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

The AI algorithm used for playing against with will be a minimax algorithm. A minimax algorithm is a decision rule used in AI system to minimize possible loss for a worst-case scenario. To use this algorithm the following algorithms were also created: selection algorithm (finds all possible moves from current system state) and node score (determines a quantifiable metric of board state for a given player).

## Selection Algorithm

The selection algorithm determines the set of next possible moves. The move set is used in minimax to determine next nodes. To determine the next possible move, the algorithm will first determine if there is more than remaining more left. If false (only move remaining), then the algorithm will save both possibilities: either the player makes the move or skips. If more than one move remaining, search for adjacent empty spaces to the last move made, and if no move is available the player has “freedom” to choose any empty space (appends any empty spaces as possible solutions). The pseudocode for the selection algorithm is below:

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

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

*// check if only one space remaining.*

*if CS only has remaining space empty:*

*// if player makes move on remaining space*

*append copy of CS with empty space = player*

*// if player does not make move on remaining space*

*append copy of CS*

*return nodeSet*

*// 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:*

*append copy any empty space in CS where the empty space is respaced with player*

*return nodeSet*

## Node Scoring Algorithm

The selection algorithm determines the set of next possible moves. The move set is used in minimax to determine next nodes. To determine the next possible move, the algorithm will first determine if there is more than remaining more left. If false (only move remaining), then the algorithm will save both possibilities: either the player makes the move or skips. If more than one move remaining, search for adjacent empty spaces to the last move made, and if no move is available the player has “freedom” to choose any empty space (appends any empty spaces as possible solutions). The pseudocode for the selection algorithm is below:

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

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

*// check if only one space remaining.*

*if CS only has remaining space empty:*

*// if player makes move on remaining space*

*append copy of CS with empty space = player*

*// if player does not make move on remaining space*

*append copy of CS*

*return nodeSet*

*// 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:*

*append copy any empty space in CS where the empty space is respaced with player*

*return nodeSet*

## 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 (node score method)*

*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*

# Results