The user is able to set the dimensions of the grid as well as the percent chance that each tile of the grid is a pothole. In this case, we create a 4 x 4 grid with a 25% chance that each tile will be a pothole.

After the grid is initialized as a two-dimensional array, a tile from the leftmost column of the grid is randomly selected as the starting point. If a pothole exists at the starting tile, the pothole is removed.

Paths are generated recursively. The tiles to the right, above, and below the starting tile will be checked for potholes in this order. If they are not potholes, each of these tiles will then go through the same process until a tile on the rightmost column is reached. To prevent backtracking, it is made sure that the tile above or below the current tile is not the same tile that had just been checked. The tile to the right of the current tile does not need to be checked for backtracking because Henny Penny will never move left, ensuring that any tile to the right of Henny Penny is a new tile.

Because the rules state that (0, 0) is the bottom-left corner of the grid and two-dimensional arrays start at the top-left corner, some calculations must be made to form the correct coordinates of the tile:

t (total number of rows) = 4 (in this case)

c = column of current tile

r = row of current tile

The correct coordinates of the tile are (c, t - 1 - r). In each call of the pathfinding algorithm, the coordinates of the current tile will be combined with the coordinates of tiles that have been checked before it to form a path string. This path will then be passed on to the next recursive call. If the rightmost column is reached, the path will be added to an ArrayList where it can be easily looped and printed.

This solution can be implemented with grids of any amount of rows or columns. The maximum amount of valid paths is equal to r^(c - 1). This number can be very high for a relatively large grid with a lack of potholes and may cause the Java heap to run out of space.