Projet Logipix Lundi 7 février 2022

INF442 — X2020

Auteur :
Yassine SOUDRI
Hiba KANBER

Version du
7 février 2022

Setting up the game

Task 1

We will use a matrix of clues to represent the puzzle :

At first we wanted to represent the clue as a pair, which occured to be uneefficient we then created a NEW class called Clue that contains all the information that we may need throughout the project :

— Value : an integer referring to the value of the clue, if it's a zero then the clue is empty.

— i,j : the position of the clue.

— isMarked : a boolean that says if the clue is empty or contains an integer, for a new puzzle if isMarked is False that means the value is 0.

— Colored : Going through the program we seemed to need this information in the backtracking and in the combination and exclusion approach , it distinguishes between a clue that we already visited and may or may not be part of the solution and the ones that are part of the solution for sure.

— Partof : If a certain clue has colored=2 which means its part of the solution partof gives i and j of the main clue of the path that contains this one and it is set to (-1,-1) initially .

Then we created a New class called Puzzle containing the following components :

— Height, Width

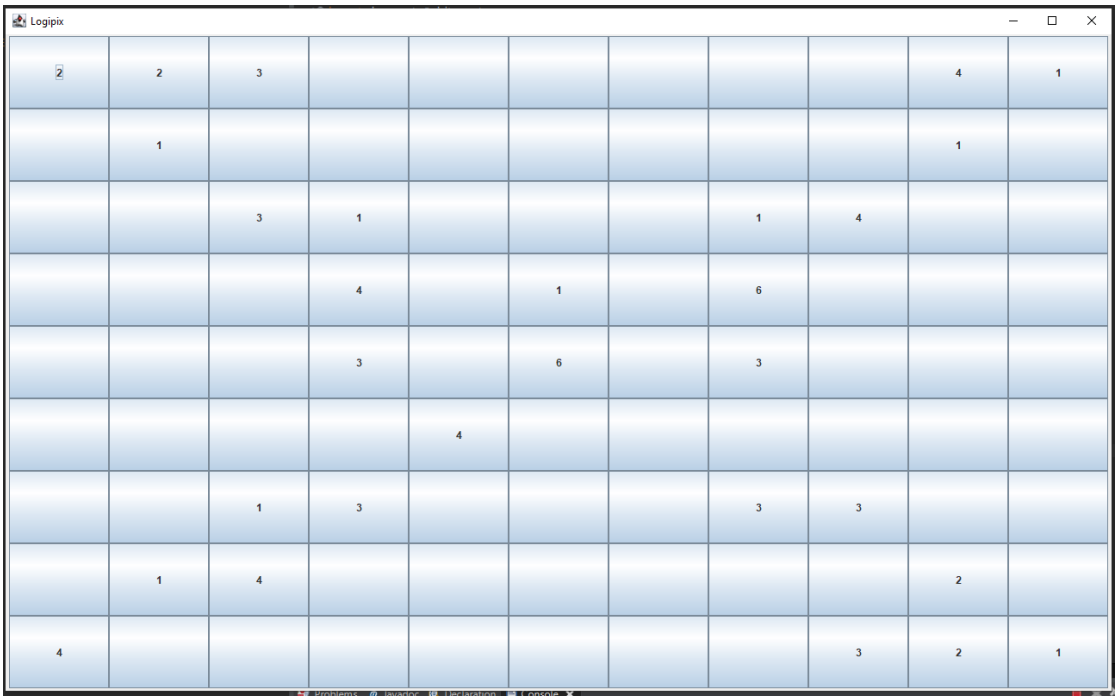
— Puzzle : A matrix containing the Clues.

— Buttons : Stocking Jframe Buttons.

We defined two constructors one that takes a matrix of Clues and the other takes a File reads it and extracts the clues that we need.

Task 2

We used a JFrame window by creating a class that extends javax.swing.JFrame class using the Buttons in the Puzzle Class.



Solving a logipix puzzle with backtracking

Task 3

For this task we chose a recursive approach it's efficient and easy to define.

First we are going to represent the solution as an Array of strings each element is a sequence of the clues' positions that belongs to the path in question, we find this representation easy to work with.

To generate all the possible lines we went through all the neighbors of the main clue that were not Colored nor Marked (except for the last part when we reach the other clue that have the same value or when we reach a clue that its colored because it is part of the solution due to the combination method) and recursively applied the same procedure on the neighbors that filled these conditions.

Task 4

We implemented the Logipix solver using Backtracking, trying to visualize all the stages that it's going through.

For each clue we had all the possible broken lines (Task 3) we then choose one of them, and color with 1 all the clues of the path in question, we also wanted to visualize clearly what it's happening by adding X on this clues in the frame and "Initial" on the main clue and "Final" on the last one so we can easily see if our solver is working correctly,in the backtracking part of this algorithm if the chosen path is not part of the solution one we discolor it.

If the list of brokenlines of a certain clue is empty we stop the tracking and return false.

Task 5

To improve the solver we had to find what makes the backtracking-based solver costly.

The first idea is that it becomes less efficient if the value of the clue increases, because the number of broken lines grows exponentially with the clue's value, so we sorted out the clues by their values starting with the smallest numbers.

The second idea using dead ends, we put the clues that have one possible broken line in the beginning of the arraylist of clues because if there is only one broken line it must be part of the solution for sure

In this dead ends algorithm we faced the ConcurrentModificationException for the first time as we tried to remove an item from a list while iterating it so we instead created a list of the items we will remove and removed them all at once after the iteration.

So that the clues with the bigger values are limited by the free space left in the grid and thus reducing the number of the broken lines and then we calculated the running times but it wasn't as expected :
for the first solver it took him around :

— 25 000 000 nanoseconds to solve the "LogiX" problem
— 1 000 000 000 nanoseconds to solve the "TeaCup" problem .

for the second solver :

— 10 000 000 nanoseconds to solve the "LogiX" problem

— 50 000 000 000 nanoseconds to solve the "TeaCup" problem .

which is better as expected for X one but unexpectedly worse for TeaCup, the difference between X and TeaCup is the size obviously but we couldn't figure out what is the exact problem here with our second Solver.

Combination and exclusion

Task6

Here we started by making two simple methods :

One that takes a clue and color every common case between all the broken lines as it will sure be part of the solution.

And the second one creates a list of all the colored clues.

So then, for our exclusion and combination algorithm we created a loop that apply the combination approach on all the clues as long as the size of the list of the colored clues is changing (this loop ends because the size of the colored clues can only increase by the combination program and it is lower than height*width so the loop must end).

Task7

We applied our exclusion and combination algorithm than made a list of the not colored clues and gave it to our solver by backtracking.

but it didn't improve the speed of our solver!!