COMPLEXITY & ALGORITHMS 3.1

[Backtracking]

FEBRUARY 2, 2020

[MICHAEL CORNELISSE (459466) LARS WIEGERS (468477)

Table of Contents

Introduction:

This report covers assignment three of complexity and algorithms performed by Michael Cornelisse and Lars Wiegers. For the third assignment we once again had the possibility to choose between two assignments, (3.1) Pawn puzzle maze and (3.2) Interplanetary maze we decided to implement assignments (3.1) Pawn puzzle maze. To be perfectly honest we weren't really sure what we were getting into when we started working on this assignment and we simply chose it because the pseudo code for the backtracking algorithms was already provided in the slides during the lecture leading up prior to that, and we felt that while the other assignment was more interesting the 3.1 pawn puzzle maze would be easier and finished earlier.

Initial analysis:

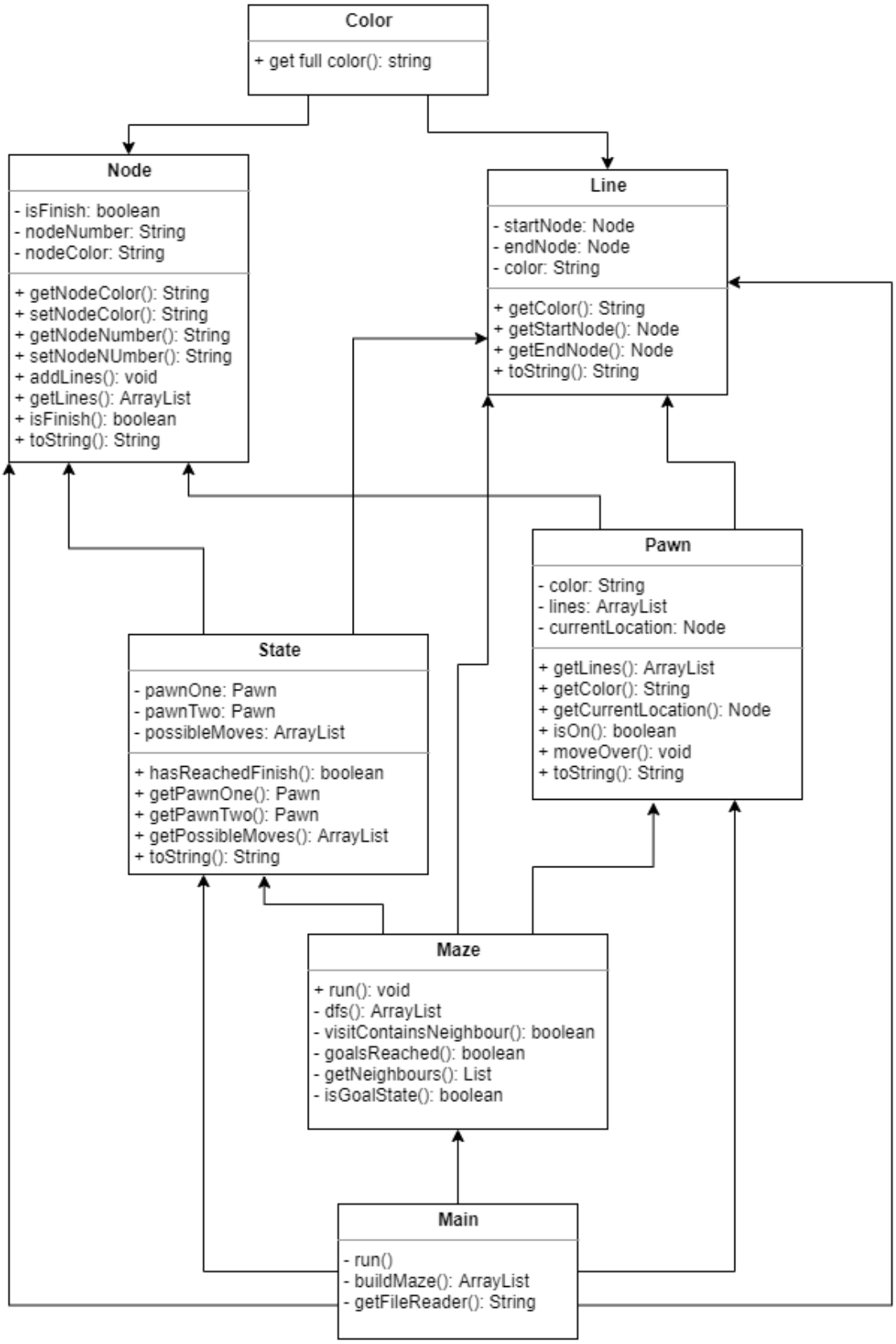
This chapter discusses the initial thoughts we had about the assignment and our initial impression of the assignment once we started working on it.

After hearing from Rene who is our teacher about how fast he finished the exercise we felt motivated to do the same. After looking at the exercise and the diagram we felt that it was something we could easily do. This later proved to be a little bit more difficult after following the code that was provided in the theory slides.

Setup:

In the following chapter we discuss our design choices for the program the following things are covered below class diagram, program overview, user manual and an elaboration on design choices and limitations.

Class diagram:



Program overview:

The program starts at Main.java. This is also where you will find your run method. From there it will read the input files and tries to find a solution to the given maze. If it can't find a solution to the finish it will return an empty array. If it can find a solution it will output all states that have happened to get there. For an example see the heading Results.

Using your own maze files:

While the project does not generate the maze / nodes for you, you yourself can make you own maze by editing the existing one. The csv files take on a simple structure that anyone should be able to understand.

Maze.csv

Start_node	End_node	Color of the line

nodes.csv

Nodes.csv is read first and contains the basic information for the maze. All the nodes that are created in here can be used in creating your actual maze in the maze.csv. Don't use any nodes in the maze that you have not defined or otherwise the maze will not be correct.

Node_name	color

Elaboration on some flaws in the design choices:

While trying to find a easy / proper way to store the maze in a text file we decided to use a string for the node Number. The reason for this is that we will also have a "finish" node which of course does not fit into an integer. After creating the program, we found this to be harder than it should be.

The code needs some unit testing. We feel this would improve the structure of the program and the confidence in the code a lot more.

Results:

The program can find a solution in the maze using a back-tracking algorithm.

Example run of the program:

Console output:

0 State{pawnOne=1 Purple [1 -> 4 Purple 1 -> 5 Black], pawnTwo=2 Black [2 -> 6 Green 2 -> 12 Purple]}

1 State{pawnOne=5 Green [5 -> 9 Orange], pawnTwo=2 Black [2 -> 6 Green 2 -> 12 Purple]}

2 State{pawnOne=5 Green [5 -> 9 Orange], pawnTwo=6 Orange [6 -> 9 Green 6 -> 10 Purple]}

3 State{pawnOne=9 Purple [9 -> 4 Green 9 -> 14 Black], pawnTwo=6 Orange [6 -> 9 Green 6 -> 10 Purple]}

4 State{pawnOne=9 Purple [9 -> 4 Green 9 -> 14 Black], pawnTwo=10 Black [10 -> 15 Green]}

5 State{pawnOne=14 Green [14 -> 20 Orange 14 -> FINISH Green], pawnTwo=10 Black [10 -> 15 Green]}

6 State{pawnOne=14 Green [14 -> 20 Orange 14 -> FINISH Green], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

7 State{pawnOne=20 Green [20 -> 19 Black 20 -> 21 Orange], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

8 State{pawnOne=21 Black [21 -> FINISH Black 21 -> 22 Orange], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

9 State{pawnOne=22 Black [22 -> 17 Orange], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

10 State{pawnOne=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

11 State{pawnOne=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple], pawnTwo=22 Black [22 -> 17 Orange]}

12 State{pawnOne=11 Orange [11 -> 12 Green 11 -> 10 Purple], pawnTwo=22 Black [22 -> 17 Orange]}

13 State{pawnOne=11 Orange [11 -> 12 Green 11 -> 10 Purple], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

14 State{pawnOne=12 Purple [12 -> 7 Green], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

15 State{pawnOne=7 Orange [7 -> 2 Green], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

16 State{pawnOne=2 Black [2 -> 6 Green 2 -> 12 Purple], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

17 State{pawnOne=2 Black [2 -> 6 Green 2 -> 12 Purple], pawnTwo=16 Green [16 -> 15 Green]}

18 State{pawnOne=6 Orange [6 -> 9 Green 6 -> 10 Purple], pawnTwo=16 Green [16 -> 15 Green]}

19 State{pawnOne=9 Purple [9 -> 4 Green 9 -> 14 Black], pawnTwo=16 Green [16 -> 15 Green]}

20 State{pawnOne=4 Green [4 -> 13 Black], pawnTwo=16 Green [16 -> 15 Green]}

21 State{pawnOne=4 Green [4 -> 13 Black], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

22 State{pawnOne=4 Green [4 -> 13 Black], pawnTwo=22 Black [22 -> 17 Orange]}

23 State{pawnOne=13 Orange [13 -> 8 Green 13 -> 18 Green], pawnTwo=22 Black [22 -> 17 Orange]}

24 State{pawnOne=13 Orange [13 -> 8 Green 13 -> 18 Green], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

25 State{pawnOne=8 Purple [8 -> 3 Purple], pawnTwo=17 Green [17 -> 11 Black 17 -> 16 Black 17 -> 12 Purple]}

26 State{pawnOne=8 Purple [8 -> 3 Purple], pawnTwo=12 Purple [12 -> 7 Green]}

27 State{pawnOne=3 Green [3 -> 1 Orange 3 -> 4 Orange], pawnTwo=12 Purple [12 -> 7 Green]}

28 State{pawnOne=3 Green [3 -> 1 Orange 3 -> 4 Orange], pawnTwo=7 Orange [7 -> 2 Green]}

29 State{pawnOne=3 Green [3 -> 1 Orange 3 -> 4 Orange], pawnTwo=2 Black [2 -> 6 Green 2 -> 12 Purple]}

30 State{pawnOne=3 Green [3 -> 1 Orange 3 -> 4 Orange], pawnTwo=6 Orange [6 -> 9 Green 6 -> 10 Purple]}

31 State{pawnOne=1 Purple [1 -> 4 Purple 1 -> 5 Black], pawnTwo=6 Orange [6 -> 9 Green 6 -> 10 Purple]}

32 State{pawnOne=1 Purple [1 -> 4 Purple 1 -> 5 Black], pawnTwo=10 Black [10 -> 15 Green]}

33 State{pawnOne=5 Green [5 -> 9 Orange], pawnTwo=10 Black [10 -> 15 Green]}

34 State{pawnOne=5 Green [5 -> 9 Orange], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

35 State{pawnOne=9 Purple [9 -> 4 Green 9 -> 14 Black], pawnTwo=15 Orange [15 -> FINISH Purple 15 -> 22 Green]}

36 State{pawnOne=9 Purple [9 -> 4 Green 9 -> 14 Black], pawnTwo=FINISH}