### **Recursive Backtracking**

* **Source**:[Jake Mills on Medium](https://jmhero05.medium.com/maze-generation-with-depth-first-search-and-recursive-backtracking-f341c8997867).
* **Key Aspects**:
  + Uses recursion to generate mazes.
  + Starts from a random cell, moves to an adjacent unvisited cell, and backtracks when it hits a dead end.
  + Simple and easy to implement in python
* **Justification**: Good for small to medium-sized mazes. It's efficient and creates mazes that appear random, but with larger maze sizes it significantly suffers in performance loss

​​​​​​

### **Prim's Algorithm**

* **Source**: [Jonathan Zong's Blog](http://jonathanzong.com/blog/2012/11/06/maze-generation-with-prims-algorithm).
* **Key Aspects**:
  + Views the maze as a graph and constructs a ‘minimum spanning tree’.
  + Starts from a random cell, moves to an adjacent unvisited cell
  + Does not generate perfect mazes, but generates structures that look more natural.
* **Justification**: Good for making non-perfect mazes that feel more natural. Also I want to include this algorithm in my program so that the user can see differences between prim’s algorithm and recursive backtracker, and to add extra complexity to the program and make it more interesting to the user

​​

### **A\* Search Algorithm**

* **Source**: [Laurent Luce's Blog](https://www.laurentluce.com/posts/solving-mazes-using-python-simple-recursivity-and-a-search/).
* **Key Aspects**:
  + Pathfinding algorithm that calculates the shortest path in a maze.
  + Uses a cost function combining the actual distance from the start point and an estimated distance to the end point.
  + Good at finding the optimal path in the kind of mazes that my program will be generating, small-medium sized complex mazes
* **Justification**: Its use of heuristics to estimate distances makes it faster and more efficient than Dijkstra's Algorithm for pathfinding in mazes

​​​​