Application of MiniMax algorithm in Two-Player Games

41202, 41203, 41205: Aditi Wikhe, Jagruti Agrawal, Anuraag Shankar

Virtual games have thrown off the world by a storm. Within this sector, multiplayer games have gained special recognition. Humans have been playing such games for decades now and are trying to figure ways to master these games. On the other hand, researchers have been working on different algorithms to achieve superhuman levels in these games. In 2016, DeepMind achieved one of the biggest breakthroughs by beating World Champion Lee Sedol in the game of Go four games to one. The project shows how the MiniMax algorithm can be applied to two-player games namely Tic-Tac-Toe and FrostFire.

1 INTRODUCTION

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. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

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.

Every board state has a value associated with it. In a given state if the maximizer has upper hand, then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state, then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

One important factor in solving games using the MiniMax algorithm is the size of the state space involved. Following are the state space sizes for a few commonly known two-player games:

1. Tic-Tac-Toe: 3^9
2. Backgammon: 10^{20}
3. Abalone: 10^{25}
4. Chess: 10^{46}
5. Go: 10^{170}

This project takes two games as case studies:

1. Tic-Tac-Toe
2. FrostFire (simplified version of Abalone)

While Tic-Tac-Toe can be considered as a small game in terms of state space size, FrostFire is a large game. Thus, while the game of Tic-Tac-Toe can be solved completely, FrostFire cannot. This is not possible even after applying optimizations such as Alpha-Beta Pruning.

2 PROJECT SCOPE

Humans have been studying games such as Chess and Go for decades now. In South Korea, the game of Go is something that is introduced into the school curriculum at a very early stage. On February 10, 1996, Chess Grandmaster Garry Kasparov lost to the IBM Deep Blue Artificial Intelligence robot in the first of six games. The IBM computer was capable of evaluating 200 million moves per second. It was later in 2008 that StockFish was released. It was the world’s best chess engine and had an ELO score much above any other human player. It was not until 2017 that DeepMind came up with AlphaZero. It was a bot which had no prior knowledge of the game of chess. It was trained for nine hours through self-play only. It was then matched against StockFish for a hundred matches. To everyone’s surprise, AlphaGo beat StockFish in 28 of the games. The rest of the 72 games were all drawn. StockFish could not outclass AlphaGo in a single game. DeepMind came up with a similar algorithm named AlphaGo, which beat the reigning world champion Lee Sedol 4 games to 1. The game of Go is considerably tougher than the game of chess, due to its massive state space. Even the world’s top players believe that one of the most important factors while making the correct decision in Go is the factor of intuition. Yet, it was AlphaGo that prevailed.

As it can be seen that artificial intelligence algorithms can master games that humans have been practicing for decades now. Not only mastery, but the important factor is also that the algorithms master such games in a matter of days and even hours in some cases. While they may seem only restricted to the virtual world, the applications of such algorithms extend to the real world as well in fields such as economics.

3 Requirements

The project has been built using C++ and Python. The STL library of C++ was required to optimize the game and allow the bot to go to larger depths.

The configurations are as follows:

1. Processor Name: Intel i5-7300HQ
2. Processor Speed: 2.50 GHz
3. Operating System: Windows 10 Home 64-bit
4. RAM: 16.0 GB

4 ALGORITHM

MiniMax is a tree search algorithm. This means that it creates a state space tree and chooses the moves with the highest value in the state space. This state space is small for Tic-Tac-Toe and allows the algorithm to search the entire space. This is not possible for FrostFire and thus it is necessary to mention beforehand how deep the algorithm must run. Depth refers to how many moves in the future should the algorithm see to makes its prediction. The algorithm is as follows:

```
function  minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1,
FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1,
TRUE))
        return value

// initial call
minimax(origin, depth, TRUE)
```

Algorithm 1. MiniMax Algorithm

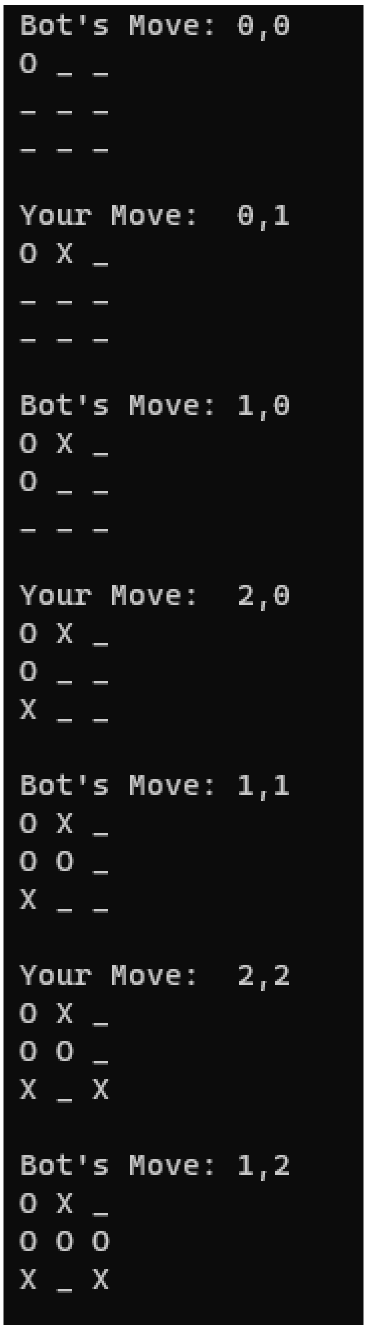
Another optimization that is applied to MiniMax is AlphaBeta pruning, that does not explore branches of the state space that will give less than optimum results. It can be applied as follows:

```
function minimax(node, depth, isMaximizingPlayer, alpha,
beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha,
beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha,
beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

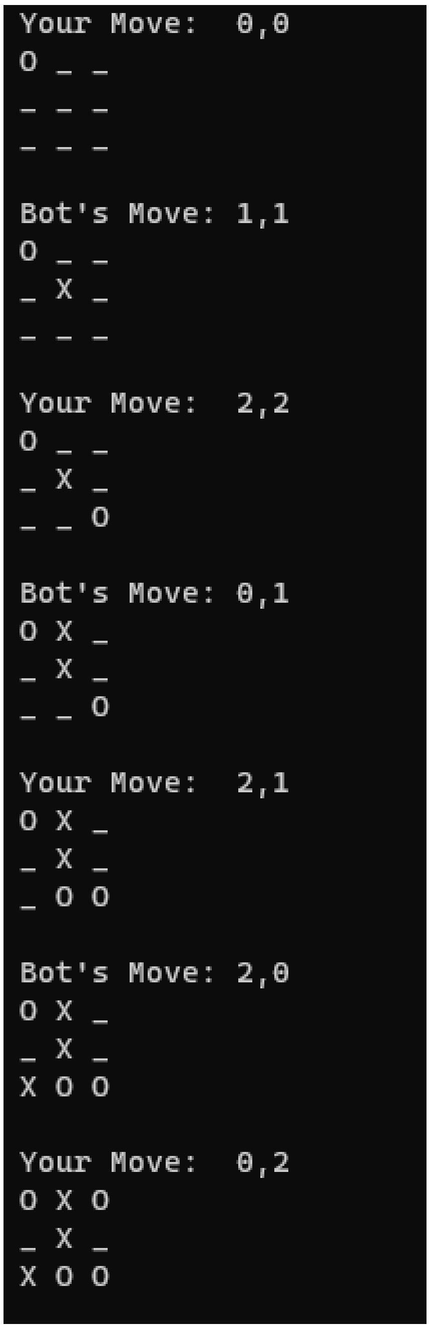
Algorithm 2. Apha-Beta Pruning

5 RESULTS

It can be seen that the Tic-Tac-Toe AI is able to perfectly play the game. This ensures that it can never lose the game.



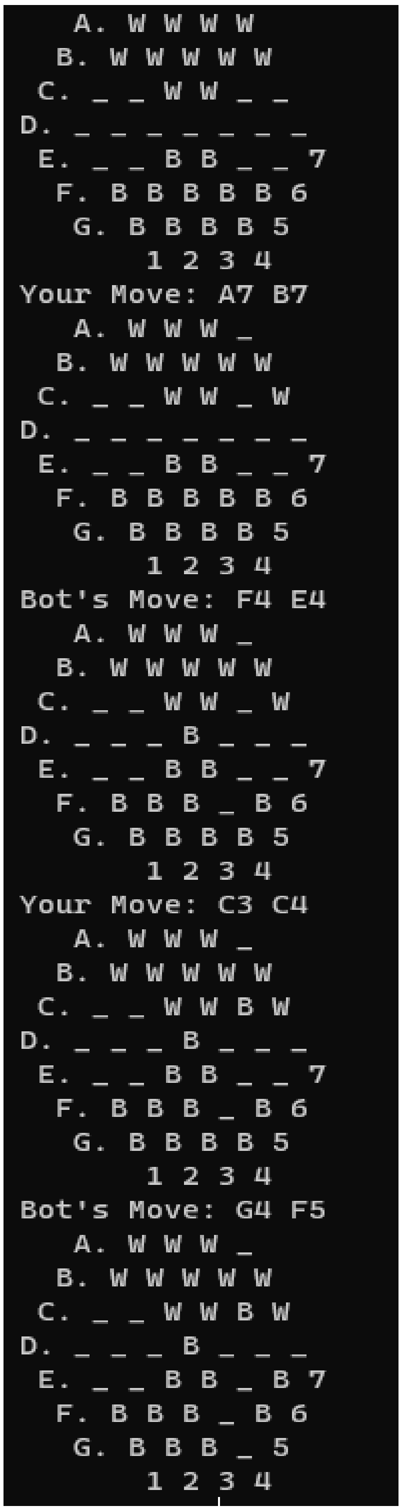
```
Bot's Move: 0,0
O _ _
_ _ _
_ _ _

Your Move:  0,1
O X _
_ _ _
_ _ _

Bot's Move: 1,0
O X _
O _ _
_ _ _

Your Move:  2,0
O X _
O _ _
X _ _

Bot's Move: 1,1
O X _
O O _
X _ _

Your Move:  2,2
O X _
O O _
X _ X

Bot's Move: 1,2
O X _
O O O
X _ X
```



```
Your Move:  0,0
O _ _
_ _ _
_ _ _

Bot's Move: 1,1
O _ _
_ X _
_ _ _

Your Move:  2,2
O _ _
_ X _
_ _ O

Bot's Move: 0,1
O X _
_ X _
_ _ O

Your Move:  2,1
O X _
_ X _
_ O O

Bot's Move: 2,0
O X _
_ X _
X O O

Your Move:  0,2
O X O
_ X _
X O O
```

(a) (b)

Fig. 1. Bot playing Tic-Tac-Toe as (a) O (b) X



```
   A. W W W W
  B. W W W W W
 C. _ _ W W _ _
D. _ _ _ _ _ _ _
 E. _ _ B B _ _ 7
  F. B B B B B 6
   G. B B B B 5
       1 2 3 4
Your Move: A7 B7
   A. W W W _
  B. W W W W W
 C. _ _ W W _ W
D. _ _ _ _ _ _ _
 E. _ _ B B _ _ 7
  F. B B B B B 6
   G. B B B B 5
       1 2 3 4
Bot's Move: F4 E4
   A. W W W _
  B. W W W W W
 C. _ _ W W _ W
D. _ _ _ B _ _ _
 E. _ _ B B _ _ 7
  F. B B B _ B 6
   G. B B B B 5
       1 2 3 4
Your Move: C3 C4
   A. W W W _
  B. W W W W W
 C. _ _ W W B W
D. _ _ _ B _ _ _
 E. _ _ B B _ _ 7
  F. B B B _ B 6
   G. B B B B 5
       1 2 3 4
Bot's Move: G4 F5
   A. W W W _
  B. W W W W W
 C. _ _ W W B W
D. _ _ _ B _ _ _
 E. _ _ B B _ B 7
  F. B B B _ B 6
   G. B B B _ 5
       1 2 3 4
```

Fig. 2. Bot playing FrostFire against user.

6 Conclusion

This project shows two bots playing two-player games using the MiniMax algorithm. The games shown were Tic-Tac-Toe and FrostFire. It could be seen that the bot performed better when pruning was applied. This was because it could explore deeper into the state space of the game. The first bot could search through the entire game and was thus undefeatable. If applied in combination with Machine Learning algorithms, these can achieve superhuman levels in two-player games.