


TRƯỜNG: ĐH CT TP.HCM KHOA: CÔNG NGHỆ THÔNG TIN BỘ MÔN: KHDL&TTNT MH: TH TRÍ TUỆ NHÂN TẠO	BUỔI 3. GIẢI THUẬT TÌM KIẾM ĐỐI KHÁNG	
---	--	---

A. MỤC TIÊU

- Cài đặt thuật toán tìm kiếm đối kháng gồm minimax, cắt tỉa alpha-beta.
- Các ứng dụng.

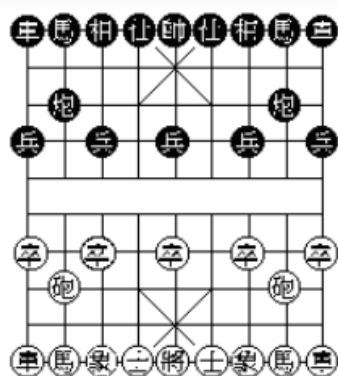
B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

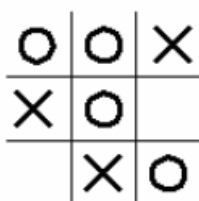
Thuật toán tìm kiếm đối kháng là một quyết định tối ưu trong trò chơi. Nghĩa là tìm một đường đi bảo đảm chiến thắng cho người chơi.



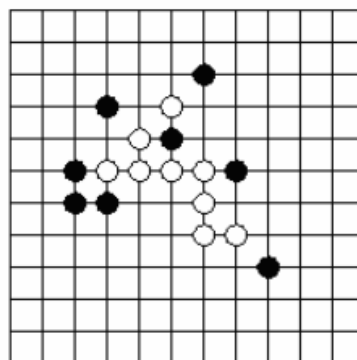
a) Cờ Tướng



b) Cờ Vua (cờ Quốc Tế)



c) Tictactoe



d) Go-moku (cờ caro)

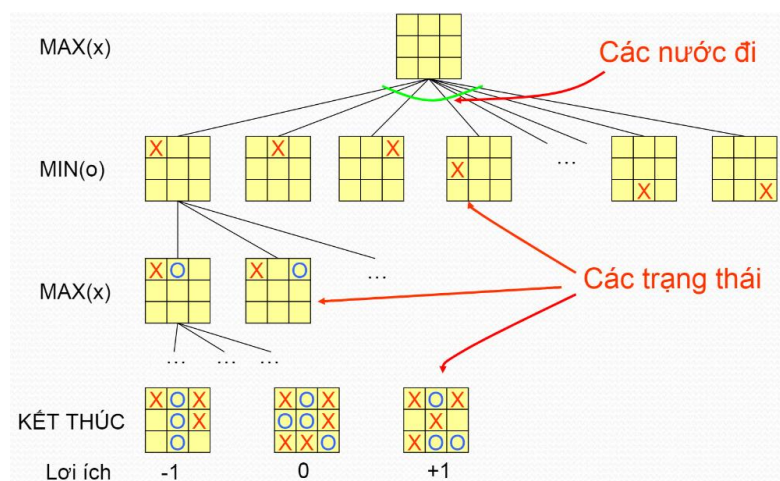
Các thành phần:

- Trạng thái ban đầu (initial state): Trạng thái của trò chơi + Người chơi nào được đi nước đầu tiên.

- Trạng thái kết thúc (terminal state): Kiểm tra kết thúc trò chơi.
- Hàm chuyển trạng thái (Sucessor): Trả về thông tin (nước đi, trạng thái). Nghĩa là, các nước đi hợp lệ từ trạng thái hiện tại, trạng thái mới được chọn.
- Hàm lợi ích (utility function): đánh giá trạng thái và kết thúc.

Hàm lượng giá heuristic $E(n) = X(n) - O(n)$ với $X(n)$ số khả năng thắng của quân X và $O(n)$ số khả năng thắng của quân O.

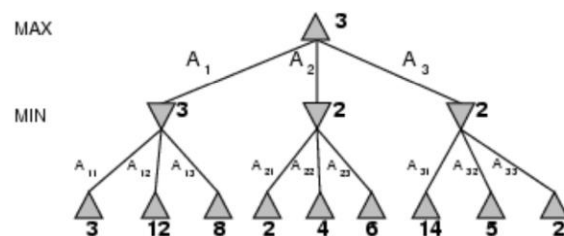
a. Thuật giải Minimax ứng dụng vào bài toán TicTacToe



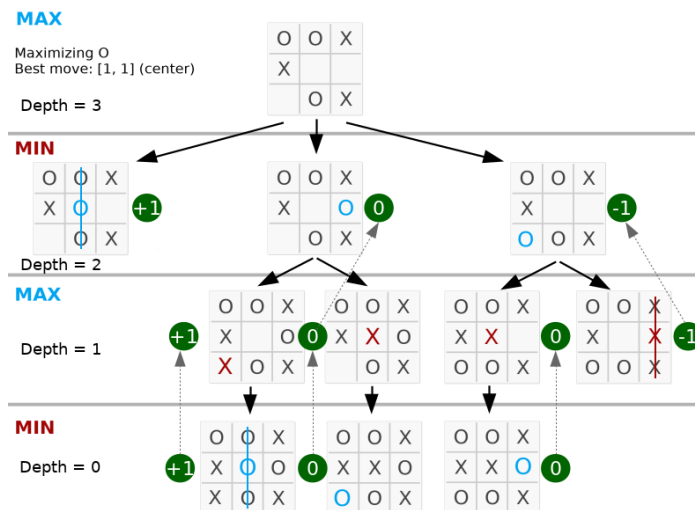
Trong đó,

MAX tối đa hóa hàm lợi ích.

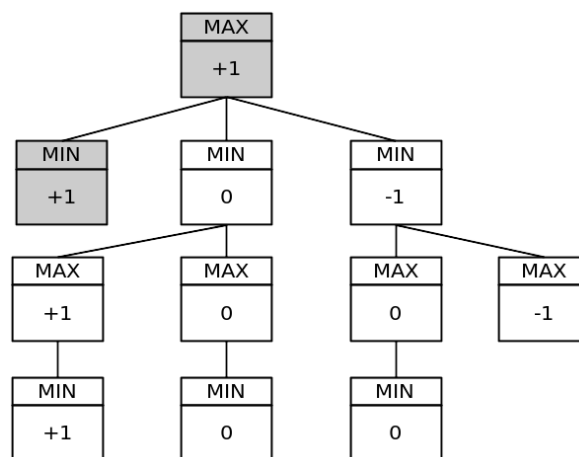
MIN tối thiểu hóa hàm lợi ích.



Ví dụ: Khảo sát cây dưới đây để hiểu về phương pháp minimax.



Giải thích:



Thuật giải Minimax.

function MINIMAX-DECISION(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\text{state})$

return the *action* in **SUCCESSORS**(*state*) with value *v*

function MAX-VALUE(*state*) *returns a utility value*

if **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)

$v \leftarrow -\infty$

for *a, s* in **SUCCESSORS**(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

return *v*

function MIN-VALUE(*state*) *returns a utility value*

if **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)

$v \leftarrow \infty$

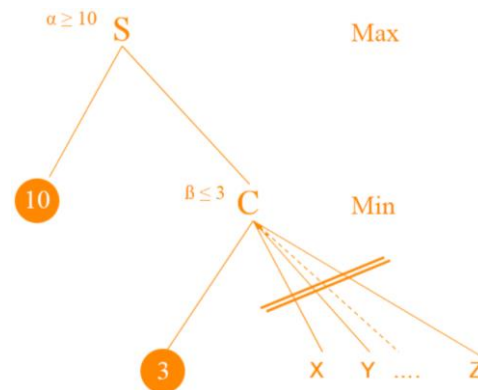
for *a, s* in **SUCCESSORS**(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

return *v*

b. Thuật toán cắt tỉa Alpha-beta với ứng dụng vào bài toán TicTacToe

Nút Max có một giá trị alpha (lớn hơn hoặc bằng alpha – luôn tăng), nút min có một giá trị beta (nhỏ hơn hoặc bằng beta – luôn giảm). Khi chưa có alpha và beta xác định thì thực hiện tìm kiếm sâu (depth-first) để xác định được alpha, beta, và truyền ngược lên các nút cha.



Giải thích:

Vấn đề tại sao chúng ta có thể cắt bỏ toàn bộ những nút con của C trên cây trò chơi trên.

Đầu tiên là xét cây từ trái sang phải ta sẽ thấy S là Max, theo chiến lược đưa ra vậy chúng ta sẽ có giá trị $\alpha \geq 10$ tại S.

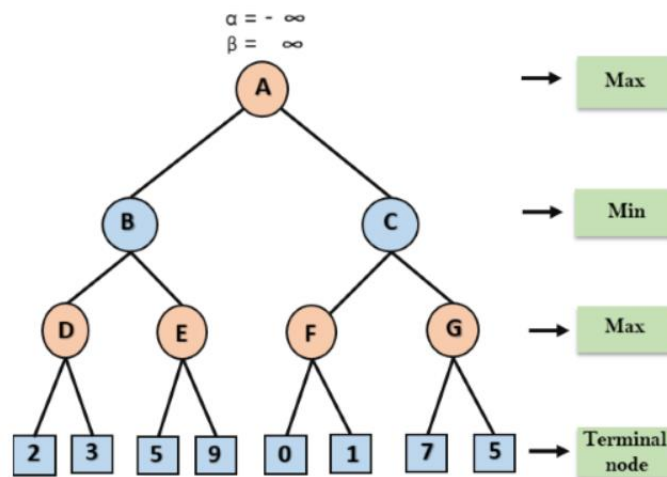
Tiếp theo, ở C ở đây là nút Min (trạng thái trò chơi dành cho Max) tức là sẽ lấy giá trị nhỏ nhất của các nút con ở dưới. Nếu như vậy thì giá trị chúng ta phải lấy là $\beta \leq 3$.

Sau khi xác định được alpha và beta, chúng ta có thể dễ dàng xác định việc có cắt tỉa hay không. Ở nút S (Max), giá trị alpha luôn ≥ 10 (luôn tăng) nhưng ở C (Min) thì giá trị luôn luôn ≤ 3 (luôn giảm), nên việc xét các con còn lại ở C là không cần thiết.

Nếu theo khoảng thì hiện tại chúng ta chỉ nhận khoảng ≥ 10 tại nút gốc S, vậy thì đâu cần bận tâm đến việc khoảng ≤ 3 tại nút C.

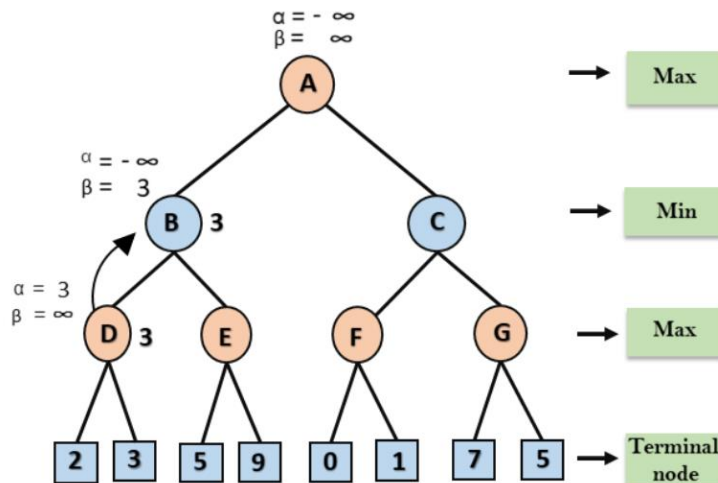
Ví dụ: Khảo sát cây dưới đây để hiểu về phương pháp cắt tỉa Alpha-beta.

Bước 1: Ở bước đầu tiên, người chơi Max sẽ bắt đầu di chuyển đầu tiên từ nút A nơi $\alpha = -\infty$ và $\beta = +\infty$, những giá trị alpha và beta này được truyền lại cho nút B nơi lại $\alpha = -\infty$ và $\beta = +\infty$ và Nút B chuyển cùng một giá trị cho nút con D của nó.



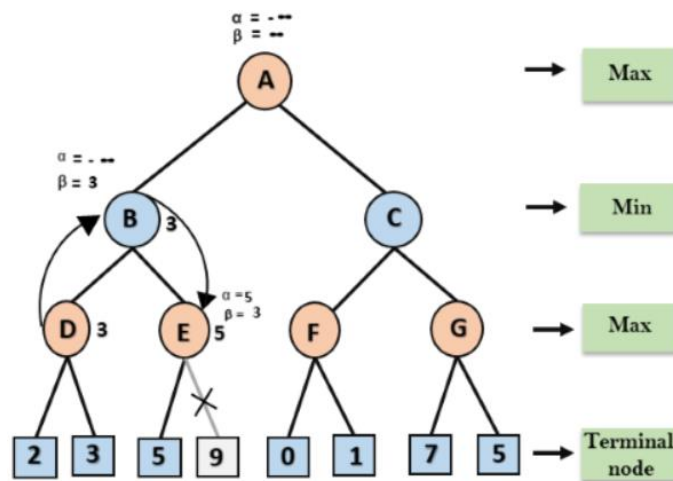
Bước 2: Tại nút D, giá trị của α sẽ được tính theo lượt của nó cho Max. Giá trị của α được so sánh với đầu tiên là 2 và sau đó là 3, và $\max(2, 3) = 3$ sẽ là giá trị của α tại nút D và giá trị của nút cũng sẽ là 3.

Bước 3: Bây giờ thuật toán quay ngược lại nút B, trong đó giá trị của β sẽ thay đổi vì đây là lượt của Min, Bây giờ $\beta = +\infty$, sẽ so sánh với giá trị của các nút tiếp theo có sẵn, tức là $\min(\infty, 3) = 3$, do đó tại nút B bây giờ $\alpha = -\infty$ và $\beta = 3$.



Trong bước tiếp theo, thuật toán duyệt qua nút kế tiếp tiếp theo của nút B là nút E, và các giá trị của $\alpha = -\infty$ và $\beta = 3$ cũng sẽ được chuyển.

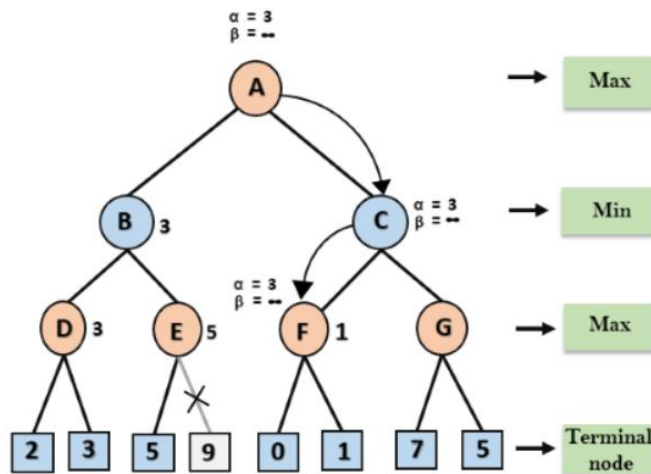
Bước 4: Tại nút E, Max sẽ đến lượt, và giá trị của alpha sẽ thay đổi. Giá trị hiện tại của alpha sẽ được so sánh với 5, vì vậy $\max(-\infty, 5) = 5$, do đó tại nút E $\alpha = 5$ và $\beta = 3$, trong đó $\alpha > \beta$, vì vậy kế thừa bên phải của E sẽ bị lược bỏ, và thuật toán sẽ không đi qua nó, và giá trị tại nút E sẽ là 5.



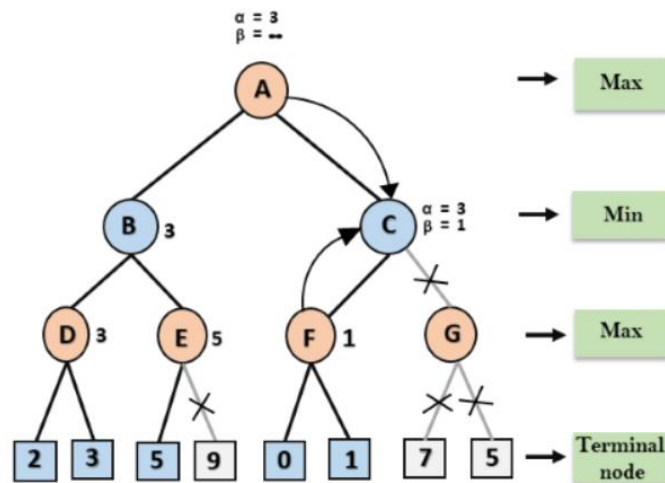
Bước 5: Ở bước tiếp theo, thuật toán lại chiếu ngược cây, từ nút B đến nút A. Tại nút A, giá trị của alpha sẽ được thay đổi giá trị lớn nhất có sẵn là 3 như $\max(-\infty, 3) = 3$ và $\beta = +\infty$, hai giá trị này bây giờ được chuyển đến người kế nhiệm bên phải của A là Nút C.

Tại nút C, $\alpha = 3$ và $\beta = +\infty$, và các giá trị tương tự sẽ được chuyển cho nút F.

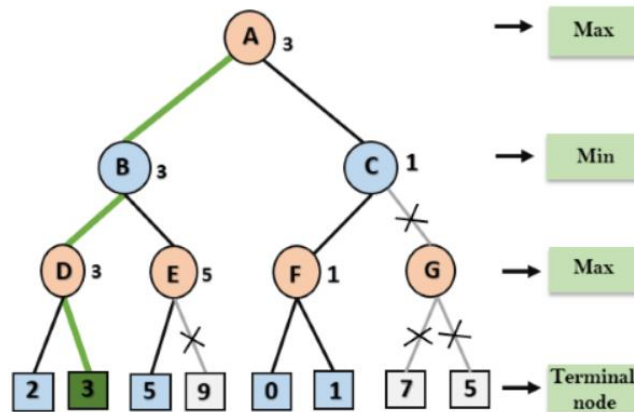
Bước 6: Tại nút F, một lần nữa giá trị của α sẽ được so sánh với nút con bên trái là 0 và $\max(3, 0) = 3$, sau đó so sánh với nút con bên phải là 1 và $\max(3, 1) = 3$ vẫn còn α vẫn là 3, nhưng giá trị nút của F sẽ trở thành 1.



Bước 7: Nút F trả về giá trị nút 1 cho nút C, tại C $\alpha = 3$ và $\beta = +\infty$, ở đây giá trị beta sẽ được thay đổi, nó sẽ so sánh với 1 nên $\min(\infty, 1) = 1$. Bây giờ tại C, $\alpha = 3$ và $\beta = 1$, và một lần nữa nó thỏa mãn điều kiện $\alpha \geq \beta$, vì vậy con tiếp theo của C là G sẽ bị lược bớt, và thuật toán sẽ không tính toàn bộ cây con G.



Bước 8: C bây giờ trả về giá trị từ 1 đến A ở đây giá trị tốt nhất cho A là $\max(3, 1) = 3$. Sau đây là cây trò chơi cuối cùng là hiển thị các nút được tính toán và các nút chưa bao giờ tính toán. Do đó, giá trị tối ưu cho bộ cực đại là 3 cho ví dụ này.



Thuật giải cắt tỉa Alpha-beta.

```
function MINIMAX-DECISION(state) returns an action
```

```
   $v \leftarrow \text{MAX-VALUE}(\textit{state})$ 
```

```
  return the action in SUCCESSORS(state) with value  $v$ 
```

```
function MAX-VALUE(state) returns a utility value
```

```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
   $v \leftarrow -\infty$ 
```

```
  for  $a, s$  in SUCCESSORS(state) do
```

```
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
```

```
  return  $v$ 
```

```
function MIN-VALUE(state) returns a utility value
```

```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
   $v \leftarrow \infty$ 
```

```
  for  $a, s$  in SUCCESSORS(state) do
```

```
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$ 
```

```
  return  $v$ 
```

II. Bài tập hướng dẫn mẫu

a. Thuật giải Minimax ứng dụng vào bài toán TicTacToe

```
import copy
import math
import random
import numpy

X = "X"
O = "O"
EMPTY = None
user = None
ai = None

def initial_state():
    """
    Returns starting state of the board.
    """
    return [[EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY]]

def player(board):
    """
    Returns player who has the next turn on a board.
    """
    count = 0
    for i in board:
        for j in i:
            if j:
                count += 1
    if count % 2 != 0:
        return ai
    return user

def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """
    res = set()
    board_len = len(board)
    for i in range(board_len):
        for j in range(board_len):
            if board[i][j] == EMPTY:
                res.add((i, j))
    return res

def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    curr_player = player(board)
    result_board = copy.deepcopy(board)
    (i, j) = action
    result_board[i][j] = curr_player
    return result_board

def get_horizontal_winner(board):
    # check horizontally
    winner_val = None
    board_len = len(board)
    for i in range(board_len):
        winner_val = board[i][0]
        for j in range(board_len):
            if board[i][j] != winner_val:
```



```

        winner_val = None
    if winner_val:
        return winner_val
    return winner_val

def get_vertical_winner(board):
    # check vertically
    winner_val = None
    board_len = len(board)
    for i in range(board_len):
        winner_val = board[0][i]
        for j in range(board_len):
            if board[j][i] != winner_val:
                winner_val = None
    if winner_val:
        return winner_val
    return winner_val

def get_diagonal_winner(board):
    # check diagonally
    winner_val = None
    board_len = len(board)
    winner_val = board[0][0]
    for i in range(board_len):
        if board[i][i] != winner_val:
            winner_val = None
    if winner_val:
        return winner_val
    winner_val = board[0][board_len - 1]
    for i in range(board_len):
        j = board_len - 1 - i
        if board[i][j] != winner_val:
            winner_val = None
    return winner_val

def winner(board):
    """
    Returns the winner of the game, if there is one.
    """
    winner_val = get_horizontal_winner(board) or get_vertical_winner(board)
    or get_diagonal_winner(board) or None
    return winner_val

def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """
    if winner(board) != None:
        return True
    for i in board:
        for j in i:
            if j == EMPTY:
                return False
    return True

def utility(board):
    """
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.
    """
    winner_val = winner(board)
    if winner_val == X:
        return 1
    elif winner_val == O:
        return -1

```

```

    return 0

def maxValue(state):
    if terminal(state):
        return utility(state)
    v = -math.inf
    for action in actions(state):
        v = max(v, minValue(result(state, action)))
    return v

def minValue(state):
    if terminal(state):
        return utility(state)
    v = math.inf
    for action in actions(state):
        v = min(v, maxValue(result(state, action)))
    return v

def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """
    current_player = player(board)
    if current_player == X:
        min = -math.inf
        for action in actions(board):
            check = minValue(result(board, action)) # FIXED
            if check > min:
                min = check
                move = action
    else:
        max = math.inf
        for action in actions(board):
            check = maxValue(result(board, action)) # FIXED
            if check < max:
                max = check
                move = action
    return move

if __name__ == "__main__":
    board = initial_state()
    ai_turn = False
    print("Choose a player")
    user = input()
    if user == "X":
        ai = "O"
    else:
        ai = "X"
    while True:
        game_over = terminal(board)
        playr = player(board)
        if game_over:
            winner = winner(board)
            if winner is None:
                print("Game Over: Tie.")
            else:
                print(f"Game Over: {winner} wins.")
            break;
        else:
            if user != playr and not game_over:
                if ai_turn:
                    move = minimax(board)
                    board = result(board, move)
                    ai_turn = False

```

```

        print(numpy.array(board))
    elif user == playr and not game_over:
        ai_turn = True
        print("Enter the position to move (row,col)")
        i = int(input("Row:"))
        j = int(input("Col:"))
        if board[i][j] == EMPTY:
            board = result(board, (i, j))
            print(numpy.array(board))

```

b. Thuật toán cắt tỉa Alpha-beta với ứng dụng vào bài toán TicTacToe

```

# I found this article very helpful:
# https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-
introduction/

import os, math

def GetWinner(board):
    """
    Returns the winner in the current board if there is one, otherwise it
    returns None.
    """
    # horizontal
    if board[0] == board[1] and board[1] == board[2]:
        return board[0]
    elif board[3] == board[4] and board[4] == board[5]:
        return board[3]
    elif board[6] == board[7] and board[7] == board[8]:
        return board[6]
    # vertical
    elif board[0] == board[3] and board[3] == board[6]:
        return board[0]
    elif board[1] == board[4] and board[4] == board[7]:
        return board[1]
    elif board[2] == board[5] and board[5] == board[8]:
        return board[2]
    # diagonal
    elif board[0] == board[4] and board[4] == board[8]:
        return board[0]
    elif board[2] == board[4] and board[4] == board[6]:
        return board[2]

def PrintBoard(board):
    """
    Clears the console and prints the current board.
    """
    os.system('cls' if os.name=='nt' else 'clear')
    print(f'''
{board[0]}|{board[1]}|{board[2]}
{board[3]}|{board[4]}|{board[5]}
{board[6]}|{board[7]}|{board[8]}
''')

def GetAvailableCells(board):
    """
    Returns a list of indices containing all available cells in a board.
    """
    available = list()
    for cell in board:
        if cell != "X" and cell != "O":
            available.append(cell)
    return available

```

```

def minimax(position, depth, alpha, beta, isMaximizing):
    """
    The AI algorithm responsible for choosing the best move. Returns best
    value of a move.
    """
    # evaluate current board: if maximizing player won -> return 10
    #                             if minimizing player won -> return -10
    #                             if no one is winning (tie) -> return 0

    # NOTE: Even though the following AI plays perfectly, it might
    #         choose to make a move which will result in a slower victory
    #         or a faster loss. Lets take an example and explain it
    #         Assume that there are 2 possible ways for X to win the game
    from a give board state.
    #         Move A : X can win in 2 move
    #         Move B : X can win in 4 moves
    #         Our evaluation will return a value of +10 for both moves A and
    B. Even though the move A
    #         is better because it ensures a faster victory, our AI may
    choose B sometimes. To overcome
    #         this problem we subtract the depth value from the evaluated
    score. This means that in case
    #         of a victory it will choose a the victory which takes least
    number of moves and in case of
    #         a loss it will try to prolong the game and play as many moves
    as possible.
    winner = GetWinner(position)
    if winner != None:
        return 10 - depth if winner == "X" else -10 + depth
    if len(GetAvailableCells(position)) == 0:
        return 0

    if isMaximizing:
        maxEval = -math.inf
        for cell in GetAvailableCells(position):
            position[cell - 1] = "X"
            Eval = minimax(position, depth + 1, alpha, beta, False)
            maxEval = max(maxEval, Eval)
            alpha = max(alpha, Eval)
            position[cell - 1] = cell
            if beta <= alpha:
                break # prune
        return maxEval
    else:
        minEval = +math.inf
        for cell in GetAvailableCells(position):
            position[cell - 1] = "O"
            Eval = minimax(position, depth + 1, alpha, beta, True)
            minEval = min(minEval, Eval)
            beta = min(beta, Eval)
            position[cell - 1] = cell
            if beta <= alpha:
                break # prune
        return minEval

def FindBestMove(currentPosition, AI):
    """
    This will return the best possible move for the player.
    Will Traverse all cells, evaluate minimax function for all empty cells.
    And return the cell with optimal value.
    Parameters:
        currentPosition (list): The current board to find best move for.
        AI (str): The AI Player ("X" or "O").
    Returns:

```

```

        int: Index of best move for the current position.
    """
    bestVal = -math.inf if AI == "X" else +math.inf
    bestMove = -1
    for cell in GetAvailableCells(currentPosition):
        currentPosition[cell - 1] = AI
        moveVal = minimax(currentPosition, 0, -math.inf, +math.inf, False
    if AI == "X" else True)
        currentPosition[cell - 1] = cell
        if (AI == "X" and moveVal > bestVal):
            bestMove = cell
            bestVal = moveVal
        elif (AI == "O" and moveVal < bestVal):
            bestMove = cell
            bestVal = moveVal
    return bestMove

def main():
    player = input("Play as X or O? ").strip().upper()
    AI = "O" if player == "X" else "X"
    currentGame = [*range(1, 10)]
    # X always starts first.
    currentTurn = "X"
    counter = 0
    while True:
        if currentTurn == AI:
            # NOTE: if the AI starts first, it'll always choose index 0 so
            to save time you could play it.
            cell = FindBestMove(currentGame, AI)
            currentGame[cell - 1] = AI
            currentTurn = player
        elif currentTurn == player:
            PrintBoard(currentGame)
            while True:
                humanInput = int(input("Enter Number: ").strip())
                if humanInput in currentGame:
                    currentGame[humanInput - 1] = player
                    currentTurn = AI
                    break
            else:
                PrintBoard(currentGame)
                print("Cell Not Available.")
        if GetWinner(currentGame) != None:
            PrintBoard(currentGame)
            print(f"{GetWinner(currentGame)} WON!!!")
            break
        counter += 1
        if GetWinner(currentGame) == None and counter == 9:
            PrintBoard(currentGame)
            print("Tie.")
            break

if __name__ == "__main__":
    main()

```

III. Bài tập ở lớp

Bài 1. Cài đặt thuật toán minimax cho bài toán tictactoe với ma trận 3x3.

Bài 2. Cài đặt thuật toán cắt tỉa Alpha-beta với ứng dụng vào bài toán TicTacToe với ma trận 3x3.

IV. Bài tập về nhà

Bài 3. Cài đặt thuật toán minimax cho bài toán tictactoe với ma trận 5×5 , 10×10 , $N \times N$.

Bài 4. Cài đặt thuật toán cắt tỉa Alpha-beta cho bài toán tictactoe với ma trận 5×5 , 10×10 , $N \times N$.

Bài 5. Ứng dụng kĩ thuật cắt tỉa Alpha-beta vào game cờ tướng, cờ vua và cờ vây.

Bài 6. Xây dựng giao diện đồ họa cho trò chơi tictactoe.