Task 7: Game Playing

Group 10 01/01/2014

1 Brief Description of Algorithms

1.1 Minimax Algorithm

Minimax is a decision rule used in artificial intelligence, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario.

We can see that this algorithm makes a tree of positions as it explores all possible moves by the opponent. When it reaches the required depth, it assesses the position using a heuristic function. It evaluates the best move such that it minimizes the best move of the opponent.

1.2 Alpha Beta Pruning

Alpha-Beta Pruning seeks to minimize the number of positions explored in the search tree by the Minimax algorithm. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. Basically, it prunes the search tree such that the outcome of the algorithm remains unchanged.

2 Heuristic functions considered

2.1 Coin Parity

This heuristic returns the lead of the player with respect to their opponent.

Listing 1: Pseudocode for Coin Parity heuristic

```
def coinParity(position):
    if position.turn == BLACK
        return board.getBlackCount() - board.getRedCount()
    else:
        return board.getRedCount() - board.getBlackCount()
```

2.2 Corners Captured

We see that the number of corners occupied improves the chances of winning drastically. This heuristic returns the difference in the corners occupied

Listing 2: Pseudocode for Corners Captured heuristic

```
def cornersCaptured(position):
    myCorners, oppCorners = 0 ,0
    for corner in position.corners():
        if corner == position.turn:
            myCorners++
        if corner == position.turn:
            oppCorners++
    return ( myCorners - oppCorners )
```

2.3 Mobility

If we have more choices, it is likely to be the case that our best choice is better. This heuristic tries to improve our freedom to make a variety of moves with respect to our opponent.

Listing 3: Pseudocode for Mobility heuristic

```
def mobility(position):
    myMoves = position.getValidMoves(position.turn).size()
    oppMoves = position.getValidMoves(position.turn.opponent).size()
    return myMoves - oppMoves
```

3 Trees to show my particular moves are chosen for at least 6 moves given the board configuration

3.1 Minimax Algorithm

- Tree 1: [Coin Parity heuristic] -> trees/minimax/tree_1.txt
- Tree 2: [Corners Captured heuristic] -> trees/minimax/tree_2.txt
- Tree 3: [Mobility heuristic] -> trees/minimax/tree_3.txt

3.2 Alpha Beta Pruning

- Tree 1: [Coin Parity heuristic] -> trees/alphaBeta/tree_1.txt
- Tree 2: [Corners Captured heuristic] -> trees/alphaBeta/tree_2.txt
- Tree 3: [Mobility heuristic] -> trees/alphaBeta/tree_3.txt

4 Comparison of Minimax and Alpha-Beta Pruning

We may expect the two algorithms to perform equally well given that Alpha-Beta Pruning is just an optimized version of the Mini-Max algorithm.

4.1 Space and Time Complexity

The Alpha-Beta pruning algorithm has lesser space and time complexity as moves that are guaranteed to be worse than previously examined moves, are not further explored. By eliminating worse states that need not be explored, the space complexity is reduced. Since, lesser states are explored, in comparison to Mini-max, the time complexity is also lesser.

4.2 Winning Criteria

The two second constraint to play the next move gives the Alpha Beta bot the advantage of exploring greater depths compared to the Mini-max Bot. When unbounded by time constraints, both the bots are expected to play equally well. The two bots are compared using the same heuristic as comparing their performances with different heuristics would be more reflective of the nature of the heuristic rather than that of the algorithm. Simulations reveal that the two bots play nearly equally well and that there is a general trend of the winning bot being the one that starts the game first.