Project 3 Report

Hongquan (Alex) Xiao, Section 1
Yiyang (YOHO) Chen, Section 2
Lang (Lauren) Chen, Section 1

1 Specification

The goal of this project is to write a program that will automatically generate and solve mazes. Each time you run the program, it will generate and print a new random maze and the solution. You will use depth-first search (DFS) and breadth-first search (BFS).

2 Design

Maze Generator

To generate a maze, first start with a grid of rooms with walls between them. The grid contains r rows and r columns for a total of $r \times r$ rooms.
Our objective here is to create a perfect maze , the simplest type of maze for a computer to generate and solve. A perfect maze is defined as a maze which has one and only one path from any point in the maze to any other point. This means that the maze has no inaccessible sections, no circular paths, no open areas.
Now, begin removing interior walls to connect adjacent rooms. The difficultly in generating a perfect maze is in choosing which walls to remove. Walls should be removed to achieve the following maze characteristics:

- Randomized: To generate unpredictable different mazes, walls should be selected randomly as candidates for removal.
- Single solution: There should be only one path between the starting room and the finishing room. Unnecessarily removing too many walls will make the maze too easy to solve. Therefore, a wall should not be removed if the two rooms on either side of the wall are already connected by some other path.
- Fully connected: Enough walls must be removed so that every room (therefore also the finishing room) is reachable from the starting room. There must be no rooms or areas that are completely blocked off from the rest of the maze.

```
create a CellStack (LIFO) to hold a list of cell locations

set TotalCells= number of cells in grid
choose the starting cell and call it CurrentCell
set VisitedCells = 1

while (VisitedCells < TotalCells)
{
  find all neighbors of CurrentCell with all walls intact

  if (one or more found choose one at random)
  {
    knock down the wall between it and CurrentCell
    push CurrentCell location on the CellStack
    make the new cell CurrentCell
    add 1 to VisitedCells
  }
  else
  {
    pop the most recent cell entry off the CellStack
    make it CurrentCell
  }
}
```

DFS Solution

```
create a CellStack (LIFO) to hold a list of cell locations

choose the starting cell and call it CurrentCell

while (currentCell != the right bottom right cell (ending))
{
   if (any way out from current cell)
   {
      randomly push one neighbor cells that current cell can reach into CellStack
      make it the new cell CurrentCell
   }
   else
   {
      pop the most recent cell entry off the CellStack
      make it CurrentCell
   }
}

whatever left in CellStack is the solution path
```

BFS Solution

```
create a CellQueue (FIFO) to hold a list of cell locations

choose the starting cell and call it CurrentCell

while (currentCell != the right bottom right cell (ending))
{
   detect all ways out from current cell
   enqueue all neighbor cells that current cell can reach into CellQueue
   set the parent of all neighbor cells that current cell can reach to current cell

   dequeue the most recent cell entry off the CellQueue
   make it CurrentCell
}

backtrack the parent from the right bottom right cell (ending) all the way until the top right cell (starting) to
    find the solution path
```

3 Conclusion

Final Result