Ultimate Tic Tac Toe using Minimax with Alpha-Beta Pruning Algorithm

1 Introduction

Ultimate Tic Tac Toe is a significantly more complex and strategic version of the classic game. Its dynamic board behavior and indirect move targeting present unique challenges for building an intelligent agent. This report explores how Minimax, enhanced with Alpha-Beta Pruning, was adapted and optimized to design a high-performing bot that plays strategically, anticipates future states, and avoids redundant computations.

2 Understanding the Game Complexity

Unlike regular Tic Tac Toe (which has 9 cells), Ultimate Tic Tac Toe has:

- 9 boards, each with 9 cells, totaling 81 cells.
- The board to play in next is determined by the last move of the opponent.
- The game's outcome depends not only on local board victories but also on the macro board.

This dramatically increases the search space and makes brute-force play infeasible in competitive time constraints.

3 The Core: Minimax Algorithm

What is Minimax?

Minimax is a recursive decision-making algorithm used in turn-based games. It simulates every possible move (and counter-move), assuming both players play optimally.

Intuition

- If it's the AI's turn (Maximizer), choose the move that maximizes the score.
- If it's the opponent's turn (Minimizer), assume they will pick the move that minimizes the score for the AI.

4 Enhancing with Alpha-Beta Pruning

Why Alpha-Beta?

While Minimax works, it explores every branch. That's painfully expensive with 81 cells and multiple valid boards at each step.

Alpha-Beta Pruning adds an optimization layer:

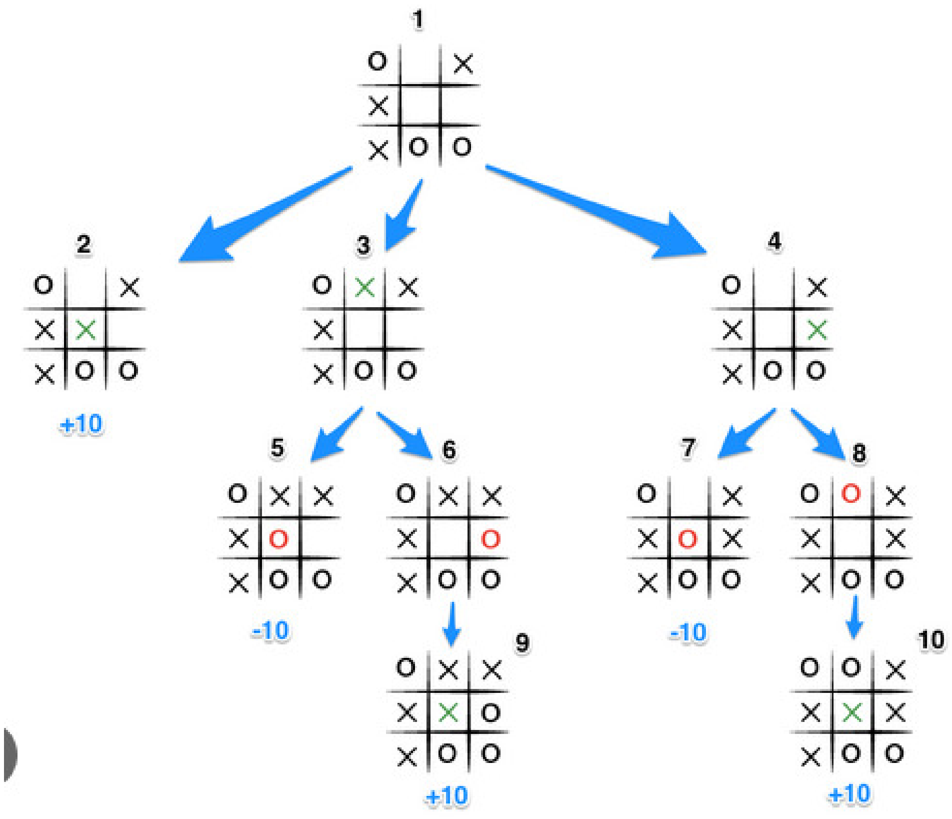


Figure 1:

- It eliminates branches that won't affect the final decision.
- This allows deeper search within the same time.

How It Works

- `alpha`: best score guaranteed for the maximizer so far.
- `beta`: best score guaranteed for the minimizer so far.
- If `beta` $\leq$ `alpha`, the current branch can be pruned.

This reduces the branching factor significantly. For Ultimate Tic Tac Toe, it helped search deeper into strategic turns (depth 3 or more), making the AI anticipate board control and not just immediate wins.

5 Game-Specific Optimization for Ultimate Tic Tac Toe

1. Targeted Valid Moves

Unlike regular Tic Tac Toe:

- A move in a cell dictates the next small board the opponent must play in.
- If that board is already won/full, the opponent can play anywhere.

Optimizations:

- Track only valid sub-boards and generate moves within them.
- Reduce the number of branches to explore at each depth.

2. Board Evaluation Function

`evaluate()` function uses:

- +100 if the AI has won the main board.
- +10 per small board won by the AI.
- -100 / -10 for opponent wins.
- 0 for neutral boards.

This gives the AI multi-scale vision:

- Think globally: win the main board.
- Act locally: secure individual boards tactically.

6 Why Use Alpha-Beta Pruning?

- Without pruning: Minimax explores all branches → exponential time.
- With pruning:
  - Useless branches are skipped.
  - Faster decision-making.

7 Player Setup

```
Player = 'X'
Player_AI = 'O'
```

8 Small Board Class

```
class SmallBoard:
    def __init__(self):
        self.board = [['␣' for _ in range(3)] for _ in range(3)]
        self.winner = None
```

9 Small Board Functionalities

```
def is_full(self): ...
def empty_cells(self): ...
def check_winner(self, player): ...
def make_move(self, row, col, player): ...
def display(self): ...
```

10 Ultimate Board Initialization

```
class UltimateBoard:
    def __init__(self):
        self.board = [[SmallBoard() for _ in range(3)] for _ in range(3)]
        self.big_board = [['␣' for _ in range(3)] for _ in range(3)]
        self.winner = None
```

11 Making Moves on the Ultimate Board

```
def make_move(self, br, bc, sr, sc, player): ...
```

12 Check for Big Board Winner

```
def check_winner(self, player): ...
```

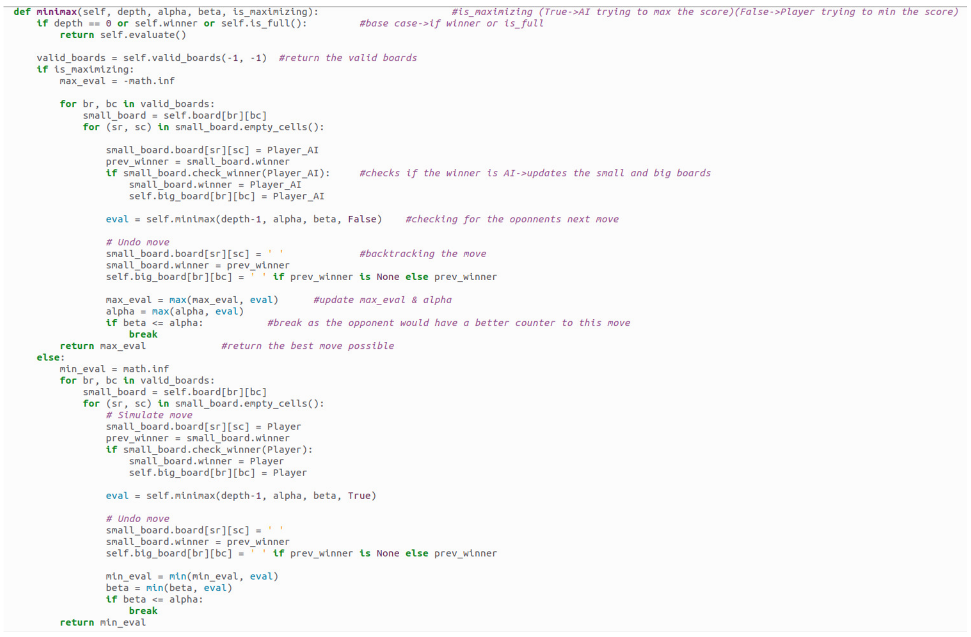
13 Board Status & Valid Moves

```
def is_full(self): ...
def valid_boards(self, last_sr, last_sc): ...
```

14 Evaluation Function

```
def evaluate(self): ...
# +100 for AI win
# -100 for Player win
# +10/-10 for each mini board owned by AI/Player
```

15 Minimax Algorithm with Alpha-Beta Pruning



```
def minimax(self, depth, alpha, beta, is_maximizing):                  #is_maximizing (True->AI trying to max the score)(False->Player trying to min the score)
    if depth == 0 or self.winner or self.is_full():         #base case->if winner or is_full
        return self.evaluate()

    valid_boards = self.valid_boards(-1, -1)  #return the valid boards
    if is_maximizing:
        max_eval = -math.inf

        for br, bc in valid_boards:
            small_board = self.board[br][bc]
            for (sr, sc) in small_board.empty_cells():

                small_board.board[sr][sc] = Player_AI
                prev_winner = small_board.winner
                if small_board.check_winner(Player_AI):     #checks if the winner is AI->updates the small and big boards
                    small_board.winner = Player_AI
                    self.big_board[br][bc] = Player_AI

                eval = self.minimax(depth-1, alpha, beta, False)    #checking for the oponnents next move

                # Undo move
                small_board.board[sr][sc] = ' '             #backtracking the move
                small_board.winner = prev_winner
                self.big_board[br][bc] = ' ' if prev_winner is None else prev_winner

                max_eval = max(max_eval, eval)      #update max_eval & alpha
                alpha = max(alpha, eval)
                if beta <= alpha:          #break as the opponent would have a better counter to this move
                    break
        return max_eval             #return the best move possible
    else:
        min_eval = math.inf
        for br, bc in valid_boards:
            small_board = self.board[br][bc]
            for (sr, sc) in small_board.empty_cells():
                # Simulate move
                small_board.board[sr][sc] = Player
                prev_winner = small_board.winner
                if small_board.check_winner(Player):
                    small_board.winner = Player
                    self.big_board[br][bc] = Player

                eval = self.minimax(depth-1, alpha, beta, True)

                # Undo move
                small_board.board[sr][sc] = ' '
                small_board.winner = prev_winner
                self.big_board[br][bc] = ' ' if prev_winner is None else prev_winner

                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break
        return min_eval
```

```python
def minimax(self, depth, alpha, beta, is_maximizing): ...
```

16 AI Move Decision

```python
def best_move(self, valid_boards): ...
```

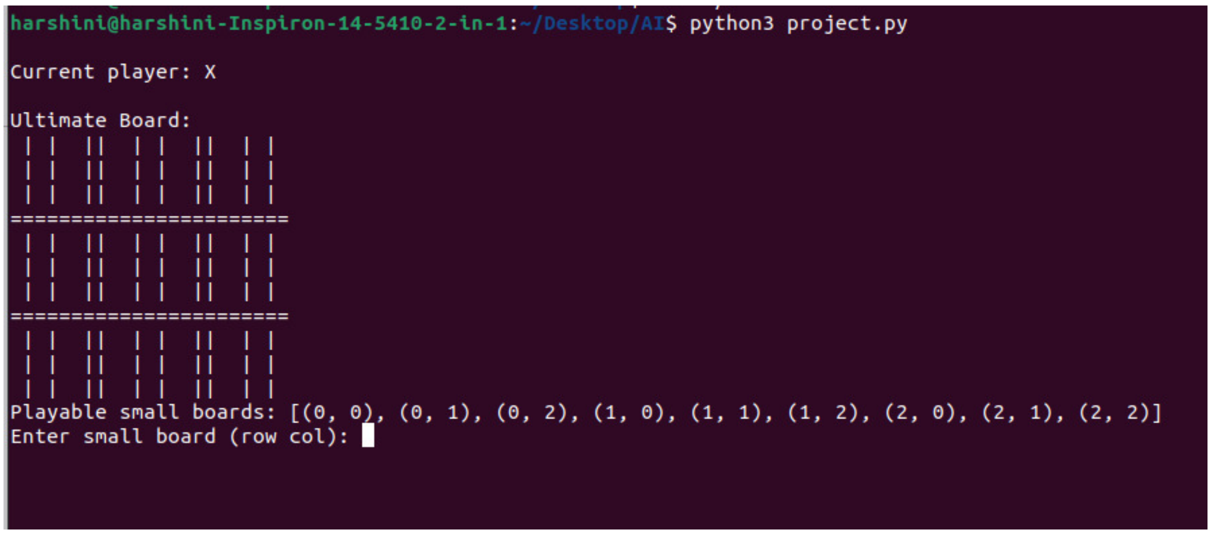
17 Display Ultimate Board

```python
def display(self): ...
```

18 Game Loop

```python
def play_ultimate_game(): ...
# Alternates between human and AI
# Ends when someone wins or board is full
```

19

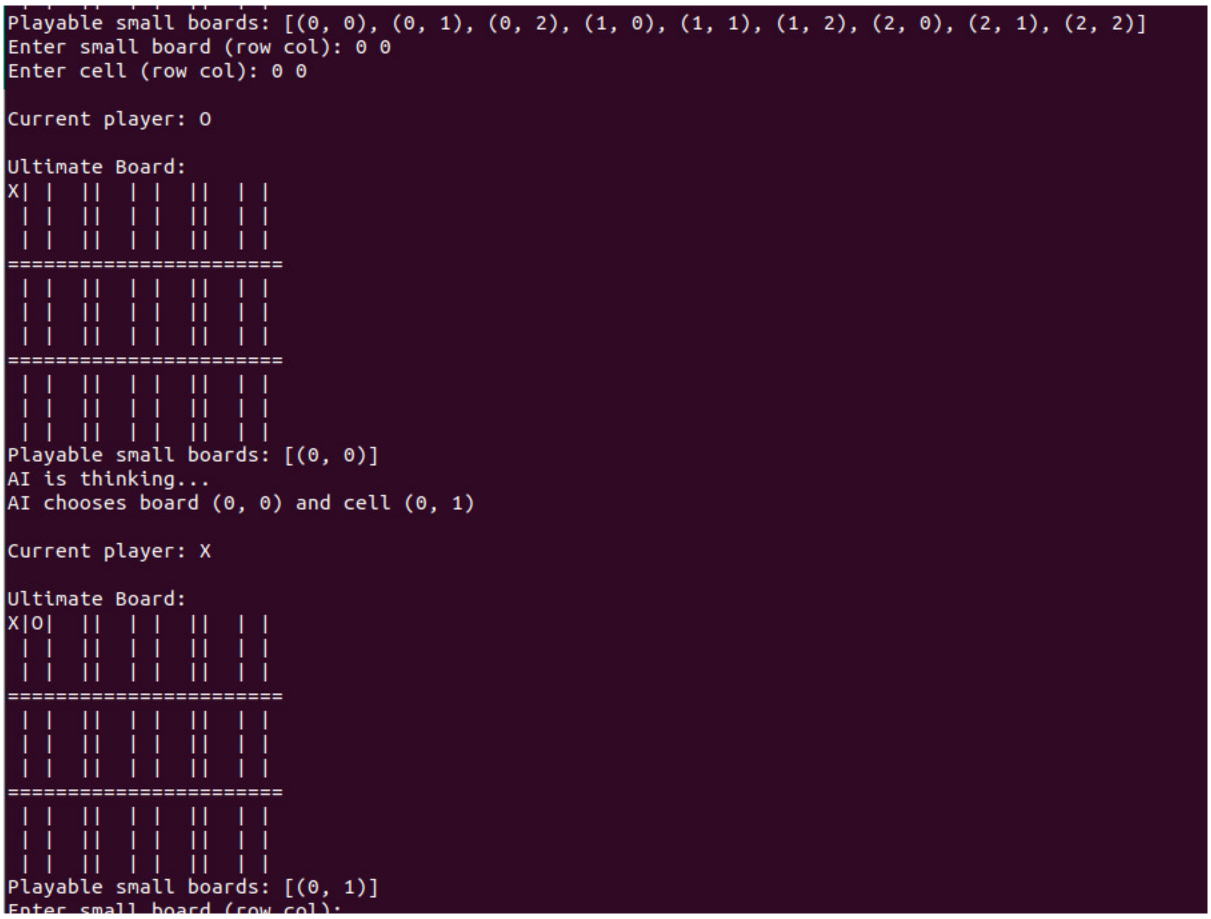


```
harshini@harshini-Inspiron-14-5410-2-in-1:~/Desktop/AI$ python3 project.py

Current player: X

Ultimate Board:
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
Playable small boards: [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
Enter small board (row col): 
```

Figure 2: Ultimate Tic Tac Toe board



```
Playable small boards: [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
Enter small board (row col): 0 0
Enter cell (row col): 0 0

Current player: O

Ultimate Board:
X| |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
Playable small boards: [(0, 0)]
AI is thinking...
AI chooses board (0, 0) and cell (0, 1)

Current player: X

Ultimate Board:
X|O|  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
=======================
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
 | |  ||  | |  ||  | |
Playable small boards: [(0, 1)]
Enter small board (row col):
```

Figure 3: Ultimate Tic Tac Toe board after a few moves