TASK 5: Game playing

1 Brief Description of Algorithms

1.1 Minimax Algorithm

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.
In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
Similarly,in our case of algorithm we can observe that it is making a tree of positions as it explores all possible combinations of moves possible to be explored by the opponent player. so, when it reaches the expected depth,it calculates the position using the heuristic function. It evaluates the best move such that it is able to minimize the best move of the opponent.

1.2 Alpha Beta Pruning

Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithms. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

- Alpha is the best value that the maximizer currently can guarantee at that level or above.
- Beta is the best value that the minimizer currently can guarantee at that level or above.

Alpha-Beta Pruning seeks to minimize the number of positions explored in the search tree by the Minimax algorithm. As soon as one possibility is found that proves the next move to be worse than previously examined move it stops evaluating a move. Basically, it prunes the search tree such that the outcome of the algorithm remains unchanged.

2 Heuristic functions considered

2.1 Coin Parity

Heuristic function in our case returns the value with which the player is leading with respect to its opponent and finds the difference between them.

```
heuristic(turn):
    if .turn == BLACK
        return BlackCount() - RedCount()
    else:
        return RedCount() - BlackCount()
```

2.2 choice

More choices means that our best choice is better. So in this way the heuristic tries to optimize the freedom to make more moves with respect to our opponent.

```
mobility(turn):
    myMoves = ValidMoves(turn).size()
    oppMoves = ValidMoves(turn.opponent).size()
    return myMoves - oppMoves
```

2.3 corners chosen

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

```
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 )
```

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

Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithms. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available.

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 there is no time constraints then we can expect that both play equally well. So simulations reveal that almost both bots perform equally and there is a trend of bot which plays first more than one which plays later one. 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. Although minimax has more possibility if we don’t consider first moving case.