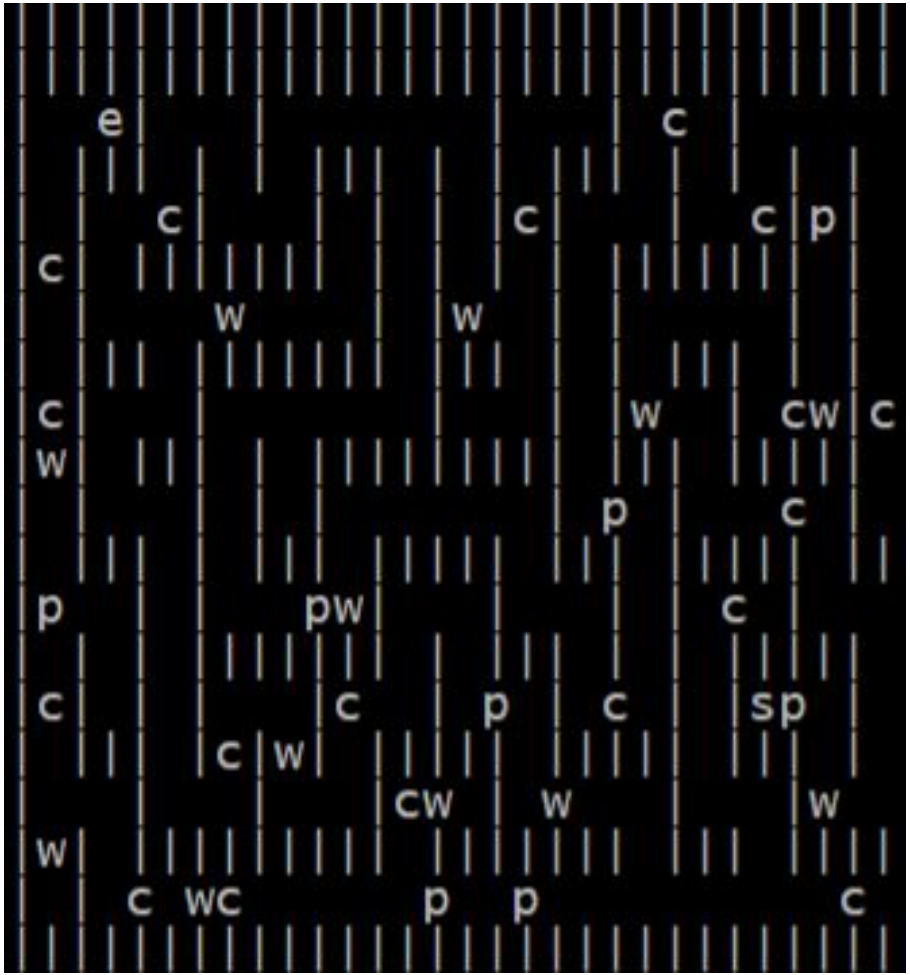
Enter The Maze

Build and Solve a Maze using Python

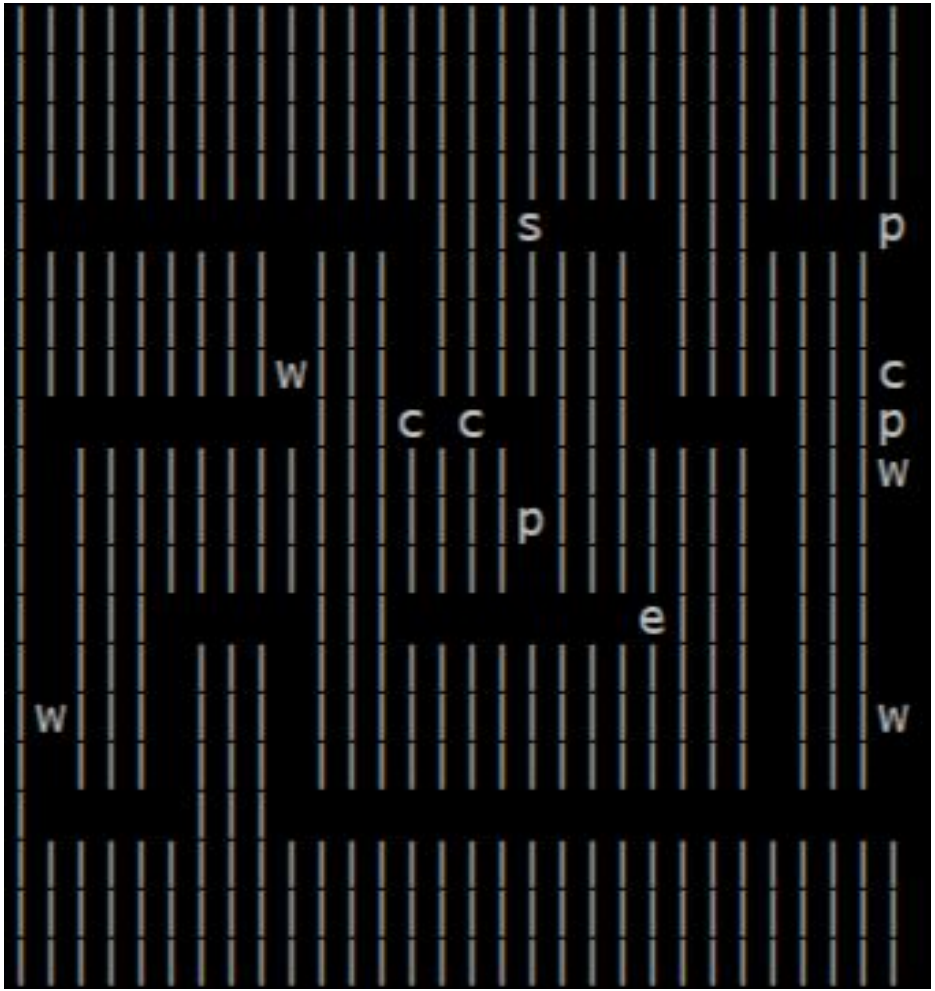
Maze (2 spaces)

Created using a method that moves 2 spaces when ‘Mowing’ the walls down to create open space.



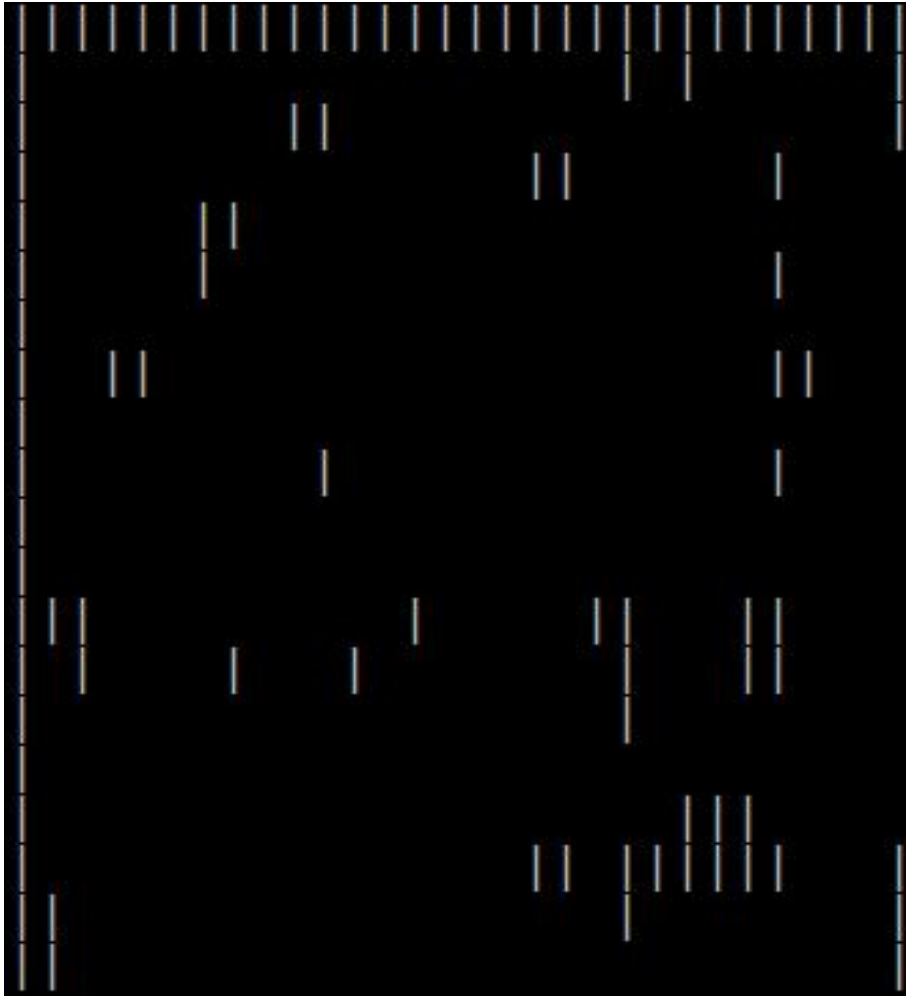
Maze_more (4 Spaces)

What happens if we ‘Mow’ more than 2 spaces to create a maze? Here we ‘Mow’ 4 spaces when creating the maze and end up with thick walls and not much of a maze.



Maze_rand (3 - 5 Spaces)

What happens if the number of spaces we move for ‘Mowing’ is not constant? Here is a maze where we move 3,4 or 5 spaces, picked randomly, before ‘Mowing’. This creates a maze with mostly open space and few walls. Not much of a maze either.



Solve The Maze

Which Graph Search Algorithm to Use?

We will be looking at the first maze where 2 spaces were moved when ‘mowing’

Breadth First Search (BFS) or Depth First Search (DFS)?

Before solving the maze we should create a copy of the maze and label each ‘Mowed’ space as either a path or a vertex.

A path is a space that has only 2 missing walls, that one that we enter from, and the one we can leave from.

A vertex has more than 2 missing walls and thus leads forward in more than one way or only has 1 missing wall and is a dead end.

Solve The Maze

BFS

With each move in the maze, the BFS algorithm with check if the space is a path or a vertex. If it is a path space it will continue to the next non-wall space as there will be only one. If the next space is a vertex it will travel down all paths until it finds another vertex or the End space. If all it finds are vertices, it will repeat this process with each possible vertex until it finds the End space.

Pros - find the shortest path

Cons - possibly use a lot of memory keeping track of collected swag from each path

Solve The Maze

DFS

With each move in the maze, the DFS algorithm with check if the space is a path or a vertex. If it is a path space it will continue to the next non-wall space as there will be only one. If the space is a vertex, it will pick a path and travel down it to the next vertex space. If the next vertex space is a dead end vertex, it will return the previous vertex and try another path. It will repeat this process at each vertex until it find the End space.

Pros - Might find a path very quickly

Cons - if there are many possibilities, it might overflow the stack with recursion

Solve The Maze

The algorithm used to make the maze uses aspects of both BFS and DFS. While mowing ro make empty spaces, recursion is used similar to DFS. While exploring the maze to add swag, a queue is used similar to BFS. When I tried making the maze with a non-constant number move movements, I would run into a recursion error when the maze go too big. This leads me to think that using BFS would be a better Idea to solve a maze of any size without error.

Store and Sort

As a patron moves through the maze using a BFS algorithm, there will be a few different paths explored before finding the End and we won't know which path is the correct path until we find the End. For this reason, I recommend using a dictionary to store the swag that is collected. Each new path explored during the solving of the maze can correspond to a key in the dictionary and a list of collected swag can be assigned to that key. When the correct solution to the maze is found, lists of swag from the paths between each vertex that were used to reach the End can be added together to produce the final swag list.

Store and Sort

In order to sort this list I would use Merge/Sort algorithm. The Merge/Sort algorithm takes and unsorted list recursively splits it into smaller lists by dividing them in half until each list is 1 item long. It merges the lists back together by looking at the first element of two lists and comparing them, appending the smaller of the elements first. What you are left with is a sorted list with a run time of O(N*log(N)).

The other possible algorithm is the Bubble sort but it has a run time of O(n^2) and so is a little bit slower than the Merge/Sort.