CSC Problem Set 04 +X Report

Name: John Cheek
Email: jwcheek2@ncsu.edu
Section: 001
Collaborators: N/A

Summary:

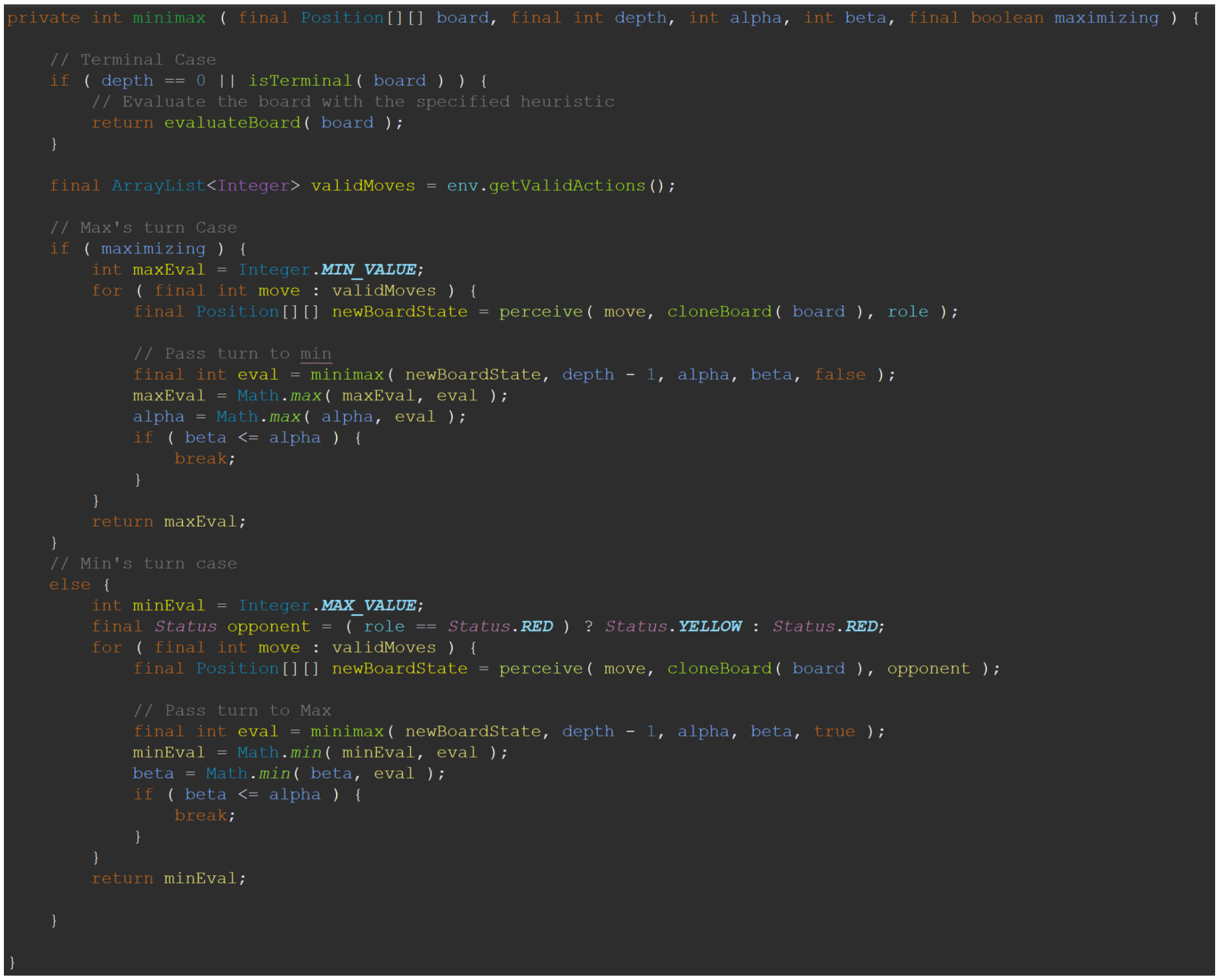
For my +X percent I revised my initial min-max search to utilize A/B pruning. This resulted in a great increase in performance, at the cost of being slightly less effective at winning matches against the random robot (by 1-3 percent in some cases).

"+X" Concept:

When I first completed my initial solution it was very effective but ran very slowly, taking about a minute and a half to complete all the test cases. Since A/B pruning was a concept I had difficulty with in class I wanted to keep my initial approach rather simple and not build A/B pruning into my first solution. While my solution was very effective, after experiencing the long runtimes when testing, I felt like going back and adding A/B pruning could significantly decrease the runtime. I felt like it would be a good choice for my +X and I could see just how much it could improve the efficiency and reduce the runtime of my agent.

Technical Implementation of +X:

The costly nature of my initial min-max approach makes sense as the heuristic I created utilizes many for loops, iterating through all possible moves (each running in O(n) time) then, when evaluating the connections it, scans the entire board (running in O(n^2) time upon each iteration and essentially each call of minimax. No changes were made to the heuristic logic, but with A/B pruning the algorithm now trims off a good number of bad moves, reducing the timely calls to the minimax method. The revised minimax method is as follows:



```java
private int minimax ( final Position[][] board, final int depth, int alpha, int beta, final boolean maximizing ) {

    // Terminal Case
    if ( depth == 0 || isTerminal( board ) ) {
        // Evaluate the board with the specified heuristic
        return evaluateBoard( board );
    }

    final ArrayList<Integer> validMoves = env.getValidActions();

    // Max's turn Case
    if ( maximizing ) {
        int maxEval = Integer.MIN_VALUE;
        for ( final int move : validMoves ) {
            final Position[][] newBoardState = perceive( move, cloneBoard( board ), role );

            // Pass turn to min
            final int eval = minimax( newBoardState, depth - 1, alpha, beta, false );
            maxEval = Math.max( maxEval, eval );
            alpha = Math.max( alpha, eval );
            if ( beta <= alpha ) {
                break;
            }
        }
        return maxEval;
    }
    // Min's turn case
    else {
        int minEval = Integer.MAX_VALUE;
        final Status opponent = ( role == Status.RED ) ? Status.YELLOW : Status.RED;
        for ( final int move : validMoves ) {
            final Position[][] newBoardState = perceive( move, cloneBoard( board ), opponent );

            // Pass turn to Max
            final int eval = minimax( newBoardState, depth - 1, alpha, beta, true );
            minEval = Math.min( minEval, eval );
            beta = Math.min( beta, eval );
            if ( beta <= alpha ) {
                break;
            }
        }
        return minEval;

    }

}
```

When minimax is first called in getAction, Alpha is initialized to Integer.MIN_VALUE and beta to Integer.MAX_Value. This A/B pruning approach is based off of Swarthmore's pseudocode provided in class. [1] It involves calculating and updating alpha and beta upon each of max or min turns respectively, then comparing the calculated result to the alpha and beta values, then passing the updated alpha and beta to the next iteration of minimax. if the beta value is ever less than or equal to the current alpha value it will break the loop, skipping through the rest of the valid moves.

Evaluation and Results:

A/B Pruning vastly improved the runtime of my algorithm however, it did slightly decrease its accuracy for tests against the RandomAgent. Below is a table comparing my initial minimax solution to the one using A/B pruning.

	Minimax	**Minimax with A/B pruning**
Runtime:	1m 44s	0m 22s
Test Results (% of trials won by the agent)		
Test 01:	100	97
Test 02:	100	98
Test 03:	100	100
Test 04:	100	100
Test 05:	100	100
Test 06:	100	100
Test 07:	100	100
Test 08:	100	100

As shown here, A/B pruning saves about one minute and twenty seconds on the runtime. The A/B minimax agent passed 100% of the trials against the vertical, horizontal, and greedy agents; however, where the normal Minimax agent won 100% of the time against the random agent, the A/B pruning agent lost a few points of accuracy against this robot. This makes sense simply due to the random nature of the agent. My heuristic evaluates every potential move and depends on the opponent to move somewhat predictably and create chains of tokens. When acting randomly, the opponent will never never do this, which causes some difficulty for my heuristic. As a result, A/B pruning may cut off some of the potential moves that would be beneficial to my agent ultimately winning.

Works Referenced:

1. *Swarthmore College. Alpha-Beta Minimax Pseudocode. Accessed: 03/16/2025. [Online]. Available: https://www.cs.swarthmore.edu/~meeden/cs63/f05/minimax.html*