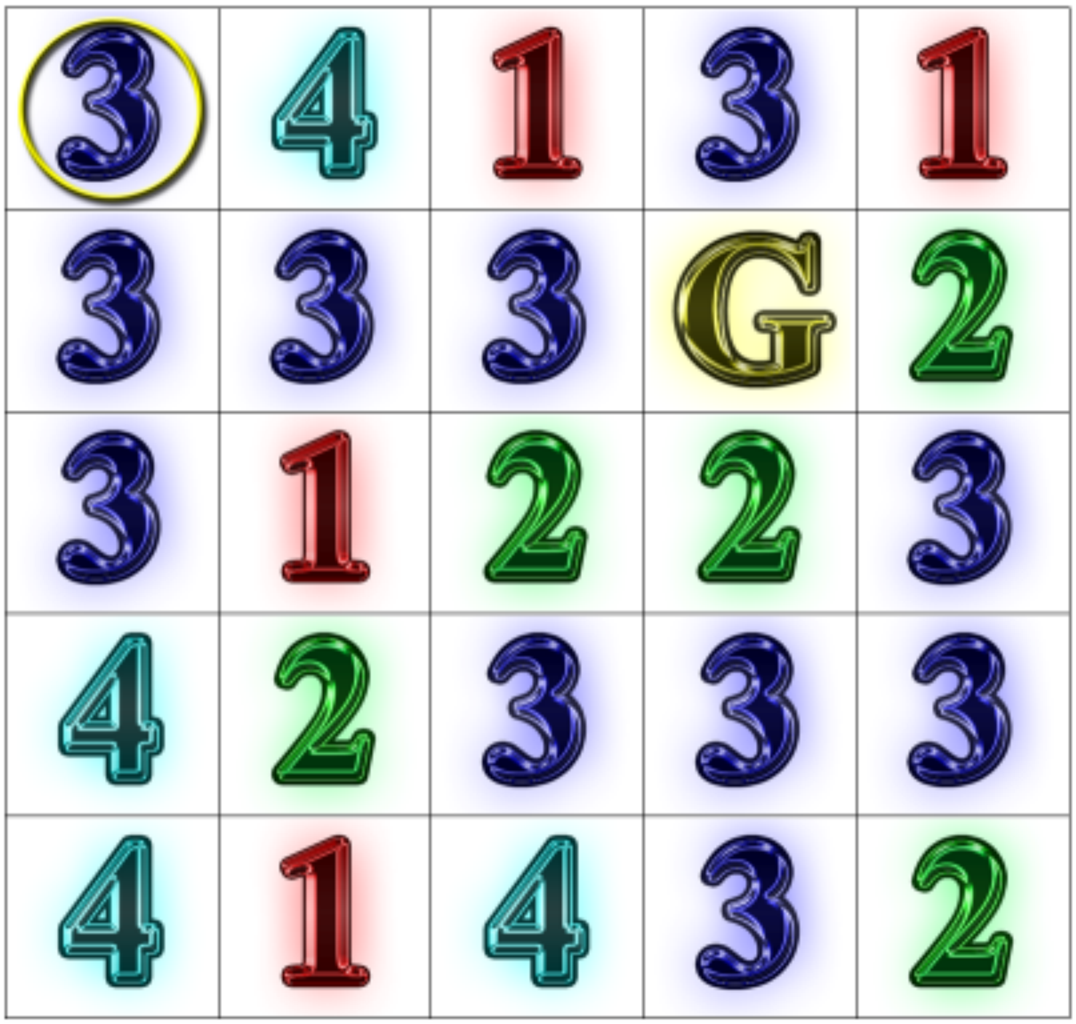
Rook Jumping Maze

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.



The solution to this maze is 13 jumps.

More information on Rook Jumping Mazes (including the solution to the maze above)

PART I

Question 1: What algorithm would you use to solve a rook jumping maze. Why? If a heuristic is used, what is the heuristic?

Either breadth-first or depth-first search. DFS will probably solve rook jumping mazes faster on average. BFS should be used if one cares about the shortest number of jumps to solve the maze. UCS is equivalent to BFS because all action costs are 1.
A cannot be used because there is no known heuristic that helps. For example, Manhattan distance doesn't work because sometimes one can jump really far and being close to the G doesn't guarantee that one can get to G quickly.*

Question 2: What is the branching factor of a state search algorithm that starts at the circle and ends at the G? What is the branching factor of a state search algorithm that starts at the G and ends at the circle? Should one implement forward search or backward search?

Search algorithms can be run backward by making the goal the initial state and the initial state the goal. One needs to implement a successor function, that can compute all the states that one could have come from (a predecessors function). Branching factor is the average number of successors from each state. The branching factor going backward is slightly smaller, so one should implement backward search.

PART II

Suppose instead of solving a rook jumping maze, we wanted to *generate* a rook jumping maze. That is, given a completely blank grid, what numbers should we put into each cell, and where should the G go? If we could do this, we could create a system that endlessly generates rook jumping mazes for other people to play.

Not all rook jumping mazes are created equally. For example, putting a 1 in every cell would make for a very easy maze. Choosing random numbers for each cell might result in an un-solvable maze. In this part, we look at how to use hill-climbing go generate solvable rook jumping mazes of differing difficulties.

Question 3: What is the state representation we should use for a hill-climbing algorithm? What should the start state look like?

A grid of numbers. The start doesn't really matter. It could be all blanks. It could be random numbers and a single, randomly placed G.

Question 5: Describe a neighborhood and a successor function.

Can be just about whatever you want. It also depends on the representation and the start state. One reasonable thing would be for the neighborhood to be all grids that differ by one cell. A successor function would then be to pick a cell randomly and change the number to any other valid number. One might also want to change the location of the G (but make sure there is only one G). This

> *works with blind hill-climbing, simulated annealing, and also as a mutation operation in genetic algorithms. Another possibility that works for cross-over in genetic algorithms is to break two grids (of two states) into quadrants and the swap quadrants.*

Question 4: Describe an evaluation function that converts a possible maze into a real number. The evaluation function way want to consider: (a) the solvability of the maze, (b) how many dead-ends (where there are no successors to a cell), (c) loops, (d) length of shortest solution, (e) number of solutions, (f) whether easier to solve going forward or backward.

> *There are many solutions. However, most evaluation functions will probably have to perform some sort of breadth-first search or depth-first search to determine the solvability. This alone is somewhat problematic because a lot of states will be unsolvable and thus have the same value (zero) and hill-climbing can be kind of random unless the landscape is smooth. Adding a bunch of factors together can help to give more variety to the return values of the evaluation function. Difficulty can also be evaluated as the length of the shortest solution, allowing for easier or harder mazes.*

Question 5: Which optimization search should one choose?

> Doesn’t really matter as long as the successor function supports the algorithm (for example a genetic algorithm might need a mutation and a cross-over successor function). Generally, simulated annealing or genetic algorithm will be preferable because they are good at exploration.