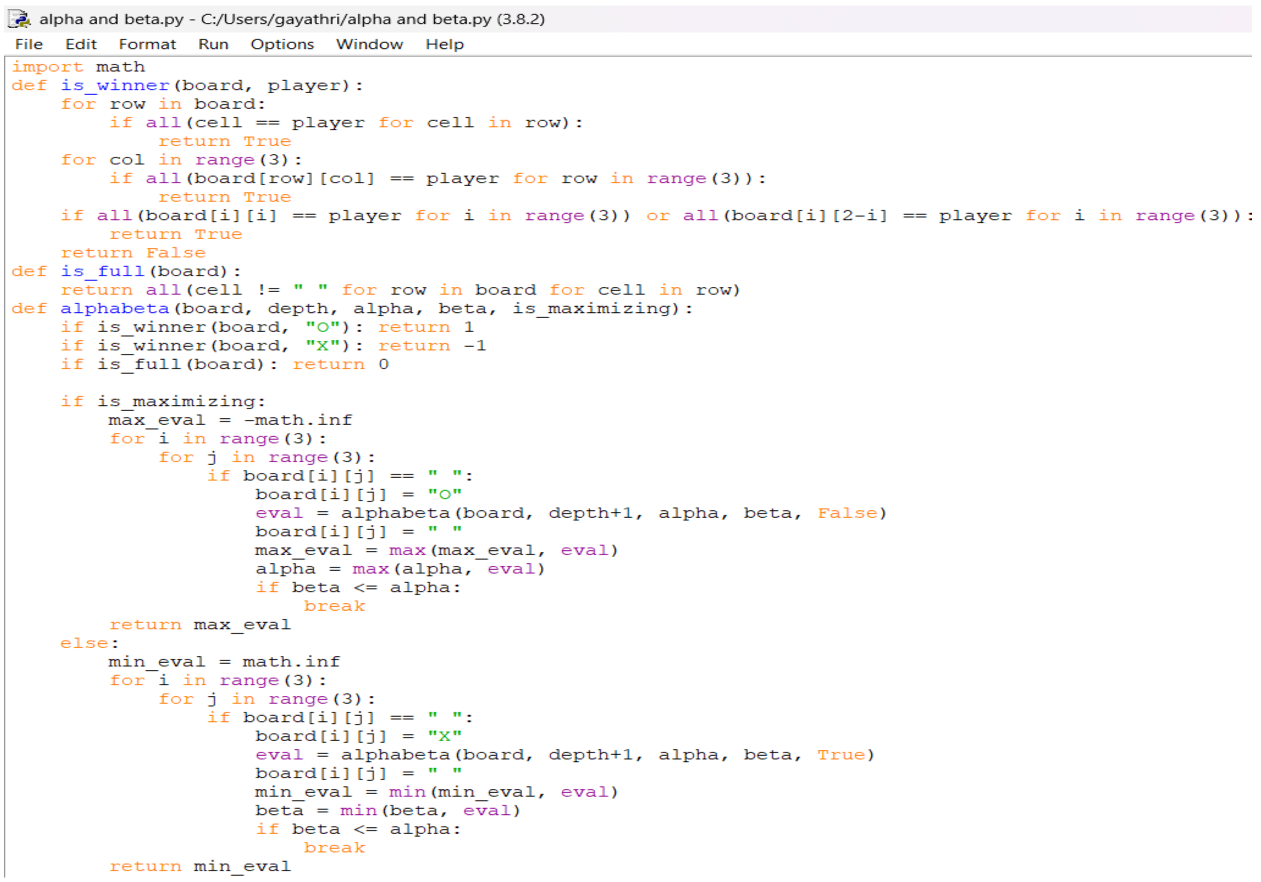
Write the python program to implement Apha & Beta pruning algorithm for gaming

AIM

To implement **Alpha-Beta Pruning Algorithm** in Python to optimize the Minimax search by pruning branches of the game tree that do not affect the final decision.

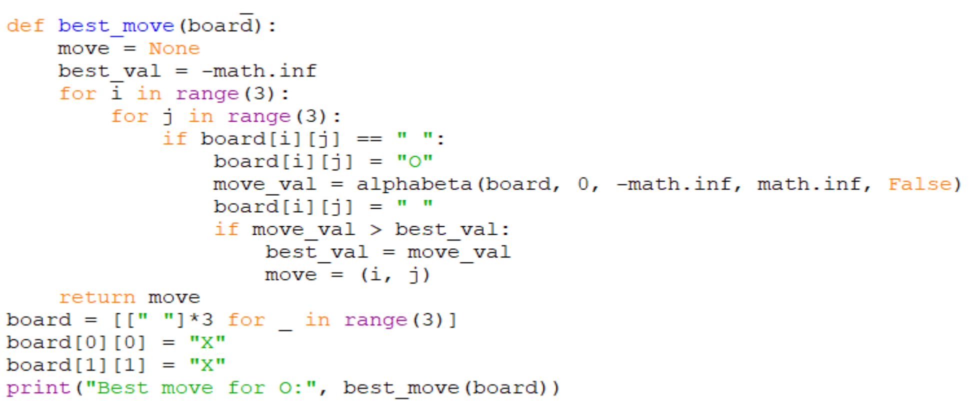
Algorithm

1. Start with root node, initialize alpha = -∞, beta = +∞.
2. If the node is a **leaf node**, return its value.
3. If it's a **Maximizing Player**:
    a. Initialize best = -∞.
    b. For each child:
        i. Call recursively with minimizing player.
        ii. Update best = max(best, val).
        iii. Update alpha = max(alpha, best).
        iv. If beta <= alpha, prune remaining branches.
4. If it's a **Minimizing Player**:
    a. Initialize best = +∞.
    b. For each child:
        i. Call recursively with maximizing player.
        ii. Update best = min(best, val).
        iii. Update beta = min(beta, best).
        iv. If beta <= alpha, prune remaining branches.
5. Return the best value.

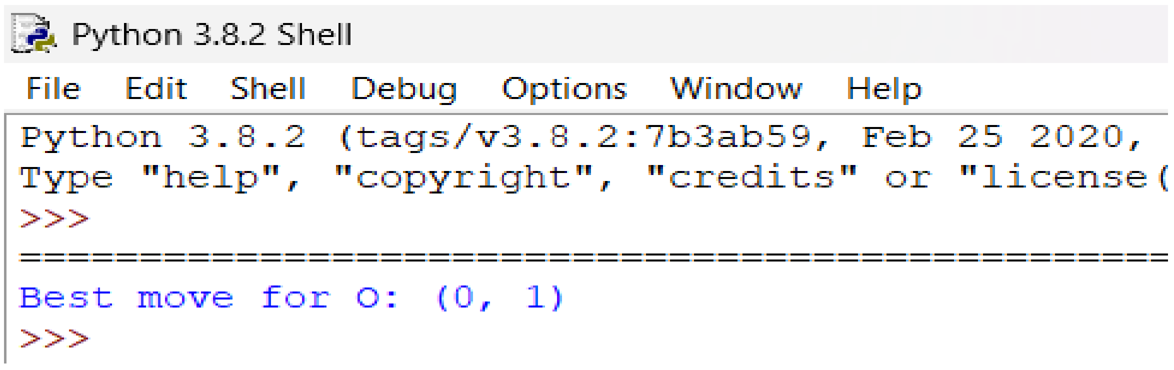

alpha and beta.py - C:/Users/gayathri/alpha and beta.py (3.8.2)
File Edit Format Run Options Window Help

```python
import math
def is_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] == player for i in range(3)):
        return True
    return False
def is_full(board):
    return all(cell != " " for row in board for cell in row)
def alphabeta(board, depth, alpha, beta, is_maximizing):
    if is_winner(board, "O"): return 1
    if is_winner(board, "X"): return -1
    if is_full(board): return 0

    if is_maximizing:
        max_eval = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    eval = alphabeta(board, depth+1, alpha, beta, False)
                    board[i][j] = " "
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    eval = alphabeta(board, depth+1, alpha, beta, True)
                    board[i][j] = " "
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval
```



```python
def best_move(board):
    move = None
    best_val = -math.inf
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "O"
                move_val = alphabeta(board, 0, -math.inf, math.inf, False)
                board[i][j] = " "
                if move_val > best_val:
                    best_val = move_val
                    move = (i, j)
    return move
board = [[" "]*3 for _ in range(3)]
board[0][0] = "X"
board[1][1] = "X"
print("Best move for O:", best_move(board))
```



```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020,
Type "help", "copyright", "credits" or "license(
>>>
==================================================
Best move for O: (0, 1)
>>>
```

Result / Output

For leaf nodes: [3, 5, 6, 9, 1, 2, 0, -1]