# Project 3 report

Maze Solving Analysis

20B11806

Saw Kay Khine Oo

1. **Explanation of the project**

   In this project, we will create and solve a maze having starting point and goal point. The main alterations we can make are the number of size of the maze and the probability of walls inside the maze, which will contribute to the success of solving the maze. Thus, we can make simulations and analyze how the percentage of success of solving the maze changes according to different combinations.

2. **How the project works**

   **2.1 Creating the maze**

   We will create a square maze with the number of small cells being $n^2$. There will or will not be a wall between two cells because it will be determined by the given probability. The cells will also be numbered from top row to bottom row and for each row, left column to right column. Since we will start from 0 for top left cell, the goal, that is, the bottom right cell will be $n^2$-1.

   We will also create adjacent list, which will contain information for the name of adjacent cells we can go through from the current cell. The adjacent list will have number of $n^2$ sub-lists and each sub-list can only have four elements as the most because obviously, each cell being a square can only have four adjacent cells as the most. Then, we will print the maze with S as starting point written in cell 0 and G as goal point written in cell $n^2$-1.

   **2.2 Exploring the paths from Start to Goal**

   The idea here is to mark cross for every cell we can go through from starting cell and then check if there is a cross in the goal cell. If the goal cell is visited, then we can assume that there is a path from starting point to goal. So, the maze can be successfully solved.

   In order to accomplish this, we will have the Boolean list “visited” for all the available cells so that we can keep track of which cell has been successfully visited. Then, we will have two main lists: lastVisitedCells and nextVisitedCells. We use nextVisitedCells to keep track of the next reachable cells given by the adjacent list. During the looping of appending the value to nextVisitedCells, we will also update the corresponding element in “visited” list to True. LastVisitedCells list will only have the value 0 in the beginning as it is the starting cell. After that, we will update the list with the new value from nextVisitedCells. When there is no more available cell in the nextVisitedCells, that means the Breadth-First Search is finished, so, we can break the loop. Then, we will check if the goal $n^2$-1 is visited and return True of False. We can also print out the message saying whether there is a successful path from starting point to goal or not.

While defining the function, I added two more parameters: start and goal so starting point and goal don't always need to be 0 and $n^2$-1. However, during the simulation, only 0 and $n^2$-1 will be used as starting point and goal.

## 3. Simulations and Analysis

We will have 3 main function for simulation section. The first simulateFixedParameters is for simulation of fixed maze size (will be referred as n later) and fixed probability (will be referred as p later). The second simulate_fixed_n is to fix n and loop through the given list of probabilities. The third function simulate_fixed_p is to fix p and loop through the given list of maze sizes. Each function will be explained first and then followed by the analysis.

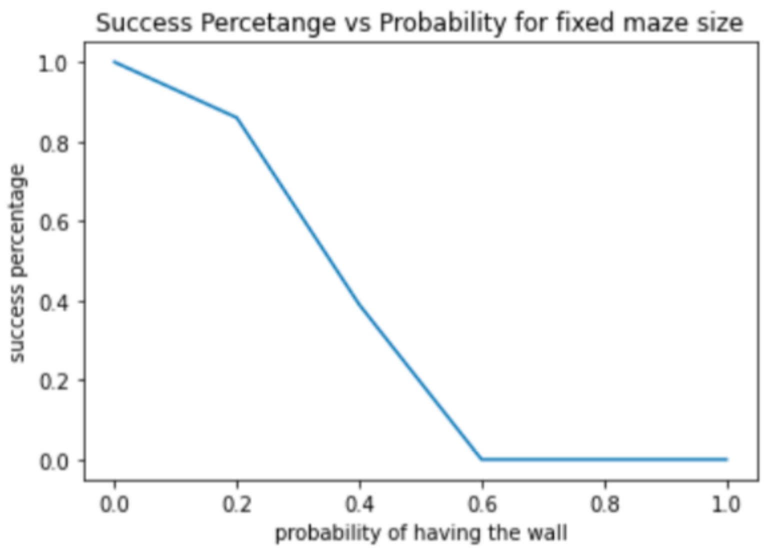
### 3.1 simulateFixedParameters (fixed n and p)

This function will take 3 parameters: n, p and number of simulations. We will loop through the number of simulations and generate the maze each time and check if the Breadth-First Search was successful. If it is successful, we will add 1 to the "success" variable each time. After successfully looping, we will divide "success" by number of simulations, this will give the success percentage.

### 3.2 simulate_fixed_n

This function will take 3 parameters: n, list of probabilities (listP) given and number of simulations. The purpose of this function is to loop through the listP with fixed n. In each loop, we can simply recall the simulateFixedParameters function to calculate success percentage.

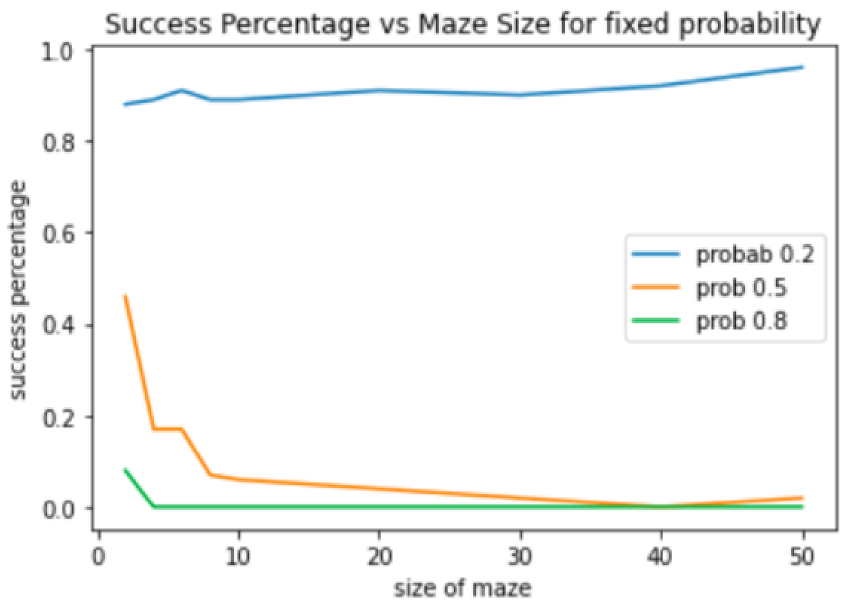
The graph below is the result of this simulation. Since probability only ranges from 0 to 1, a list of 0.2 interval is used. For the simulation, a maze size of 100 is used and number of simulations is done 100 times to produce better results. The overall result is as expected because higher probability of having the walls between cells obviously contributes to lower success percentage. However, I was surprised to find that even from a probability of 0.6, it is almost impossible to succeed solving the maze.



### 3.3 simulate_fixed_p

This function will take 3 parameters: list of maze sizes (listN), given p and number of simulations. The purpose of this function is to loop through the listN with fixed P. In each loop, we can simply recall the simulateFixedParameters function to calculate success percentage.

We can guess that even if the probability is fixed, success percentage will be different if we alter the maze size. The graph below is generated with 2 to 50 maze sizes and 100 simulations. From the graph below, we can see that the success percentage of small maze size is clearly higher than the large maze size for a given probability. Thus, if the maze is small enough, the success percentage won't necessarily be zero with 0.6 probability. I expected that with 0.5 probability, success percentage will be around 0.5 and it seems like that is true for small maze size. When the maze size becomes bigger, that doesn't fit anymore.



## 4 Conclusion

Through this project, I learnt how to implement the Breadth-First Search algorithm and how most traversal graphs work. Moreover, I also learnt how to compare different combinations and realized that how important it is to pay attention to the variables that are contributing to the results.