CSI 32 Final Projects List

Easy to do projects:

1. **Hangman** (using graphics)
Design and implement a simple version of the classic game Hangman using graphics library cs1graphics.

2. **Tic-Tac-Toe** (using graphics)
Design and Implement a Tic-Tac-Toe game where you can play against another person, using graphics library cs1graphics.

3. **"Line of Best Fit"** (Statistics) (using graphics)
Write a program that reads a data set of two-dimensional points and calculates the "line of best fit" for that point set. Plot the data and resulting line graphically.

4. **Frequency Distribution and Histograms** (Statistics, using graphics)
Design and implement a program that's given some numerical data and number of classes to use, processes the data and produces a histogram.

5. **A simple graphical editor** (using graphics)
Write a program using cs1gaphics module that will allow the user to do simple graphical pictures.

6. **Sliding Tile game** (using graphics)
Write a sliding tile game (with numbers 1 through 15). There are tiles numbered 1 to 15 randomly placed on a 4×4 grid, with one empty space. If a tile adjacent to the empty position is clicked on, then it should move to that spot. When any other tile is clicked on, nothing should happen.

7. **Towers of Hanoi** (solved by Recursion) (using graphics)
The game begins with three pegs and a tower of n disks on the first peg, stacked from largest at bottom to smallest at top. The goal is to move them all to the third peg, moving only one disk at a time. Moreover, you are not allowed to place a disk on top of a smaller disk. Write a program that generates a solution to the problem + show it graphically.

8. **Matrix arithmetic** (using file input and graphics)
Design and implement program that will allow the user to play with matrix operations (addition, multiplication and transpose of a matrix).
Input: two matrices (from files)
The rest: in the graphics window

9. **Coloring Turtle** (using graphics)

Consider that you have a finite grid of square cells that are initially all white. You also have a 'turtle' sitting on one of those cells. There is also a table that identifies the action to be taken by the turtle depending on the color of the square it is currently occupying. The actions take the form (change color of current cell−that could be to the same color −turn left or right, no turn, or reverse, and move to the cell it is now facing.

For example, a very simple set of rules would be:

- If white, change to red and turn left.
- If red, change to blue and turn right
- If blue, no change, no turn

Initially limit yourself to only a few colors. Four is plenty to start with.

Note that some sets of rules produce very boring results and others produce quite startling and unexpected ones.

Moderate Problems:

1. A Maze

Write a program that for a given maze (labyrinth) and a given start point will find a path out of the labyrinth.
The labyrinth will be given with 0's and 1's. Where 1 stands for wall, and 0 stands for path, S stands for Start point, F stands for Finish point. The maze won't have loops, but some paths will lead to a dead end and will be given as a file to the program.

Example of a maze (input):

```
F00111111111111111111111111111111111111111111111111111111111111111111111
11011111111100000000011110000001111111111110000111111111111111111111111111
11011111111101111111011110111100000000011110110111111111111111111111111111
11011111111101111111000000111111111111011110110000000000000000000000011111
11011111111101111111111111111111111111011110111111111111111111111111011111
11011111111101111100000011110000000000000011100000000000000000000000011111
11011100000001111101111011110111111111111111111101111111111111111111111111
11011101111111111001111000000111111111111111110001111111110000000000111
11010001111111111011111111111111100000000011111110111111111101111111110111
11011111111111111011111111111111001111110011111110111111111100000001110111
11011111111000000111111111111S000111111001111110011111111111111101110111
11011111100001110111111111111111100000001111111001111111110000000111111
11011111101111111011111111111111110001111100111100000000000011111111110011
11011110001111111000000111111111100111100000011111111111110111111000011
11011111111000011011111111111111110000001111011110000111110000111101111
11011111111011011011111111111111111011111111011110100111110110000011111
11011110000011000011111111111111110111111111111010011100011011111111111
11011101111111111111111111111111111011110001110000110111011110000001111
11011100000000000000000000000111111000000010110010011011101111011011111
11011101110111111111111111110011111111111110110011011000001100001110000011
11000001110111111111111111111011111111111110000111011101111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111
```

Here are few of the possible approaches:

1. **Wall follower** (*left-hand rule* or the *right-hand rule*):
   Player keeps one hand in contact with one wall of the maze. In this case the player is guaranteed not to get lost and will reach a different exit if there is one; otherwise, he or she will return to the entrance.
   Good for: mazes with all walls connected and without loops
2. **Tremaux's algorithm**
   is an efficient method to find the way out of a maze that requires drawing lines on the floor to mark a path, and is guaranteed to work for all mazes that have well-defined passages. A path is either unvisited, marked once or marked twice. Every time a direction is chosen it is marked by drawing a line on the floor (from junction to junction). In the beginning a random direction is chosen (if there is more than one). On arriving at a junction that has not been visited before (no other marks), pick a random direction (and mark the path). When arriving at a marked junction and if your current path is marked only once then turn around and walk back (and mark the path a second time). If this is not the case, pick the direction with the fewest marks (and mark it, as always). If there is no exit, this method will take you back to the start where all paths are

marked twice. In this case each path is walked down exactly twice, once in each direction. The resulting walk is called a bidirectional double-tracing.

3. **Shortest path algorithm**
   When a maze has multiple solutions, the solver may want to find the shortest path from start to finish. This algorithm finds the shortest path by implementing a *breadth-first search*. The algorithm uses a *queue* to visit cells in increasing distance order from the start until the finish is reached. Each visited cell needs to keep track of its distance from the start or which adjacent cell nearer to the start caused it to be added to the queue. When the finish location is found, follow the path of cells backwards to the start, which is the shortest path.

For more information about the mazes and algorithms visit:
http://en.wikipedia.org/wiki/Maze
http://en.wikipedia.org/wiki/Maze_solving_algorithm

A possible path:

```
F00111111111111111111111111111111111111111111111111111111111111111111111111111
1101111111110000000001111000000111111111111000011111111111111111111111111111
1101111111110111111101111011110000000001111011011111111111111111111111111111
1101111111110111111100000011111111111101111011000000000000000000000000011111
1101111111110111111111111111111111111101111011111111111111111111111111011111
1101111111110111111000000111100000000000000011100000000000000000000000011111
1101111000000011111011110111101111111111111111110111111111111111111111111111
1101110111111111110011110000001111111111111111110001111111111000000000000111
1101000111111111101111111111111110000000001111111011111111101111111110111
1101111111111111101111111111111100111111001111111011111111100000011101111
1101111111111000001111111111111S000111111001111111001111111111111111101110111
1101111110000111101111111111111111000000011111110011111111100000001111111
1101111110111111101111111111111111000111100111100000000000011111111110011
1101111000111111100000111111111110011110000001111111111111101111111000011
1101111111100001101111111111111111000001111011110000111110000111011111
1101111111101101101111111111111111101111111011110100111110110000011111
1101111000001100001111111111111111101111111111110100111000110111111111
1101111011111111111111111111111111110111100011100001101110111100000011111
1101111000000000000000000000011111100000010110010011011101111011011111
1101110111011111111111111111001111111111111011001101100000110000111000011
1100000111011111111111111111101111111111111000011101110111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111
```

For testing purposes don't use the given above maze. Use something simple first, and then the example of the maze given above. For example:

```
11111       111111F11       11F11111        1111111111
11S01       111111011       11000011        1111111111
11101       111111011       11111011        F000001111
111F1 then  111111011  then 11S00011  then  1111100101
            11S100111       11111111        111S110001
            110001111       11111111        1110000111
            111111111       11111111        1111111111
```

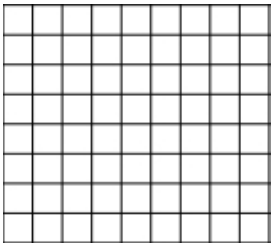
2. **Game of Life**

The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970

The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input from humans. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

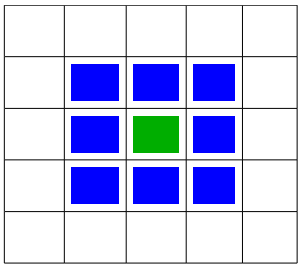
Rules

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square *cells*,



each of which is in one of two possible states, *live* or *dead*.

Every cell interacts with its eight *neighbors*, which are the cells that are directly horizontally, vertically, or diagonally adjacent.



At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
2. Any live cell with more than three live neighbors dies, as if by overcrowding.
3. Any live cell with two or three live neighbors lives on to the next generation.
4. Any dead cell with exactly three live neighbors becomes a live cell.

The initial pattern constitutes the *seed* of the system.

The first generation is created by applying the above rules simultaneously to every cell in the seed — births and deaths happen simultaneously, and the discrete moment at which this happens is sometimes called a *tick* (in other words, each generation is a pure function of the one before). The rules continue to be applied repeatedly to create further generations.

Given above information and even more can be found here:

http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

http://www.math.com/students/wonders/life/life.html

or just Google for Game of Life

3. **Fractals: Kosh Snowflake**

http://math.rice.edu/~lanius/frac/koch.html - explanation
http://math.rice.edu/~lanius/frac/koch/koch.html - demonstration
http://library.thinkquest.org/26242/full/fm/fm16.html
http://en.wikipedia.org/wiki/Koch_snowflake
http://www.google.com/url?sa=t&source=web&cd=3&ved=0CCsQFjAC&url=http%3A%2F%2Fwww.cs.wfu.edu%2F~burg%2Fnsf-due-0340969%2Fworksheets%2FKochSnowflakeProgrammingAssignment.pdf&rct=j&q=Kosh%20snowflake%20algorithm&ei=qn-OTab7KujE0QHkhLy_Cw&usg=AFQjCNEzMtDEmzWWQneedwQirmdmox3m9g&cad=rja

4. **Othello / reversi game**

http://en.wikipedia.org/wiki/Reversi/Othello
http://www.freegames.ws/games/boardgames/othello/othello.htm