# Computer Science 300 – Spring 2014
# Assignment #3

**80 points** **Due: Monday, 4/07/14 at 11:59PM**

The code for this assignment is available in the file `A3.zip` on D2L.

1. (30 points) In this problem, you will implement the alpha-beta pruning algorithm. You will implement it in the file `AlphaBeta.java` by modifying the methods `minimax` and `maximin` ONLY. To test your implementation, a fully implemented game, namely Tic-Tac-Toe, has been provided. The relevant files for this game are:

   - `Player.java` : A player is either the MAX player (which always goes first in the game) or the MIN player. A player can search to a given depth (number of plies) and evaluates nodes at that depth using a static evaluation function. Finally, a player remembers the numbers of games it has played, won, and lost. This class is abstract and must be extended by a concrete class for a given game.

   - `Game.java` : A game has two players, namely the MAX and the MIN players, as well as a winner. This class is abstract and must be extended by a concrete class for a given game.

   - `Node.java` : A node in the MINIMAX search tree has a depth (with the root node at depth 0) and a move that led to it (for the root node, the move is empty). This class is abstract and must be extended by a concrete class for a given game. Each concrete game class defines its own data structure for the board.

   - `EvalFunction.java` : A static evaluation function that takes a game node and returns a real number assessing the value of this node for whatever player is using the function. This class is abstract and must be extended by a concrete class for a given player.

   - `AlphaBeta.java` : Implements the MINIMAX search algorithm as a set of two mutually-recursive methods, and needs to be completed to include alpha-beta pruning. Instances of this class wrap around a game instance.

   - `TicTacToePlayer.java` : A concrete class derived from `Player`.

   - `TicTacToeGame.java` : A concrete class derived from `Game`.

   - `TicTacToeNode.java` : A concrete class derived from `Node`, with the obvious 3x3 2D array of characters as a representation for the board.

   - `NilssonEvalFunction.java` : A concrete class derived from `EvalFunction` that implements an evaluation function for TicTacToe proposed by Nils Nilsson.

- `RandomEvalFunction.java` : A concrete class derived from `EvalFunction` that implements the evaluation function that returns a random value between 0.0 and 1.0. Of course, this evaluation function may be used as a baseline function in any game.

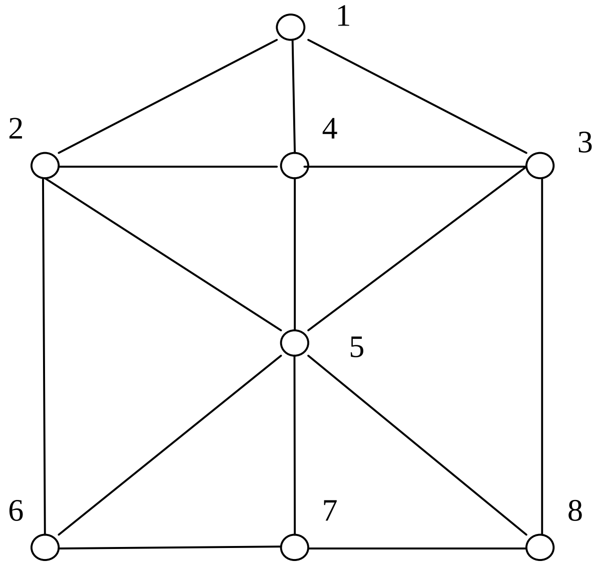
- `TicTacToeDriver.java` : The main program to be used to test your alpha-beta implementation.

To run your program, run the main method in the `TicTacToeDriver` class. Giving "`?`" as the command line argument will run one game using predefined players. The number of nodes visited by MINIMAX search on this turn will be also printed. After you correctly implement alpha-beta pruning this number should go down significantly.

When you are done with implementing alpha-beta pruning, spend time experimenting with the given code. In particular, investigate what happens when you modify the driver code, say, to change the max depth of each player, to switch MAX and MIN, to have a Nilsson player go against another Nilsson player, with or without the same depth. . . . To help you complete this experimentation, a tournament method in the driver program has been setup, which you can invoke by giving "`t`" as the command line argument.

2. (50 points) Consider the simple **Pennies and Quarter** game (PQ) described below.

   Alice has three pennies, which are initially placed in locations 1, 2, and 3 on the board shown below. Bob chooses the initial location of the quarter, from among locations 5, 6, 7, 8. The pennies always move first, and Alice can move one penny per turn. Pennies and the quarter move from one node to an adjacent empty node when the nodes are joined by an edge. The pennies can only move sideways or forwards, while the quarter can move in any direction. Alice wins if the pennies trap the quarter, and Bob wins if the quarter gets to location 1, or if the same position is repeated three times.



Implement this game using the API described for question 1. So, you will need to provide the classes `PQGame.java`, `PQNode.java`, `PQPlayer.java` and `PQDriver.java`. You may also provide additional files, as needed. Then test your program using the `AlphaBeta.java` class you defined earlier, and answer this question.

A. What do you notice, if anything, from running the minimax/minimax with alpha-beta pruning algorithms on this problem?

**Submission**

**You may do this assignment with a partner, if you choose**. If so, it is expected that you both understands all parts of the code you submit, and you will both receive identical grades.

Submit the following files, zipped up in a single folder called `Assignment3.zip` to the `Assignment3` drop box on D2L by the deadline specified:

1) A `Readme` file with the following information in it:
   a) Name of partner, if any:
   b) Statement that you understood all the code that was submitted, even if done by your partner: (Yes/No)
   c) Question 1:
      - Successfully completed: Yes/No/Yes but...
   d) Question 2:
      - Successfully completed: Yes/No/Yes but...
      - Your answer to question A at end of question 2
2) `AlphaBeta.java` for Question 1.
3) All relevant java files for Question 2 (at least 4 must be there).