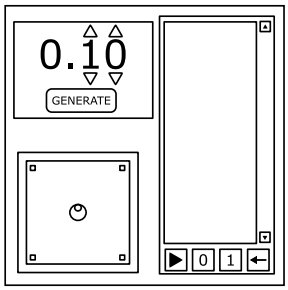
On the Subject of Robit Programming



This is not simply Maze Identification nor Robot Programming. This module is on a whole other level.

On the module is a giant square which is the interface used for viewing the maze. Initially, the module will show a "Generate" button and a numeric display denoting how much delay the module generates its mazes, 0.00 being instant, 1.00 being 1 step every 1 second.

Pressing the "Generate" button will start creating the maze by carving out the giant square at a specified delay which can be customized for the defuser. The defuser can press the "Generate" button to generate a new maze until the defuser finds one that is sufficient. Generating the maze for the first time on the module also reveals a terminal on the right of the maze.

To disarm the module, the defuser must figure out how the maze was generated, and the expert must modify the starting instructions to grab the combination of bits that tells the robit to move in a specific direction. The robit must be guided to the 4 corners of the provided quadrants, noted by the small square on each, and then move towards the middle row/column of the maze to disarm the module. The four quadrants may also modify the robit's behavior when its corner has not been visited yet.

The terminal consists of a screen with a highlight denoting the current pair the robit is scanning. On the initial activation or when inputs are cleared, there would be no highlighted instruction. The defuser can input a series of 1's or 0's on the terminal and can scroll the instruction sets with the up or down keys. The backspace button to the right of the 0 and 1 buttons can be used to delete the previous inputs. If that button is held for a few seconds, it will clear ALL inputted bits from the terminal. The play button on the terminal starts the robit and allows the robit to process the provided bits. The defuser will not be able to regenerate a maze or adjust the delay on how the maze is generated, so caution is advised when pressing the play button.

Once the expert figures out each quadrant's quirks and starting instructions, determined from steps 2 and 3 from the manual, have the defuser request the directions to all four corners in any order, and then anywhere in the center row/column. Modify these directions based on the quadrant the robit will be on and then use the bit table determined on step 2 to create a string of bits to input. If an odd number of bits were inputted onto the terminal and the robit reaches the lone bit, the lone bit will not be processed instead and the highlight will not mark at the lone bit instead. A strike will occur if the robit crashes into a wall or runs out of processable commands while in an quadrant whose corner is not collected.

Step 1: Figuring Out How The Maze Was Generated

The maze generated from this module is random however how the maze generates can be predicted with repeated regenerations of that maze.

For ease of access, a red cube is placed to point where the current position is during its corridor generating and checking procedures. Dark blue cubes may also be placed to denote special properties of that algorithm. This table consists of each algorithm used in this module and their respective descriptions on how the algorithm determines when generating a maze.

Maze Algorithm	Description
Recursive Backtracking	The simplest algorithm to understand and to implement. Creates mainly long winding corridors with few dead ends. If it finds no unvisited cells around itself, it will simply "backtrack" to the last cell and try to continue from there. The maze is considered generated if it backtracks to the starting cell and finds no unvisited cells around that area.
Binary Tree	A crazily simple algorithm that can be used to generate simple mazes by only looking at one cell at a time. The maze generated has a strong diagonal bias alongside being skewed to a specified corner. The skew is easily determined by examining the long corridors on the edge of the maze.
Hunt And Kill	Similar to Backtracking Generator except if it finds no unvisited cells around itself, it will scan either randomly, row by row, or column by column for any unvisited cells adjacent to at a visited cell and stop on the first one it finds. It will then continue generating the corridor by connecting that unvisited cell with a visited cell and resume normally like the Backtracking Generator. The maze is considered generated when all the cells are visited.
Prim's	One of the "minimal spanning tree algorithms." Chooses an unvisited cell based off of the starting cell as a new potential cell to connect. Randomly visit a new cell based off any potential cells currently and connects it to any other visited cells if needed. Adds new adjacent unvisited cells to this visited cell if needed. Repeat until all cells are visited and connected.
Kruskal's	One of the "minimal spanning tree algorithms." Grabs all the possible edges from the given maze and randomizes the set. Select a random edge to connect and check if the cell pair connected to that edge are in the same group. Connect the cells connecting to that edge if either are not in a group or both cells are in different groups. Repeat until all edges on that set have all been used at least once.

Step 1: Figuring Out How The Maze Was Generated (Continued)

Maze Algorithm	Description
Aldous-Broder	One of the "uniform spanning tree algorithms." Starting at the provided cell, walk in ANY random direction to visit that cell. Connect if the cell was previous unvisited. Continuously walk in a random directions within the maze until all cells have been visited at least once. **This algorithm may take a long time to generate a maze.**
Wilson's	One of the "uniform spanning tree algorithms." Marks the starting cell as a branch to start off. Then starting at a random unbranched cell, walk in ANY random direction and leave a mark on the previous cell denoting the direction for later. Continuously walk in a random directions within the maze until entering a branch. Use the starting unbranched cell and the final path given to draw a branch that connects that given cell. Clear all directions not used. Repeat with a new unbranched cell as a new starting cell until all cells are branched. **This algorithm may take a long time to generate a maze.**
Sidewinder	Another simple and fast algorithm not fully researched for maximum capabilities. After the first row/column, randomly pick between connecting in that row/column or connecting towards the first row/column with the current group. At the end of the current row/column, connect randomly towards the first row/column. The difference between Binary Tree and this algorithm is the first spanning corridor given at the very edge of this maze generated with this algorithm.
Eller's	A very complicated algorithm to understand utilizing grouping cells only based off of the previous row. This algorithm will connect each row randomly, and connect at least 1 cell downwards in that group. On the last row, connect all of the remaining cells as one group. Utilized to generate infinite mazes in linear time.
Growing Tree	A configurable maze generator utilizing generating mazes based on how cells are picked. The main difference from Prim's and Recursive Backtracking algorithms is determined by modifying the odds of picking any visited cell in particular. This can even mimic other algorithms such as Recursive Backtracking and Prim's if configured.
Recursive Division	This special algorithm utilizes a different way of generating passages. Instead of starting with a maze with a bunch of walls, the algorithm will instead generate as many walls as possible while leaving a possible corridor. The algorithm will always try to split based off of what side is longer down to a 1-wide or 1-tall corridor.

Step 2: Obtaining The Starting Instructions

Once the defuser has figured out how the maze was generated, the expert must then grab the set of starting instructions based on how the maze was generated. Certain mazes that are generated will have their respective biases noted in the table which may alter the starting instructions. In the table provided, "Up", "Down", "Left", and "Right" are all abbreviated as a single letter for that given direction in the bit pair as "U", "D", "L", "R" respectively.

<table>
<tr><th>Maze Algorithm</th><th>Starting Instructions</th></tr>
<tr><td>Recursive Backtracking</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>U</td><td>D</td><td>L</td><td>R</td></tr></table></td></tr>
<tr><td>Aldous-Broder</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>L</td><td>U</td><td>R</td><td>D</td></tr></table></td></tr>
<tr><td>Binary Tree</td><td><u>Determined Skew</u>
<table><tr><th colspan="4">North-West Skew</th></tr><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>D</td><td>L</td><td>U</td><td>R</td></tr></table><table><tr><th colspan="4">South-West Skew</th></tr><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>L</td><td>D</td><td>R</td><td>U</td></tr></table><table><tr><th colspan="4">North-East Skew</th></tr><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>U</td><td>R</td><td>D</td><td>L</td></tr></table><table><tr><th colspan="4">South-East Skew</th></tr><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>R</td><td>U</td><td>L</td><td>D</td></tr></table></td></tr>
<tr><td>Prim's</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>U</td><td>L</td><td>D</td><td>R</td></tr></table></td></tr>
<tr><td>Kruskal's</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>U</td><td>R</td><td>L</td><td>D</td></tr></table></td></tr>
<tr><td>Growing Tree</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>L</td><td>R</td><td>D</td><td>U</td></tr></table></td></tr>
<tr><td>Wilson's</td><td><table><tr><td>00</td><td>01</td><td>10</td><td>11</td></tr><tr><td>D</td><td>L</td><td>R</td><td>U</td></tr></table></td></tr>
</table>

Step 2: Obtaining The Starting Instructions (Continued)

<table>
<tr><th>Maze Algorithm</th><th>Starting Instructions</th></tr>
<tr><td>Hunt And Kill</td><td>
<u>Unvisited Cell Scanning Order</u>
<table>
<tr><th colspan="4">Scanning Randomly</th><th colspan="4">Scanning Row per Row</th><th colspan="4">Scanning Otherwise...</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td><td>00</td><td>01</td><td>10</td><td>11</td><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>L</td><td>R</td><td>U</td><td>D</td><td>L</td><td>U</td><td>D</td><td>R</td><td>L</td><td>D</td><td>U</td><td>R</td></tr>
</table>
</td></tr>
<tr><td>Sidewinder</td><td>
<u>First Edge Corridor Generated From This Algorithm</u>
<table>
<tr><th colspan="4">North Corridor</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>D</td><td>R</td><td>L</td><td>U</td></tr>
<tr><th colspan="4">West Corridor</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>R</td><td>D</td><td>L</td><td>U</td></tr>
</table>
<table>
<tr><th colspan="4">East Corridor</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>R</td><td>D</td><td>U</td><td>L</td></tr>
<tr><th colspan="4">South Corridor</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>D</td><td>R</td><td>U</td><td>L</td></tr>
</table>
</td></tr>
<tr><td>Eller's</td><td>
<u>Grouping Order</u>
<table>
<tr><th colspan="4">Row Per Row</th><th colspan="4">Col Per Col</th></tr>
<tr><td>00</td><td>01</td><td>10</td><td>11</td><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>U</td><td>D</td><td>R</td><td>L</td><td>D</td><td>U</td><td>L</td><td>R</td></tr>
</table>
</td></tr>
<tr><td>Recursive Division</td><td>
<table>
<tr><td>00</td><td>01</td><td>10</td><td>11</td></tr>
<tr><td>D</td><td>U</td><td>R</td><td>L</td></tr>
</table>
</td></tr>
</table>

Step 3: Determine the Quadrant's Quirks

Each quadrant is colored one of 7 different possibilities with each of these modifying the behavior of the robit. When the move is performed on that quadrant, the robit may move in a direction that does not exactly match up with the bit table obtained from the previous step. Once the robit has reached the corner of the given quadrant or leaves the given quadrant, the quadrant's quirk will stop applying. When the robit is at the axis of the maze (the center row/column), no quirk would apply to that command. Performed commands/moves refer to the previous number of moves/commands processed. If the robit crashes while in the quadrant, the number of performed moves will not change unless the instructions are modified before restarting the robit. Modifying the instructions also resets the number of performed commands down to 0 for that robit.

Common colors are found on this page. These colors on this table may repeat on the module.

Quadrant Color	Quirk
Red	The robit moves in opposite directions while in that quadrant in repsect of the command provided. (L → R, R → L, U → D, D → U)
Magenta	While in this quadrant, if the number of performed commands so far is divisible by 4, the robit may not move for 1 command. This will skip the highlighted instruction given by this robit. Skipped instructions WILL count for performed commands.
Green	If the number of performed commands so far is even, perform the red quirk while the robit is in the quadrant. Otherwise, perform like there is no quirk.
Blue	Perform the red quirk if the Xth character of the serial number is a digit, where X is the number of commands the robit has performed so far, modulo 6, plus 1. Otherwise, perform like there is no quirk.

Step 3: Determine the Quadrant's Quirks (Continued)

This table continues from the previous page. These are rarer colors that can show up on the module. At most one of these can show up.

Quadrant Color	Quirk
Yellow	While the robit is in this quadrant, rotate the directions the robit will actually go by 90° clockwise for every command the robit has performed overall. For clarification: For 4N commands, (U → U, R → R, D → D, L → L). For 4N+1 commands, (U → R, R → D, D → L, L → U). For 4N+2 commands, (U → D, R → L, D → U, L → R). For 4N+3 commands, (U → L, R → U, D → R, L → D).
Gray	If the number of moves the robit has performed overall is divisible by 10 while the robit is in this quadrant, forcibly pause the robit after the next instruction. This requires the defuser to manually resume the robit's instructions.
Orange	Take the number of commands the robit has performed so far and modulo this by 15. Add 1 and count this many tiles in this table provided underneath, starting from the left. If that tile is a "-", the red quirk will be applied. Otherwise, if that tile is "X", the robit will be forcably paused at the end of the move. Otherwise, there is no quirk. X | * | * | - | - | * | * | - | * | - | * | - | * | * | -