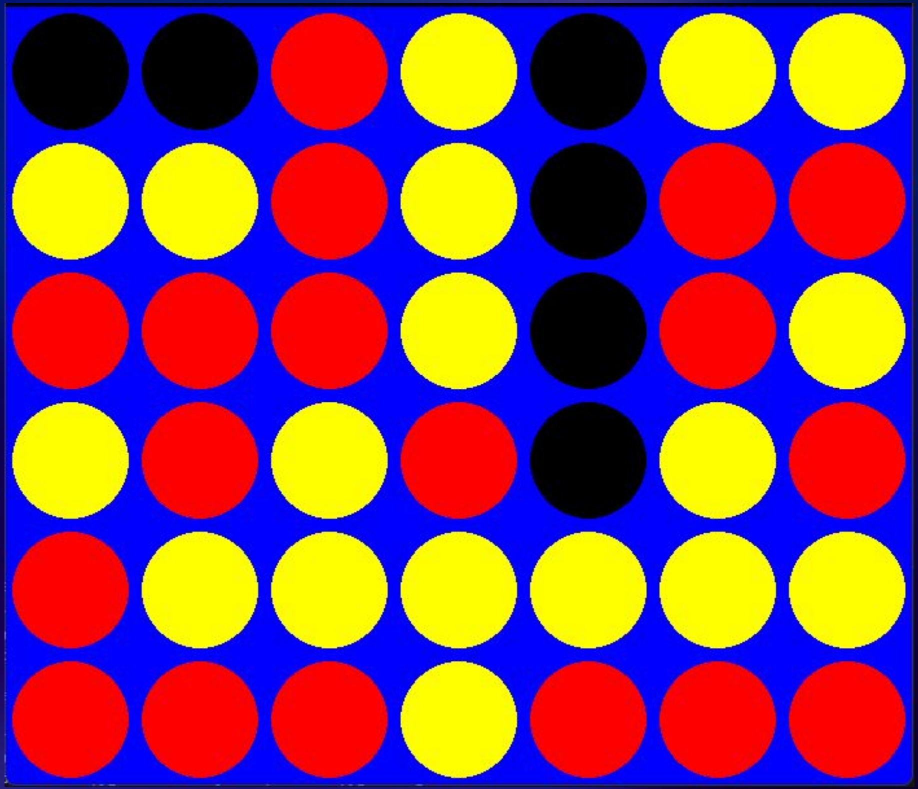
CPSC 481 Project

Bryan, Jericho, Logan

Connect 4

- A two-player connection game where players drop colored discs into a 6-row, 7-column vertical grid.
- The first player who forms a horizontal, vertical, or diagonal line of their own four discs wins.
- The problem we will solve is to design an artificial intelligence agent that can play Connect 4 at a competitive level.

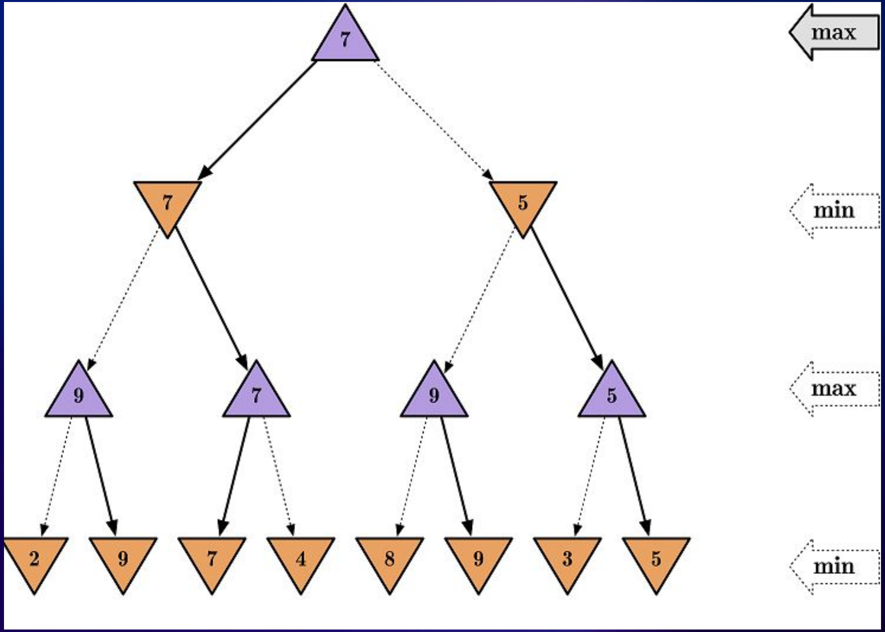


AI approach

- Minimax Algorithm
- Alpha-beta Pruning
- Heuristic Evaluation Function

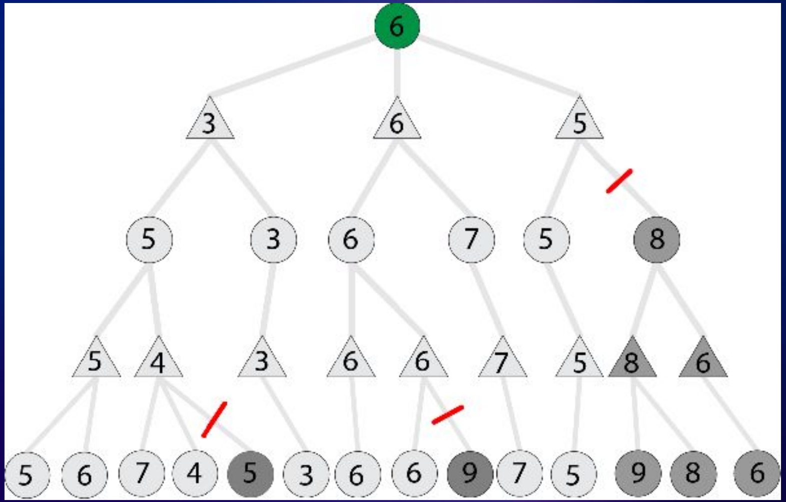
Minimax Algorithm

- Allows computer to plan ahead
- Evaluate possible future moves
- Ensures AI plays optimally for the given search depth.
- Prevents mistakes that lets the opponent win



Alpha-Beta Pruning

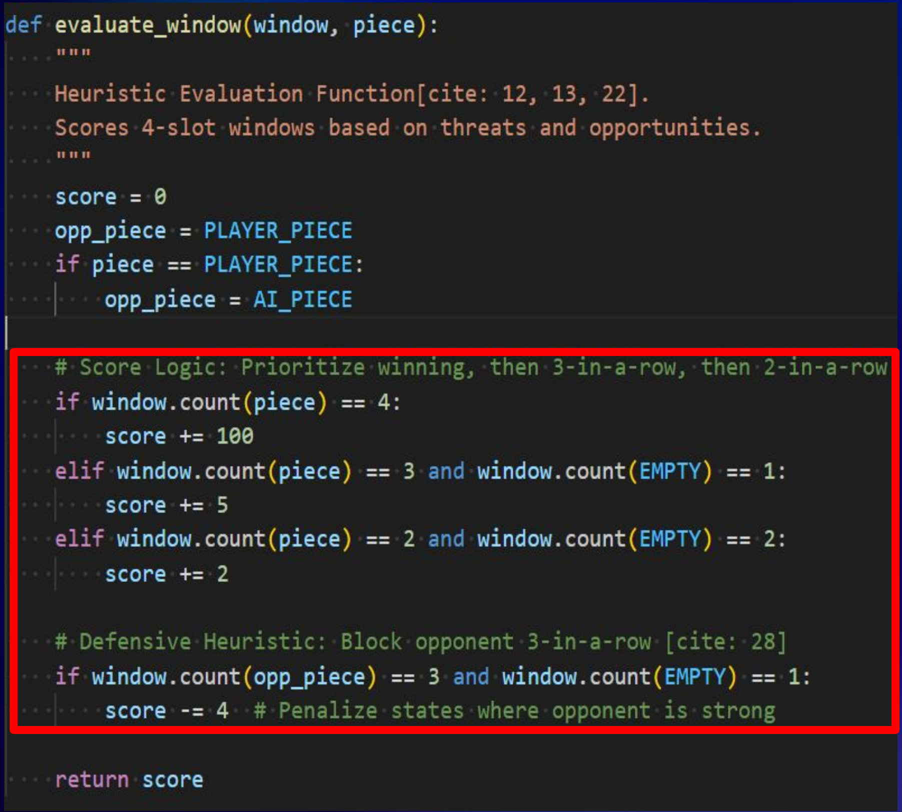
- Speeds up minimax algorithm by cutting off the branches that do not matter
- Avoids useless calculations
- Does not change the minimax result



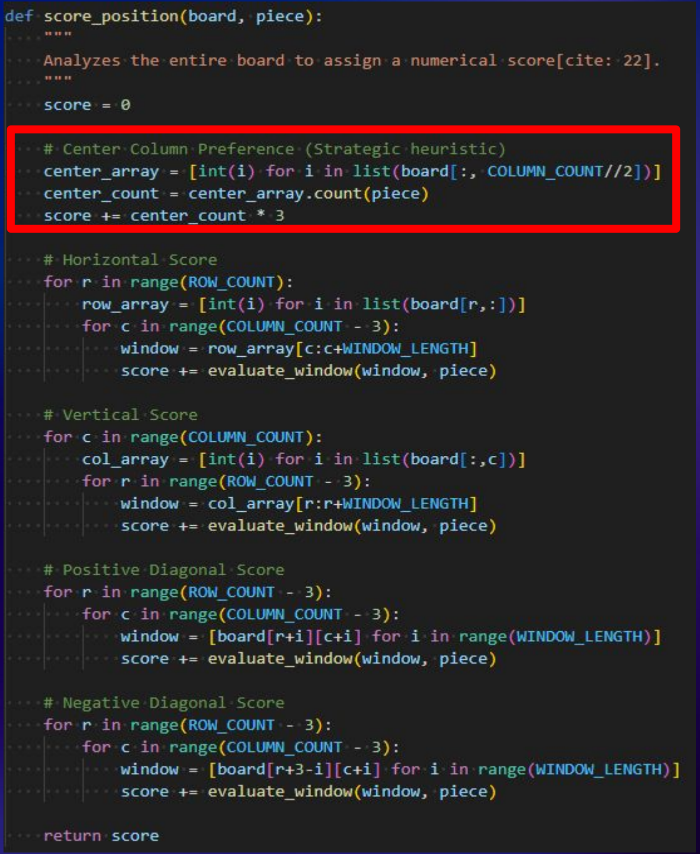
Heuristics

- Without heuristics, minimax would be blind
- Heuristics allows the AI to make intelligent decisions before seeing a final outcome
- Makes the AI behave like a human player

AI approach code

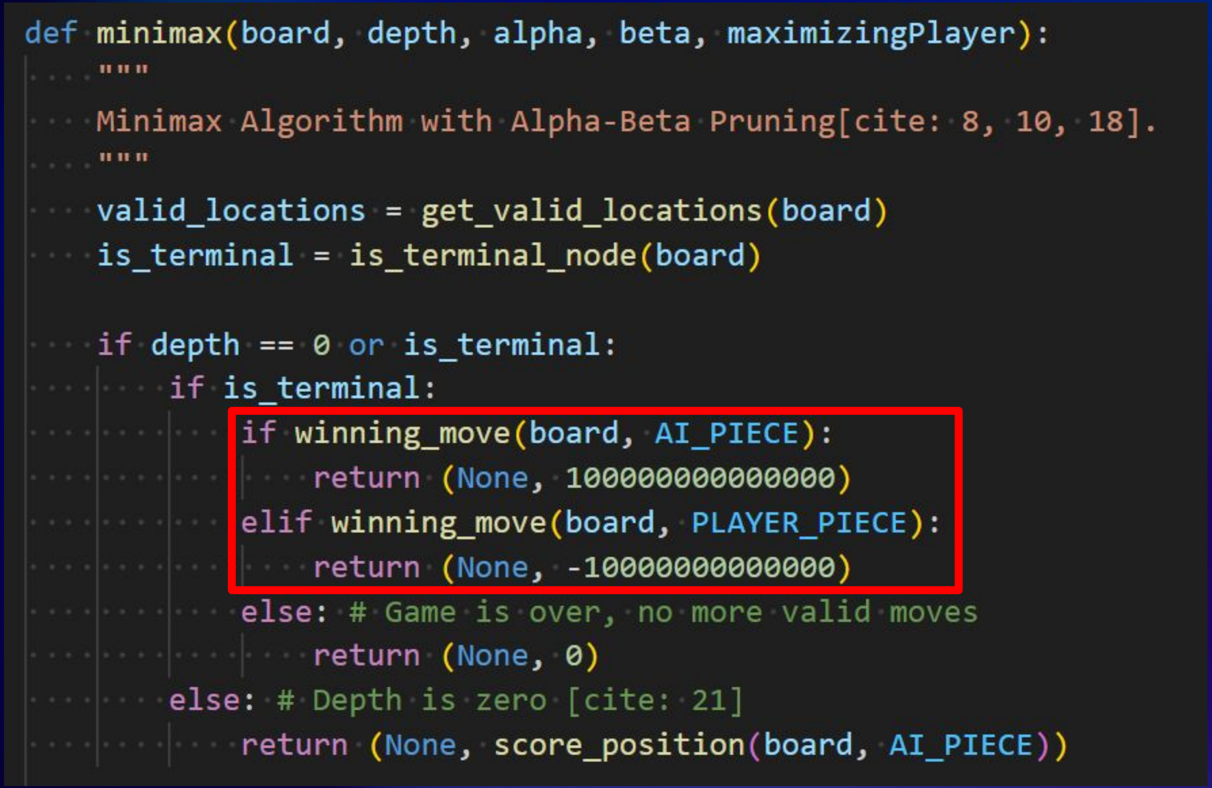


```python
def evaluate_window(window, piece):
    """
    Heuristic Evaluation Function[cite: 12, 13, 22].
    Scores 4-slot windows based on threats and opportunities.
    """
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
        opp_piece = AI_PIECE

    # Score Logic: Prioritize winning, then 3-in-a-row, then 2-in-a-row
    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 5
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 2

    # Defensive Heuristic: Block opponent 3-in-a-row [cite: 28]
    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 4  # Penalize states where opponent is strong

    return score
```



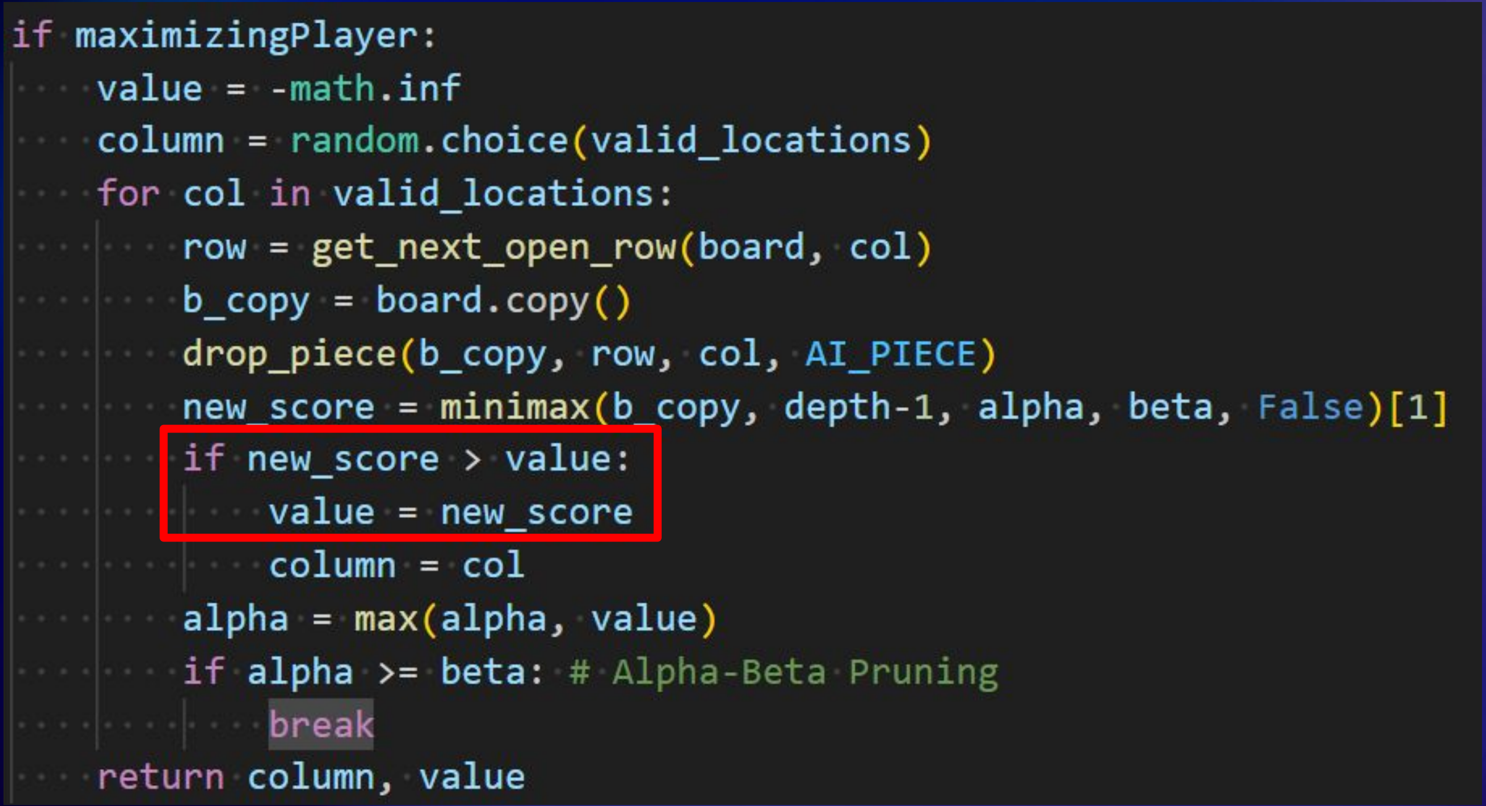
```python
def score_position(board, piece):
    """
    Analyzes the entire board to assign a numerical score[cite: 22].
    """
    score = 0

    # Center Column Preference (Strategic heuristic)
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 3

    # Horizontal Score
    for r in range(ROW_COUNT):
        row_array = [int(i) for i in list(board[r,:])]
        for c in range(COLUMN_COUNT - 3):
            window = row_array[c:c+WINDOW_LENGTH]
            score += evaluate_window(window, piece)

    # Vertical Score
    for c in range(COLUMN_COUNT):
        col_array = [int(i) for i in list(board[:,c])]
        for r in range(ROW_COUNT - 3):
            window = col_array[r:r+WINDOW_LENGTH]
            score += evaluate_window(window, piece)

    # Positive Diagonal Score
    for r in range(ROW_COUNT - 3):
        for c in range(COLUMN_COUNT - 3):
            window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

    # Negative Diagonal Score
    for r in range(ROW_COUNT - 3):
        for c in range(COLUMN_COUNT - 3):
            window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

    return score
```

AI approach code



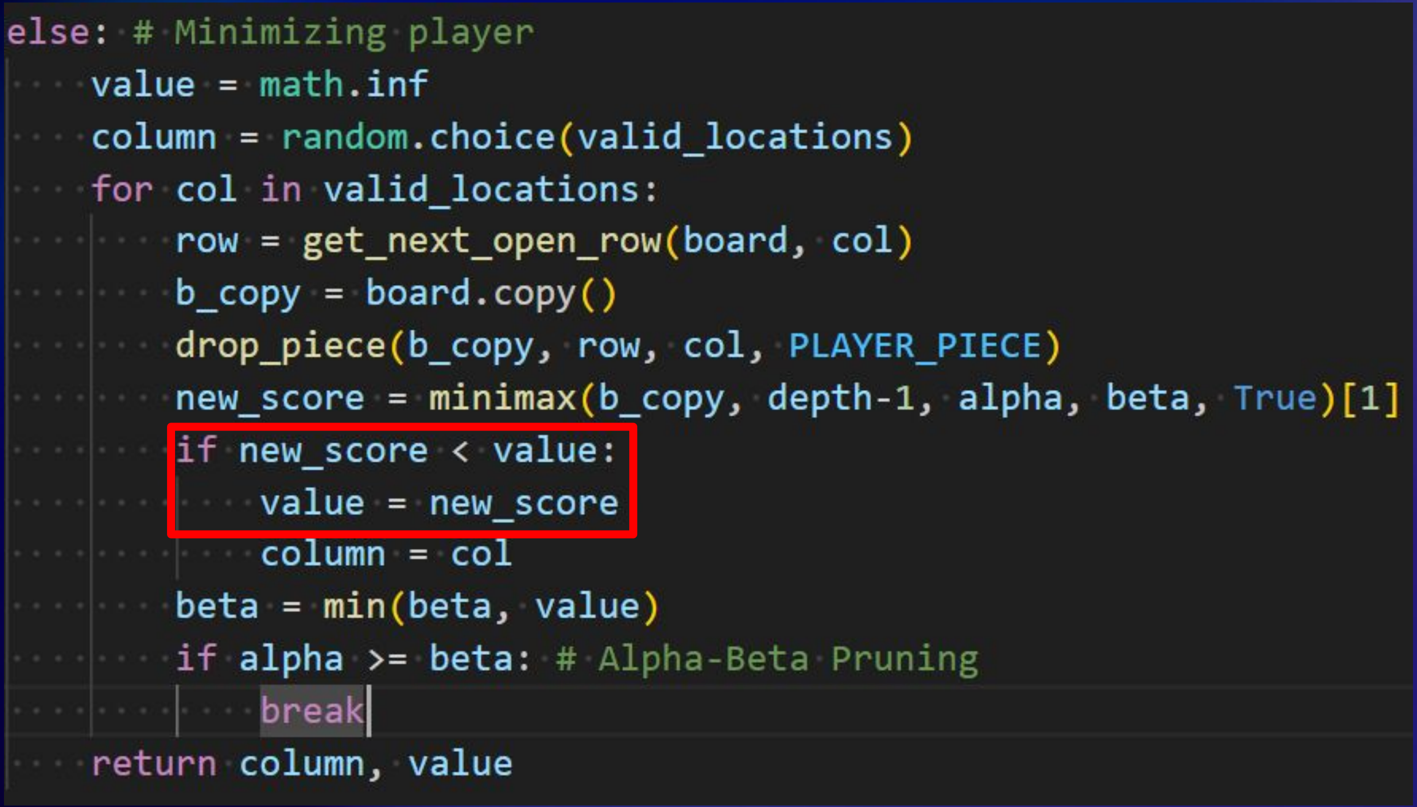
```python
def minimax(board, depth, alpha, beta, maximizingPlayer):
    """
    Minimax Algorithm with Alpha-Beta Pruning[cite: 8, 10, 18].
    """
    valid_locations = get_valid_locations(board)
    is_terminal = is_terminal_node(board)

    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 100000000000000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -10000000000000)
            else: # Game is over, no more valid moves
                return (None, 0)
        else: # Depth is zero [cite: 21]
            return (None, score_position(board, AI_PIECE))
```

AI approach code



```
if maximizingPlayer:
    value = -math.inf
    column = random.choice(valid_locations)
    for col in valid_locations:
        row = get_next_open_row(board, col)
        b_copy = board.copy()
        drop_piece(b_copy, row, col, AI_PIECE)
        new_score = minimax(b_copy, depth-1, alpha, beta, False)[1]
        if new_score > value:
            value = new_score
            column = col
        alpha = max(alpha, value)
        if alpha >= beta: # Alpha-Beta Pruning
            break
    return column, value
```

AI approach code



```
else: # Minimizing player
    value = math.inf
    column = random.choice(valid_locations)
    for col in valid_locations:
        row = get_next_open_row(board, col)
        b_copy = board.copy()
        drop_piece(b_copy, row, col, PLAYER_PIECE)
        new_score = minimax(b_copy, depth-1, alpha, beta, True)[1]
        if new_score < value:
            value = new_score
            column = col
        beta = min(beta, value)
        if alpha >= beta: # Alpha-Beta Pruning
            break
    return column, value
```

Demo

Key Results

- Successful implementation of a functional Connect Four game
- AI can make intelligent decisions using minimax
- Alpha-Beta Pruning improves performance without affecting the results
- Heuristic Evaluation Function makes AI performs with Human-like behavior
- GUI rendering works smoothly

Contributions

Bryan Maus – Backend Game Logic

Jericho Salonga – AI Implementation

Logan Clampitt – Frontend/GUI

References

https://sandipanweb.wordpress.com/2017/03/06/using-minimax-with-alpha-beta-pruning-and-heuristic-evaluation-to-solve-2048-game-with-computer/

https://www.semanticscholar.org/paper/Alpha-beta-Pruning-in-Chess-Engines-Marckel/3c5cdc5fb590fc56ac429884216f8e0ce31c8164