**CS 4346- Project #2**

* **The Problem Description and team member contributions:**

We are testing multiple evaluation functions for our minimax algorithm. By using a Tic Tac Toe game for an example, we can visualize the results and track the stats of each evaluation function. Each member of the team created their own evaluation function, and one member created two. We all worked together on creating the tic-tac-toe game including the Minimax function.

* **The Domain:**

The score obtained by our heuristic function “evaluate”. This score is minimized and maximized to determine the best position for our game piece on the board. The domain of this score is [-1000, 1000]. The problem domain is maximizing the path to obtain the goal state of a winning piece (3 in a row).

* **Methodologies:**

Minimax and the changing heuristic function “evaluate”. Our methodologies involved different ways to evaluate a “good spot”. One team member indicated that an evaluation of a “good spot” can be broken down and compared. They were then able to identify a filled location on the board with 2 in-a-line empty spots nearby. This would indicate a “good spot” for a game piece, but not a “better spot” than 2 pieces touching each other with one spot nearby. We decided to break that heuristic down among our 4 evaluation functions to determine which heuristic is the most efficient.

* **Analysis of the program:**

Instead of only checking for 3 game pieces of the same type in one row, we modified the evaluation functions to determine two game pieces in a row, with an empty spot nearby. In addition to that, we modified it further to check for two empty spots nearby a neighboring game piece. These modifications should allow for a shorter path to the goal state of 3 game pieces in a row.

w is the winner, f(x) is the evaluation function number:

F(x)| w | length / Nodes | time(s) |

(1) | X | 199 / 161 | 0.158 |

(2) | O | 443 / 212 | 0.111 |

(3) | O | 189 / 413 | 0.135 |

(4) | X | 258 / 509 | 0.131 |

* **Conclusion:**

In conclusion, the game showed the shortest path when checking for 2 empty spots neighboring a game piece as well as 3 game pieces in a row. The game path was consistently the shortest over multiple rounds. This evaluation function is listed as “Evaluation 2” in the source code. The longest execution time was our first evaluation function, that was only evaluating if it could find three in a row by placing a piece in the next location. This generates a shorter path but takes more time to evaluate. The longest path we found was the second function. This function addition includes a check for two neighbors that have one empty spot as well as 3 in a row. I think because of the many different ways two neighbors are defined the program takes a longer game path, even though there are less nodes. The highest amount of nodes generated was the fourth evaluation function. This function combines the additions of the first three and we believed would lead to the shortest game path and time. It wasn’t far off for time, but it does have the shortest path/nodes generated. This would indicate it is finding a shorter path in less amount of time than the other heuristic functions as expected, but due to the subsequent “unnecessary” checks it causes the program to slow down in execution time.

Program Run

 