CSC 210 Program #5

Project Corndog ● Feb 20, 2008

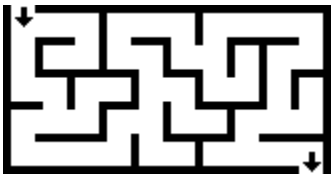
Logistics

Program #5 Project Corndog is:

- **Worth** 10 points, or 10% of your grade
- **Due** Saturday Mar 8, 2008 (noon)
- **Covers** graphs, breadth-first search, depth-first search, spanning trees, shortest path, backtracking, sets, recursion and/or stacks, and more!

Description

In Project Corndog, we will write software to play with mazes. Oh, I don't know… like this little guy:



This is called a "perfect maze" because it has one solution; it has a path to each "cell", or square, within the maze; and it has no borderless, open spaces.

You can almost close your eyes and imagine that this maze is a grid (rows and columns) of square cells. The example above has 5 rows and 10 columns. Yes? Each cell can have a wall on the top, right, bottom and left.

In fact, you can model this maze as a graph. Cells are vertices. An edge exists between cells that do **not** have a wall between them… because they are "connected." (At this point, someone raise their hand and ask me to draw this on the board. Thanks.)

In fact (again), if a perfect maze is as I described it with one path to each cell, then you would call that a spanning tree of the graph. Stay tuned.

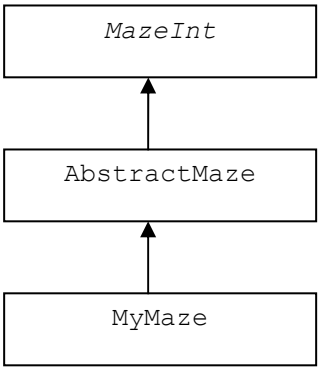
Project Corndog steps (pretty much in order of attack):

1. Design your data structures/classes to represent a maze
2. Draw a maze.
3. Read and write mazes to a file.
4. Generate a maze.
5. And finally, solve a maze.
6. Do your flair

Piece a cake. Gulp.

1. Design

Let's use the standard Java Collections interface/abstract class two-step. Here's my UML, sort of:



So, `MazeInt` is an interface, defining the maze contract. `AbstractMaze` is an abstract class that implements some of the interface methods without actually defining a data structure for the maze. Then, `MyMaze` does the heavy lifting.

A couple of other classes worth mentioning:

- `Cell` – represents a square in the maze
- `MazeFactory` – creates mazes from a file or generates them out of mid-air, with the greatest of ease
- `MazePanel` – put your maze in a handy panel surrounded by some GUI critters and paint it

Please don't change `MazeInt`. You can add to `AbstractGraph` if you like. If the `MazeInt` contract is cramping your style for some reason, let me know and we'll consider changing/improving it.

Please note that I have not defined the data structure or approach you will use to represent your maze. That's up to you. I'm not about to tell you what to do on the last program here. Gasp.

2. Draw

I think this is a pretty easy step. I just set my cell size (32, 40, whatever pixels square). Then, go cell by cell and draw its walls.

```
foreach row in your maze
  foreach column in this row
    draw the cell walls at (row, col)
```

You can spruce up this drawing to make it as pretty as you like.

3. Maze Files

Project Corndog should read and write maze descriptions to/from a file. This way we can exchange fun mazes and solve them.

I have a (gross) maze file format. I tinkered with the idea of using XML, and then I got over it. The good news is that I have skeleton code for you to read the files. Some links:

- csc210_maze_file_format_description.txt – text description of our files
- test01.txt – a small example file

I'll put my file reader skeleton on the k: drive.

4. Generate

While you are setting up your scaffolding (steps 1-3), you should be thinking about the task of automatically creating and solving mazes.

If your maze can be represented as a graph, then a perfect maze can be represented as a spanning tree on that graph. So, how can you create a "random" spanning tree? Three (similar) ways are depth-first, Prim's spanning tree, and Kruskal's spanning tree.

A depth-first approach starts at a random cell in the graph and then grows the spanning tree like a crystal from there. Like this:

1. Start at a random cell
2. Choose for a random neighbor cell you haven't visited yet
3a. If you find one, connect to it (or knock down its wall)
3b. Else try a previously visited cell (on a stack?)
4. Repeat steps 2 and 3 until you have visited (and connected to) each cell

In our book, Prim's algorithm is used to find a minimum spanning tree. Well, here all our edges have the same weight, so we just want a spanning tree. You can still use Prim's though by randomly adding an edge connected to the current set, rather than adding the smallest edge. Ditto for Kruskal's algorithm.

Please implement either Prim's or Kruskal's algorithm. You can do depth-first as well, if you want.

5. Solve

Actually, there are two maze programs described in our book:

- Pages 388-392 show a recursive/backtracking method of solving a maze
- Pages 655-658 show a breadth-first search method of solving a maze. I recommend this approach for your maze solver as well. I might, however, recommend using a stack rather than recursion, but that's up to you.

The book's mazes are not perfect. He talks about how the first approach with backtracking might not find the best (shortest) solution, whereas the second approach will. Our mazes are perfect, so there is only one solution. I'd love to throw a non-perfect maze at you though… we'll see.

One interesting design decision that you have: do the breadth-first search directly on your maze structure, or convert it to a more conventional graph data structure and solve it there. Sort of like this:

1. Convert your maze to a graph (cells adjacent if no wall between them)
2. Do your breadth-first search on the graph.
3. Write your solution back onto your maze (color/mark cells on solution path).

It's up to you.

6. Flair

"I do want to express myself, okay.
And I don't need 37 pieces of flair to do it."

- Waitress in "Office Space"

There are so many fun flairs you can add to this project. Just some:

- Add GUI and data structure support so that you can show generating and/or solving a maze, step by step. That's cool!
- Make your maze algorithms and data structures more efficient. I can give you a handout on handling disjoint sets more efficiently.
- Generate and solve non-perfect mazes or mazes of different shapes.
- Output your maze as a JPG. There's code from a previous program somewhere around here to help you do this.

Of course, the best flair is always something you came up with on your own and you feel passionate about it.

7. Special Corndog Notes

There's lots of fun (and scary) stuff on mazes out there on that internet. I have no problem with you looking at other mazes and algorithms. Indeed, I would love to see everyone do some searching and then post a comment/link or two on the class blog! I am ever the optimist. I may even post a couple ditties myself.

Please be aware, however, that **copying code** from the internet (or anyone else) is considered **plagiarism**. Get algorithm ideas and Java help from the internet and others, but do your own coding. Any instances of plagiarism get bumped up to the Dean.

Some applicable bromides: start early; think, then code; run faster.

I'm here if you want help: wtkrieger@noctrl.edu

Grading

I'll be honest. On this final big-money program, I expect pristine design, code, documentation, and results. Anything otherwise will be judged harshly. Go:

1. **Your README file** – describing the state of your program
2. **Your Net Beans folder** – including your Java source code, class files, etc.
3. **Your Javadoc** – generate using the "Build/Generate Javadoc" menu
4. **Your applet** – create a web page at w:/index.htm with an Applet of one of your favorite trees. Please include a link to your README and Javadoc.
5. **Your printout** – You do **not** have to turn in code for Project Corndog. I'll peek at your code on the k: drive.

Special deliverables for Project Corndog:

6. **Email me your maze** – Email me a maze file that you have generated by noon on **Wednesday March 5, 2008**. Please use our CSC 210 maze file format. I will collect these and place them on the k: drive for all to enjoy.
7. **Print your solutions** – Print your solutions to each of the mazes posted on the k: drive. You can use <ALT><PrintScreen> to copy your solution to the cut-paste buffer. Then paste it somewhere like Word and print it out. Or, perhaps your flair will be to save maze solutions to a JPG file.

Start early and have fun with this one.

one deca-point… yow, bill

Additional Notes

I'll post additional notes directly on the blog.