Lab 12
Lijnenspel revisited

24 November 2015

Reading the code

The drill this week is to read, analyse, and answer questions about code.

Regardless of how far you got on Lab 8, it will be instructive to read someone else's implementation—mine—and see what's done the same and what's done differently. I've put my implementation in `/home/shared/160-3/lab12`. Copy my implementations into your directory so you can edit them and compile them.

The files `lijnenspel.h`, `lijnenspel.cpp`, and `check_board.cpp` represent a complete implementation of the requirements of Lab 8; `check_board.cpp` has the `main` function and everything else is in the `lijnenspel` files. The file `play_board.cpp` is a further extension designed to support a user "playing" a Lijnenspel board (i.e. trying to solve it).

As you read through the code, you will find it helpful to refer back to the Lab 8 handout. You should also feel free to compile and run the code, and make your own changes to it in order to better understand what each expression or line of code accomplishes. Nothing in any of these files is meant as a particularly fancy C++ trick (although there are one or two things you haven't seen); if you don't understand what something does, you should ask.

To guide your exploration of the code, and also to structure the assignment, I've posed a number of how-and-why questions about my code, which I handed out in class Monday (also available as PDF from the course website). Concise answers are fine (if they're correct!), but make sure they're clear. Include line numbers where relevant. Bring them with you to lab; I'll expect to see an attempted answer for each question. We'll go over the answers early in the lab period before moving on with the lab.

Also: I don't usually put the features of the day in the drill handout, but since they have to do with searching through files, you might find them really helpful to use *during* the drill, so here they are!

Command line FOTD: `grep`

The `grep` command is a general search tool that lets you find occurrences of some pattern in a whole batch of files. For instance, if you go to the `/home/shared/160-3/lab12` directory (where my implementations are) and type

```
grep grid lijn*.*
```

you'll get a listing of every time the `grid` variable shows up (in either file); what it's doing is going through each source file and printing out every line that contains the string "`grid`".

But `grep` is more powerful than that. Its first argument is what's called a "regular expression" or "regex", and lets you search for some pretty complicated things. You can get more information on this on your own, but a few quick tricks:

- By enclosing the pattern in (single) quotes, you can search for strings with spaces in them:

  ```
  grep 'const Matrix' lijn*.cpp
  ```

- If you want to match any single character, use a period:

  ```
  grep 'int . = 0' lijn*.cpp
  ```

  matches any line that inits a single-letter variable.

- To match any amount of any text, use the period wildcard with an asterisk:

  ```
  grep 'if.*grid' lijn*.cpp
  ```

- To match the beginning or end of the line, use caret and dollar-sign respectively.

  ```
  grep '^int' lijn*.cpp
  ```

  will give just those lines that *start* with `int`.

Vim FOTD: searching

From command mode, if you hit the forward slash key, it's a little bit like colon mode: the cursor moves to the bottom of the screen and awaits further input. But what it's waiting for now is a regular expression to search for.

Having just explained regexes in the context of `grep`, there's not much more to explain here; they work essentially the same way. After the initial slash, you type a regex and hit enter, and Vim will find the next place in the file that matches that regex, or if there are none it will tell you that.

Also inside command mode, the `n` command will repeat the previous search. So pressing `n` repeatedly will cycle through all matches in a file. Using `N` instead goes through matches in reverse order.

The `n` command together with the period command (which repeats the previous command) is a workhorse combination: first, search for a pattern and do something; then alternate `n.n.n.` until you've done your action every place that pattern occurs.