Programming an Adaptive Artificial Intelligence using the Monte Carlo Tree Search Algorithm for Turn-Based Games

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The most intuitively exhaustive and conventional form of artificial intelligence (AI) for board games is minimax. Minimax involves constructing the complete game tree of possible moves from the current state of a game and choosing the branch that leads to most positive terminal results. Although this may be a very thorough form of AI, for even moderately complex games the game tree is much too immense to be able to iterate through the complete game tree efficiently.

Thus, the solution to this is the Monte Carlo Tree Search Algorithm (MCTS), a heuristic algorithm, which uses only knowledge of the possible moves and the Monte Carlo method to selectively sample branches of the complete game tree with many playouts and reach conclusions based on these samples. The game which MCTS was applied to for this project was Connect Four, because its complete game tree is much too massive for minimax to be a viable option. The goal of this project was to program an efficient artificial intelligence, implementing the Monte Carlo Tree Search Algorithm, to make decisions on a turn-by-turn basis for the game Connect Four (given only the knowledge of the possible moves).

First, using an object-oriented approach in Java, a game framework for Connect Four was created with an established domain of moves. Then, MCTS was implemented in four parts: *selection* of nodes, *expansion* of children, *simulation* of game playouts, and *backpropagation* to update node stats. Repeating these four steps for a predetermined number of times gave rise to knowledge that allowed the AI to make a decision.

Data was collected in order to determine the effectiveness and efficiency of the MCTS implementation, including many trials of AI vs AI playouts, timing the speed of the AI, comparing the different MCTS play-styles, and comparing MCTS to the e-greedy and minimax selection policies.

Ultimately, it was found that the MCTS implementation was indeed efficient and showed a high level of success in both efficiency and effectiveness at different levels of playouts for the game Connect Four. Future implications include giving the AI further domain knowledge to account for sampling error, continuing to increase its efficiency, and fully optimizing the playstyle and number of playouts. Using this knowledge of MCTS and the frameworks of this implementation, it can be applied to a vast array of games as well as more practical applications such as amino and nucleic acid polymerizations in bioinformatics.