# Lab 5 Backtracking with An Explicit Stack

## Goal

In this lab you will develop an animated program that searches for a target square in a maze.

## Resources

- Chapter 5: Stacks
- Chapter 7: Recursion
- docs.oracle.com/javase/8/docs/api/ —API documentation for the Java Stack class
- Maze.jar—The working application
- Lab Manual Appendix —An Animation Framework

In `javadoc` directory
- *`Maze.html`*—API documentation for the class `Maze`, which contains all the maze data

## Java Files

- *`FindDefaultDirectory.java`*
- *`Maze.java`*
- *`MazeActionThread.java`*
- *`MazeApplication.java`*

*There are other files used to animate the application. For a full description, see Appendix: An Animation Framework of this manual.*

## Input Files

- *`maze1.java`*
- *`maze2.java`*
- *`maze3.java`*
- *`maze4.java`*
- *`maze5.java`*
- *`maze6.java`*

**Commented [CH1]:** LM10 Maze1

## Directed Lab Work

All of the classes needed for the `MazeApplication` exist. This application is based on the `AnimatedApplication` framework. If you have not already, you should look at the description of it in Appendix: An Animation Framework. The class that you will be working on is `MazeActionThread`. There is one other class that is specific to this application that you will need to work with. This class is `Maze`. Take a look at them now, if you have not done so already.

*In the labs so far, no file input or output has been done. Today's application will read from a file to determine the configuration of the maze. Different Java run time environments use different directories as their default when opening a file. It might be the directory that the class is in or it may be somewhere else. The first goal is to find where the default directory is.*

**Step 1.** Compile and then run `main` from the class `FindDefaultDirectory`.

**Step 2.** Leave your Java environment temporarily and search for the file name `DefDirHere.txt`.

**Step 3.** Move the files `maze1.txt` through `maze6.txt` to the directory your particular implementation of Java reads from and writes to.

**Step 4.** Compile the class `MazeApplication`. Run the `main` method in `MazeApplication`.

*Checkpoint: If all has gone well, you should get a graphical user interface with step controls along the top and application setup controls on the bottom. There should be one text field where you can enter the name of a file containing the maze data. Type maze2.txt for the text file and then press enter. There should be a message indicating that it is now the maze input file. If not, check to make sure that you copied the file to the correct place. The application will read the maze. It should correspond, except for the start and goal, to the first maze from the pre-lab PPT. The start should be at location (0,0). The goal should be at location (1,1). Enter 2 and 5 for the start. Press enter. The position of the start star should change. Enter 0 and 3 for the goal. Press enter. The position of the goal circles should change. Now the picture should match the pre-lab. Step twice. The application should finish.*

**Step 5.** Create a new class named `MazeFrame`. Make it a private inner class in `MazeActionThread`. It will be the frame class that will be used to simulate the recursive search of the maze. Create any needed methods for the `MazeFrame` class.

**Step 6.** Refer to the pre-lab and implement the code for the method `searchMaze()` in `MazeActionThread`. Don't forget to visit the goal when you find it.

**Step 7.** The code to indicate steps in the animation needs to be added. Put the following line immediately after every call to `visitSquare()` or `scheduleSquare()`:

```
   animationPause();
```

**Step 8.** Call `searchMaze()` in the method `executeApplication()`. Set the variable `goalFound` with the value returned by the call.

*Final Checkpoint: Run MazeApplication. Set up the application with maze2.txt. Use a start of (2,5). Use a goal of (0,3).*

*Step the application twice. A red dot should appear in the start square indicating that it was the most recently visited.*

*Step the application once more. A cyan circle should appear in one of the neighbors. This is a scheduled square. It should be the first one pushed on the stack.*

*Step the application twice more. The other two open squares next to the start should be filled with cyan circles, indicating that they have been scheduled.*

*Step the application once more. The last cyan dot should turn red as it is visited. The start will turn blue, which is the indication of a visited square that is not most recent.*

*Continue stepping the application. Eventually, the application should end with the red dot filling the center of the goal and a message will appear that it was found.*

*Run MazeApplication. Set up the application with maze2.txt. Use a start of (0,3). Use a goal of (0,7).*

*This time the application should search every square except goal. It will finish and the message should not appear.*

*Try the other mazes. Create ones of your own. Enjoy!*

## ***Post-Lab Follow-Ups***

1. Modify the Maze application to display the path from the starting point to the current location in blue.