Yifan Guo

[Yifan.guo@vanderbilt.edu](mailto:Yifan.guo@vanderbilt.edu)

Estimated time: 24 hours

Actual time spent: 15 hours

4. I found reading the spec several times, writing out the sample execution provided in the project spec, and seeing how the project was supposed to work during lecture helpful for this project.

5. I did not reference any other material besides this project description.

6. To test the final program is correct, I first executed the program on test1.txt and test2.txt with both FIFO and LIFO agendas, and traced the path of nodes printed by the program by hand to verify the path was valid. To account for edge conditions, I then removed some walls from both mazes, and repeated the step above. To test the stack implementation, I excluded the Main.cpp file from the project and added a testQueue\_t.cpp file which tests all the methods declared in the Queue\_t.h file, then added a testStack\_t.cpp file which tests all the methods in declared in the Stack\_t.h file.

7. LIFO agenda seemed to be faster at finding a solution because LIFO does depth-first search and will find the solution quickly if it proceeds down the right path. Since FIFO does breadth-first search, it takes the “safe-way” in getting to the end, by going down a little in every path until it finds the end, taking more iterations.

8. If you failed to mark maze locations as visited, both the FIFO and LIFO solver would not be able to determine if a maze was solvable for all solvable mazes (unless the finish was right next to the start location). This is because points that were already visited (but not marked) would be pushed onto the agenda an infinite amount of times, and the program would continue to process those same points forever.

9. I enjoyed writing a program that was both generic and robust, since the MazeSolver can solve mazes in 2 different ways and won’t crash if a wall or two was missing.

10. Project description was not lacking, just took a long time to read through.

11. In the future, I would like to see how to use the depth-first search to find the length of the longest valid path in a maze, and breadth-first search to find the length of the quickest path in a maze.

12. No other information to include.