CS 150: Project 3 - Maze Game

C W Liew

Version as of: 20:41 Monday 30th April, 2018
Due: 11:55pm, Saturday, May 4, 2018

1 Introduction

For project 3 you will design and implement a turn based multi-player maze game. Each player will start at some position in the maze and will try to exit the maze in the minimum number of rounds - where at each round, every player gets a turn. At each turn, each player will roll a dice for the number of steps they can move and then proceed to move the specified number of steps.

2 Project Description

The maze is represented as an undirected weighted graph with nodes connected by weighted edges. The weights specify the distance (number of steps) between nodes. The maze will be described in a graph with some of the nodes identified as exit nodes. There is a limit to how far each player can "see", i.e., how far ahead they can search. For example if the limit is 100, then the player cannot "see" any node that is more than 100 steps away from the current position. Each player will use an algorithm to determine the direction in which they will move.

3 Program Behavior

The program starts with reading in several parameters from the command line - the **limit**, the number of players, the max value for the dice, the graph description filename, a file with the starting node and the exit nodes. The program will then read the graph description and build the maze - a weighted undirected graph. The second file will provide the information about which node to start at and the names (labels) of the exit nodes. The specified number of players will be created and placed at the starting node and the game begins.

At the beginning of each round where every player takes a turn, the program will prompt the user for a command:

- 'x': exit the program
- 'c': continue running the program without stopping to prompt the user
- 'p': print the position of every player

- 'i': continue with the round, making sure every player has a turn.

At each player's turn, she will:

1. roll the dice and save the value as the number of steps to take. If the user is currently at a node, or the steps takes the user past the next node then go on to the next step to determine the direction to move past the node. Otherwise, just continue in the current direction for the number of steps.
2. use an algorithm to select the new direction, and then move to the node and past it in the new direction.

3.1 More Detailed Project Requirements

- there are at least 3 different players
- there are at least 3 different algorithms being used and one of them is either the shortest or farthest algorithm (see below)

Here are some different algorithms that you might want to design and implement. I have added some alternative interpretations/choices. You can choose any alternative you like. **: BTW, a player cannot choose the node that she just came from unless it is the only choice**

1. random: the player chooses a direction at random at every turn except for when the distance to an exit node is within the **limit.** If an exit node is within the limit, the player will take the shortest path to the exit node.

   Alternative: The player chooses a node at random from its neighbors unless one of its neighbors is an *exit* node. In that case, it will select the *exit* node.

2. first: the player always chooses the first alternative at every turn except for when the distance to an exit node is within the **limit.**

   Alternative: The players chooses the first alternative amongst its neighbors unless one of its neighbors is an *exit* node. In that case, it will select the *exit* node.

3. last: the player always chooses the last alternative at every turn except for when the distance to an exit node is within the **limit.** If an exit node is within the limit, the player will take the shortest path to the exit node.

   Alternative: The players chooses the last alternative amongst its neighbors unless one of its neighbors is an *exit* node. In that case, it will select the *exit* node

4. shortest: the player always heads towards the closest node (found within limit distance) except for when the distance to an exit node is within the **limit.** If an exit node is within the limit, the player will take the shortest path to the exit node.

5. farthest: the player always heads towards the furthest node (found within limit distance) except for when the distance to an exit node is within the **limit.** If an exit node is within the limit, the player will take the shortest path to the exit node.

Algorithms 4 and 5 use a modified version of Dijkstra's shortest path algorith where if the distance from the current node to a neighbor node is further than the **limit** value passed to the program, then the neighbor node is not seen/visited, i.e., it appears not to be connected to the current node. All of the algorithms assume that you will not return to the node that you just visited (no bouncing back and forth).

Project Constraints

The following constraints apply to the project:

1. The project is to be completed individually. The only person you can consult is the instructor.
2. You are only to use containers from the Java Collections Framework. The only exception is when you use a method from an API that requires the use of arrays (e.g., main()).

4 Simplifications

You can simplify your project in one (or more) of the following ways:

- only have 2 players,
- only have 2 algorithms but one of them is either the 4 or 5 algorithm.
- have more than 3 algorithms but they do not include the 4 or 5 algorithm.

5 Report

Your report should try to answer questions (you should design your own questions) like the following:

1. how does the **limit** value affect the performance of the algorithms?
2. which algorithm is best at various limit values?
3. what is the complexity of the game? How does the time required change with the number of players?

6 Submission

The project submission is in 4 parts:

1. (2.5 pts) - due Wednesday, April 25: the "story" of your program
2. (2.5 pts) - due Saturday, April 28: class diagrams for your program
3. (85 pts) - due Saturday May 5: the rest including a draft report
4. (10 pts) - due Wednesday, May 9: the final report

s

Your submission for Part 3 will be composed of the following:

1. source files (*.java) that are commented and have javadoc directives
2. test files, one test file per class
3. a README.txt file that contains instructions on how to run your program
4. a draft of the project report (see project report guidelines `www.cs.lafayette.edu/~liew/courses/cs150/writeup-guidelines.pdf`) - 50% of the report grade is assigned to this. The remaining 50% will be given to a final report. The final report will be due several days after the draft is corrected and returned.