



CMPSC 442: Artificial Intelligence (Fall 2021)

Project-1
Tic Tac Toe using Minimax Algorithm & Alpha Beta Pruning Algorithm

Hansi Seitaj

Dr. Vinayak Elangovan

Submitted On:
10/09/2022

TABLE OF CONTENTS

1	INTRODUCTION	3
2	BACKGROUND (optional)	3
3	DESIGN & IMPLEMENTATION	8
4	RESULTS / SAMPLE OUTPUTS	11
5	CONCLUSION	11
6	REFERENCES	11

1. INTRODUCTION

In this project, I have created an AI that is capable of playing the game of Tic Tac Toe. I have used the Minimax algorithm with Alpha-Beta pruning to do so. First, I created a class that represents our Tic Tac Toe board. This class has few methods that will be used to check the game state, make moves, check for legal moves, allow the user to play with AI. Furthermore, the game state can be one of three things: 'X', 'O', or 'tie'. If 'X' is displayed, that means that player 'X' has won the game. If 'O' is displayed, that means that player 'O' has won the game. If 'tie' is displayed, that means that the game has ended in a draw.

2. BACKGROUND

The Minimax algorithm is a simple algorithm used for optimal decision-making in game theory and artificial intelligence. The algorithm relies on systematic searching, and a simple evaluation function. Let's assume that every time during deciding the next move we search through a whole tree, all the way down to leaves. Effectively we would investigate all the possible outcomes and every time we would be able to determine the best possible move. In addition, since these algorithms rely on being efficient, the Minimax algorithm's performance can be improved by using alpha-beta pruning.

Along the same lines, Alpha-beta algorithm is an improved minimax using a heuristic function. It stops evaluating a move when it makes sure that it's worse than previously examined move. Such moves need not to be evaluated further. In other words, compared to minimax algorithm, it gives the same output, but cuts off the branches that can't affect the final decision; therefore, improving the time complexity and performance.

3. DESIGN & IMPLEMENTATION

Part 1

- Max algorithm

- 1 I assign a value higher than the worst case (-2) and we need -1 for loss, 0 for tie, 1 for win.
- 2 Ask for the result from the is_end() function to check what state we are currently.
- 3 The 'O' player makes a move and calls Min() to allow the other player to fill an empty spot and continue with the game

- Min algorithm

- 1 I assign a value higher than the worst case (+2) and we need -1 for win, 0 for tie, 1 for loss.
- 2 Ask for the result from the is_end() function to check what state we are currently.
- 3 The 'X' player makes a move and calls Max() to follow up with the other player to fill an empty spot

```

# Player 'O' is max, in this case AI
def Max(self):

    # Possible values for maxv are:
    # -1 - loss
    # 0 - a tie
    # 1 - win

    # Initially setting it to -2 as worse than the worst case:
    maxValue = -2

    px = None
    py = None

    result = self.is_end()

    # If the game came to an end, the function needs to return
    # the evaluation function of the end.
    # That can be:
    # -1 - loss
    # 0 - a tie
    # 1 - win
    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                # On the empty field player 'O' makes a move and calls Min
                # That's one branch of the game tree.
                self.current_state[i][j] = 'O'
                (m, min_i, min_j) = self.Min()
                # Fixing the maxv value if needed
                if m > maxValue:
                    maxValue = m
                    px = i
                    py = j
                # Setting back the field to empty
                self.current_state[i][j] = '.'
    return (maxValue, px, py)

```

```

# Player 'X' is min, in this case human/user
def Min(self):

    # Possible values for minv are:
    # -1 - win
    # 0 - a tie
    # 1 - loss

    # Initially setting it to 2 as worse than the worst case:
    minValue = 2

    qx = None
    gy = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'X'
                (m, max_i, max_j) = self.Max()
                if m < minValue:
                    minValue = m
                    qx = i
                    gy = j
                self.current_state[i][j] = '.'

    return (minValue, qx, gy)

```

Part 2

- MaxAlphaBeta algorithm

- 1 I assign a value higher than the worst case (-2) and we need -1 for loss, 0 for tie, 1 for win.
- 2 Ask for the result from the is_end() function to check what state we are currently.
- 3 The 'O' player makes a move and calls MinAlphaBeta() to allow the other player to fill an empty spot and continue with the game
- 4 The difference between the Max() algorithm is that we are pruning the branch we find to be higher than the max value compared to an alpha and beta value.

```
def MaxAlphaBeta(self, alpha, beta):

    maxValue = -2
    px = None
    py = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'O'
                (m, min_i, min_j) = self.MinAlphaBeta(alpha, beta)
                if m > maxValue:
                    maxValue = m
                    px = i
                    py = j
                self.current_state[i][j] = '.'

            # Next two ifs in Max and Min are the only difference between regular algorithm and minimax
            if maxValue >= beta:
                return (maxValue, px, py)

            if maxValue > alpha:
                alpha = maxValue

    return (maxValue, px, py)
```

- MinAlphaBeta algorithm

- 1 I assign a value higher than the worst case (+2) and we need -1 for win, 0 for tie, 1 for loss.
- 2 Ask for the result from the is_end() function to check what state we are currently.
- 3 The 'X' player makes a move and calls MaxAlphaBeta () to follow up with the other player to fill an empty spot.
- 4 The difference between the Min() algorithm is that we are pruning the branch we find to be lower than the min value compared to an alpha and beta variable.

```

def MinAlphaBeta(self, alpha, beta):

    minValue = 2

    qx = None
    qy = None

    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'X'
                (m, max_i, max_j) = self.MaxAlphaBeta(alpha, beta)
                if m < minValue:
                    minValue = m
                    qx = i
                    qy = j
                self.current_state[i][j] = '.'

            if minValue <= alpha:
                return (minValue, qx, qy)

            if minValue < beta:
                beta = minValue

    return (minValue, qx, qy)

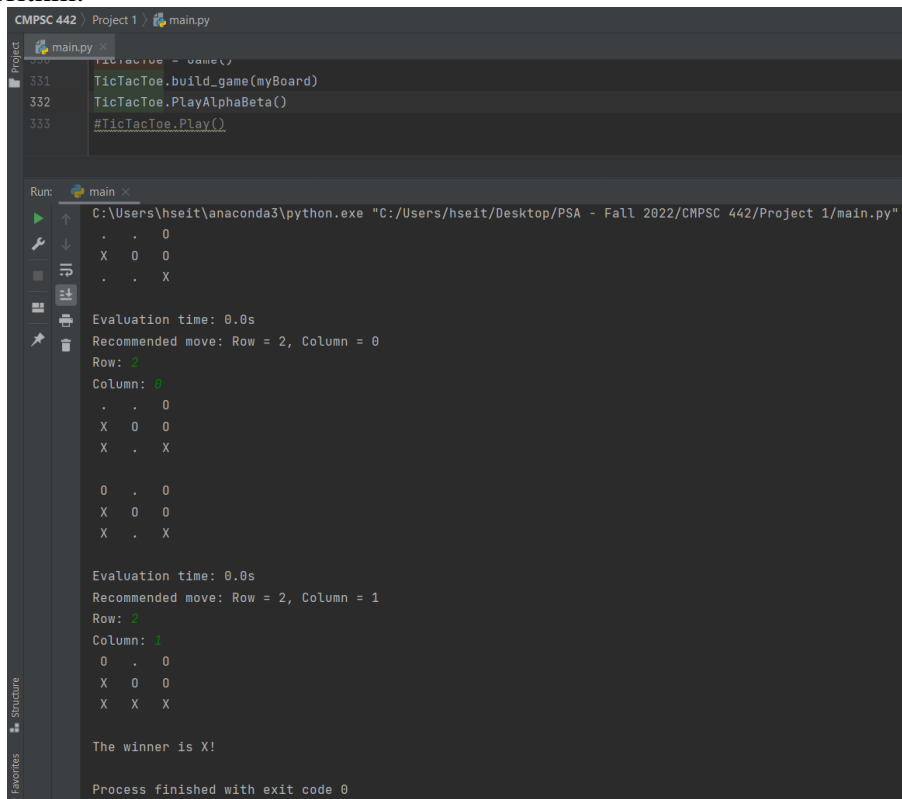
```

Answer the following:

1. What will be your optimal move according to Minimax and also with Alpha beta pruning?
2. Does both the algorithm suggest the same move?
3. How many nodes does it visit?
4. Perform a hand calculation for the above set and compare the result with the program result. Is it the same?

The optimal move according to Minimax and Alpha beta pruning algorithms are row 2 and column 0. Therefore, both algorithms suggest the same move. In a given board, the algorithms reformulate the strategy and based on the resultValue they do the calculations and the nodes needed to be visited. In case of 4 empty spot the algorithm will iterate through the board and do the calculations accordingly to maximize or minimize the chances for winning. Finally, after a hand calculation on the board given, the results are pretty much the same. The 'X' plyer has to select the move on row 2 col 0 since picking any other move 'O' wins. Therefore, the best recommended move by hand match the recommendation given by both algorithms.

AlphaBeta algorithm:



The screenshot shows a code editor with the following Python code in `main.py`:

```
TicTacToe = Game()
TicTacToe.build_game(myBoard)
TicTacToe.PlayAlphaBeta()
#TicTacToe.Play()
```

The Run window displays the execution output:

```
C:\Users\hseit\anaconda3\python.exe "C:/Users/hseit/Desktop/PSA - Fall 2022/CMPSC 442/Project 1/main.py"
. . 0
X 0 0
. . X

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 0
Row: 2
Column: 0
. . 0
X 0 0
X . X

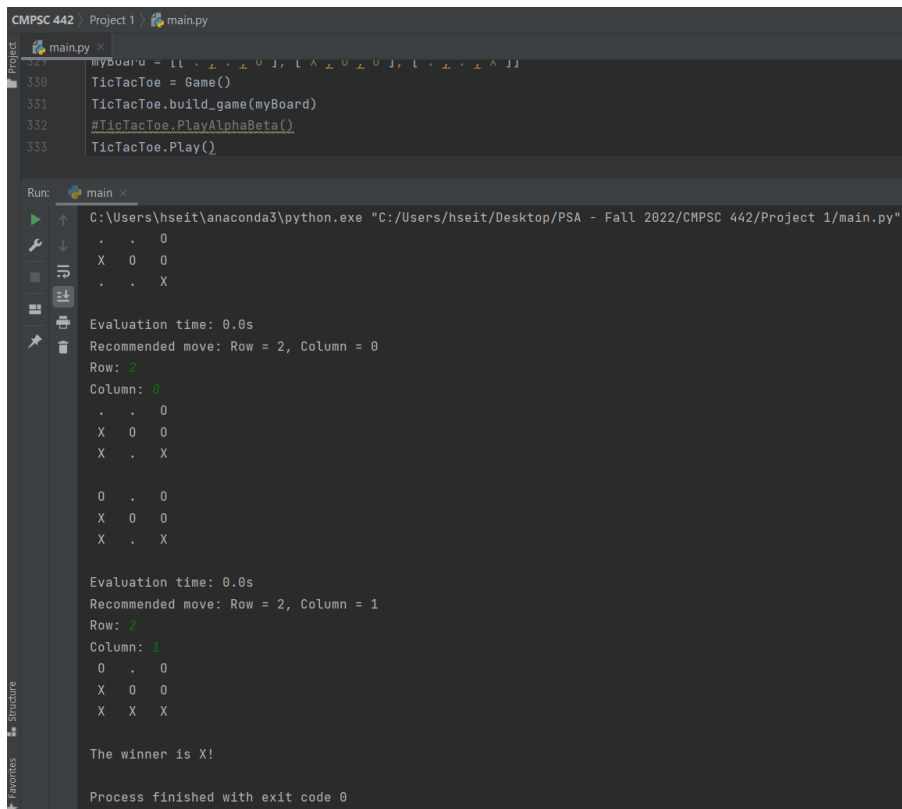
0 . 0
X 0 0
X . X

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 1
Row: 2
Column: 1
0 . 0
X 0 0
X X X

The winner is X!

Process finished with exit code 0
```

Minimax:



The screenshot shows a code editor with the following Python code in `main.py`:

```
myBoard = [[1, 2, 3, 0], [1, 2, 0, 0], [1, 2, 3, 0]]
TicTacToe = Game()
TicTacToe.build_game(myBoard)
#TicTacToe.PlayAlphaBeta()
TicTacToe.Play()
```

The Run window displays the execution output:

```
C:\Users\hseit\anaconda3\python.exe "C:/Users/hseit/Desktop/PSA - Fall 2022/CMPSC 442/Project 1/main.py"
. . 0
X 0 0
. . X

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 0
Row: 2
Column: 0
. . 0
X 0 0
X . X

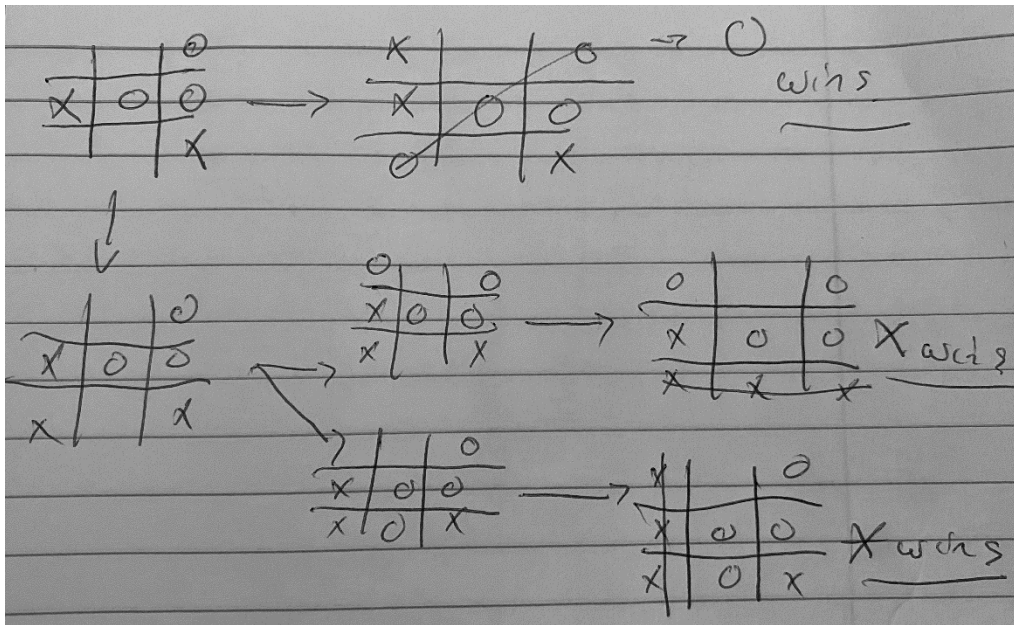
0 . 0
X 0 0
X . X

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 1
Row: 2
Column: 1
0 . 0
X 0 0
X X X

The winner is X!

Process finished with exit code 0
```

Hand calculations:



4. RESULTS / SAMPLE OUTPUTS

Comparison Table

Algorithm	Minimum time	Maximum time
Minimax	3.14s	3.68s
Alpha-beta pruning	0.12s	0.13s

The same steps for both versions:

Minimax:

```
C:\Users\hseit\anaconda3\python.exe "C:/Users/hseit/Desktop/PSA - Fall 2022/CMPSC 442/Project 1/main.py"
Tic Tac Toe!

. . .
. . .
. . .

Evaluation time: 3.6510875s
Recommended move: Row = 0, Column = 0
Row: 0
Column: 0
X . .
. . .
. . .

X . .
. 0 .
. . .

Evaluation time: 0.0312498s
Recommended move: Row = 0, Column = 1
Row: 0
Column: 1
X X .
. 0 .
. . .

X X 0
. 0 .
. . .
```



```
main x
↑ Evaluation time: 0.0s
↓ Recommended move: Row = 2, Column = 0
| Row: 2
| Column: 0
| X X 0
| . 0 .
| X . .
|
| X X 0
| 0 0 .
| X . .
|
| Evaluation time: 0.0s
| Recommended move: Row = 1, Column = 2
| Row: 1
| Column: 2
| X X 0
| 0 0 X
| X . .
|
| X X 0
| 0 0 X
| X 0 .
|
| Evaluation time: 0.0s
| Recommended move: Row = 2, Column = 2
| Row: 2
| Column: 2
| X X 0
| 0 0 X
| X 0 X
|
| It is a tie!
```

AlphaBeta algorithm:

```
main x
↑ C:\Users\hseit\anaconda3\python.exe "C:/Users/hseit/Desktop/PSA - Fall 2022/CMPSC 442/Project 1/main.py
↓ Tic Tac Toe!
| . . .
| . . .
| . . .
|
| Evaluation time: 0.1249971s
| Recommended move: Row = 0, Column = 0
| Row: 0
| Column: 0
| X . .
| . . .
| . . .
|
| X . .
| . 0 .
| . . .
|
| Evaluation time: 0.0s
| Recommended move: Row = 0, Column = 1
| Row: 0
| Column: 1
| X X .
| . 0 .
| . . .
|
| X X 0
| . 0 .
| . . .
```

```
Run: main x
Evaluation time: 0.0s
Recommended move: Row = 2, Column = 0
Row: 2
Column: 0
X X 0
. 0 .
X . .

X X 0
0 0 .
X . .

Evaluation time: 0.0s
Recommended move: Row = 1, Column = 2
Row: 1
Column: 2
X X 0
0 0 X
X . .

X X 0
0 0 X
X 0 .

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 2
Row: 2
Column: 2
X X 0
0 0 X
X 0 X

It's a tie!
```

Both algorithms recommend the same patterns and get to the same result (tie):

Pattern for Minimax (row, column): 0,0 – 0,1 – 2,0 – 1,2 – 2,2

Pattern for AlphaBeta (row, column): 0,0 – 0,1 – 2,0 – 1,2 – 2,2

The nodes visited by the AI in both algorithms in the first moves are the same for applying the same pattern (8 nodes per case). However, the more moves are made, the AlphaBeta prunes the branches further (from 8 to 6 and 4 than 2) and it doesn't need to check them again which it takes less time to evaluate.

- 3 different starting trials:

Row 1, Col 0:

```
Evaluation time: 0.0s
Recommended move: Row = 2, Column = 0
Row: 2
Column: 0
0 X 0
X 0 .
X . X

0 X 0
X 0 .
X 0 X

Evaluation time: 0.0s
Recommended move: Row = 1, Column = 2
Row: 1
Column: 2
0 X 0
X 0 X
X 0 X

It's a tie!

Process finished with exit code 0
```

Row 1, Col 1:

```
Evaluation time: 0.0s
Recommended move: Row = 0, Column = 2
Row: 0
Column: 2
0 X X
X X 0
. 0 .

0 X X
X X 0
0 0 .

Evaluation time: 0.0s
Recommended move: Row = 2, Column = 2
Row: 2
Column: 2
0 X X
X X 0
0 0 X

It's a tie!

Process finished with exit code 0
```

Row 2, Col 2:

```
Evaluation time: 0.0s
Recommended move: Row = 0, Column = 2
Row: 0
Column: 2
X 0 X
. 0 .
0 X X

X 0 X
. 0 0
0 X X

Evaluation time: 0.0s
Recommended move: Row = 1, Column = 0
Row: 1
Column: 0
X 0 X
X 0 0
0 X X

It's a tie!

Process finished with exit code 0
```

With these starting inputs and following the recommended move by the algorithm I get to the same Tie result.

Trials/Start Values	Row 1, Col 0	Row 1, Col 1	Row 2, Col 2
Results	Tie	Tie	Tie

5. CONCLUSION

- What have you learnt from this project?

In this project, I learned how to implement the Minimax and Alpha Beta pruning algorithm in a real intuitive game. I learned how these two algorithms calculate the minimum and maximum values and feed that output to the algorithm to learn and chose the better option. Moreover, I was able to implement an AI that is almost impossible to win against and understand the role of the time complexity between the algorithms. Finally, both algorithms were effective in addressing the problem, but Alpha Beta pruning is significantly faster then Minimax which makes it optimally better for selection into games as Tic-tac-toe, etc. Finally, keeping in mind the user experience and implementing a crucial algorithm made this project for me a new way of approaching the software world.

- What ways you can expand this project?

I believe that the project addresses the goal very neatly and there are no serious additional parts to add for new learning experiences.

6. REFERENCES

<https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/>