Program 1: Search

Program 1 is all about search; both traditional tree/graph search and local search methods. The problem domain is one of **generating** "Rook Jumping Mazes" (evaluation of the mazes you are generating will also involve **solving** them).

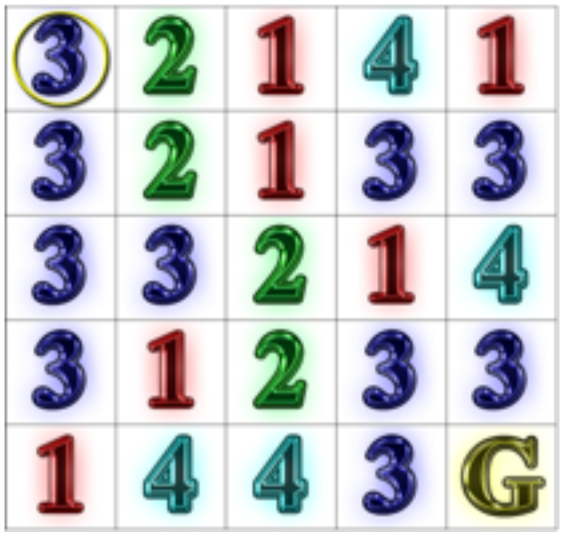
Several problems include some written questions or analyses. Make sure you turn in:

- Your code
- Instructions on how the TA can run your code for each problem
- A written report answering the questions

Rook Jumping Mazes

The origin of the Rook Jumping Maze (a.k.a. Number Maze) is unknown, but some attribute its creation to the great puzzle innovator Sam Loyd. Loyd's "Back from the Klondike" Queen Jumping Maze, which additionally allows diagonal moves, appeared April 24th, 1898 in the New York Journal and Advertiser. It also appears on page 106 of the Cyclopedia of Puzzles, a collection of Loyd's work compiled by his son. There are many additional variations. This problem setting is from Todd Neller.

Rook Jumping Maze: *Starting at the circled square in the upper-left corner, find a path to the goal square marked "G". From each numbered square, one may move that exact number of squares horizontally or vertically in a straight line. How many moves does the shortest path have?*



Solution (select to reveal): [ ]

PROBLEM 1: Maze representation [20 pts]

Generate and print/display a random *n*-by-*n* Rook Jumping Maze (RJM) ($5 < n < 10$) where there is a legal move (jump) from each non-goal state.

Let array row and column indices be (r_{min}, ..., r_{max}) and (c_{min}, ..., c_{max}), respectively, where

$$r_{max} - r_{min} + 1 = c_{max} - c_{min} + 1 = n.$$

The RJM is represented by a 2D-array of *jump numbers*. A cell's *jump number* is the number of array cells one must move in a straight line horizontally or vertically from that cell. The start cell is located at (r_{min}, c_{min}). For the goal cell, located at (r_{max}, c_{max}), let the jump number be zero. For all non-goal cells, the randomly generated jump number must allow a legal move. In the example 5-by-5 maze above, legal jump numbers for the start cell are {1, 2, 3, 4}, whereas legal jump numbers for the center cell are {1, 2}. In general, the minimum legal jump number for a non-goal cell is 1, and the maximum legal jump number for a non-goal cell [r, c] is the maximum of r_{max} - r, r - r_{min}, c_{max} - c, and c - c_{min}. This defines a directed graph where vertices are cells, edges are legal jumps, and the outdegree is 0 for the goal cell vertex and positive otherwise.

Inputs: a number between 5 and 10 inclusive.

Outputs: display a randomly-generated RJM of the given size (note you may want to modularize creating an RJM and displaying it (Model vs. View)—read through the entire assignment to see how the parts all fit together.

Examples:

```
Rook Jumping Maze size (5-10)? 5
3 2 4 4 4
3 1 2 2 2
2 1 1 2 2
1 3 2 2 1
2 4 4 3 0
```

```
Rook Jumping Maze size (5-10)? 6
5 2 1 1 1 2
4 3 4 3 1 5
5 2 3 1 4 1
3 1 2 1 4 1
3 4 1 1 4 5
4 3 4 5 2 0
```

```
Rook Jumping Maze size (5-10)? 10
6 2 4 6 6 8 3 7 2 5
3 7 1 6 2 6 8 8 8 5
8 8 7 4 3 6 5 6 5 2
1 8 1 6 1 4 5 7 5 1
7 5 6 4 5 4 1 7 4 9
3 8 4 2 3 3 3 6 4 9
8 3 1 2 3 5 1 4 5 4
5 8 3 1 7 2 7 3 5 9
9 1 4 5 7 5 7 6 8 2
4 6 3 7 6 2 5 9 1 0
```

PROBLEM 2: Maze Evaluation [20 pts]

There are many features of a good RJM. One obvious feature is that the maze has a solution, i.e. one can reach the goal from the start. One simple measure of maze quality is the minimum number of moves from the start to the goal. The larger the minimum value, the better [harder!] the maze. For simplicity, we will limit our attention to this, although consideration of other features is an interesting exercise (see extra credit)

Using a suitable search algorithm, compute the minimum distance (i.e. depth, number of moves) to the goal from the start cell. Create an **objective function** (a.k.a. energy function) that returns the **negated** distance from start to goal, or a large positive number (e.g. 1,000,000) if no path from start to goal exists. [Then the task of maze generation can be reformulated as a search through changes in the maze configuration so as to minimize this objective function.]

Input: a random RJM

Output: a negative integer or a very large positive integer representing no solution

PROBLEM 3: Maze Generation
[First-Choice Gradient Descent] [20 pts]

Using a form of stochastic local search called *First-Choice Gradient Descent*, search the space of 5-by-5 Rook Jumping Mazes (RJMs) for a given number of iterations and print/display the best maze evaluated according to the objective function specified by your objective function (problem 2).

Given a number of iterations from the user, first generate a random 5-by-5 Rook Jumping Maze (RJM) as in Problem 1 and evaluate it according to your objective function. Then for the given number of iterations:

- For a random, non-goal cell, change the jump number to a different, random, legal jump number.
- Re-evaluate the objective function on this new RJM.
- If the objective function has not increased (worsened), accept the change, and furthermore remember this RJM if its evaluation is the best (minimum) evaluated so far. Otherwise, reject the change and revert the cell to its previous jump number.

Finally, print/display the RJM with the best (minimum) objective function evaluation that you found.

More formally, let *jump function j* be a mapping from cells to legal jump numbers, or 0 in the case of the goal cell. Let *objective function* (or *energy function*) *e* be a function we seek to minimize over jump functions. Let *step* be a function that takes a jump function *j*, and generates a "neighboring" jump function j' by making a single, stochastic change: function *step* chooses from non-goal cells with equal probability, and for that cell *c* chooses $j'(c)$ from all legal jump numbers not equal to $j(c)$ with equal probability. For all other cells, $j'(c) = j(c)$. Then we may describe our algorithm as follows:

```
• Let $j$ be chosen at random, and $j_{best} \leftarrow j$.
• For a given number of iterations:
    • $j' \leftarrow step(j)$
    • if $e(j') \leqq e(j)$
        • $j \leftarrow j'$
        • if $e(j) < e(j_{best})$
            • $j_{best} \leftarrow j$
• return $j_{best}$
```

Inputs: size of maze, number of iterations

Outputs: Display the "Best" maze found [note: again, read ahead---you'll want to separate display from simply returning the "best" maze structure]

Problem 3 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with the number of iterations?*

2. *Why should we allow sideways moves (objective function is the same)? That is, how might sideways moves help stochastic local search?*

PROBLEM 4: First-choice Gradient Descent With Random Restarts [20 pts]

One problem with pure gradient descent is that stochastic local search may become trapped in local minima, where every local step is uphill, making things worse. One *escape strategy* is to restart the search periodically. Another way of viewing this is that we iteratively perform pure first-choice gradient descent, starting each descent at a random initial state. The end result is the *best result from all descents*.

Inputs: size of maze, number of restarts, number of iterations per restart.

Outputs: display best maze found

Problem 4 Questions:

1. *Empirically, keeping the total number of iterations (i.e. descents times iterations per descent) constant (e.g. 10,000), approximately how many restarts yields the best average generated maze?*

2. *For which kind of optimization tasks is it better not to restart search? That is, for which kinds of functions or iteration constraints is it better to do a single descent for all iterations?*

PROBLEM 5: First-choice Gradient Descent with Random Uphill Steps [20 pts]

Another escape strategy from local minima allows **uphill** steps with some small probability. Thus, with some probability, any local minimum can be escaped. In this exercise, you will modify your code from Problem 3 (simple first-choice gradient descent) to allow uphill steps with a given fixed probability *p*.

Inputs: size of maze, number of iterations, probability *p* of moving to a state even if it is worse

Outputs: display best maze found

Problem 5 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with (1) the number of iterations, and (2) the probability of accepting uphill steps?*

2. *Is the probability of taking an uphill step the same as the probability of escaping a local minimum? Why or why not?*

Extra Credit

Extra Credit 1: Simulated Annealing [20 pts]

One problem with your solution in problem 5 is that all uphill steps are equally likely. A small uphill step would generally be more desirable than a large uphill step. *Simulated annealing* is a stochastic local search technique based on an analogy to energy distributions in heated materials as they cool (i.e. anneal) and "seek" a lower energy state. For example, blacksmiths long ago observed that *quenching*, i.e. rapid cooling, would lead to a harder, more brittle metal than annealing, i.e. slow cooling, which yields a more malleable metal with a lower energy and more crystalline configuration.

When the material is heated (high energy input), atoms reconfigure among different possible energies much like a random walk. When the material is rapidly cooled (low/no energy input), atoms reconfigure to lower energy states often getting trapped in local minima. The local minima escape strategy of simulated annealing concerns a temperature schedule, called an *annealing schedule*, that gradually shifts search from a free random walk to a final descent, while favoring smaller uphill steps over larger ones. In a nutshell, for local minima, there are temperatures where one is more likely to escape than reenter. For more on simulated annealing, see the book. You may use a simple annealing schedule $T \leftarrow T*d$ for $0<d<1$.

EC1 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with (1) the number of iterations, and (2) the initial temperature, and (3) the decay rate? Experiment to develop a good annealing schedule and compare your annealing performance with others according to a common measure, e.g. average iterations per production of a maze with a shortest solution path of 18 or more.*

2. *For a fixed number of iterations (e.g. 10000), experiment and develop a best annealing schedule and best Problem 4 (gradient descent with random restarts) policy. Which yields the lowest average energy Rook Jumping Maze after the given number of iterations?*

Extra Credit 2: Better Evaluation Function [20 pts]

Define a *better* maze evaluation function and argue why it leads to improved maze quality. Compare and contrast the results obtained with your new evaluation function versus the previous evaluation function using the same stochastic local search technique for each.

There are many features of a good Rook Jumping Maze (RJM). Previously, we evaluated RJMs according to solvability and the shortest distance to the goal. However, there are many other features that may be considered as well. Here, we describe a few:

- **Black holes**[1] - A black hole is a dead-end. Define a **reachable** cell to be a cell one can reach from the start through a sequence of legal moves. Define a **reaching** cell to be a cell from which one can reach the goal through a sequence of legal moves. A cell is part of a black hole if it is a reachable, non-reaching cell.
- **White holes**[1] - Some puzzlers seek to trace backwards from the goal to the start. A white hole is a back-tracing dead end. A cell is part of a white hole if it is an unreachable, reaching cell.
- **Start and goal positions** - Is anything gained/lost by allowing the position of the start and/or goal cell to vary?
- **Shortest solution uniqueness** - How does knowledge that there is a unique shortest solution affect the maze solving experience?
- **Forward/backward decisions** - Sometimes there is only a single legal "forced" move from a cell. How do the number of forward decisions affect the maze solving experience? This is related to a game-tree branching factor. Initial forced moves are especially noticeable. For back-tracing puzzlers, consider also the number of backward decisions, i.e. the number of cells for which there are multiple possible predecessors.
- **Same jump clusters** - Same jump clusters are sets of cells that all have the same jump number and reach each other without leaving the cluster. For example, consider the maze above. The two leftmost 3s of the second row and the 3 of the bottom row form a 3-jump cluster. How do same-jump clusters affect the maze solving experience? Are some better than others?
- **Solution direction variability** - Again, consider the maze above. Part of the shortest solution begins at the second 3 in the third row and proceeds right-left-right-left. How does such back-and-forth within the same row or column affect the maze solving experience?

Select one or more of the features above that you consider most important, compute measures for maze evaluation, and seek to improve upon the previous RJM evaluation function. Generate sets of puzzles using the old and new evaluation functions, compare and contrast the puzzles, and argue why your new evaluation function improves the average quality of your mazes.

Possible background readings include:

- Cormen et al. Introduction to Algorithms, 3rd edition, Chapter 25 (All-pairs shortest path)
- Wikipedia "All pairs shortest path" algorithm articles (e.g. Floyd-Warshall algorithm)

[1] *The descriptive maze terms "black hole" and "white hole" were coined by maze designer Adrian Fisher.*