Some Minimax Notes  
(version 2, 12/6/2016, edit to typo in minimax(), see red [1])

Minimax is an algorithm that evaluates board positions and propagates a value up a DFS search tree, alternating between taking the min or the max of children nodes at each step. In tic-tac-toe, we can associate X with the MAX nodes and O with the MIN nodes. Then a winning board could have a score of +INF and a losing board a score of –INF, and a tie would be 0. (INF is a large number)

If your DFS search tree is sufficiently small, you can always search to the terminal nodes (someone always wins or ties) and return one of the above values (+INF, -INF, 0) as a board score. This is the case in tic-tac-toe. If, instead, you have to stop before the game is over, you need some evaluation function to assign a value to a board. An evaluation function mentioned in class for Tic-Tac-Toe is eval(board) = 3X2+X1 – 3O2 – O1, but you can make your own.

It is important to note that if you DO use an evaluation function, you still return +/- INF for terminal states. INF must be larger than any internal node’s eval function. Otherwise your AI will not necessarily pursue winning positions.

minimax(board, player, max\_d):

if player == X: return max\_dfs(board, player, max\_d, current\_d**)[1]**

if player == O: return min\_dfs(board, player, max\_d, current\_d)**[1]**

max\_dfs(board, player, max\_d, current\_d):

if terminal(board):

return terminal\_value(board), None

if current\_d == max\_d:

return eval(board), None

v = -INF

move = -1

for m in open\_squares(board):

new\_value = min\_dfs(assign(board, m, player), toggle(player),

max\_d, current\_d+1)[0]

if new\_value > v:

v = new\_value

move = m

return v, move

The primary difference in the above code and the version given in class on Day 1 is simply the inclusion of many details needed to make the algorithm work: keeping up with current and changed boards, remembering which move is the best, updating the player and depth, etc.