CMPUT 350 Lab 9 Prep Problems

Tic-Tac-Toe

- Played on a 3x3 board
- Two players move alternately by marking an empty square with their symbol (x and o, respectively)
- The player who first creates 3 of his symbols in a row (horizontally, vertically, or diagonally) wins
- The game ends when there are no moves left
- If at that time nobody has created a 3-in-a-row configuration, the game is counted as a draw

Sample Game:

```
---  x   --x   o    --x  x   -xx  o   -xx  x   xxx
--- ---> ---  ---> -o- ---> -o- ---> -oo ---> -oo  x wins
---      ---       ---      ---      ---      ---
```

MiniMax Search Applied to Tic-Tac-Toe

For a review of *MiniMax Search*, see the AI part 4 notes **part4.pdf**.

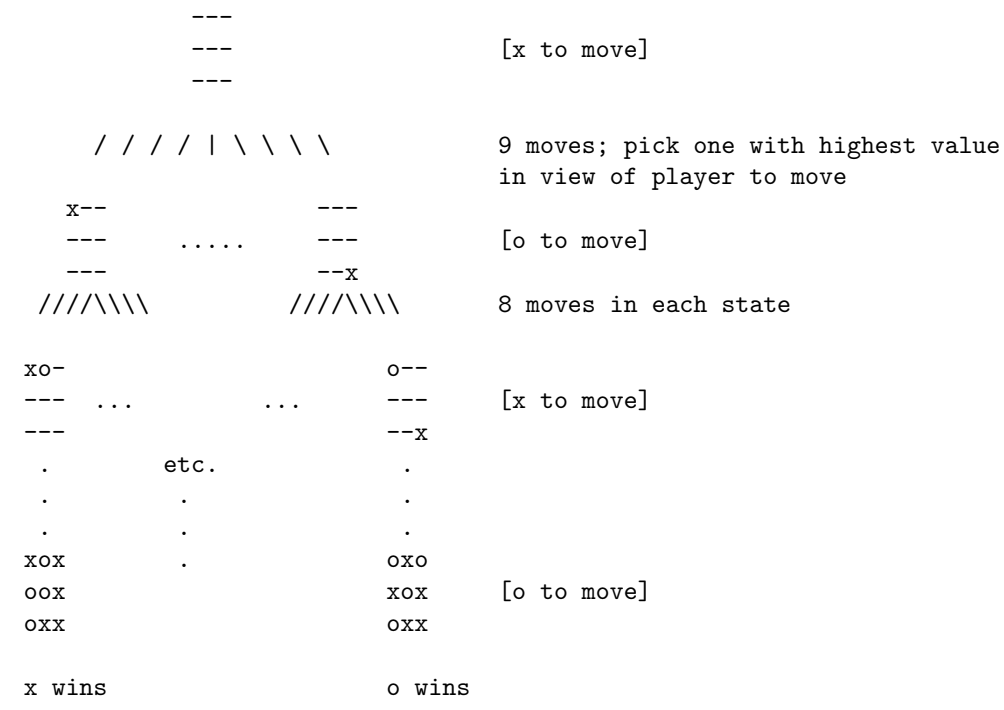
Game Tree

- Vertices represent game states
- The root vertex represents the state at the beginning of the game
- Edges represent moves
- Leaf vertices mark the end of a particular move sequence. The game ends there (terminal game states)
- Each leaf has an associated payoff value (e.g., +1 for player MAX)

MiniMax Search

- MiniMax search backs up values starting with leaves
- Assumes we have two players: MAX and MIN (x and o in this case)
- Uses the minimax rule (maximizing scores in MAX nodes, minimizing in MIN nodes) if the values are stored in view of player MAX
- **Alternative**: NegaMax search always evaluates states in view of the player to move. This simplifies the code, because we only ever need to maximize scores.
- **Important**: assuming alternating moves the returned value has to be negated

Tic-Tac-Toe Game Tree 1



Tic-Tac-Toe Game Tree 2

What is the value of this game state in view of player x?

```
xoo
--x  [x to move]
-o-
```

[draw game tree on a piece of paper]

Game Tree Search Mechanics

MiniMax or NegaMax (preferred) search can be implemented by depth-first-search, copying states and applying moves to the copied states as we go down the game tree until we reach a terminal state, which gets evaluated and its value passed back to the caller for maximization of the negated value (NegaMax) or minimization/maximization of the returned value depending on whose turn it is (MiniMax)

Please refer to lecture notes **part4.pdf** (especially page 27) for details and code snippets you may want to use.

Tuples

In this exercise you'll use `std::tuple`, which is a generalization of `std::pair`. Tuples contain a sequence of data items that don't necessarily share the same type. Individual items can be accessed by `std::get<i>(tuple)`.

Example:

```
std::tuple<int,double,char> t(0, 3.5, 'x');
std::cout << get<0>(t) << " " << get<1>(t) << " " << get<2>(t) << std::endl;
// prints 0 3.5 x
```

1. In this problem you will implement missing functions of a tic-tac-toe program that read a game state from a string, print it to stdout, and determine whether a board is completely filled.

In file `ttt1.cpp`, which is given, implement the parts marked with

```
// ... implement
```

You may want to use this script file to compile your program: `make1`