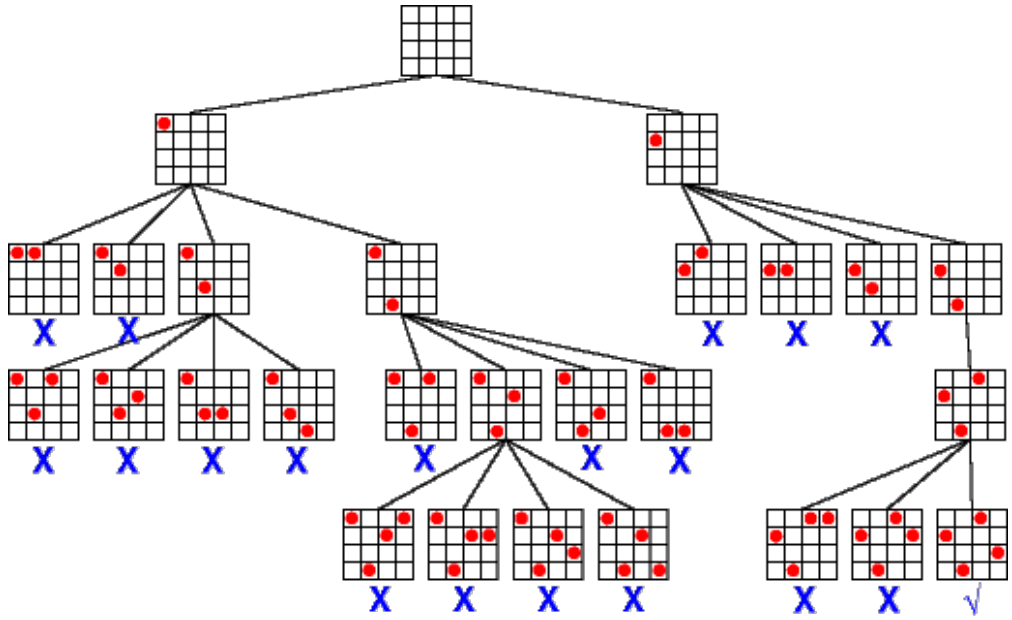
Project 2: Choose Your Puzzle (Command Line)

Approval Date: Saturday, March 15th @ 11:59pm
Requirements Date: Sunday, March 23rd @ 11:59pm
Design Date: Wednesday, April 2nd @ 11:59pm
Implementation Due Date: Wednesday, April 9th @ 11:59pm



The Problem

For the second part of your project, you will get to choose which puzzle you want to add to the solver for your wheel puzzle. The basic steps for this part of the project are:

1. Choose a puzzle and get approval from your instructor on it.
2. Write the requirements for your project, including sample test cases.
3. Design a class diagram.
4. Develop and test your solution.

The basic requirements for the puzzle you choose are:

- The board must be multidimensional (greater than 1-D)
- It must take as input two things - whether the entire solution path is desired (or not) and a file that describes the initial state
- If a solution exists, it must display it to standard output (along with the path, if specified)
- It must use the common backtracking algorithm you developed in project 1 to solve the puzzle
- Later, you will develop a playable UI for the puzzle you choose here.

To give you some ideas, here are two links to sites that have puzzles that your instructor has deemed appropriate for you to use.

- Puzzle Picnic
- nikoli.com Everybody's page

Most of these puzzles require Java to run in a web browser. You should set up your Java security preferences with exceptions for these sites.

Also please note that I do not want you to choose Sudoku, as it has been "programmed to death", in my opinion. Instead, choose one of the many variants from that genre.

Make sure that you choose a puzzle that you enjoy! You will be required in the final part of the project to develop a playable user interface for your puzzle (similar to what you see when you play with the puzzles in your browser).

Approval

Please send me an email, before the due date, indicating the name of the puzzle you want to do, and a link to a website that describes the puzzle. I will respond back promptly with an answer as to whether you are allowed to do that puzzle or not.

If you choose to use one of the puzzles from the websites provided, you can consider yourself automatically approved. You just have to let me know which one you have chosen. If you want to choose another puzzle that isn't listed, you will have to run it by me before the approval date. After the approval date I will expect you to develop the puzzle we agreed upon, unless you can convince me otherwise.

Requirements

You will develop the requirements for your puzzle and submit a document before the due date. Your requirements must include:

- A brief tutorial on what puzzle you chose and what the rules are.
- How the program is run on the command line.
- A description of a sample input file format.
- At least five sample input files.
- At least one diagram of how an input file relates to an initial board.
- A description of the output format.
- At least five sample outputs corresponding to the input files.

All programs will be run on the command line. The executable should be named appropriately. Regardless of the program name, all programs must accept up to two additional parameters:

- A string, `path`, that indicates the user wants the entire solution path to be displayed. This is optional, and would be specified directly after the program name.
- The name of the input file. This argument is required and will either be specified after the path option, or directly after the program name.

If the path is not specified correctly, or the file name cannot be open, you should display an appropriate message to standard output and exit the program.

If the input file can be opened, you can assume, for the most part, that its format is correct. Since others may want to make their own input files, you must do some "light" error checking to help the user understand what may be wrong with their input.

Your input files must specify the dimensions of the board. This should be on the first line.

Just like with the previous puzzle, you must assume that the initial configuration could either be empty, or partially completed. You can assume that a partially completed configuration is already valid.

You are deciding the output format for your puzzle. Please use only the printable ASCII characters. Also use a reasonable format that easily allows the user to identify what the components of your puzzle are. If you have questions as to whether your output is "readable" or not, please see me.

The format you choose for your document is up to you. You should zip up your document, along with your sample files and submit them to the dropbox before the deadline. Please name your files:

- `requirements.ext`: Your requirements document (ext is based on your file format).
- `input.#`: Each of your input files should be in a separate ASCII text file, numbered 1-5.
- `output.#`: The corresponding outputs, assuming the user did not select "path" on the command line.
- `output-path.#`: The corresponding outputs assuming the user selected "path" on the command line.

It is ok if when you go to implement your puzzle that you do not match the output you have specified here exactly. The solution path may be different, and that is ok. You will be submitting an updated requirements document at the end and you can make the output match your solution at that point. FYI, if you choose one of the puzzles from the websites listed above, they usually have only one solution to them.

Design

As with the previous part, you will design a class diagram which incorporates both the wheel puzzle and your new puzzle. The solver must be common and work independently of the puzzle being solved. If you do not use a common solver you will be severely penalized for using a poor design.

The two most common methods for making your solver work with different configuration types is to either use templates, or an abstract configuration class from which your concrete puzzle types inherit from.

Submit your class diagram to the dropbox before the deadline.

Implementation

Your project will have two main programs in it. One for the wheel puzzle, and one for your new puzzle. It must build and run on our CS Ubuntu machines using C++11 and

`make`.

Based on the puzzle you choose, and the size of configuration, there can be great variance in the running time to find a solution. You must do a reasonable amount of pruning when you generate your successor configurations. You will not receive credit for this assignment if you don't effectively prune and the backtracking basically becomes exhaustive enumeration. Solving this problem is up to you!

Your project must make use, whenever possible, of the new features of C++11 that we have been discussing in class. A program that is written primarily in C, ignores the Standard Template Library, or doesn't use C++ I/O streams, will receive little to no credit.

When you are ready to submit your project, zip up all the required files and submit them to the MyCourses dropbox:

- All source files required to build both programs. Put these in a folder named `Source`.
- Your final requirements documents (do not forget the test files). Put these in a folder named `Requirements`.
- A final class diagram. Put this in a folder named `Design`.

Grading

The grade breakdown for this assignment is:

- Approval: 5%
- Requirements: 10%
- Design: 10%
- Implementation: 60%
- Memory Management: 10%
- Code Style & Commenting: 5%

Submissions

All submissions will be accepted up to 1 day late with a 20% penalty.

---

updated: Mon Mar 10 2014