Jacob Young
Sai Karthik Padmanaban

Project 3 Report

For this assignment, we were tasked with implementing a machine that could play against someone in the game "4 in a line." The game is composed of an 8 x 8 grid and players take turns placing their piece on any position on that grid. If at any time a player has 4 of their pieces in a row (either by row or by column), the player will win the game.

To implement the game well, our project needed to be composed of 3 main parts: determining which positions are best to be evaluated, evaluating said positions' scores, and using an adversarial search with alpha-beta pruning to determine the next possible move. All of this must take place under a user-set time limit as well. We set the time counter to be checked during the pruning method since we can determine what our best state is at any given time and since most of the computing time is spent in the pruning method. When determining our positions to be evaluated, we implemented an assortment of checks to test if certain positions were viable. For each position, we could check if either player would win or lose in resulting turns if a piece was played there. If any of these checks turned out to be true, the position would be added to the evaluation set, otherwise, it would be ignored. If there were no pieces played yet, or if there is only one, we had set methods to place the computer's piece adjacent to where the first piece was played.

Once we had our possible positions, or successors, we used our evaluation function to determine the score of each successor. Our evaluation function was simple: we checked how many consecutive pieces there were in each direction. For instance, if we are evaluating an open space for the player (O piece), we would check rows in front and behind, as well as columns above and below. If any combination of O's were found in those checks, points were awarded to that successor. We scored 1 point if 1 piece was found adjacent to the space, 5 points for 2 consecutive

pieces, and 50 for 3. We also had one last check for if we somehow found 4 in a row and granted that 1000 points to guarantee the selection to secure the win. After tallying the points in each position, they are then sent to our adversarial search method.

Our adversarial search was constructed from the lectures in class. Once all the positions have a score, the implementation for alpha beta pruning was straightforward. It recursively calls itself until it reaches the max depth of the search, then determines the min-max for each node. If at any point our alpha is greater than or equal to our beta, we break the search from the current node and advance to the next one. To repeat, this is also where we implemented the time constraint, so while searching for the best node, we always have the "best" recorded. If at any time we hit our limit, we simply terminate and return the current best score.

The win checks are always done after a move is played, so whenever either the player or the computer places a piece, we check all consecutive positions. If 4 in a line are found, we end the program and output whoever the winner is.