Computational Thinking with Maze Generation

Nicholas Miehlbradt

March 25, 2022

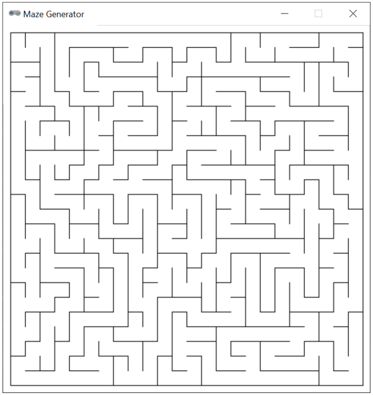
What is computational thinking?

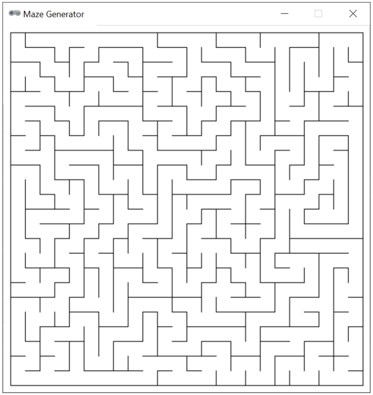
We're going to apply a computational thinking approach to solve a problem.

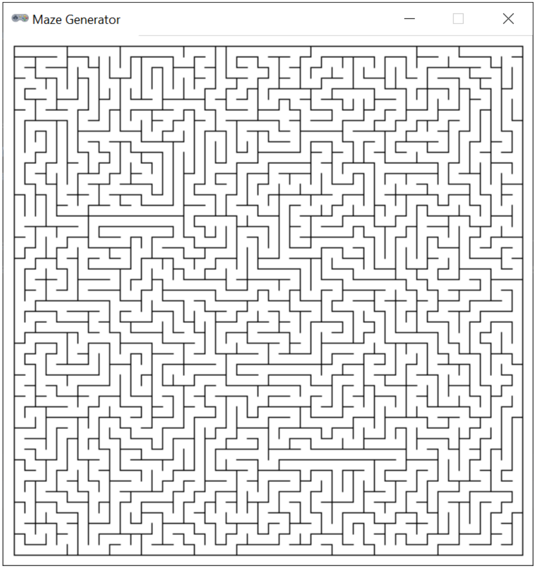
How do you generate a random maze?

1. Start with a vague problem
2. Break it down to an algorithm that can be implemented
3. Hopefully get to some coding

Maze Generation






Maze Generator

Constraints

- Working on a grid of squares, all walls at 90°
- All squares must be part of the maze (no unreachable areas)
- No loops in the maze (i.e. you can reach every point via exactly one path)

How would you solve the problem?
How do we turn this into something that we can implement?

One way of doing this?

1. Carve out a random path
2. Do this until you run into a dead end
3. Repeat for the rest of the space until everywhere is filled

This is still very general but we have some steps make more specific.

What does it mean to draw a random path?

1. Pick a starting square
2. Look at the adjacent squares that are not already part of the path
3. Choose one at random
4. Make a path between the starting square and the chosen square
5. Make the chosen square the new starting square and repeat

Ambiguities

- What does it mean for squares to be adjacent?
- How do we check if a square is not already part of the random path?
- How do we make a path between squares?
- ...

Ambiguities

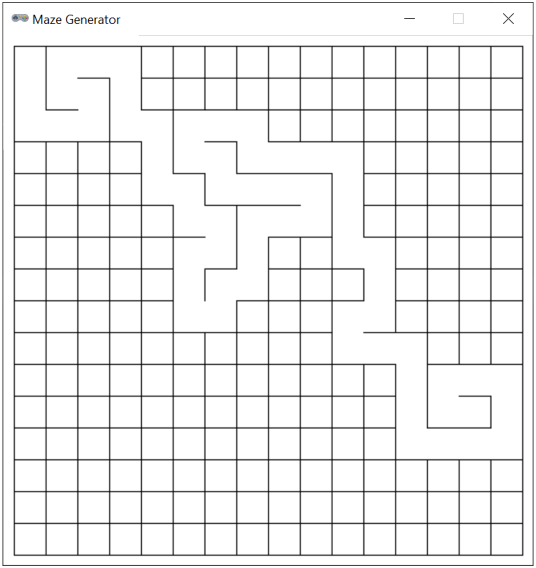
- What does it mean for squares to be adjacent?
  - We mean orthogonally adjacent (above, below, left right)
  - Use a coordinate system to talk about positions
  - Need some special cases to handle edges
  - Simple enough to write a function
- How do we check if a square is not already part of the random path?
- How do we make a path between squares?
- ...

A better algorithm

1. Initialise `current_square` variable to a starting square (e.g. (0, 0))
2. Get a list of adjacent squares which have all their walls
3. Pick a random adjacent square that has all it's walls
4. Remove the wall between the `current_square` and the chosen square
5. Update `current_square` to the chosen square

A better algorithm

1. Initialise `current_square` variable to a starting square (e.g. (0, 0))
2. Get a list of adjacent squares which have all their walls
3. If this list is empty we have reached a dead end. Terminate
4. Pick a random adjacent square that has all it's walls
5. Remove the wall between the `current_square` and the chosen square
6. Update `current_square` to the chosen square


Maze Generator

A better algorithm

1. Initialise `current_square` variable to a starting square (e.g. (0, 0))
2. Get a list of adjacent squares which have all their walls
3. If this list is empty we have reached a dead end
    1. What do we do here?
4. Pick a random adjacent square that has all it's walls
5. Remove the wall between the `current_square` and the chosen square
6. Update `current_square` to the chosen square

A better algorithm

1. Initialise `trail` list with a starting square
2. Get a list of adjacent squares which have all their walls
3. If this list is empty we have reached a dead end
    1. Step back one square and try again
4. Pick a random adjacent square that has all it's walls
5. Remove the wall between the `current_square` and the chosen square
6. Update `current_square` to the chosen square

A better algorithm

1. Initialise `trail` list with a starting square
2. Get a list of adjacent squares which have all their walls
3. If this list is empty we have reached a dead end
    1. Step back one square and try again
    2. What happens when we have no more squares to step back?
4. Pick a random adjacent square that has all it's walls
5. Remove the wall between the `current_square` and the chosen square
6. Update `current_square` to the chosen square

A better algorithm

1. Initialise `trail` list with a starting square
2. Get a list of adjacent squares which have all their walls
3. If this list is empty we have reached a dead end
    1. Step back one square and try again
    2. We are done!
4. Pick a random adjacent square that has all it's walls
5. Remove the wall between the `current_square` and the chosen square
6. Update `current_square` to the chosen square

Now that we have a good idea of how the algorithm works we can go and implement it!
Some parts are still a bit vague e.g. how do we represent and deal with walls, what does it mean to add/remove a wall. We could repeat the process until it becomes obvious.

Next steps...

How does this generalise?

What other kinds of mazes could we generate?

- Hexagonal grids?
- Irregular grids?
- Curves?
- What would we need to do to allow loops?

What other uses could this algorithm have?