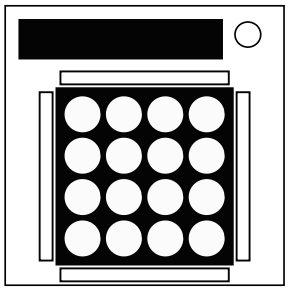
On the Subject of Algorithmia



There appears to have been an accident at the restaurant placemat factory.

- The screen at the top of the module cycles through 10 two-digit numbers. This sequence is the module's **seed**, starting with the number in blue.
- The module will generate a 4×4 maze, where each cell is represented by a bulb in its corresponding position on the module.
- Two of the bulbs will be lit. The white bulb represents your current position, while the colored bulb represents the goal. Use the bars on the sides of the grid to navigate to the goal without passing through a wall in the maze.

Determining the Walls of the Maze

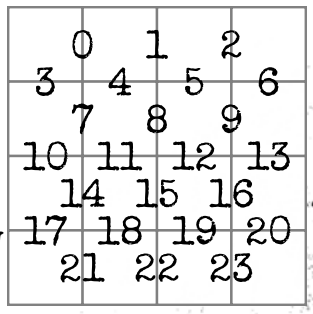
- The module generates its maze using a certain algorithm. The algorithm used is determined by the color of the non-white bulb.
- When generating the maze, the algorithm at many points requires the use of random numbers.
  - Place a "pointer" under the first number of the seed.
  - **Any** time the algorithm requests a random number, a number n will be supplied. Modulo the number above the pointer by n, to obtain the number used for the calculation.
  - Then, move the pointer to the next number of the seed, wrapping around from the end to the beginning.

Important things you probably shouldn't forget

- The edges of the 4×4 grid are **not** classified as "edges" in the following algorithms. You still cannot pass them on the module.
- All counting is **zero-indexed**: The leftmost entry of a sequence is index 0, the second from the left is index 1, and so on.
- Unless stated otherwise, all edges begin in an impassable state.

Red Bulb: Kruskal's Algorithm:

1. Begin with 16 independent "groups" representing each cell in the maze.
2. Let L be the list of all edges on the module in the order depicted to the right.
3. Generate a random number with n= the number of edges in L. Count this many entries in L.
4. Remove the edge at this position from the list, this edge is now passable, and unify the two groups connected by this edge into one group.
5. Remove any edges from L which connect a group to itself.
6. Repeat steps 3-5 until there is only one group on the module.



Green Bulb: Prim's Algorithm

1. Generate a random number using n= 16. This is the index of the starting cell. Mark it as visited.
2. Let A be the list of every cell which is adjacent to an already-visited cell in reading order.
3. Generate a random number using n= the number of cells in A. Count this many entries in A. Let this cell be called C.
4. Generate a random number using n= the number of visited cells adjacent to C. Number these cells from north going clockwise, and use the cell with that number. The edge between C and this cell is passable.
5. Repeat steps 2-4 until every cell has been visited.

Blue Bulb: Recursive Backtracker:

1. Generate a random number using n= 16. This is the index of the starting current cell. Mark it as visited.
2. Let A be the set of unvisited cells adjacent to the current cell, ordered from north going clockwise.
3. If A is empty, back up to the previously visited cell until you have reached a cell such that A is not empty.
4. Generate a random number with n= the number of cells in A. Count this many entries in A. Let this cell be called D.
5. The edge between D and the current cell is passable. Mark D as visited. D becomes the current position.
6. Repeat steps 2-5 until every cell has been visited.

Cyan Bulb: Hunt-and-Kill:

1. Generate a random number using n= 16. This is the index of the starting current cell. Mark it as visited.
2. Let A be the set of unvisited cells adjacent to the current cell, ordered from north going clockwise.
3. If A is empty, set the current position to the first unvisited cell in reading order which is adjacent to an already-visited cell. Then:
    - Let V be the list of visited cells adjacent to this new cell, ordered starting from north going clockwise.
    - Generate a random number with n= the number of cells in V. Count this many entries in V. The edge between this cell and the new cell is passable.
    - Return to step 2.
4. Generate a random number with n= the number of cells in A. Count this many entries in A. Let this cell be called D.
5. The edge between D and the current cell is passable. Mark D as visited. D becomes the current position.
6. Repeat steps 2-5 until every cell has been visited.

Magenta Bulb: Sidewinder:

1. Begin with every edge impassable **except** for the three edges connecting the cells of the top row together.
2. Let *R* be an initially empty list of cells.
3. Start at the leftmost cell of the second row and proceed in reading order. Perform the following steps on each cell, which will be called *C*.
4. Add *C* to *R*.
5. If *C* is not the rightmost cell of its row, generate a random number with *n*= 2. If this number is 0, make *C*'s **right** edge passable.
6. Otherwise, if *C* is the rightmost cell of its row, or the generated number is **1**, generate a random number with *n*= the number of cells in *R*. Count this many entries in *R*, and make that cell's **top** edge passable. Then, empty *R*'s contents.
7. Repeat steps 3-6 until every cell has been processed.

Yellow Bulb: Recursive Division:

1. Begin with all edges in a **passable** state. All 16 cells are initially one "group". Groups are always rectangular. A partition is a line which divides a group into two others. A partition's direction is the way its line points.
2. Let group *G* be the first group in reading order whose width and height are both not equal to 1.
3. Use the following rules to obtain a direction of partitioning *G*:
    - If *G* is wider than it is tall, partition it **vertically.**
    - If *G* is taller than it is wide, partition it **horizontally.**
    - If *G* is square, generate a random number with *n*= 2. If this number is 0, partition it **vertically,** otherwise partition it **horizontally.**
4. Generate a random number with *n*= the number of ways *G* can be partitioned using the obtained direction. Count that many into the list of possible partitionings in top-down and left-right order to obtain the chosen partition. Call this partition *P*.
5. Every edge along *P* becomes impassable, and the sections of *G* segmented by *P* become two new groups.
6. Generate a random number with *n*= the length of *P*'s line. Make the edge in that position of *P* passable, counting in top-down and left-right order.
7. Repeat steps 2-6 until all groups have a width or height of 1.