Crossword Solving Algorithm

*Thomas Laverghetta (*[*tlave002@odu.edu*](mailto:tlave002@odu.edu)*)*,   
Old Dominion University – Computational Modeling and Simulation Engineering

Please read over Assignment 4 System Requirements document (which accompanies this report) first to gain understanding of program overview.

# Algorithms

## Minimax Algorithm

*NEXT\_MOVE <-pointer to best move*

*Minimax(node, depth, isMaxPlayer, alpha, beta):*

*if depth == max depth OR node is leaf:*

*return value of node*

*if isMaxPlayer == MAX:*

*bestVal = -INF*

*foreach child node:*

*value = minimax(child, depth + 1, MIN, alpha, beta)*

*if depth == 0:*

*preVal = bestVal*

*bestVal = max(bestVal, value)*

*if preVal < bestVal:*

*NEXT\_MOVE = child*

*Else:*

*bestVal = max(bestVal, value)*

*alpha = max (alpha, bestVal)*

*if beta <= alpha:*

*break*

*END if*

*END for*

*return bestVal*

*else: <-isMaxPlayer == MIN*

*bestVal = +INF*

*foreach child node:*

*value = minimax(child, depth + 1, MAX, alpha, beta)*

*if depth == 0:*

*preVal = bestVal*

*bestVal = min(bestVal, value)*

*if preVal > bestVal:*

*NEXT\_MOVE = child*

*Else:*

*bestVal = min(bestVal, value)*

*beta = min (beta, bestVal)*

*if beta <= alpha:*

*break*

*END if*

*END for*

*return bestVal*

*END if*

*END*

## Selection Algorithm

*Selection(curr state (CS), last stone pos (LSP), player (W or B))*

*selection node set (nodeSet) <-set of possible moves*

*// check for empty spaces around LSP*

*if CS[LSP.row, LSP.col - 1] == 0:*

*append copy of CS with CS[LSP.row, LSP.col - 1] = player*

*do same for the rest for:*

*r + 1, c*

*r - 1, c*

*r , c + 1*

*r , c - 1*

*r + 1, c + 1*

*r - 1, c + 1*

*r + 1, c - 1*

*r - 1, c – 1*

*if nodeSet size == 0:*

*return nodeSet*

# Results