That the exterior of the maze is a rectangle. The picture on page one shows an example of a non-rectangular maze.

Drawing the Maze

1. You must draw the maze in the same manner as shown in the picture on page 1. I have provided a class, `DrawMaze` that handles some of the more complicated aspects of this. `DrawMaze` uses the `DrawingPanel` class that is provided with the textbook. I have included both `DrawMaze` and `DrawingPanel` in `assign1.zip`. You are free to use any part of `DrawMaze` in your program. In particular, I do not expect you to reproduce the `drawStart` and `drawEnd` methods in `DrawMaze`.

2. You should make no modifications to `DrawingPanel.java`.

3. `DrawMaze` uses three values to do its work. Your program will set these three values, `borderwidth`, `cellsize`, and `sleeptime`, based on command line arguments. The default value of `sleeptime` should be 50 rather than the value of 1000 that it is used to initialize it. These values can be changed using command line arguments as described below.

4. Your program must animate your search. This should be done by having the recursive search by calling the `move` method in `DrawMaze` when you make a move. You will also need to call some sort of "unmove" method when your search backtracks. The easy way to undraw a move is to draw the same line using the color `BACKGROUND`. (The default color values are defined at the beginning of `DrawMaze`.) I will demonstrate my version of the program during class on Friday.

   If you need to, as part of your animation, you can redraw any part of the maze, or the whole maze, to cleanup damage that might have been done, for example when unmaking a move.

5. You are free to change anything you want to in `DrawMaze`. I expect that you will want to change the parameter lists for some of the methods that I have provided and add new methods

of your own.

6. There are some important decisions that need to be made for DrawMaze. I discuss these in more detail under Suggested Steps, below.

Command Line Arguments

1. Your program must accept one to four command line arguments. As in the past, if there are too many or too few arguments or if the arguments are not right or usable in some way, I expect you to produce an error message and exit the program. Refer back to Lab 3 for information on how to test your program using command line arguments.
2. The command line arguments are:
    1. The name of the maze file the program is to read. This argument is mandatory.
    2. A value for `cellsize`. This argument is the number of pixels in the edge of a cell in a drawing of the maze. This argument is optional. If it is provided, it should be an `int` greater than or equal to 10. If this argument is not provided, the default value for `cellsize` is 30.
    3. A value for `borderwidth`. This argument is the number of pixels that are allowed for the width of the border. This argument is optional. If it is provided, it should be an `int` greater than or equal to 5. If this argument is not provided, the default value for `borderwidth` is 40.
    4. A value for `sleeptime`. This argument is the number of milliseconds that the animation will wait after displaying a move. This argument is optional. If it is provided, it should be an int in the range 0 to 10000. (That's too fast to see the search to agonizingly slow.) If this argument is not provided, the default value for `sleeptime` is 50. You can see this variable being used in the `move` method in `DrawMaze`.

    You should note that the `DrawMaze` class that I have provided will adjust automatically for changes in these values. Your program should do the same. As long as you use the named variables and not their default values, you should not have any problem with this.

Searching the Maze

Your program must search the maze recursively. That's part of the purpose of this assignment.

As part of the search process, add the calls to `DrawMaze` that are necessary to animate the search.

Note that this maze search algorithm can be very inefficient. The text file maze6.txt can take a looong time to search if `sleeptime` is greater than zero. There are much more efficient ways to search the maze; but, those methods are beyond the scope of this class.

Suggested Steps

It may seem like there's a lot to do for this assignment. However, if you break the problem into pieces, none of the pieces is all that big. Also, remember to test each piece before you go on to the next one. That way you will have confidence that, in general, any new problems are problems with new code you have added. Here's a suggested order for the pieces:

1. Get the command line arguments working first. The code I have provided will respond to changes in the values in `DrawMaze`. This will allow you to test your code for handling the command line without having to do any additional work.
2. Read the input file describing the maze. You will need to use some sort of two dimensional array structure(s) to store the structure of the maze. If you want to be particularly careful with testing this, see if you can print out the structure of the maze in a format similar to (or identical to) the input format. A simpler alternative is to just print out the various variables that capture the structure of the maze and manually verify that the values are correct.
3. Display the maze. Update `DrawMaze` so that it will draw the maze as show above. In order to do this, you will need to decide how `MazeSolver` and `DrawMaze` will communicate. Both classes need to use the structure of the maze (the size, the wall locations, and the start and end locations) as part of their processing. You will discover that the most convenient form for one class is probably not the most convenient for the other. You have two options: (a) read the maze in one form and convert to the other form or (b) read the maze in one form and have the other class use that form, even if it's less convenient.

   The test for this is obvious: Does the maze as drawn match the input file.
4. Now implement the recursive search as described above. Implement the animation of the search as part of this step. By watching the animation, you can watch the progress of the search and get visual feedback allowing you to see if anything is going wrong.
5. Do final testing. I have provided seven samples mazes as part of `assign1.zip`, `maze0.txt` to `maze6.txt`. Here is a brief description of each of the mazes:
   - `maze0.txt` – a small trivial maze. This one is good for some initial testing. This one does not require any backtracking for the search.
   - `maze1.txt` – a maze with no walls. Again, this can be solved with no backtracking. However, this one depends on the marking visited cells correctly.
   - `maze2.txt` – the maze used to generate the figure on page one. This one requires both backtracking and marking of visited cells.
   - `maze3.txt` – a version of maze2 with a couple of extra walls. This one has no solution.
   - `maze4.txt` – a large rectangular maze that involves some searching.
   - `maze5.txt` – a large non-rectangular maze that involves some searching.
   - `maze6.txt` – a large rectangular maze that can involve a great deal of searching.
6. Do any final cleanup before submitting your program. Remember to remove and/or comment out any code you may have inserted to assist with testing and debugging.

References

1. The `DrawingPanel` class is described in Chapter 3G of the text book.
2. There are supplemental slides for `DrawingPanel` that can be found here: http://www.buildingjavaprograms.com/slides/3ed/ch03g-graphics.ppt.

3. Lab 3 provides some references on arrays and handling command line arguments.

Coding Standards

Your program should follow the standards described as part of Lab 1.

Your program will be graded on conformance to the coding standards as well as correct functionality.

Turn in your program on Canvas

You should turn in your versions of `MazeSolver.java` and `DrawMaze.java`. You should not turn in `DrawingPanel.java`. Your program will be tested with the standard version of `DrawingPanel` so you should remember not to make any modifications to it.