A game!

Due: 4 Nov 2005

Educational objectives:

- ***Primary objectives:*** **Experience working in groups, implementing simple stack and queue classes, using stacks to implement depth first search, and using these to play well, or optimally, a game called *NIM* .**
- ***Secondary objectives:*** **Developing a user interface, implementing and using templates.**

Statement of work: (i) Implement generic *stack* and *queue* classes, and (ii) use your stack class to play the game *NIM*, as described below.

Deliverables:

- ***Oct 13, during class:*** **Turn in a hardcopy of a game-tree for *Max* starting with the state `(1, 2, 1)`. (We will discuss game trees in recitation and in class 11 Oct 2005.) Each individual (rather than each group) must turn in a solution, for this task alone.**
- ***Oct 14, 5 pm:*** **Email me the names of the people in your group. A group may contain three or four students. If you want me to assign you to a group, then please let me know. If I don't hear from you by the deadline, then I will assign you to a group of my choice!**
- ***Oct 21, 5pm:*** **Turn in a hardcopy of a simple design document that describes different classes that you will implement, the basic idea behind the algorithm for your program, the person responsible for implementing each feature, and the amount of time that you expect to take for implementing each feature.**
- ***Oct 28, 5pm:*** **Give me a hardcopy of a progress report that describes the different features that you have implemented.**
- ***Nov 4:*** **Turn in a `makefile` and all header (*.h) and cpp (*.cpp) files that are needed to build your software, as described in www.cs.fsu.edu/~asriniva/courses/DS05/HWinstructions.html. Turn in your development log too, which should be a plain ASCII text file called `LOG.txt` in your project directory. You will submit all of these using the `project4submit.sh` script.**
- ***TBA:*** **Group demonstration of your project.**

Background: This game involves two players, and starts with a certain number of heaps of objects (say sticks), with each heap containing a certain number of sticks (the number may vary from heap to heap). The players alternate in taking turns. In each turn, a player has to remove objects from exactly one heap. The player may remove any number of sticks from that heap, but must remove at least one. The game ends when there are no more sticks left in any of the heaps. The last player to remove a stick loses.

Requirements:

- **Create a subdirectory called `proj4`.**

- **You will need to have a `makefile` in this directory. In addition, all the header and cpp files needed to build your software must be present here, as well as the `LOG.txt` file.**
- **You should create the following additional files.**
  - ***stack.h:*** **This should implement a generic `stack` class with at least the following features: (i) `void push(T &)`, (ii) `void pop()`, (iii) `T &top()`, and (iv) `bool empty()`.**
  - ***queue.h:*** **This should implement a generic `queue` class with at least the following features: (i) `void push(T &)`, (ii) `void pop()`, (iii) `T &front()`, and (iv) `bool empty()`.**
  - ***Other files:*** **You may use more files.**
  - ***main.cpp:*** **This is the main program. The command: `./nim <c/h> <m> <N1> < N2> ... <Nk>` will cause a *NIM* game to be played with `k` heaps, having initial number of sticks = `<N1>`, `<N2>`, ..., `<Nk>` respectively. The argument `c` makes the computer play first, while the argument `h` lets the human play first. The argument `m` specifies the number of moves in the future that should be evaluated using the game tree. Nodes beyond that level should be evaluated using any criterion that you wish to use. For nodes within the level specified by `m`, you must use the mini-max procedure described in class. However, you may use pruning strategies and symmetries to reduce the number of nodes evaluated, if you wish to.**

Sample executable: A sample executable is available at `~cop4530/fall05/bin/nim` on `linprog`. The first person to find errors in our program will get a bonus point!

Bonus points:

You may get up to 10 additional points for providing a good GUI interface. You may also get 10 additional points if your implementation plays as well as ours!

Notes:

1. **You should not use the STL `list`, `vector`, `deque`, `stack`, or `queue` classes. You may use the `string` class. Please get my permission before using any other STL feature.**
2. **We will test your `stack` and `queue` classes on entirely different applications. So it is important for these classes to be generic and exactly as specified.**
3. **You will be graded on your user interface too.**

Last modified: 12 Oct 2005