Homework 3 — assigned 20 October — revised 27 October — revised 2 November — due Sunday 7 November

Additional questions will be added later.

2.1 Games

Pick any one of the following position games:

1. Connect Four (a 7×6 board with gravity)
2. Qubic (a $4 \times 4 \times 4$ version of tic-tac-toe)

Revisit problem 2.5 from the previous assignment. Is it possible to play the game perfectly? This means the following: if possible, guarantee a win; but if it is not possible to guarantee a win, then at least guarantee a draw. Consider this question separately from the vantage point of the first player (red) and the second player (green).

If your answer is yes, your task is to construct a strategy that implements this perfect play. (There is no need to play well in the usual sense. For instance, if we are only going to guarantee a draw, it is completely irrelevant if we ever win. However, you may decide to also do the task as for "no".) You need to explain how you came up with the strategy. You need to justify that the strategy indeed plays perfectly. This may be done with a proof, or by exhaustive testing of all possible plays by the opponent.

If your answer is no, your task is to construct a strategy that plays the game reasonably well in some heuristic sense that you will define and then experimentally validate.

If your answer is "I don't know", proceed as for "no".

You may choose any Haskell representation for game boards:

```
type Board = ...
```

The names of the functions to be written are `perfectStrategyForRed`, `perfectStrategyForGreen`, `heuristicStrategyForRed`, and `heuristicStrategyForGreen`, all of type `Board -> Board`.

Hint. Before tackling one of the games listed above, revisit the game of tic-tac-toe.

2.2 Logic

We can use the following type `Expr` to represent Boolean formulas in conjunctive normal form succinctly:

```
type Expr  =  [[Int]]
```

In this representation, for instance, `[[-1, 2, 4], [-2, -3]]` stands for the more conventional $(\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3)$.

Write a function `eval :: (Int -> Bool) -> Expr -> Bool` to compute the Boolean value of a formula under a given assignment of Boolean values to the variables that appear in the formula; here the first argument is a function that describes the assignment.

Write a function `satisfiable :: Expr -> Bool`, which determines if the given formula is satisfiable, i.e., true for some assignment of Boolean values to the variables that appear in the formula.

Write a function `readCNFFile :: String -> IO Expr` to read and parse a file with the given name; the file is assumed to be in DIMACS CNF format (see, e.g., `http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html`). Then write a main program that will accept the CNF file name as its first command-line argument, parse it, solve the satisfiability problem for it, and finally print, to a file named as the second command-line argument, the outcome, which is either a single line `UNSAT` or a line `SAT` followed by a line showing the assignment in clause format, i.e., all the variables of the expression, as positive or negative numbers depending on the assignment, terminated with a 0.

How to turn in

Submission instructions: use the `turnin` facility provided by the department; see course wiki for details.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.