Lab 8
Lijnenspel

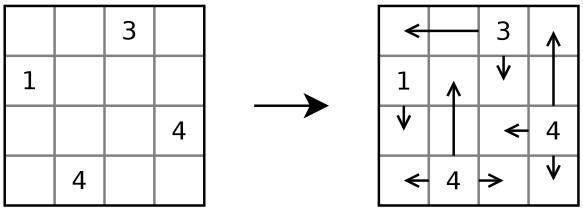
27 October 2015

The drill for this lab is given below. Come to lab on Tuesday either with it completed or with a specific written question in your notebook identifying which drill step you got to and what about it you're stuck on.

The drill this week has two semi-separate parts: one to make sure you understand the surface task (a puzzle game called Lijnenspel), and the other part to get you started on using the `Matrix` stuff we've been talking about in class. You can do the drill bits (ha!) in either order.

Drill 1: Introducing Lijnenspel

Lijnenspel[1] is a puzzle played on a grid (similar to Sudoku or a crossword puzzle). In Lijnenspel, the initial grid contains numbers in some of the squares (as on the left below), and the puzzle solver's job is to draw horizontal and vertical arrows extending out from the numbers to fill the rest of the grid. The total length in squares of all the arrows emanating from a numbered square should add up to that number, as in the solved example on the right:[2]

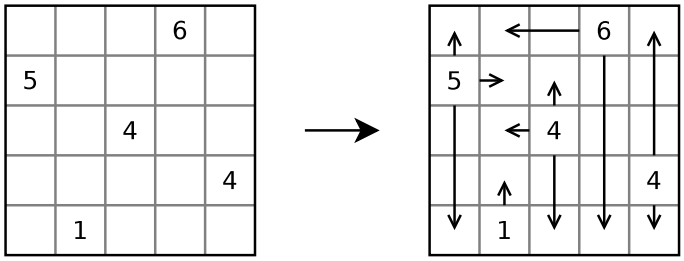


To be a proper Lijnenspel puzzle, the solution should be unique; and indeed any such puzzle with a unique solution is possible to solve without guessing (though the logic may require a bit of work). There are various tactics that can be applied. For instance, if a particular square is only "reachable" from

[1] Also known as "Line Game", but the authors of the site I pulled examples from are Dutch, and so publish it under both names. "Lijnenspel" looks cooler, no? It's pronounced "LINE-en-spell".

[2] This example comes from the description page at `http://www.puzzlepicnic.com/genre?lijnenspel` .

one numbered square, there has to be an arrow connecting them; in the following example, the second square in the top row and the fourth square in the bottom row are *only* reachable from the 6—so we could immediately "spend" six to draw arrows from the 6 to those two squares. That makes the right column unreachable except from the 4; and this sort of logic can continue through to complete the puzzle.[3]



To confirm you understand how the Lijnenspel puzzles work, work out at least the first two on the handout I gave out in class. Compare notes with other students!

Drill 2: Matrix

This part is more like the drills you've done in the past.

1. Start a program (`.cpp` file) that `#include`s the two `Matrix` headers we learned about this week, and also the `using namespace` line. See the code from `/home/shared/160-3/1026/use_matrix.cpp` if you don't remember how! For now, have it create a 4×4, 2D matrix of `int`, and print that out to the screen. Make sure this much compiles. (As *always*, try to compile as often as is reasonable, and start running your code as soon as it's feasible to do so.)

2. Before creating the matrix, read a single positive integer from the user; modify the matrix creation so that it's a square of that size. (For instance, if the user typed 3, the matrix would be a 3×3 grid.)

3. Change the contents of the matrix from `int` to `char`, add code to read in the grid from `cin` after reading the intended size of the grid, and

[3] This puzzle by Zack Butler of RIT, who also provided the inspiration for this lab.

build two test files with input appropriate to this format (a size, and then a curly-bracket-delimited matrix of characters).

4. Edit your examples to correspond to either completed or non-completed Lijnenspel puzzles. In a starting-position puzzle, each char will be either a digit from 1 to 9, or a period '`.`' for an open square. In a completed puzzle, all the periods will have been replaced with one of '`<`', '`>`', '^', or '`v`', depending on which direction their arrow was going. (Displaying and storing a length-3 arrow as "`>>>`" instead of "`-->`" will make our life easier later, but will still be easy to interpret visually.)

5. Write a function `count_numsquares` that takes a 2D grid (that is, a `Matrix<char,2>`) and counts and returns how many of the squares in the given grid are number squares. Two notes: first, to check if a character is a digit, you'll compare it to `'0'` (or `'1'`) and `'9'`—note the single quotes. Second, in this function you'll need one `for` loop to go through the rows, and another `for` loop *inside* it to go through each element in each row. In your `main` function, after you've printed the grid itself, print how many of its squares are number squares (i.e. the result of this function).

6. Write a second function `sum_numsquares` that takes a 2D grid and computes the total of all the number squares in the given grid. (So, for the Lijnenspel on the front page of this handout, it should return 12: 1+3+4+4.) Note that since we store a number square as a character rather than a number, you'll have to adjust it before doing math with it—if you remember in Lab 5 that `'a' + 3` yielded `'d'`, you might not be surprised to find that `'3' - '0'` yields the actual int value 3. Make use of this fact when computing your sum! Then, in your `main` function, also print the result of this function.