MAZE SOLVER

Ali Motalebi

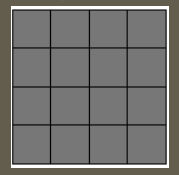
9723086

MAZE GENERATION: HUNT-AND-KILL ALGORITHM

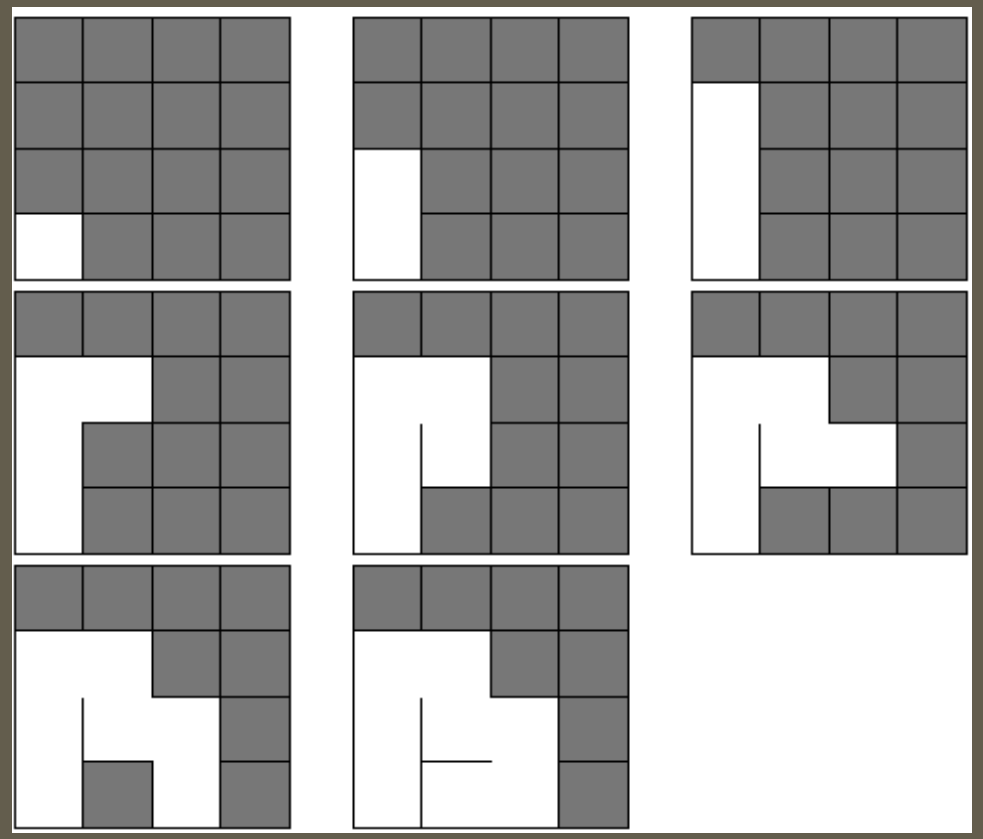
- The algorithm is the "hunt-and-kill algorithm". In a nutshell, it works like this:

1. Choose a starting location.
2. Perform a random walk, carving passages to unvisited neighbors, until the current cell has no unvisited neighbors.
3. Enter "hunt" mode, where you scan the grid looking for an unvisited cell that is adjacent to a visited cell. If found, carve a passage between the two and let the formerly unvisited cell be the new starting location.
4. Repeat steps 2 and 3 until the hunt mode scans the entire grid and finds no unvisited cells.

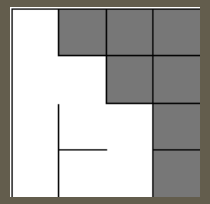
- Let's walk through an example.
- **An example:**
- I'll just use a basic 4×4 grid:



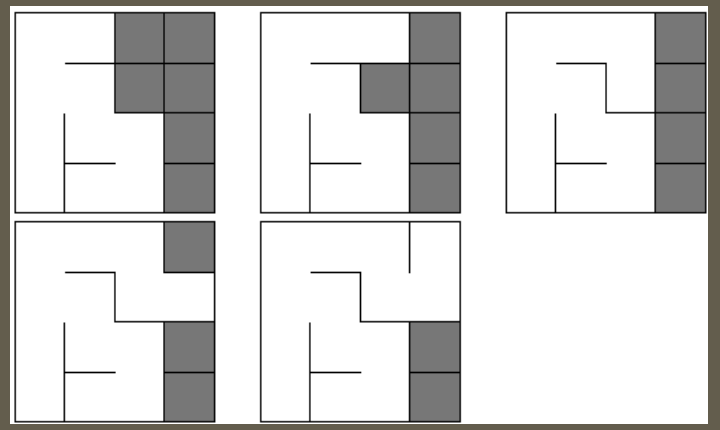
- Now, I'll give you the walk phase as a sequence of frames here. it's not that interesting, really, until it reaches a dead-end.



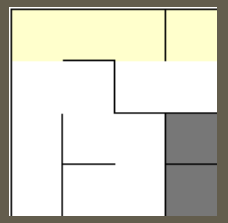
- All possible directions lead either out of bounds, or into an already-visited neighbor. At this point, the recursive backtracker would begin backtracking, looking for a previously visited cell in the stack that had unvisited neighbors. The hunt-and-kill algorithm is not nearly so sophisticated: stuck? Go hunting.
- And so we hunt. Beginning at the first row, we begin scanning each row for an unvisited cell with a visited neighbor. It turns out to be our lucky day: our very first cell is a match: unvisited, with a visited neighbor. We connect the two:



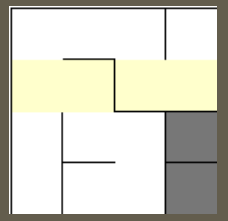
- And then we start a random walk from our new starting point:



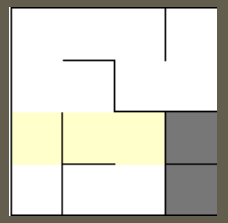
- Stuck again, so we go hunting. There are no cells in the first row that match:



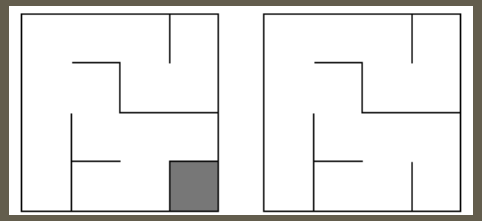
- And no matches in the second row, either. (Remember, we're looking for *unvisited* cells with *visited* neighbors.)



- The third row, however, has a match in its last cell:

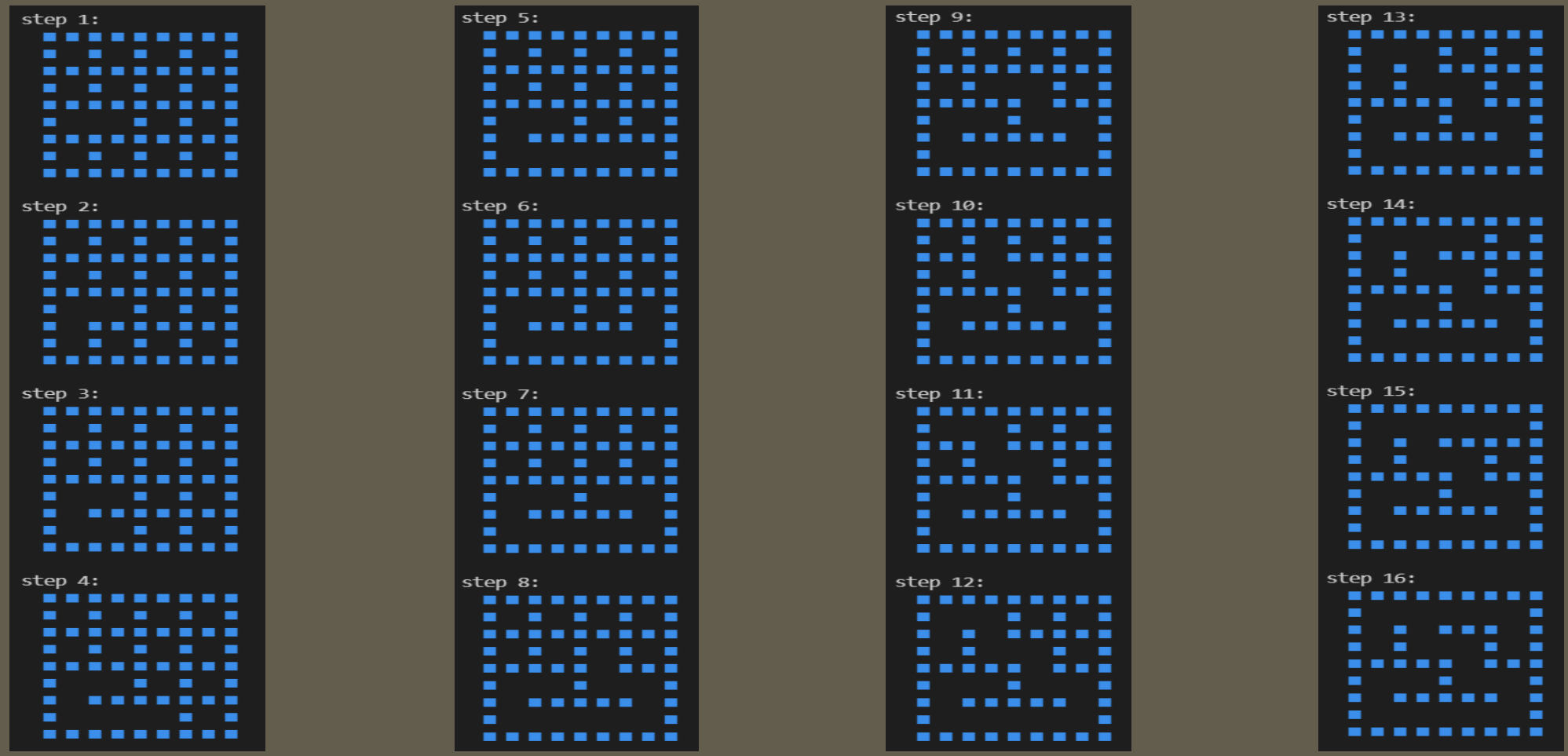


- So, we connect that unvisited cell to any one of its visited neighbors (at random), and do our random walk:



- And we again stub our digital toes on another dead-end. We're stuck, so we go hunting again, looking row-by-row for an unvisited cell.
- The scan completed without finding any unvisited cells, so the algorithm terminates and leaves us with our maze.

Example 2 (C++ code):



GRAPH CREATION

- 1.At first it set player place as the root of the tree.
- 2.Then it will add all neighbor cells that not added to the tree as its children.
- 3.Repeat step 2 for all children until all cells be added.
- Graph created!

BFS & DFS

- BFS:
- Now we have the tree and we just have to check nodes(cells) with the same depth and explores all of the nodes at the present depth prior to moving on to the nodes at the next depth level.
- DFS:
- It uses the opposite strategy of BFS. It instead explores the node branch as far as possible before being forced to backtrack and check other nodes. The sequence of checking children is right→left→down→up .

GAME MODE

- You are the red man. Let's go find your friend and have fun.

REFERENCES

- Ansi color codes:
- https://gist.github.com/zyvitski/fb12f2ce6bc9d3b141f3bd4410a6f7cf/revisions
- Maze generation algorithm:
- http://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm.html
- Symbols and ascii art:
- https://fsymbols.com

MY GITHUB REPOSITORY

- https://github.com/a-motalebi/MazeSolver