CS 7320 HW3 Submission Template

Name: Charles Bryan
ID: 49350684

Part 1: copy or paste your test results

```
Test1 Output:

/Users/charlesbryan/anaconda3/envs/002C_1242/bin/python
/Users/charlesbryan/Desktop/AI/401_1242/hw3/test1_get_child_boards.py

PASS Test 1

PASS Test 2

PASS Test 3  4x4 array


Process finished with exit code 0
```

Test2 Output:

```
/Users/charlesbryan/anaconda3/envs/002C_1242/bin/python
/Users/charlesbryan/Desktop/AI/401_1242/hw3/test2_evaluate.py

PASS Test 1 No Win

PASS Test 2  column win

PASS Test 3  row win

PASS Test 4  diag win

PASS Test 5  diag2 win

PASS Test 6  diag2 win 4x4


Process finished with exit code 0
```

Test3 Output:

```
/Users/charlesbryan/anaconda3/envs/002C_1242/bin/python
/Users/charlesbryan/Desktop/AI/401_1242/hw3/test3_is_terminal_node.py

PASS Test 1 Non Terminal Board

PASS Test 2  Terminal Board

PASS Test 3  Terminal Board

PASS Test 4  Terminal Board

PASS Test 5  Terminal Board

PASS Test 6  Terminal Board


Process finished with exit code 0
```

Part 2.

Run MiniMax Results

Board	Score	# Boards	Time
b2	-1	1109	0.05132603645324707
b3	0	549946	21.290376901626587
b4	0	8146001	306.3955090045929

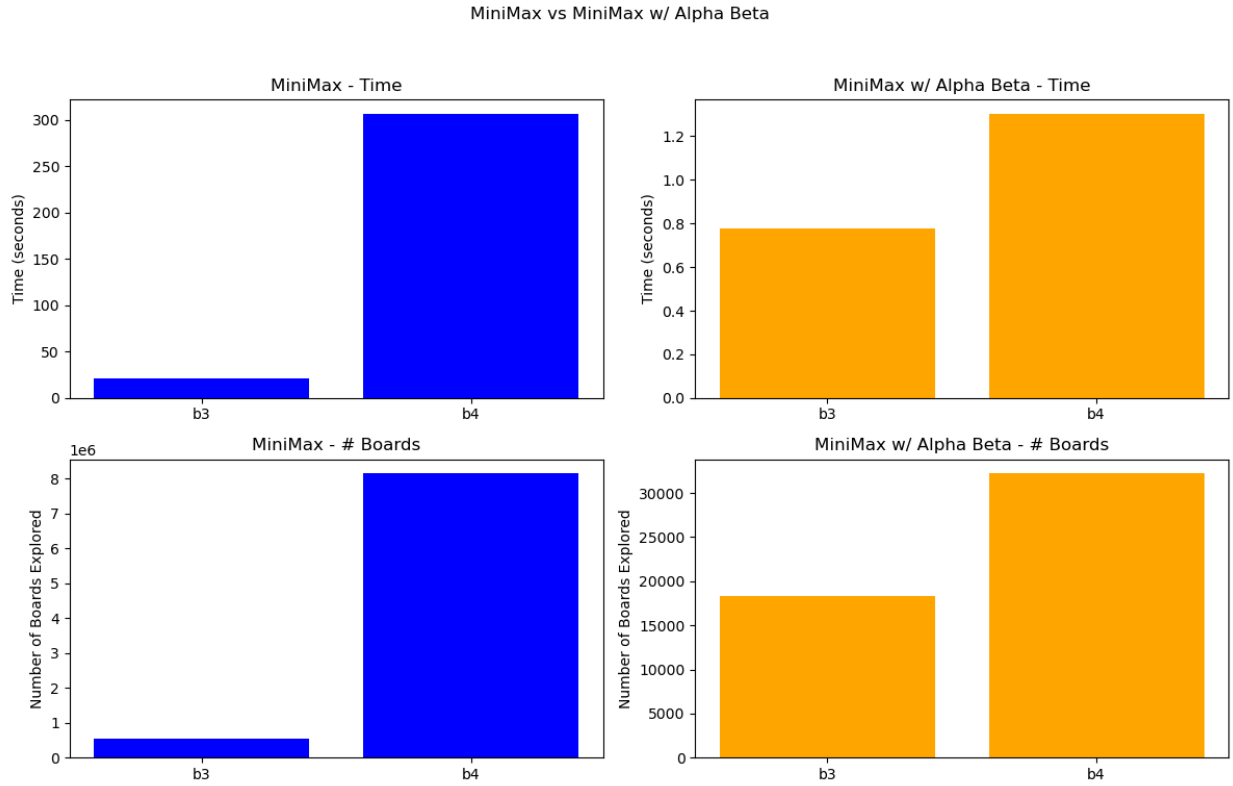
Part 3.

MiniMax Results

Board	Score	# Boards	Time
b3	0	549946	21.290376901626587
b4	0	8146001	306.3955090045929

MiniMax w/ Alpha Beta Results

Board	Score	# Boards	Time
b3	0	18297	0.7763669490814209
b4	0	32201	1.3025169372558594



Part 4.

Describe your heuristic

My heuristic assesses each line of the board for immediate wins or potential wins, assigning higher scores to boards where the player is closer to winning and lower scores to boards where the opponent is closer to winning. This heuristic will help guide my previous alpha-beta minimax function toward the "best boards" first. By adding this heuristic, my algorithm becomes more efficient in navigating the search space. Here is a breakdown of the functions -

- **tic_tac_toe_heuristic**: Calculates the overall score of a Tic-Tac-Toe board from a specific player's perspective. (Sums up the scores of all rows, columns, and diagonals)
- **evaluate_line**: Calculates the score of a single line based on whether it is a winning line, or has the potential to be a winning line for the player or the opponent.
- **is_winning_line**: Determines if a line is a winning line
- **evaluate_potential_line**: Assesses a line for its potential value by counting the number of player or opponent marks. Returns a positive score if the line has potential for the player, or a negative score if it has potential for the opponent.

Show the code for your heuristic (not your entire program)

```
def tic_tac_toe_heuristic(board, char):

    player = 1 if char == 'X' else -1

    opponent = -player

    score = 0


    for i in range(3):

        score += evaluate_line(board[i, :], player, opponent)

        score += evaluate_line(board[:, i], player, opponent)

    score += evaluate_line(np.diag(board), player, opponent)

    flipped_board = np.fliplr(board)

    score += evaluate_line(np.diag(flipped_board), player, opponent)


    return score
```

```
def evaluate_line(line, player, opponent):
    line_score = 0

    if is_winning_line(line, player):
        line_score += 100
    elif is_winning_line(line, opponent):
        line_score -= 100
    else:
        line_score += evaluate_potential_line(line, player, opponent)

    return line_score

def evaluate_potential_line(line, player_val, opponent_val):
    player_count = np.count_nonzero(line == player_val)
    opponent_count = np.count_nonzero(line == opponent_val)

    potential_win_for_player = player_count > 0 and opponent_count ==
0
    potential_win_for_opponent = opponent_count > 0 and player_count
== 0

    if potential_win_for_player:
        return player_count
    elif potential_win_for_opponent:
        return -opponent_count

    return 0
```

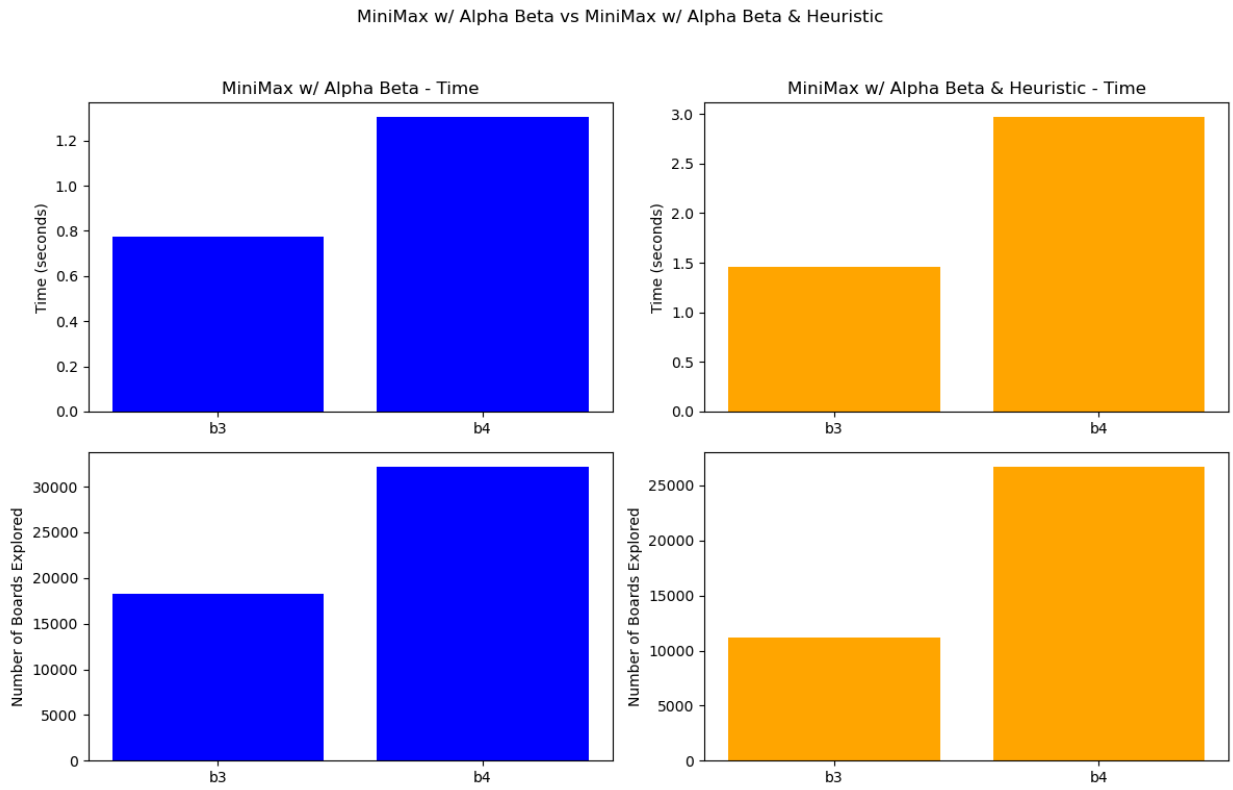
```
def is_winning_line(line, player_val):
    return np.all(line == player_val)
```

MiniMax w/ Alpha Beta

Board	Score	# Boards	Time
b3	0	18297	0.7763669490814209
b4	0	32201	1.3025169372558594

MiniMax w/ Alpha Beta Using Heuristc

Board	Score	# Boards	Time
b3	0	11152	1.4585857391357422
b4	0	26721	2.9652512073516846



Discussion.

Write a well-organized discussion of the results of your tests. Did Minimax and Alpha-Beta pruning work as expected? Did you encounter any challenges? Was your heuristic effective? Why?

Both MiniMax with Alpha-Beta pruning and the heuristic version worked as expected in reducing the number of boards explored. The heuristic further optimized the search space, demonstrating its potential effectiveness.

The increase in time with the heuristic highlights a trade-off between reducing the number of explored boards and the computational cost of evaluating those boards. This trade-off is crucial in game AI development and can vary depending on the complexity of the heuristic and the nature of the game.

Overall, the tests confirmed the utility of Alpha-Beta pruning in enhancing MiniMax efficiency and demonstrated the potential of heuristics in further optimizing the search, albeit with considerations for computational overhead.