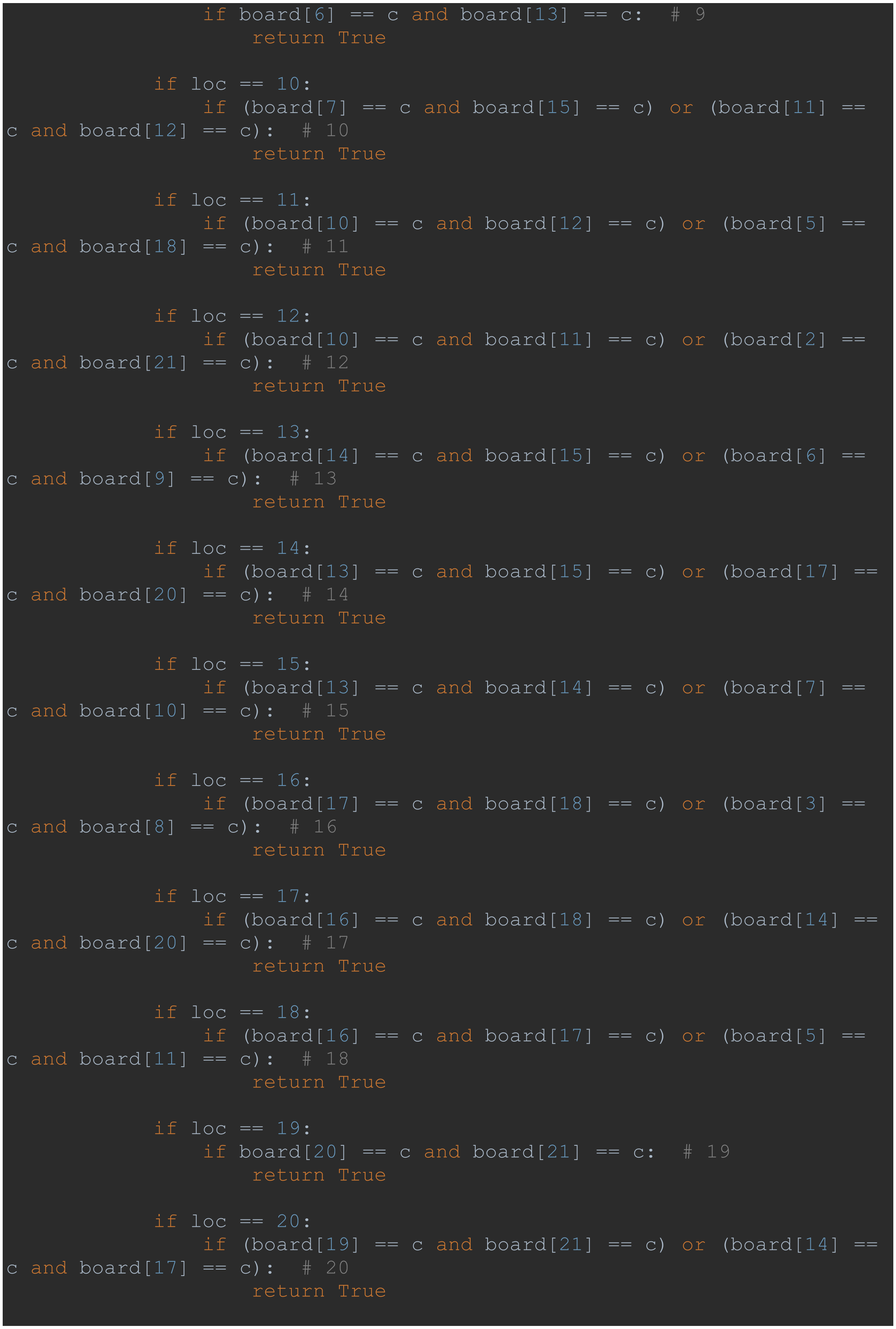
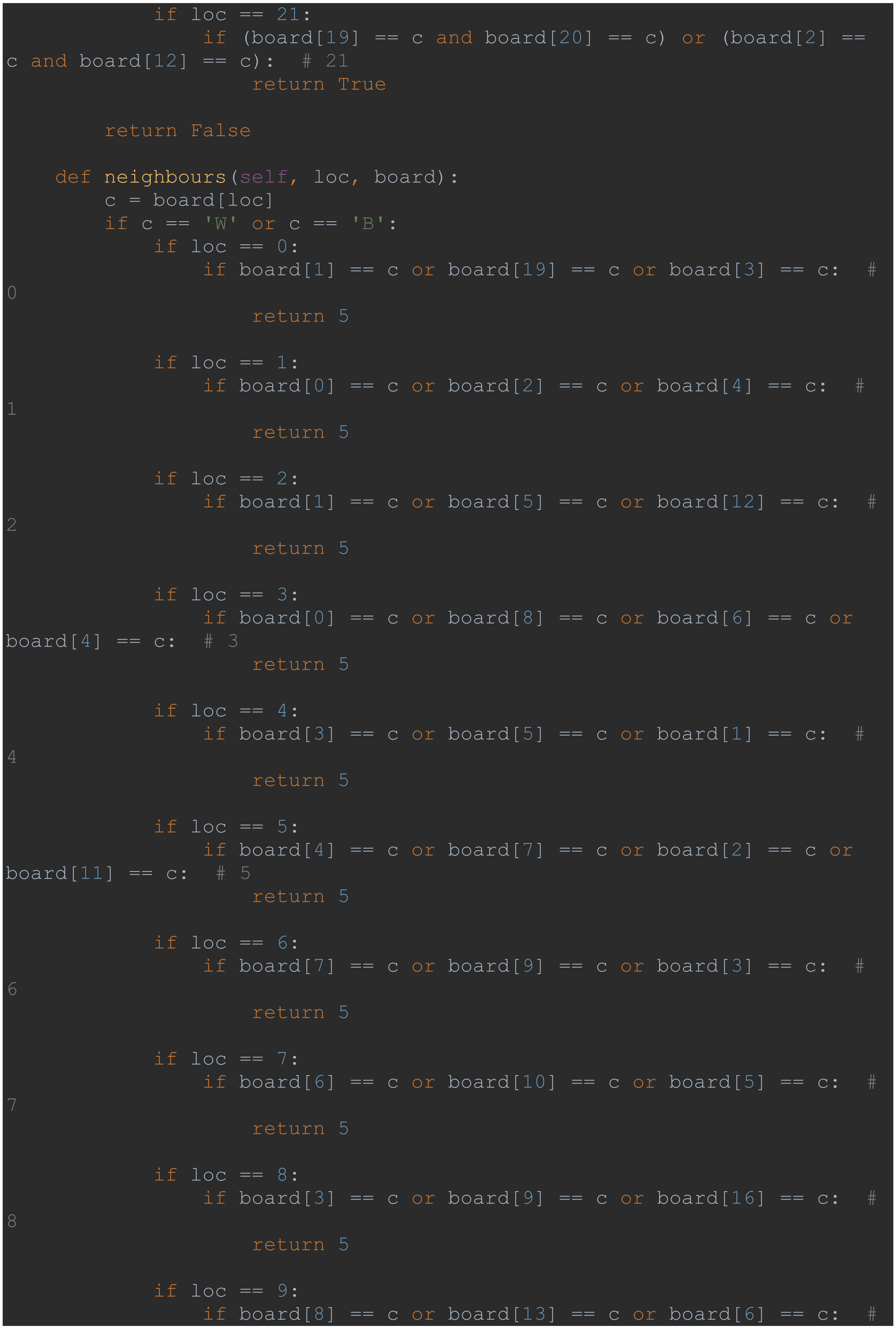
Source code

1. MiniMaxOpening

```
from io import StringIO
import sys


class MiniMaxOpening:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
```

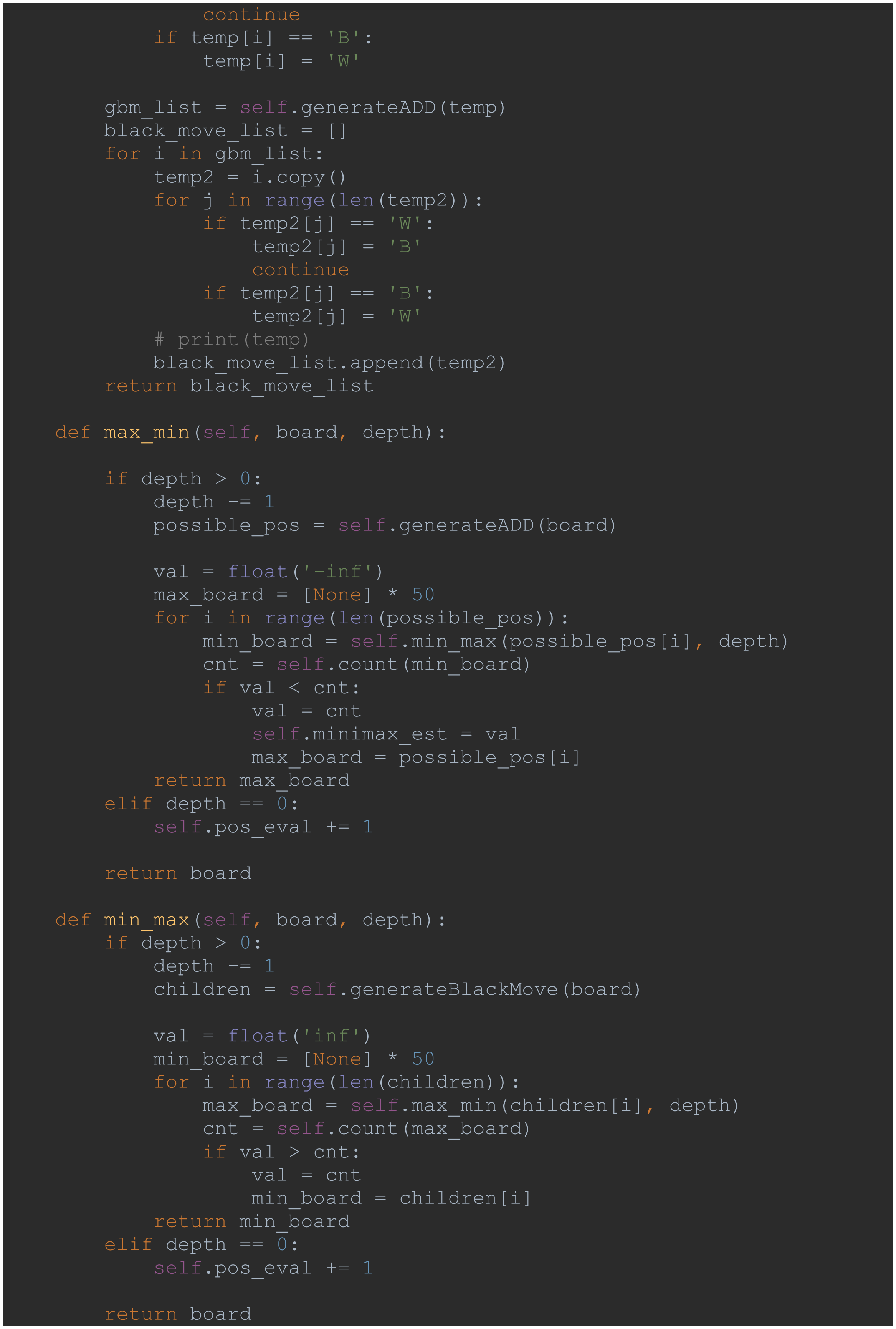


```
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True
```



```
            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c or board[19] == c or board[3] == c:  #
0
                    return 5

            if loc == 1:
                if board[0] == c or board[2] == c or board[4] == c:  #
1
                    return 5

            if loc == 2:
                if board[1] == c or board[5] == c or board[12] == c:  #
2
                    return 5

            if loc == 3:
                if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
                    return 5

            if loc == 4:
                if board[3] == c or board[5] == c or board[1] == c:  #
4
                    return 5

            if loc == 5:
                if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
                    return 5

            if loc == 6:
                if board[7] == c or board[9] == c or board[3] == c:  #
6
                    return 5

            if loc == 7:
                if board[6] == c or board[10] == c or board[5] == c:  #
7
                    return 5

            if loc == 8:
                if board[3] == c or board[9] == c or board[16] == c:  #
8
                    return 5

            if loc == 9:
                if board[8] == c or board[13] == c or board[6] == c:  #
```

```
9
                    return 5

            if loc == 10:
                if board[7] == c or board[15] == c or board[11] == c:
# 10
                    return 5

            if loc == 11:
                if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
                    return 5

            if loc == 12:
                if board[11] == c or board[2] == c or board[21] == c:
# 12
                    return 5

            if loc == 13:
                if board[14] == c or board[16] == c or board[9] == c:
# 13
                    return 5

            if loc == 14:
                if board[13] == c or board[15] == c or board[17] == c:
# 14
                    return 5

            if loc == 15:
                if board[14] == c or board[18] == c or board[10] == c:
# 15
                    return 5

            if loc == 16:
                if board[17] == c or board[19] == c or board[13] == c
or board[8] == c:  # 16
                    return 5

            if loc == 17:
                if board[16] == c or board[18] == c or board[14] == c
or board[20] == c:  # 17
                    return 5

            if loc == 18:
                if board[15] == c or board[17] == c or board[21] == c
or board[11] == c:  # 18
                    return 5

            if loc == 19:
                if board[20] == c or board[0] == c or board[16] == c:
# 19
                    return 5

            if loc == 20:
                if board[19] == c or board[21] == c or board[17] == c:
# 20
                    return 5
            if loc == 21:
                if board[20] == c or board[18] == c or board[12] == c:
# 21
                    return 5

        return False

    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def count(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1
        return w_count - b_count

    def generateADD(self, board):
        pos = []
        for i in range(len(board)):
            if board[i] == 'x':
                temp = board.copy()
                temp[i] = 'W'
                if self.closemill(i, temp):
                    pos = self.generateRem(temp, pos)
                else:
                    pos.append(temp)
        return pos

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
```



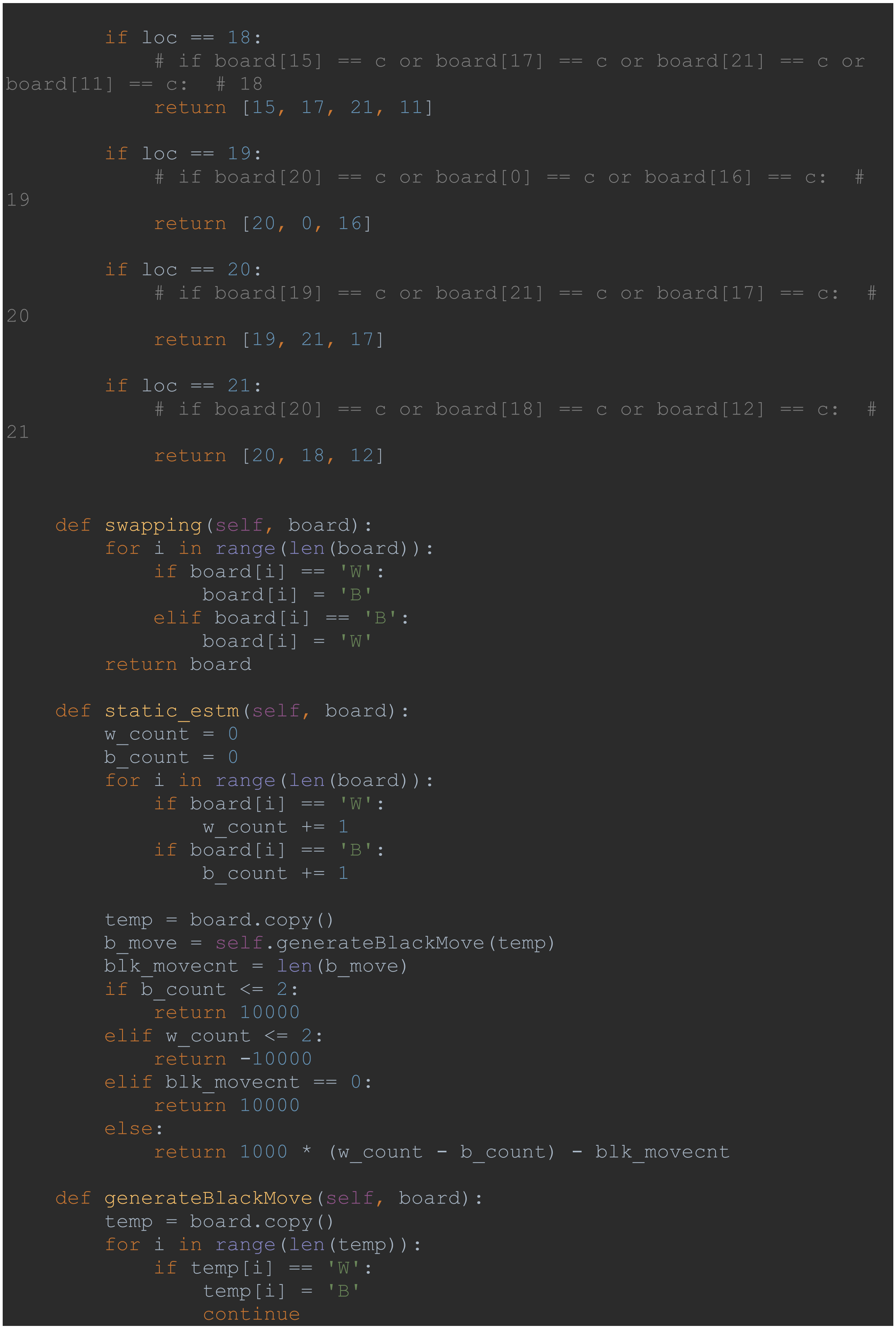
```python
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.generateADD(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            # print(temp)
            black_move_list.append(temp2)
        return black_move_list

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.generateADD(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.count(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.count(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board
```

```
if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxOpening()
        new_moves = obj.max_min(board, depth)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: '+new_s)
        print('Positions Evaluated: '+str(obj.pos_eval))
        print('MiniMax evaluation: '+str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation:
'+str(obj.minimax_est)+'\n')
```

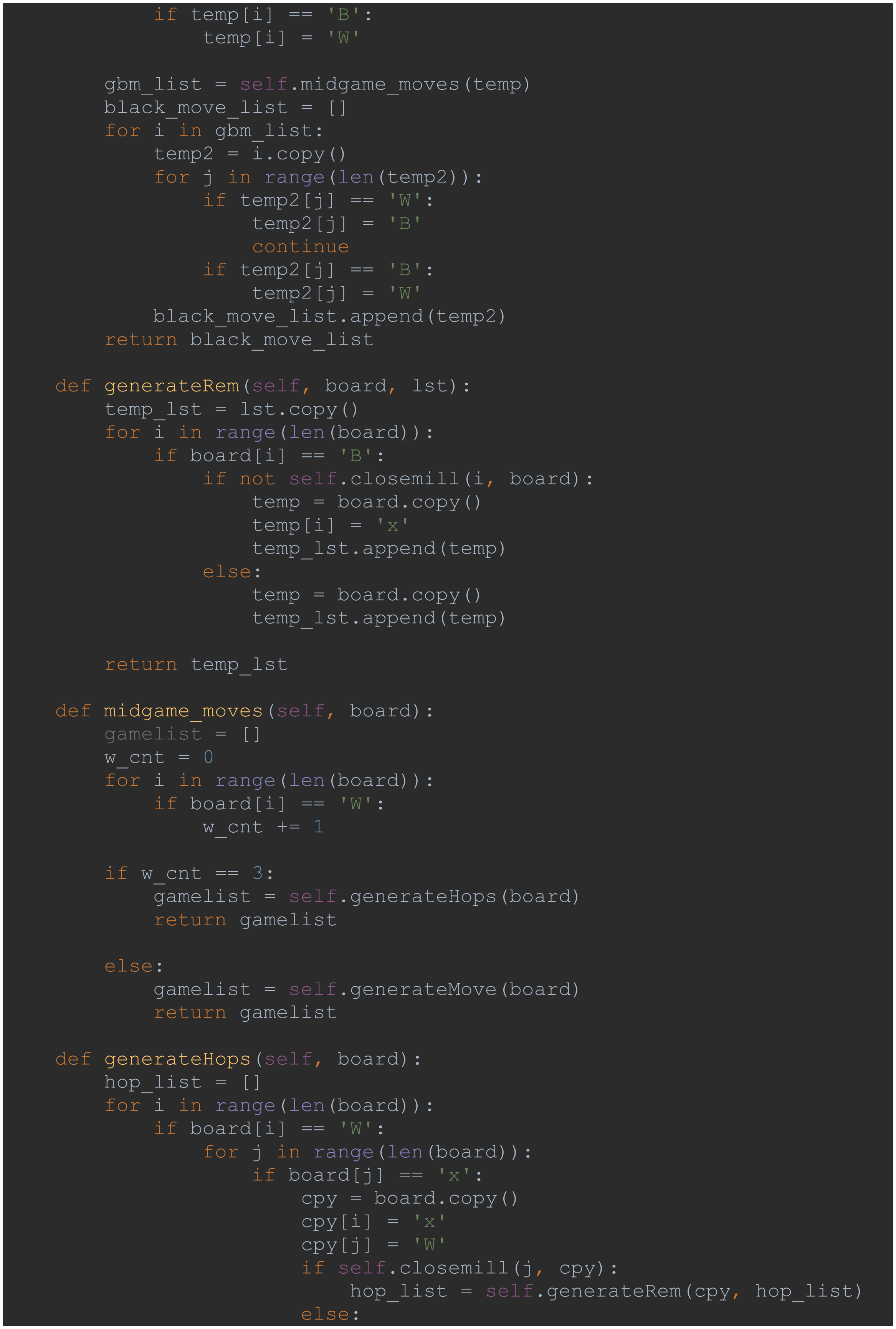
2. MiniMaxGame

```
import sys


class MiniMaxGame:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c
and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c
and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c
and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True
            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc):
        if loc == 0:
            # if board[1] == c or board[19] == c or board[3] == c:  # 0
            return [1, 3, 19]

        if loc == 1:
            # if board[0] == c or board[2] == c or board[4] == c:  # 1
            return [0, 4, 2]

        if loc == 2:
            # if board[1] == c or board[5] == c or board[12] == c:  # 2
            return [1, 5, 12]

        if loc == 3:
            # if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
            return [0, 8, 4, 6]

        if loc == 4:
            # if board[3] == c or board[5] == c or board[1] == c:  # 4
            return [3, 5, 1]

        if loc == 5:
            # if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
```

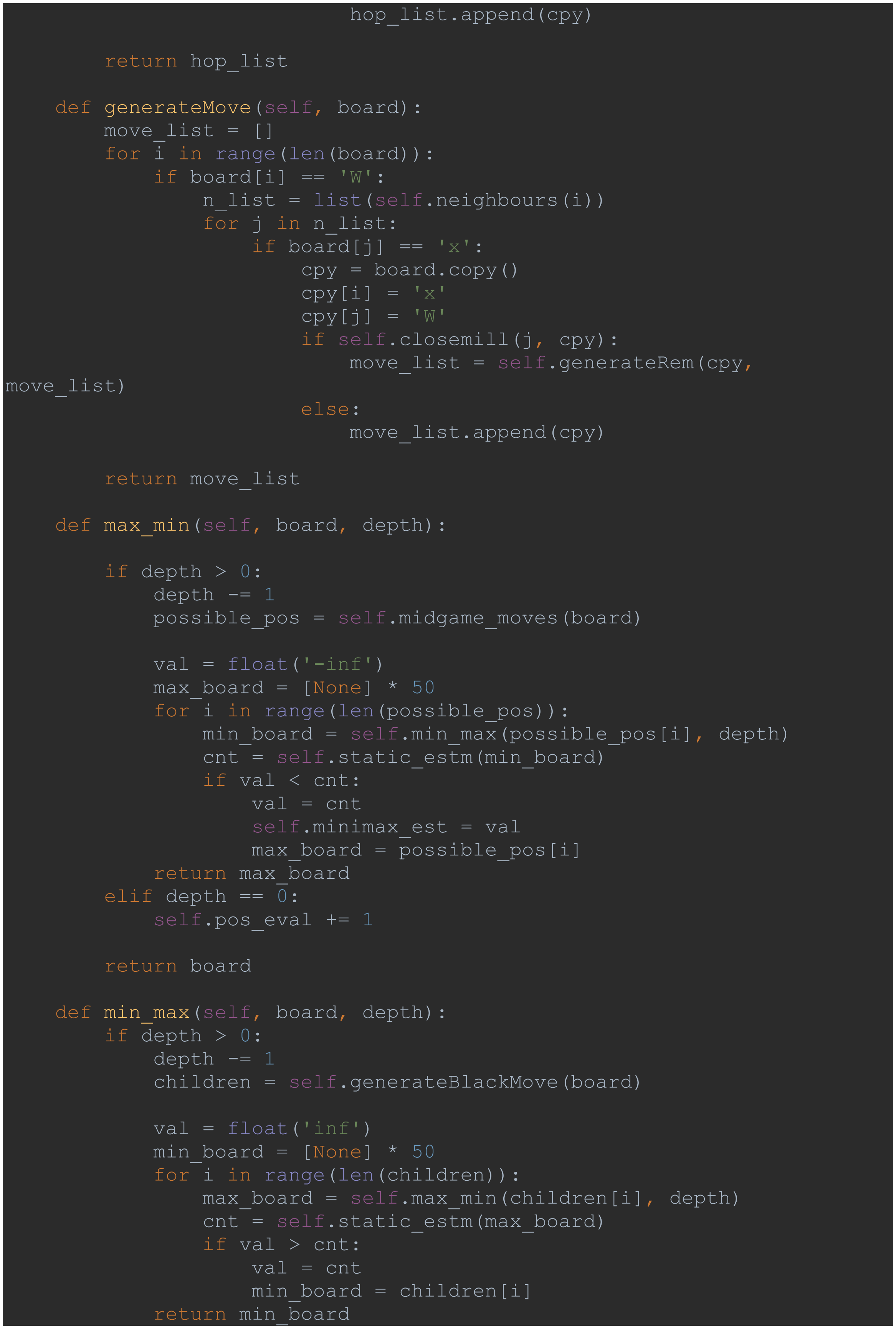
```
            return [4, 7, 2, 11]

        if loc == 6:
            # if board[7] == c or board[9] == c or board[3] == c:  # 6
            return [7, 9, 3]

        if loc == 7:
            # if board[6] == c or board[10] == c or board[5] == c:  # 7
            return [6, 10, 5]

        if loc == 8:
            # if board[3] == c or board[9] == c or board[16] == c:  # 8
            return [3, 9, 16]

        if loc == 9:
            # if board[8] == c or board[13] == c or board[6] == c:  # 9
            return [8, 13, 6]

        if loc == 10:
            # if board[7] == c or board[15] == c or board[11] == c:  #
10

            return [7, 15, 11]

        if loc == 11:
            # if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
            return [10, 12, 5, 18]

        if loc == 12:
            # if board[11] == c or board[2] == c or board[21] == c:  #
12

            return [11, 2, 21]

        if loc == 13:
            # if board[14] == c or board[16] == c or board[9] == c:  #
13

            return [14, 16, 9]

        if loc == 14:
            # if board[13] == c or board[15] == c or board[17] == c:  #
14

            return [13, 15, 17]

        if loc == 15:
            # if board[14] == c or board[18] == c or board[10] == c:  #
15

            return [14, 18, 10]

        if loc == 16:
            # if board[17] == c or board[19] == c or board[13] == c or
board[8] == c:  # 16
            return [17, 19, 13, 8]

        if loc == 17:
            # if board[16] == c or board[18] == c or board[14] == c or
board[20] == c:  # 17
            return [16, 18, 14, 20]
```



```
        if loc == 18:
            # if board[15] == c or board[17] == c or board[21] == c or
board[11] == c:  # 18
            return [15, 17, 21, 11]

        if loc == 19:
            # if board[20] == c or board[0] == c or board[16] == c:  #
19
            return [20, 0, 16]

        if loc == 20:
            # if board[19] == c or board[21] == c or board[17] == c:  #
20
            return [19, 21, 17]

        if loc == 21:
            # if board[20] == c or board[18] == c or board[12] == c:  #
21
            return [20, 18, 12]


    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def static_estm(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1

        temp = board.copy()
        b_move = self.generateBlackMove(temp)
        blk_movecnt = len(b_move)
        if b_count <= 2:
            return 10000
        elif w_count <= 2:
            return -10000
        elif blk_movecnt == 0:
            return 10000
        else:
            return 1000 * (w_count - b_count) - blk_movecnt

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
```



```python
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.midgame_moves(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def midgame_moves(self, board):
        gamelist = []
        w_cnt = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_cnt += 1

        if w_cnt == 3:
            gamelist = self.generateHops(board)
            return gamelist

        else:
            gamelist = self.generateMove(board)
            return gamelist

    def generateHops(self, board):
        hop_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                for j in range(len(board)):
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            hop_list = self.generateRem(cpy, hop_list)
                        else:
```



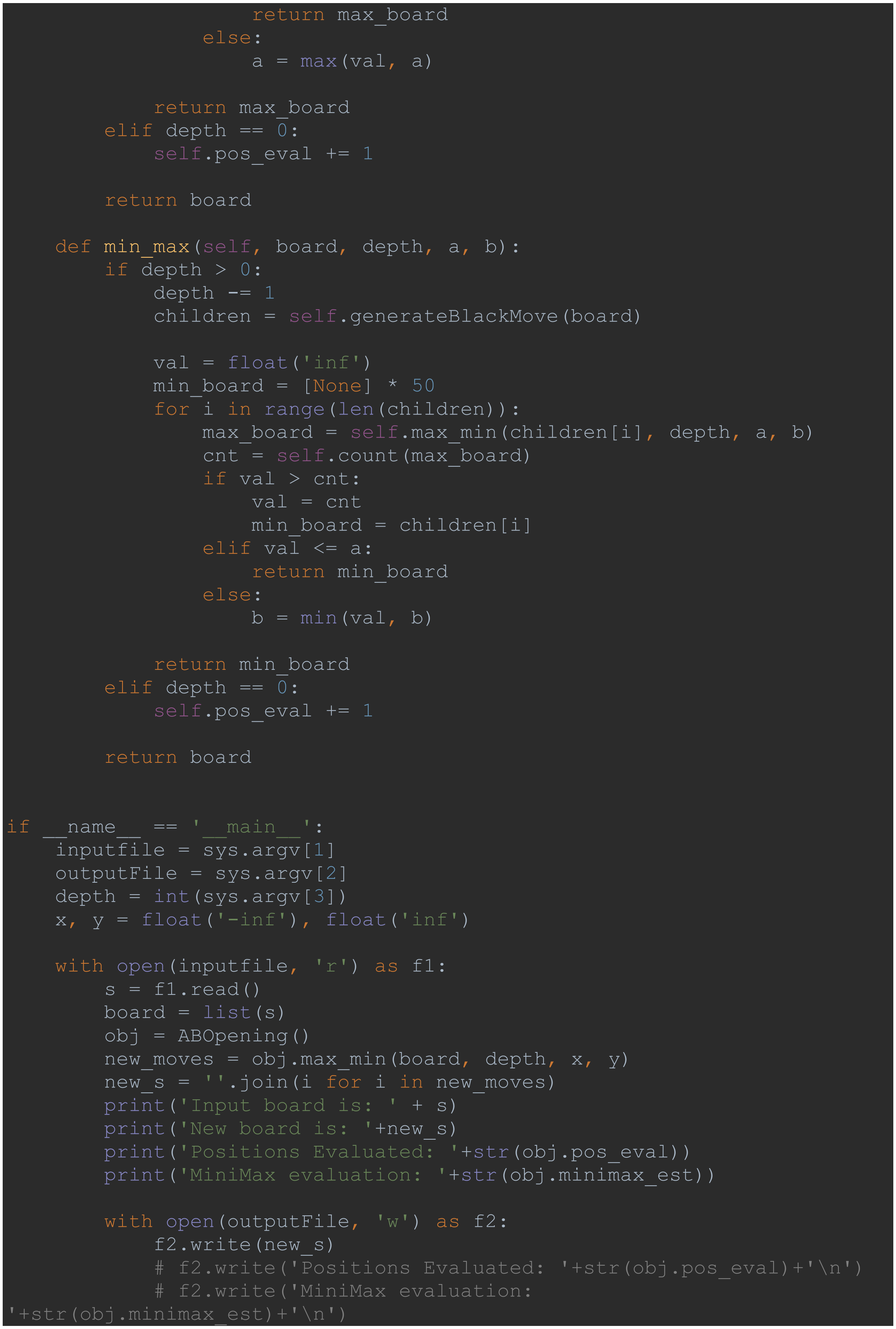
```
                            hop_list.append(cpy)

        return hop_list

    def generateMove(self, board):
        move_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                n_list = list(self.neighbours(i))
                for j in n_list:
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            move_list = self.generateRem(cpy,
move_list)
                        else:
                            move_list.append(cpy)

        return move_list

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.midgame_moves(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.static_estm(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.static_estm(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxGame()
        new_moves = obj.max_min(board, depth)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: ' + new_s)
        print('Positions Evaluated: ' + str(obj.pos_eval))
        print('MiniMax evaluation: ' + str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
```

3. ABOpening

```
import sys


class ABOpening:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c and board[16] == c):  # 3
```

```
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c
and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True
            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c or board[19] == c or board[3] == c:  #
0
                    return 5

            if loc == 1:
                if board[0] == c or board[2] == c or board[4] == c:  #
1
                    return 5

            if loc == 2:
                if board[1] == c or board[5] == c or board[12] == c:  #
2
                    return 5

            if loc == 3:
                if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
                    return 5

            if loc == 4:
                if board[3] == c or board[5] == c or board[1] == c:  #
4
```

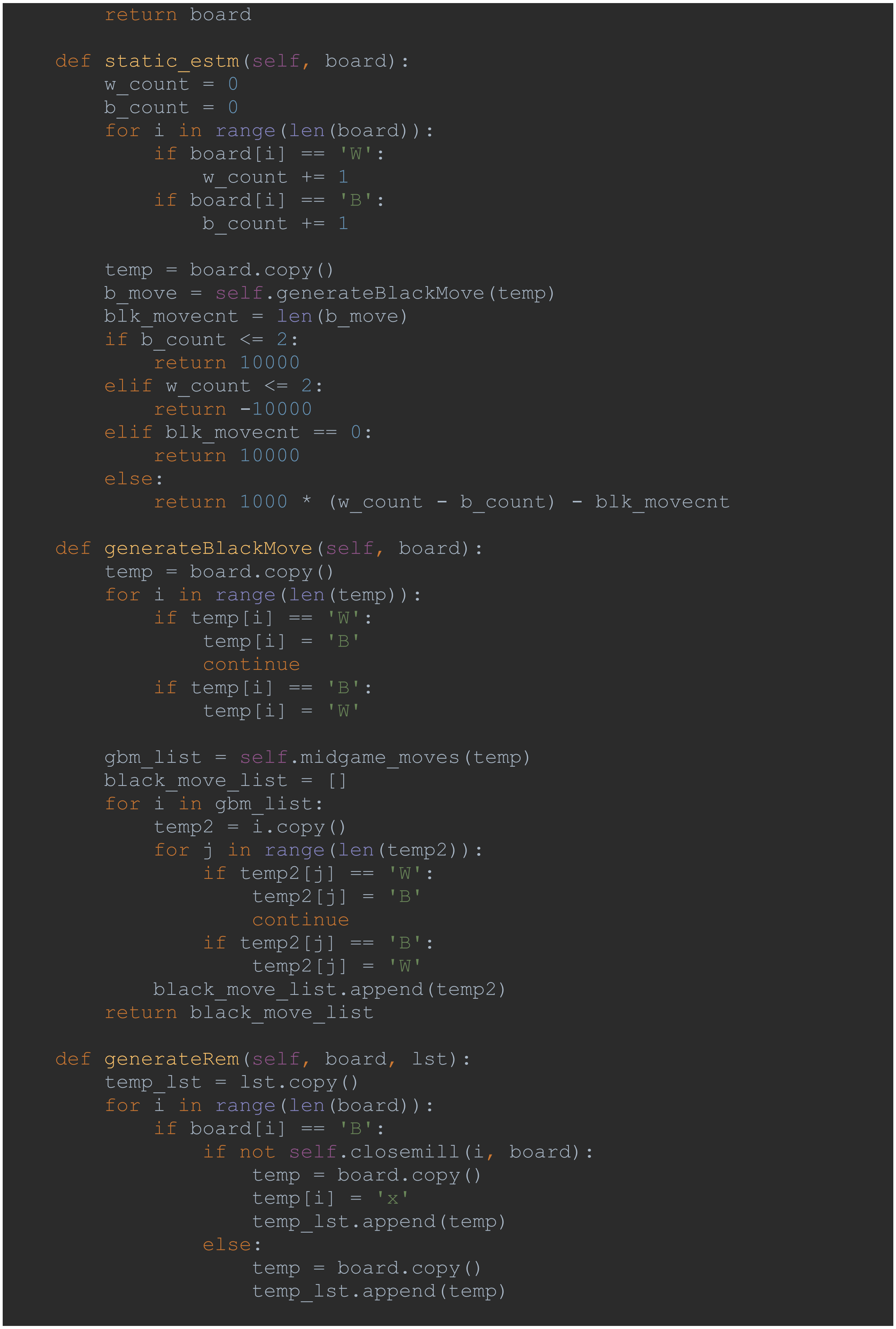
```
                    return 5

            if loc == 5:
                if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
                    return 5

            if loc == 6:
                if board[7] == c or board[9] == c or board[3] == c:  #
6
                    return 5

            if loc == 7:
                if board[6] == c or board[10] == c or board[5] == c:  #
7
                    return 5

            if loc == 8:
                if board[3] == c or board[9] == c or board[16] == c:  #
8
                    return 5

            if loc == 9:
                if board[8] == c or board[13] == c or board[6] == c:  #
9
                    return 5

            if loc == 10:
                if board[7] == c or board[15] == c or board[11] == c:
# 10
                    return 5

            if loc == 11:
                if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
                    return 5

            if loc == 12:
                if board[11] == c or board[2] == c or board[21] == c:
# 12
                    return 5

            if loc == 13:
                if board[14] == c or board[16] == c or board[9] == c:
# 13
                    return 5

            if loc == 14:
                if board[13] == c or board[15] == c or board[17] == c:
# 14
                    return 5

            if loc == 15:
                if board[14] == c or board[18] == c or board[10] == c:
# 15
                    return 5
            if loc == 16:
                if board[17] == c or board[19] == c or board[13] == c
or board[8] == c:  # 16
                    return 5

            if loc == 17:
                if board[16] == c or board[18] == c or board[14] == c
or board[20] == c:  # 17
                    return 5

            if loc == 18:
                if board[15] == c or board[17] == c or board[21] == c
or board[11] == c:  # 18
                    return 5

            if loc == 19:
                if board[20] == c or board[0] == c or board[16] == c:
# 19
                    return 5

            if loc == 20:
                if board[19] == c or board[21] == c or board[17] == c:
# 20
                    return 5

            if loc == 21:
                if board[20] == c or board[18] == c or board[12] == c:
# 21
                    return 5

        return False

    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def count(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1
        return w_count - b_count

    def generateADD(self, board):
        pos = []
        for i in range(len(board)):
            if board[i] == 'x':
                temp = board.copy()
                temp[i] = 'W'
                if self.closemill(i, temp):
                    pos = self.generateRem(temp, pos)
                else:
                    pos.append(temp)
        return pos

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.generateADD(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list

    def max_min(self, board, depth, a, b):

        if depth > 0:
            depth -= 1
            possible_pos = self.generateADD(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth, a, b)
                cnt = self.count(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
                elif val >= b:
```



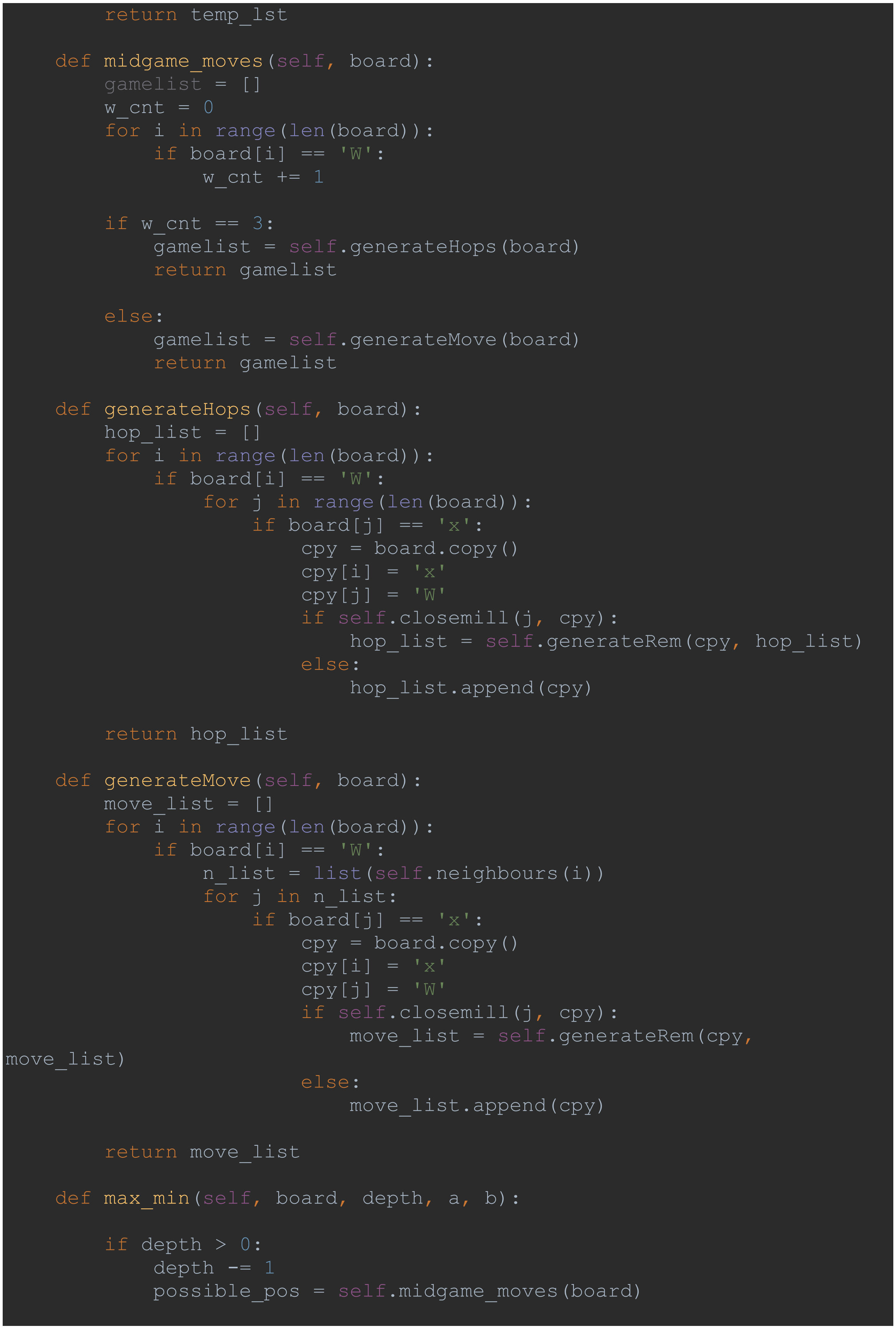
```
                    return max_board
                else:
                    a = max(val, a)

            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth, a, b):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth, a, b)
                cnt = self.count(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
                elif val <= a:
                    return min_board
                else:
                    b = min(val, b)

            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])
    x, y = float('-inf'), float('inf')

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = ABOpening()
        new_moves = obj.max_min(board, depth, x, y)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: '+new_s)
        print('Positions Evaluated: '+str(obj.pos_eval))
        print('MiniMax evaluation: '+str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation: 
'+str(obj.minimax_est)+'\n')
```

4. ABGame

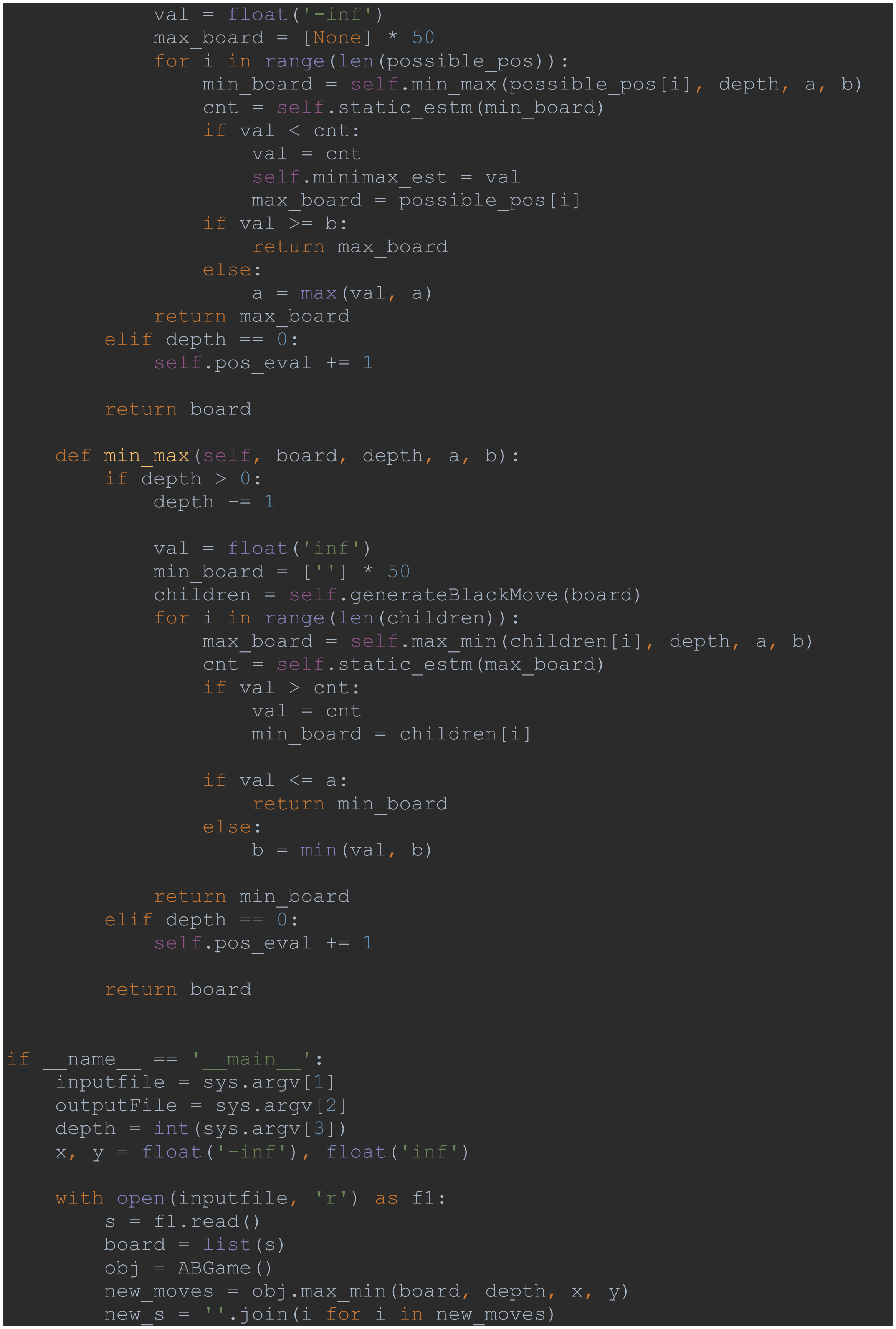
```
import sys


class ABGame:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c
and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c
and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c
and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True
            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc):
        if loc == 0:
            # if board[1] == c or board[19] == c or board[3] == c:  # 0
            return [1, 3, 19]

        if loc == 1:
            # if board[0] == c or board[2] == c or board[4] == c:  # 1
            return [0, 4, 2]

        if loc == 2:
            # if board[1] == c or board[5] == c or board[12] == c:  # 2
            return [1, 5, 12]

        if loc == 3:
            # if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
            return [0, 8, 4, 6]

        if loc == 4:
            # if board[3] == c or board[5] == c or board[1] == c:  # 4
            return [3, 5, 1]

        if loc == 5:
            # if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
            return [4, 7, 2, 11]

        if loc == 6:
            # if board[7] == c or board[9] == c or board[3] == c:  # 6
            return [7, 9, 3]

        if loc == 7:
            # if board[6] == c or board[10] == c or board[5] == c:  # 7
            return [6, 10, 5]

        if loc == 8:
            # if board[3] == c or board[9] == c or board[16] == c:  # 8
            return [3, 9, 16]

        if loc == 9:
            # if board[8] == c or board[13] == c or board[6] == c:  # 9
            return [8, 13, 6]

        if loc == 10:
            # if board[7] == c or board[15] == c or board[11] == c:  #
10
            return [7, 15, 11]

        if loc == 11:
            # if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
            return [10, 12, 5, 18]
        if loc == 12:
            # if board[11] == c or board[2] == c or board[21] == c:  #
12
            return [11, 2, 21]

        if loc == 13:
            # if board[14] == c or board[16] == c or board[9] == c:  #
13
            return [14, 16, 9]

        if loc == 14:
            # if board[13] == c or board[15] == c or board[17] == c:  #
14
            return [13, 15, 17]

        if loc == 15:
            # if board[14] == c or board[18] == c or board[10] == c:  #
15
            return [14, 18, 10]

        if loc == 16:
            # if board[17] == c or board[19] == c or board[13] == c or
board[8] == c:  # 16
            return [17, 19, 13, 8]

        if loc == 17:
            # if board[16] == c or board[18] == c or board[14] == c or
board[20] == c:  # 17
            return [16, 18, 14, 20]

        if loc == 18:
            # if board[15] == c or board[17] == c or board[21] == c or
board[11] == c:  # 18
            return [15, 17, 21, 11]

        if loc == 19:
            # if board[20] == c or board[0] == c or board[16] == c:  #
19
            return [20, 0, 16]

        if loc == 20:
            # if board[19] == c or board[21] == c or board[17] == c:  #
20
            return [19, 21, 17]

        if loc == 21:
            # if board[20] == c or board[18] == c or board[12] == c:  #
21
            return [20, 18, 12]

    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
```



```
        return board

    def static_estm(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1

        temp = board.copy()
        b_move = self.generateBlackMove(temp)
        blk_movecnt = len(b_move)
        if b_count <= 2:
            return 10000
        elif w_count <= 2:
            return -10000
        elif blk_movecnt == 0:
            return 10000
        else:
            return 1000 * (w_count - b_count) - blk_movecnt

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.midgame_moves(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)
```



```
        return temp_lst

    def midgame_moves(self, board):
        gamelist = []
        w_cnt = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_cnt += 1

        if w_cnt == 3:
            gamelist = self.generateHops(board)
            return gamelist

        else:
            gamelist = self.generateMove(board)
            return gamelist

    def generateHops(self, board):
        hop_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                for j in range(len(board)):
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            hop_list = self.generateRem(cpy, hop_list)
                        else:
                            hop_list.append(cpy)

        return hop_list

    def generateMove(self, board):
        move_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                n_list = list(self.neighbours(i))
                for j in n_list:
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            move_list = self.generateRem(cpy,
move_list)
                        else:
                            move_list.append(cpy)

        return move_list

    def max_min(self, board, depth, a, b):

        if depth > 0:
            depth -= 1
            possible_pos = self.midgame_moves(board)
```



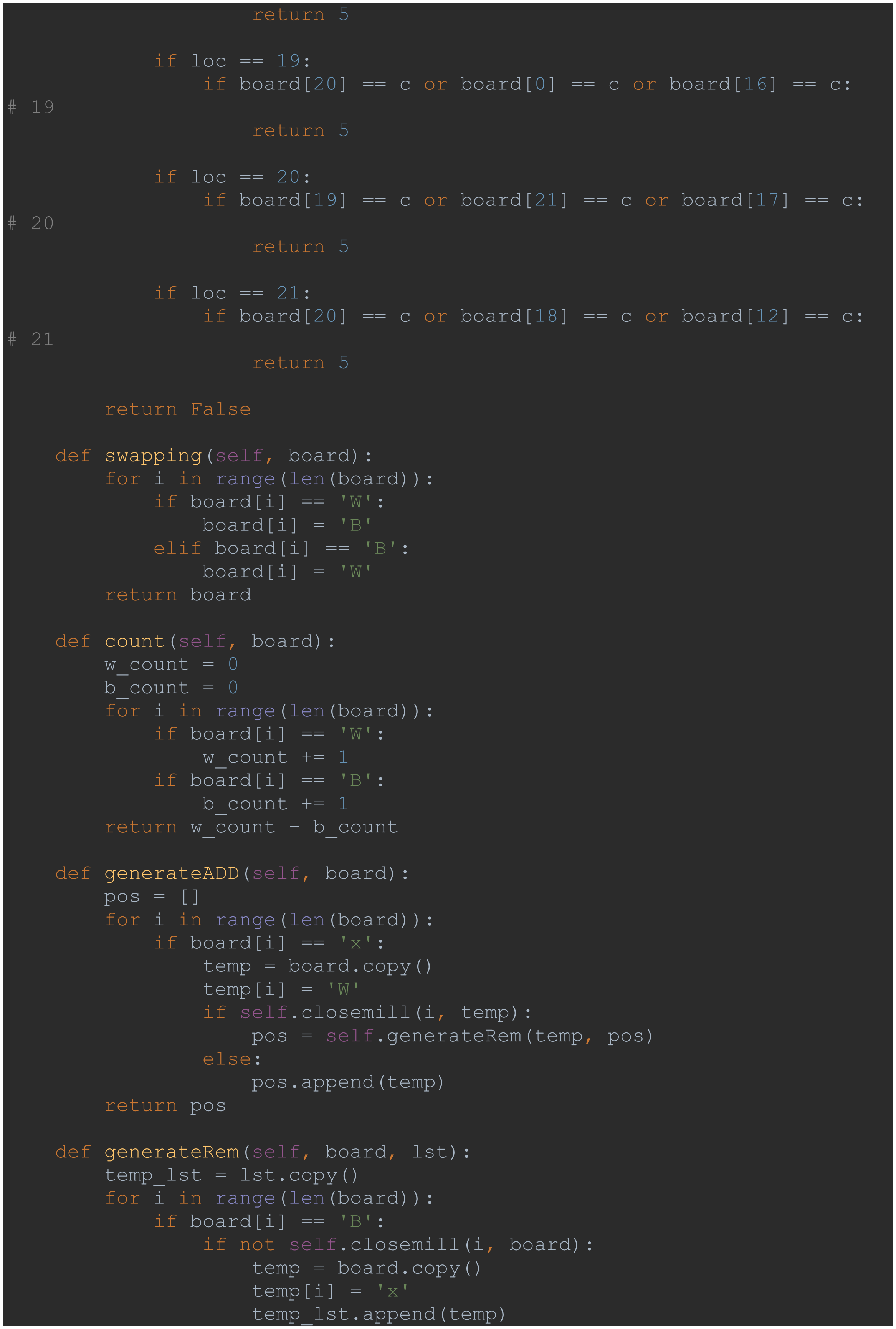
```
            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth, a, b)
                cnt = self.static_estm(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
                if val >= b:
                    return max_board
                else:
                    a = max(val, a)
            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth, a, b):
        if depth > 0:
            depth -= 1

            val = float('inf')
            min_board = [''] * 50
            children = self.generateBlackMove(board)
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth, a, b)
                cnt = self.static_estm(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]

                if val <= a:
                    return min_board
                else:
                    b = min(val, b)

            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])
    x, y = float('-inf'), float('inf')

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = ABGame()
        new_moves = obj.max_min(board, depth, x, y)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: ' + new_s)
        print('Positions Evaluated: ' + str(obj.pos_eval))
        print('MiniMax evaluation: ' + str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation: 
'+str(obj.minimax_est)+'\n')
```

5. MiniMaxOpeningBlack

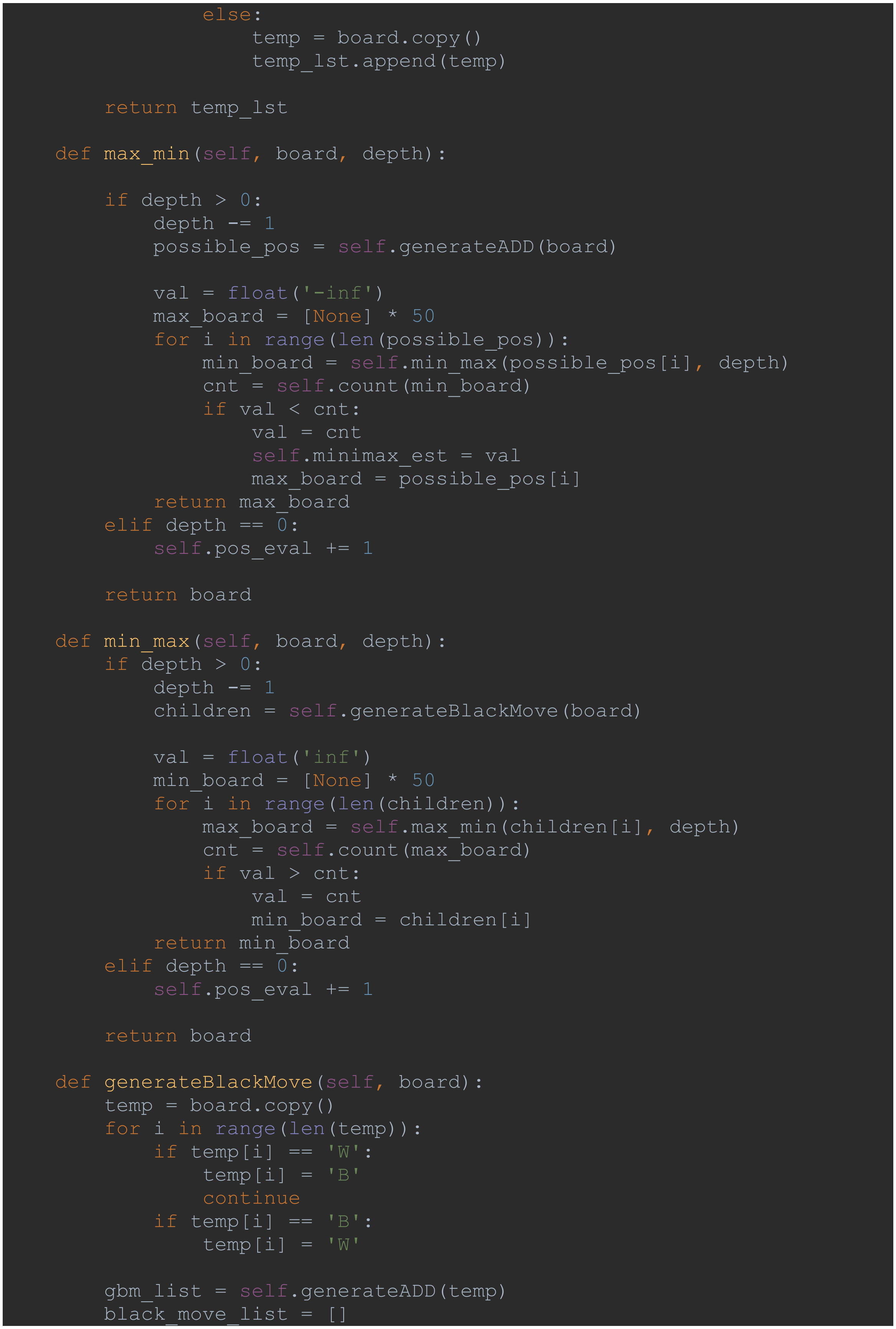
```
from io import StringIO
import sys


class MiniMaxOpeningBlack:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c 
and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c 
and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c 
and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
```

```
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
```

```
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c or board[19] == c or board[3] == c:  #
0
                    return 5

            if loc == 1:
                if board[0] == c or board[2] == c or board[4] == c:  #
1
                    return 5

            if loc == 2:
                if board[1] == c or board[5] == c or board[12] == c:  #
2
                    return 5

            if loc == 3:
                if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
                    return 5

            if loc == 4:
                if board[3] == c or board[5] == c or board[1] == c:  #
4
                    return 5

            if loc == 5:
                if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
                    return 5

            if loc == 6:
                if board[7] == c or board[9] == c or board[3] == c:  #
6
                    return 5

            if loc == 7:
                if board[6] == c or board[10] == c or board[5] == c:  # 
7
                    return 5

            if loc == 8:
                if board[3] == c or board[9] == c or board[16] == c:  # 
8
                    return 5

            if loc == 9:
                if board[8] == c or board[13] == c or board[6] == c:  # 
9
                    return 5

            if loc == 10:
                if board[7] == c or board[15] == c or board[11] == c:  
# 10
                    return 5

            if loc == 11:
                if board[10] == c or board[12] == c or board[5] == c or 
board[18] == c:  # 11
                    return 5

            if loc == 12:
                if board[11] == c or board[2] == c or board[21] == c:  
# 12
                    return 5

            if loc == 13:
                if board[14] == c or board[16] == c or board[9] == c:  
# 13
                    return 5

            if loc == 14:
                if board[13] == c or board[15] == c or board[17] == c:  
# 14
                    return 5

            if loc == 15:
                if board[14] == c or board[18] == c or board[10] == c:  
# 15
                    return 5

            if loc == 16:
                if board[17] == c or board[19] == c or board[13] == c 
or board[8] == c:  # 16
                    return 5

            if loc == 17:
                if board[16] == c or board[18] == c or board[14] == c 
or board[20] == c:  # 17
                    return 5

            if loc == 18:
                if board[15] == c or board[17] == c or board[21] == c 
or board[11] == c:  # 18
```



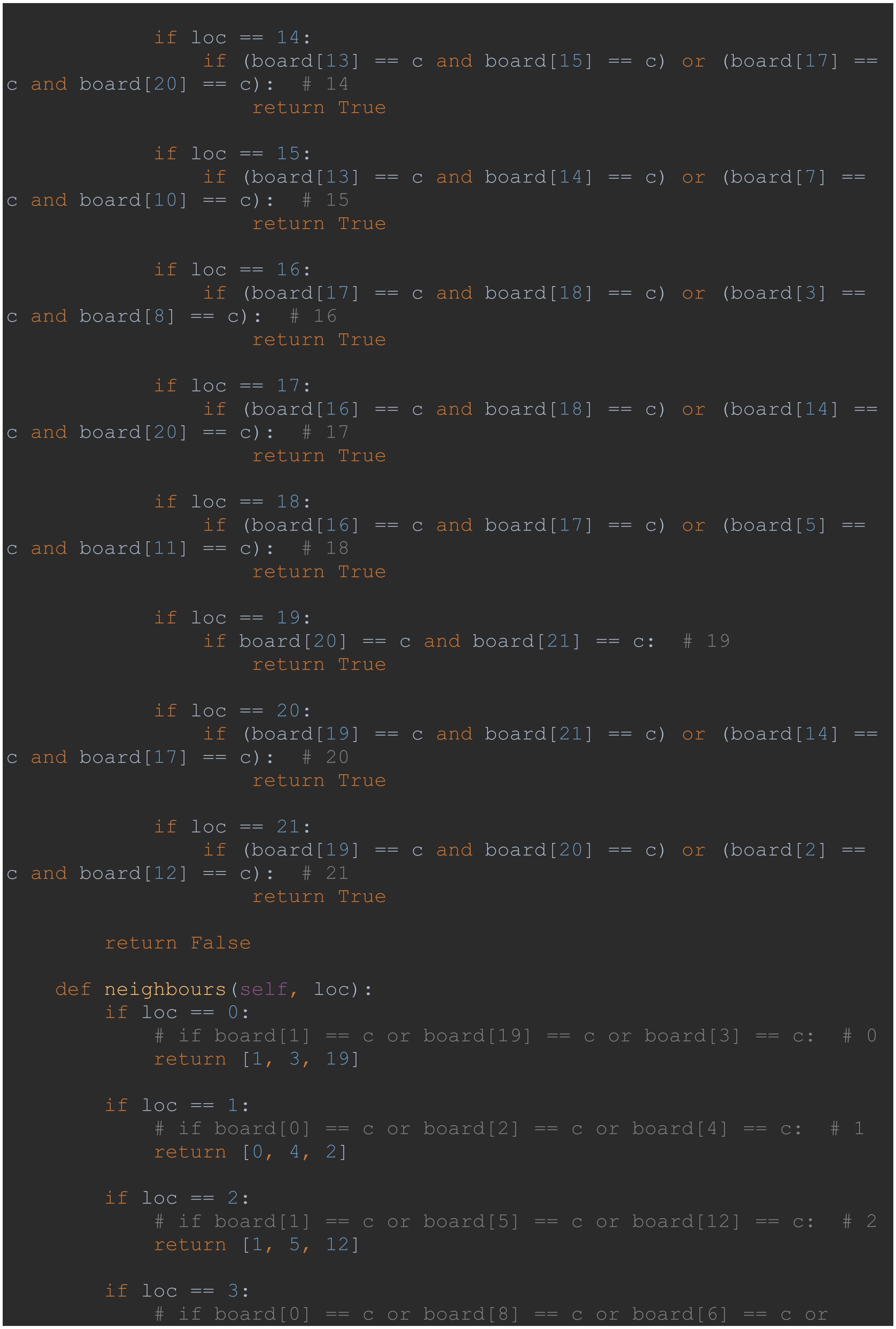
```
                    return 5

            if loc == 19:
                if board[20] == c or board[0] == c or board[16] == c:
# 19
                    return 5

            if loc == 20:
                if board[19] == c or board[21] == c or board[17] == c:
# 20
                    return 5

            if loc == 21:
                if board[20] == c or board[18] == c or board[12] == c:
# 21
                    return 5

        return False

    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def count(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1
        return w_count - b_count

    def generateADD(self, board):
        pos = []
        for i in range(len(board)):
            if board[i] == 'x':
                temp = board.copy()
                temp[i] = 'W'
                if self.closemill(i, temp):
                    pos = self.generateRem(temp, pos)
                else:
                    pos.append(temp)
        return pos

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
```



```
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.generateADD(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.count(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.count(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.generateADD(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxOpeningBlack()
        brd = obj.swapping(board)
        new_moves = obj.max_min(brd, depth)
        new_moves = obj.swapping(new_moves)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: '+new_s)
        print('Positions Evaluated: '+str(obj.pos_eval))
        print('MiniMax evaluation: '+str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation: 
'+str(obj.minimax_est)+'\n')
```

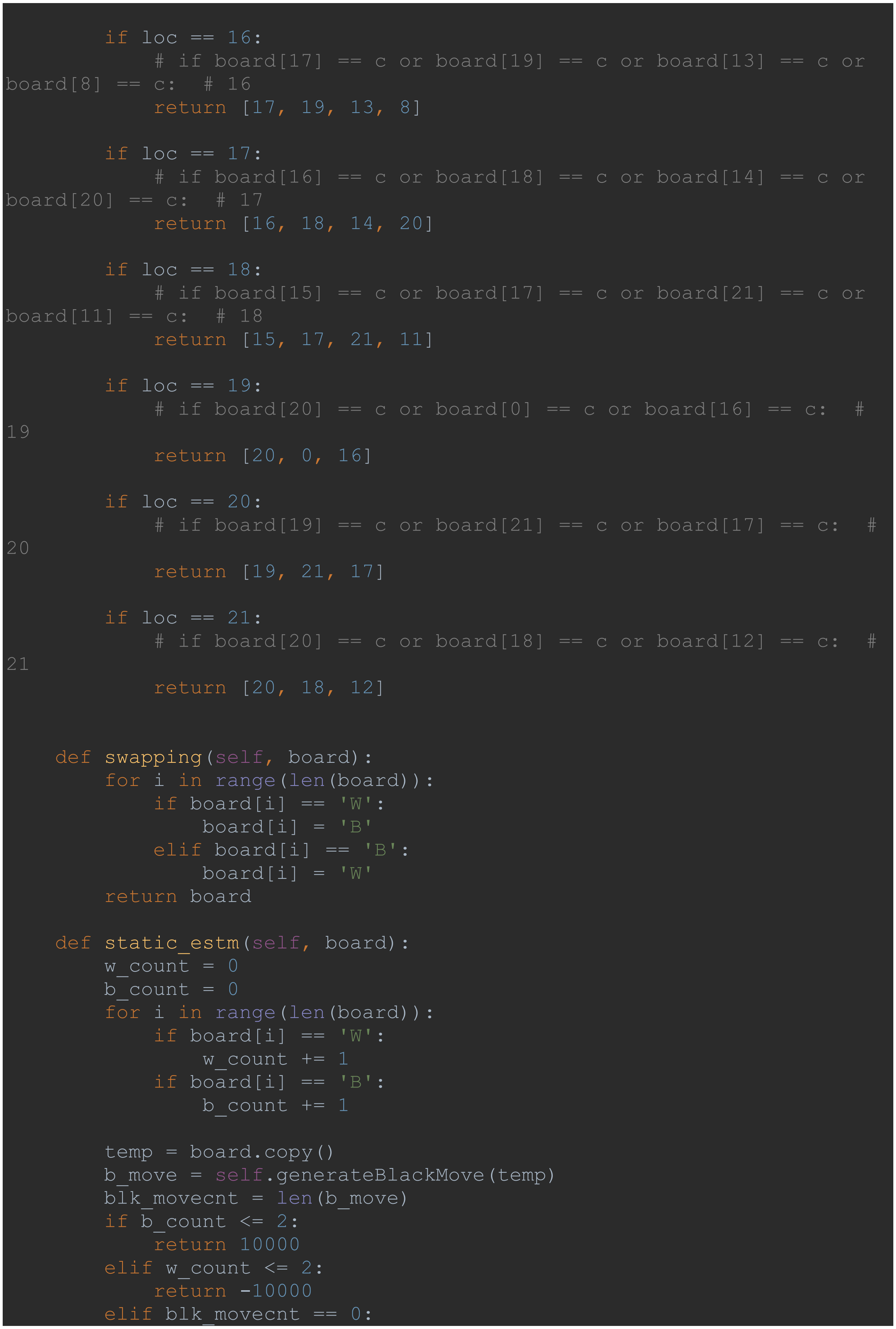
6. MiniMaxGameBlack

```
import sys


class MiniMaxGameBlack:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c and board[21] == c):  # 2
                    return True

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] == c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] == c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] == c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] == c and board[9] == c):  # 13
                    return True
```

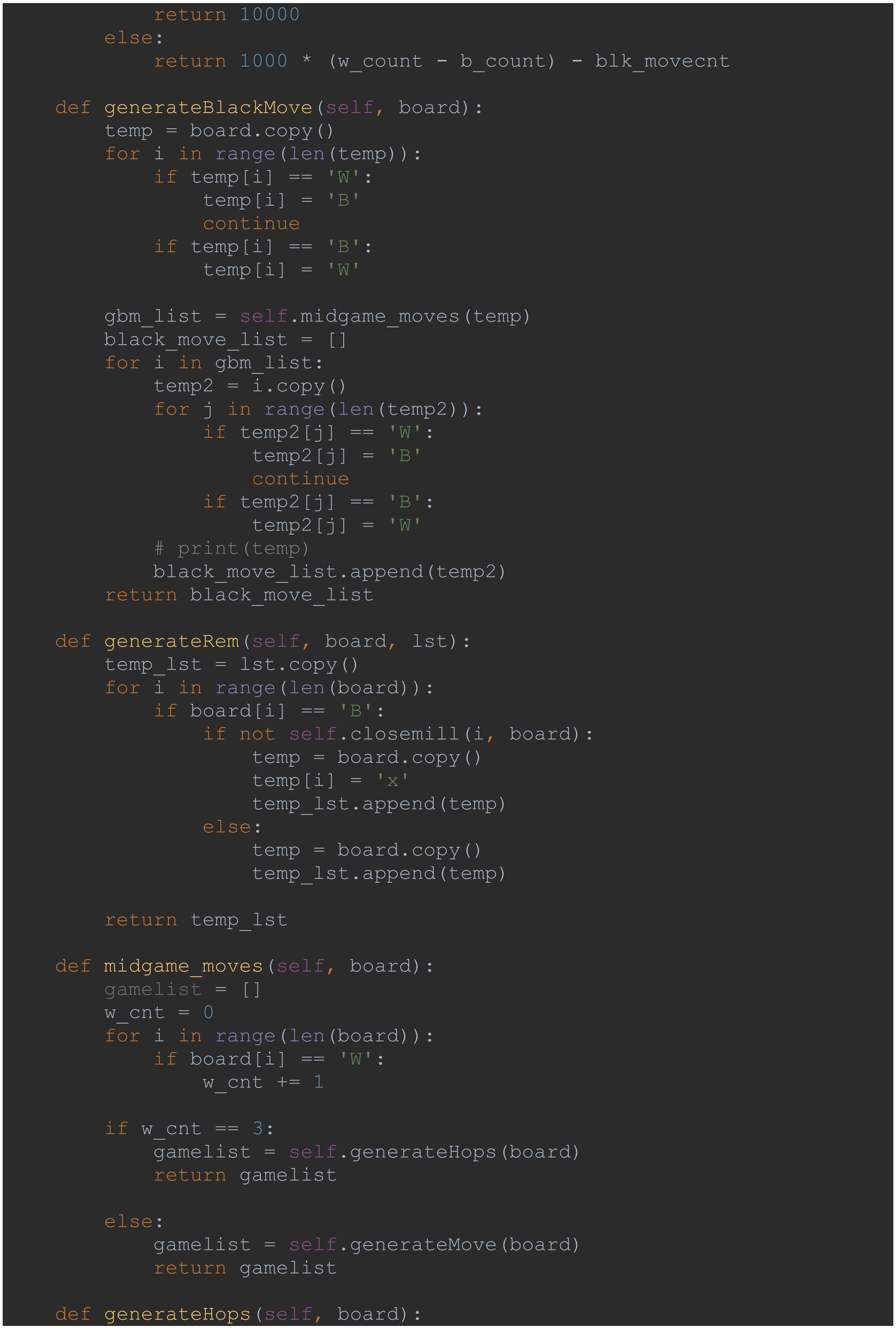


```
            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc):
        if loc == 0:
            # if board[1] == c or board[19] == c or board[3] == c:  # 0
            return [1, 3, 19]

        if loc == 1:
            # if board[0] == c or board[2] == c or board[4] == c:  # 1
            return [0, 4, 2]

        if loc == 2:
            # if board[1] == c or board[5] == c or board[12] == c:  # 2
            return [1, 5, 12]

        if loc == 3:
            # if board[0] == c or board[8] == c or board[6] == c or
```

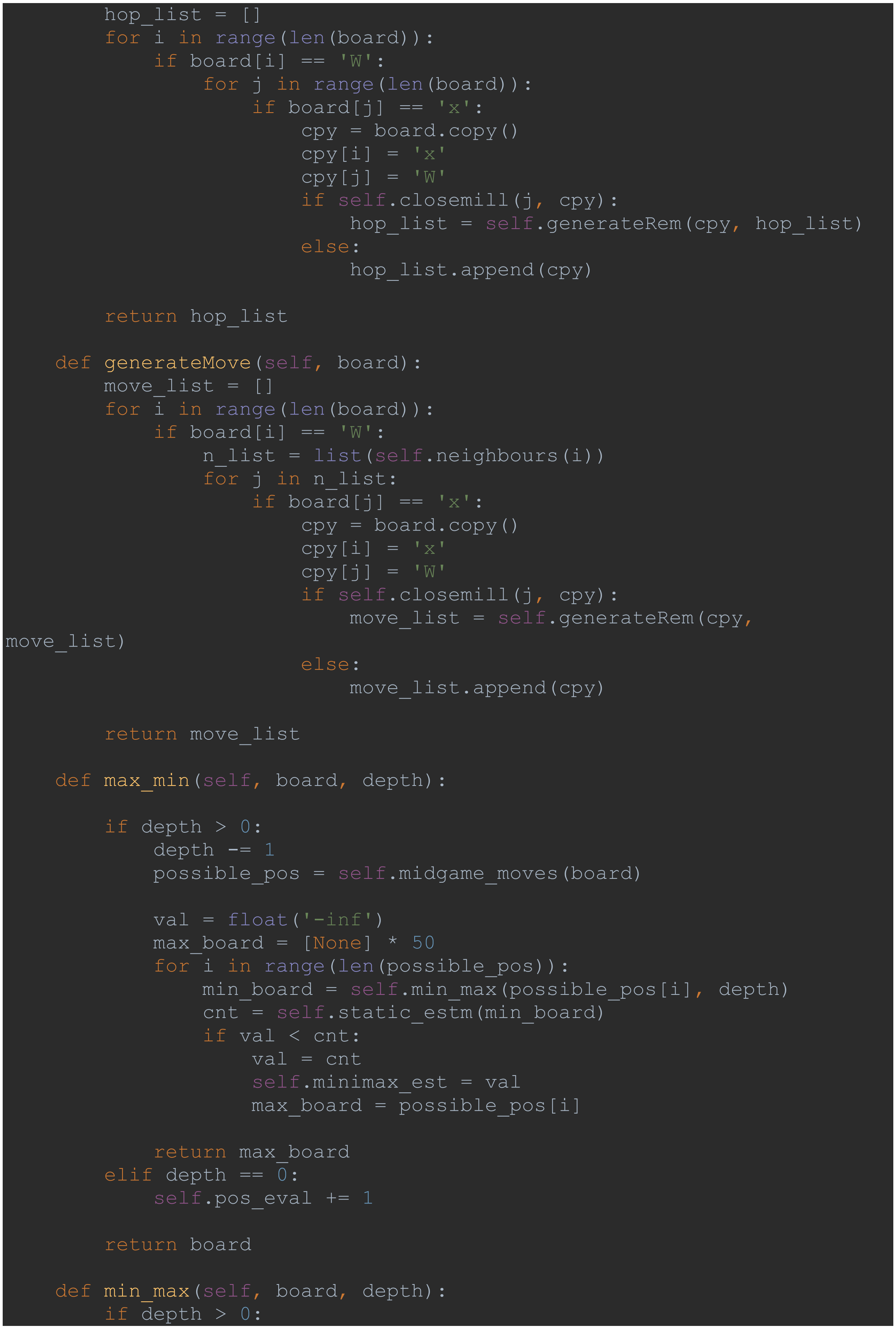
```
board[4] == c:  # 3
            return [0, 8, 4, 6]

        if loc == 4:
            # if board[3] == c or board[5] == c or board[1] == c:  # 4
            return [3, 5, 1]

        if loc == 5:
            # if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
            return [4, 7, 2, 11]

        if loc == 6:
            # if board[7] == c or board[9] == c or board[3] == c:  # 6
            return [7, 9, 3]

        if loc == 7:
            # if board[6] == c or board[10] == c or board[5] == c:  # 7
            return [6, 10, 5]

        if loc == 8:
            # if board[3] == c or board[9] == c or board[16] == c:  # 8
            return [3, 9, 16]

        if loc == 9:
            # if board[8] == c or board[13] == c or board[6] == c:  # 9
            return [8, 13, 6]

        if loc == 10:
            # if board[7] == c or board[15] == c or board[11] == c:  #
10

            return [7, 15, 11]

        if loc == 11:
            # if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
            return [10, 12, 5, 18]

        if loc == 12:
            # if board[11] == c or board[2] == c or board[21] == c:  #
12

            return [11, 2, 21]

        if loc == 13:
            # if board[14] == c or board[16] == c or board[9] == c:  #
13

            return [14, 16, 9]

        if loc == 14:
            # if board[13] == c or board[15] == c or board[17] == c:  #
14

            return [13, 15, 17]

        if loc == 15:
            # if board[14] == c or board[18] == c or board[10] == c:  #
15

            return [14, 18, 10]
```



```
        if loc == 16:
            # if board[17] == c or board[19] == c or board[13] == c or
board[8] == c:  # 16
            return [17, 19, 13, 8]

        if loc == 17:
            # if board[16] == c or board[18] == c or board[14] == c or
board[20] == c:  # 17
            return [16, 18, 14, 20]

        if loc == 18:
            # if board[15] == c or board[17] == c or board[21] == c or
board[11] == c:  # 18
            return [15, 17, 21, 11]

        if loc == 19:
            # if board[20] == c or board[0] == c or board[16] == c:  #
19
            return [20, 0, 16]

        if loc == 20:
            # if board[19] == c or board[21] == c or board[17] == c:  #
20
            return [19, 21, 17]

        if loc == 21:
            # if board[20] == c or board[18] == c or board[12] == c:  #
21
            return [20, 18, 12]


    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def static_estm(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1

        temp = board.copy()
        b_move = self.generateBlackMove(temp)
        blk_movecnt = len(b_move)
        if b_count <= 2:
            return 10000
        elif w_count <= 2:
            return -10000
        elif blk_movecnt == 0:
```



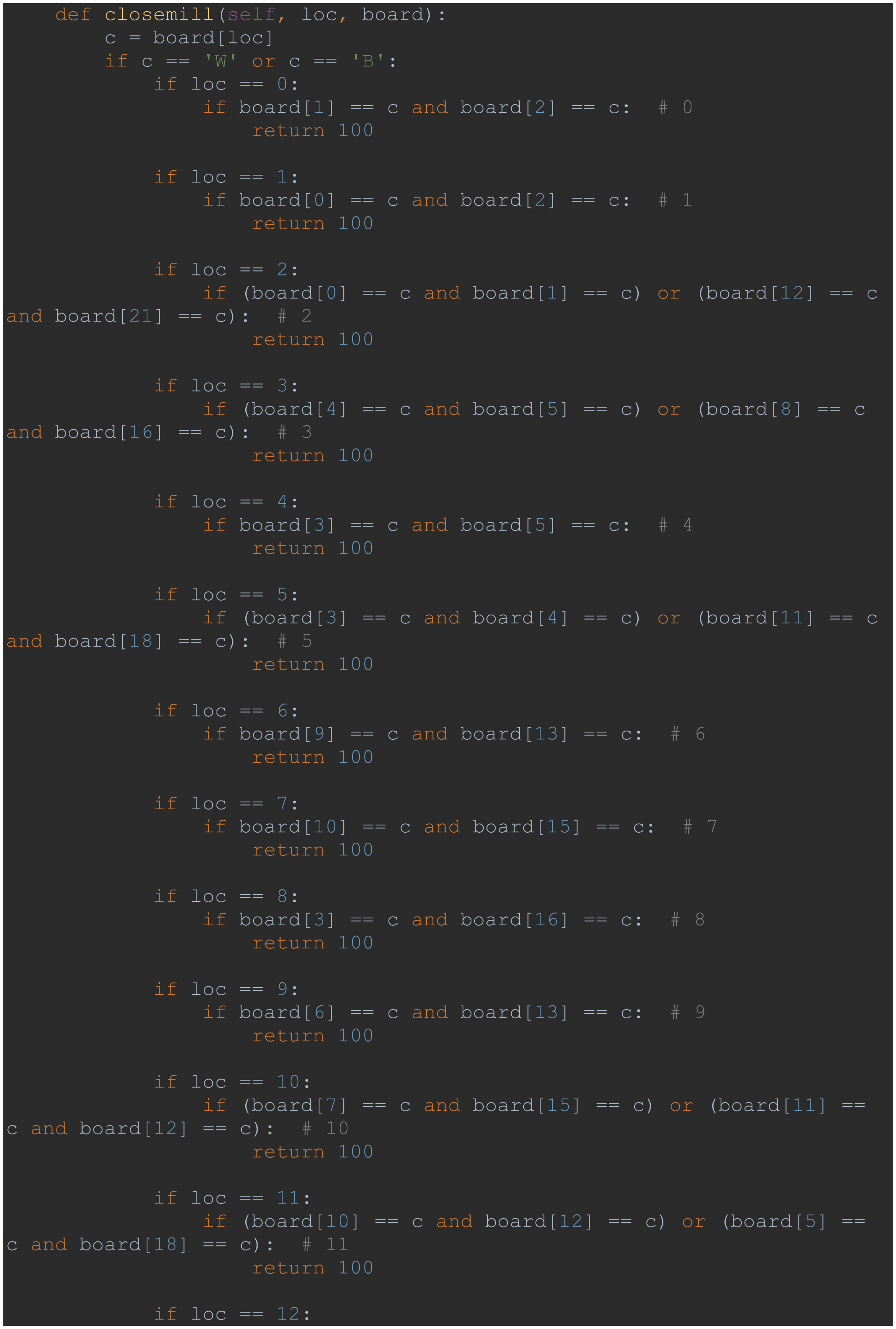
```
            return 10000
        else:
            return 1000 * (w_count - b_count) - blk_movecnt

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.midgame_moves(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            # print(temp)
            black_move_list.append(temp2)
        return black_move_list

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def midgame_moves(self, board):
        gamelist = []
        w_cnt = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_cnt += 1

        if w_cnt == 3:
            gamelist = self.generateHops(board)
            return gamelist

        else:
            gamelist = self.generateMove(board)
            return gamelist

    def generateHops(self, board):
```



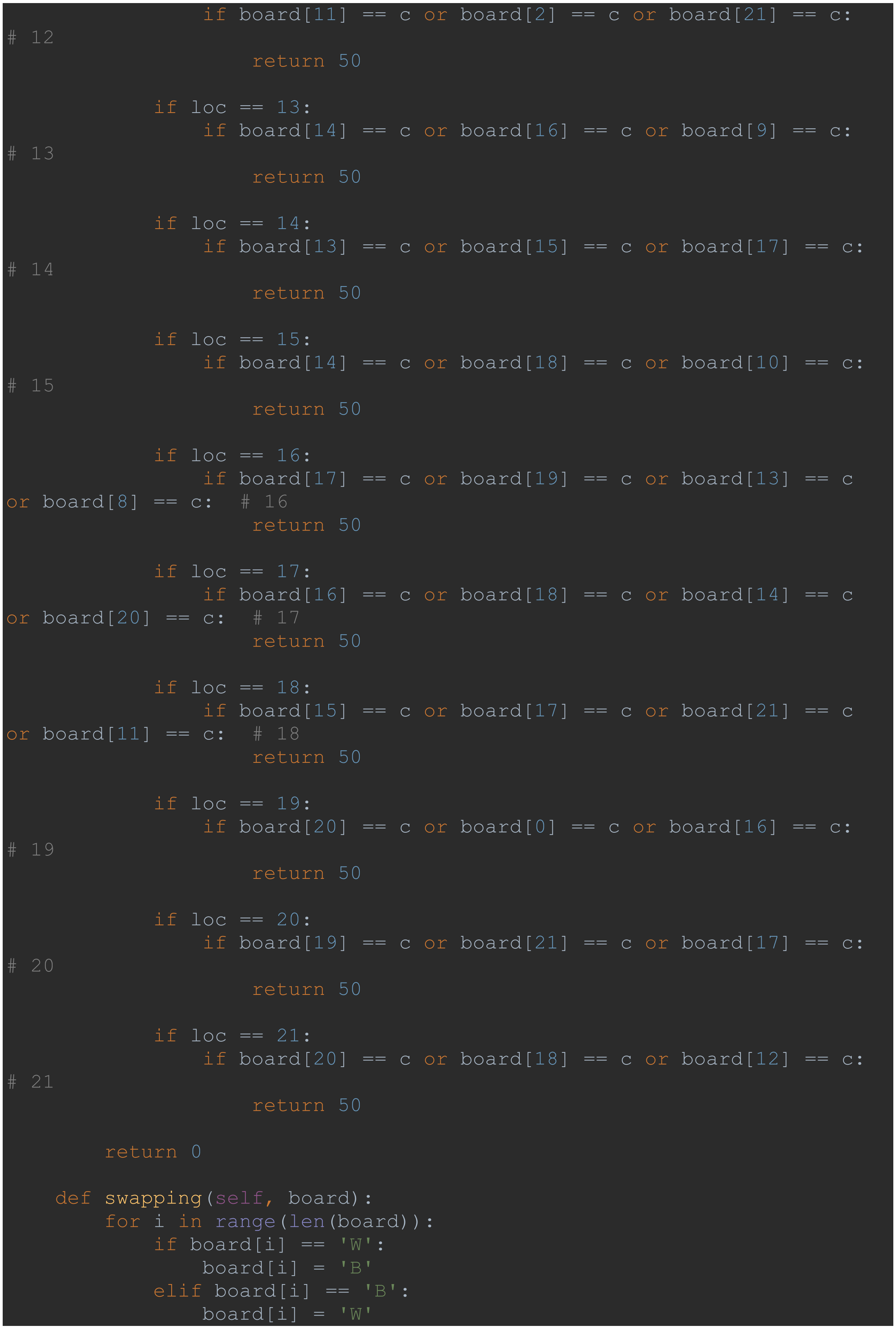
```
        hop_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                for j in range(len(board)):
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            hop_list = self.generateRem(cpy, hop_list)
                        else:
                            hop_list.append(cpy)

        return hop_list

    def generateMove(self, board):
        move_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                n_list = list(self.neighbours(i))
                for j in n_list:
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            move_list = self.generateRem(cpy,
move_list)
                        else:
                            move_list.append(cpy)

        return move_list

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.midgame_moves(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.static_estm(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]

            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.static_estm(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]

            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxGameBlack()
        brd = obj.swapping(board)
        new_moves = obj.max_min(board, depth)
        new_moves = obj.swapping(new_moves)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: ' + new_s)
        print('Positions Evaluated: ' + str(obj.pos_eval))
        print('MiniMax evaluation: ' + str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation: 
'+str(obj.minimax_est)+'\n')
```

7. MiniMaxOpeningImproved

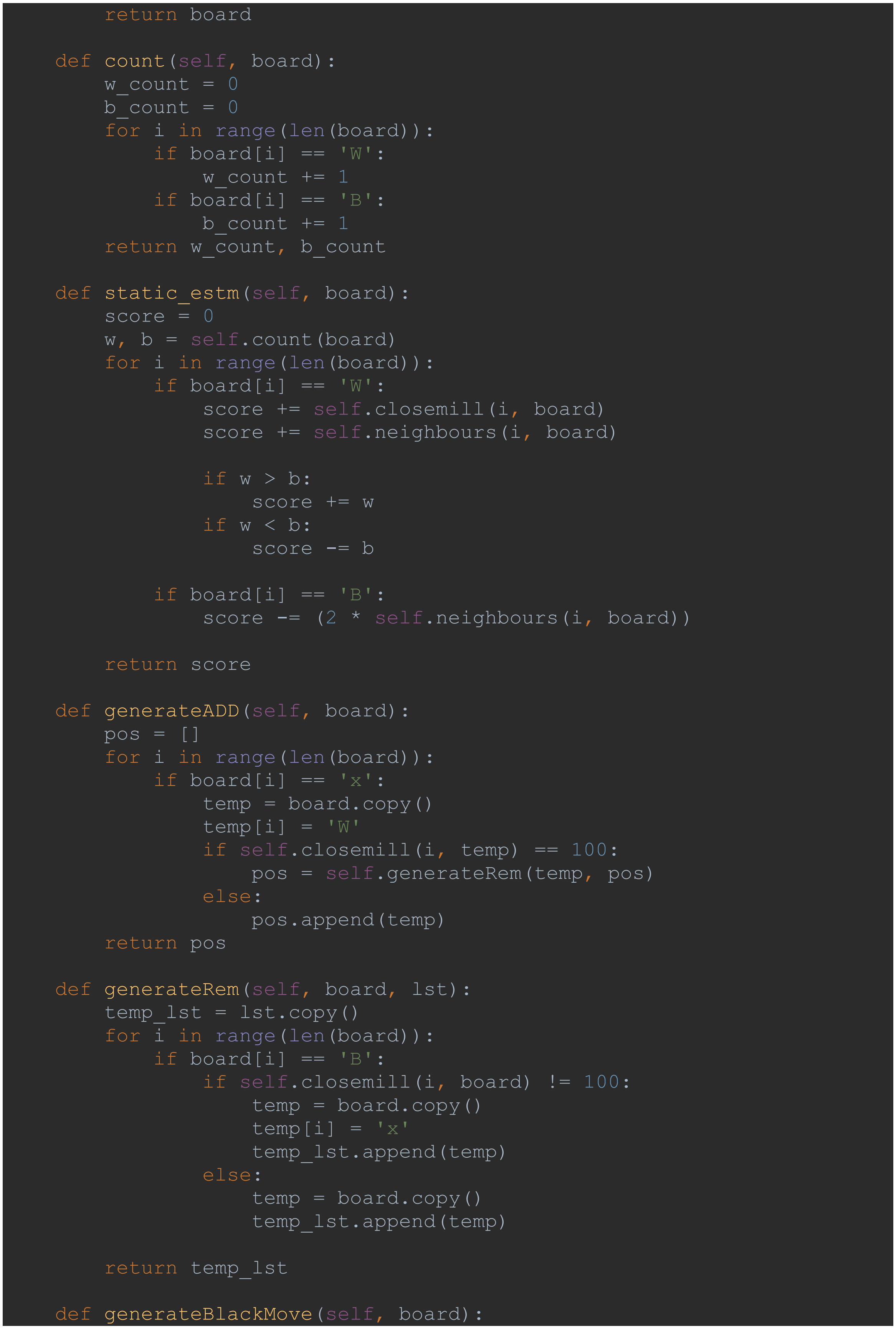
```
from io import StringIO
import sys


class MiniMaxOpening:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0
```



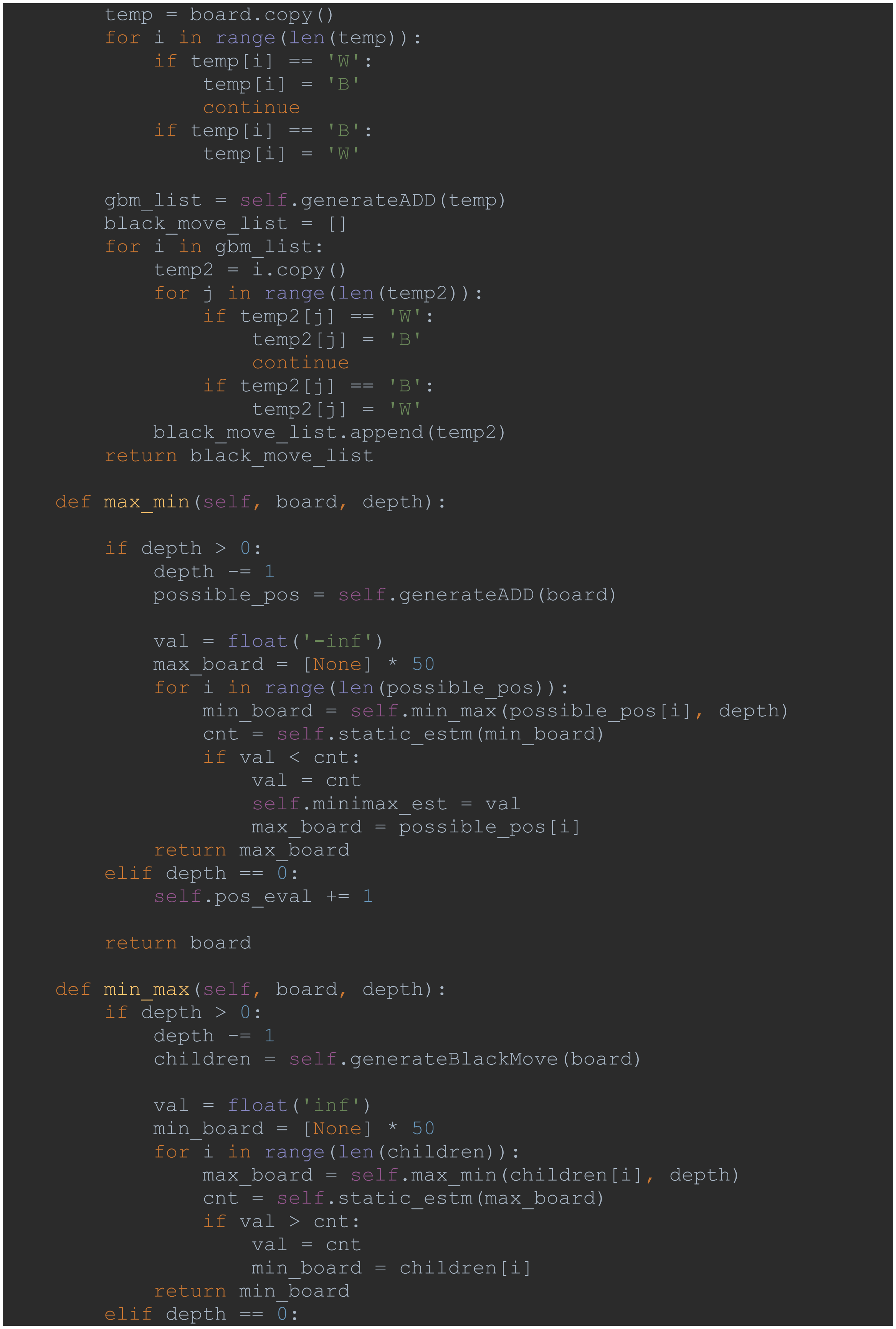
```
    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return 100

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return 100

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c
and board[21] == c):  # 2
                    return 100

            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c
and board[16] == c):  # 3
                    return 100

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return 100

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c
and board[18] == c):  # 5
                    return 100

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return 100

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return 100

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return 100

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return 100

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return 100

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return 100

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return 100

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return 100

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return 100

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return 100

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return 100

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return 100

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return 100

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return 100

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return 100

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return 100

        return 0

    def neighbours(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c or board[19] == c or board[3] == c:  #
0
                    return 50
            if loc == 1:
                if board[0] == c or board[2] == c or board[4] == c:  #
1
                    return 50

            if loc == 2:
                if board[1] == c or board[5] == c or board[12] == c:  #
2
                    return 50

            if loc == 3:
                if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
                    return 50

            if loc == 4:
                if board[3] == c or board[5] == c or board[1] == c:  #
4
                    return 50

            if loc == 5:
                if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
                    return 50

            if loc == 6:
                if board[7] == c or board[9] == c or board[3] == c:  #
6
                    return 50

            if loc == 7:
                if board[6] == c or board[10] == c or board[5] == c:  #
7
                    return 50

            if loc == 8:
                if board[3] == c or board[9] == c or board[16] == c:  #
8
                    return 50

            if loc == 9:
                if board[8] == c or board[13] == c or board[6] == c:  #
9
                    return 50

            if loc == 10:
                if board[7] == c or board[15] == c or board[11] == c:
# 10
                    return 50

            if loc == 11:
                if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
                    return 50

            if loc == 12:
```



```
                if board[11] == c or board[2] == c or board[21] == c:
# 12
                    return 50

            if loc == 13:
                if board[14] == c or board[16] == c or board[9] == c:
# 13
                    return 50

            if loc == 14:
                if board[13] == c or board[15] == c or board[17] == c:
# 14
                    return 50

            if loc == 15:
                if board[14] == c or board[18] == c or board[10] == c:
# 15
                    return 50

            if loc == 16:
                if board[17] == c or board[19] == c or board[13] == c
or board[8] == c:  # 16
                    return 50

            if loc == 17:
                if board[16] == c or board[18] == c or board[14] == c
or board[20] == c:  # 17
                    return 50

            if loc == 18:
                if board[15] == c or board[17] == c or board[21] == c
or board[11] == c:  # 18
                    return 50

            if loc == 19:
                if board[20] == c or board[0] == c or board[16] == c:
# 19
                    return 50

            if loc == 20:
                if board[19] == c or board[21] == c or board[17] == c:
# 20
                    return 50

            if loc == 21:
                if board[20] == c or board[18] == c or board[12] == c:
# 21
                    return 50

        return 0

    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
```



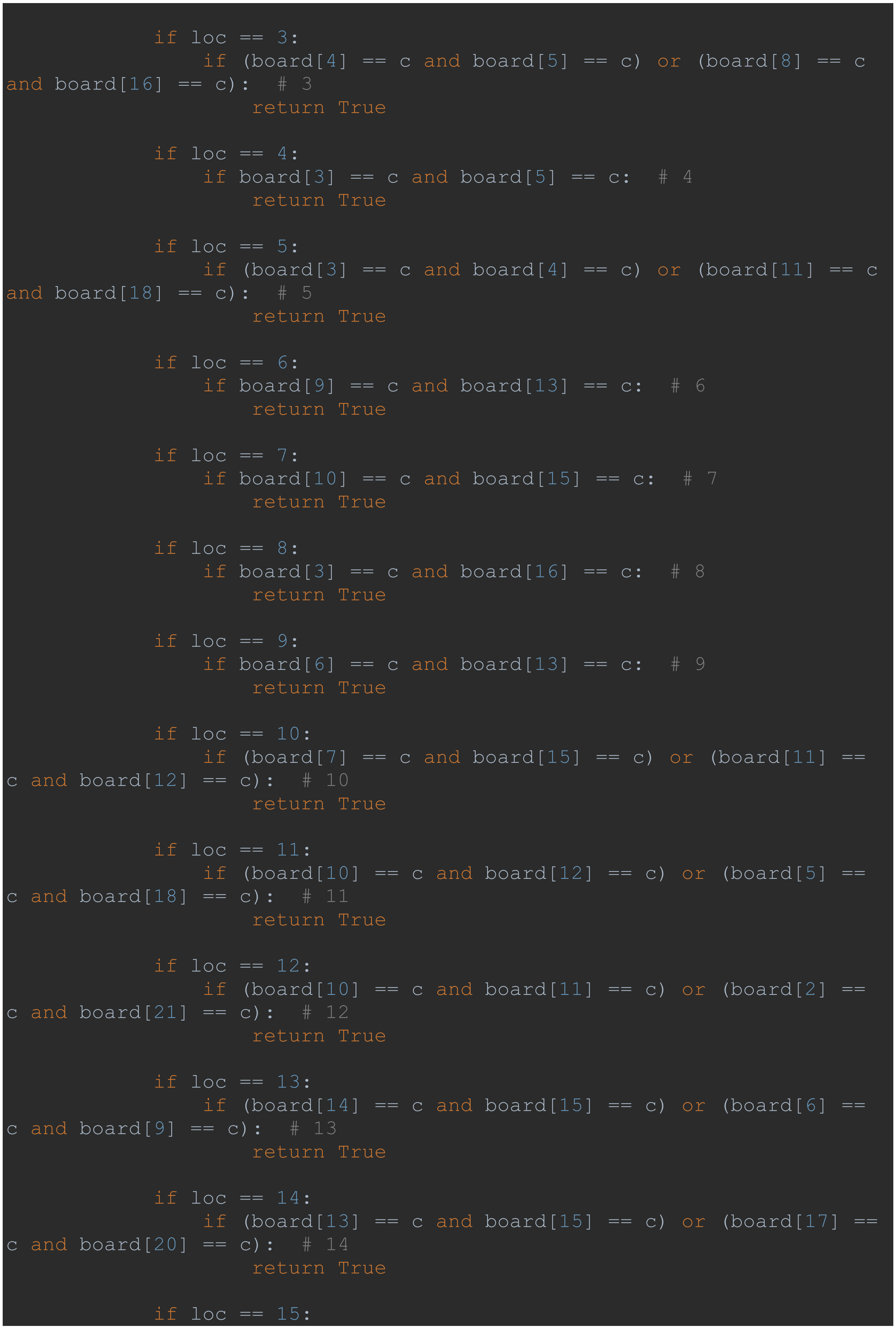
```
        return board

    def count(self, board):
        w_count = 0
        b_count = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_count += 1
            if board[i] == 'B':
                b_count += 1
        return w_count, b_count

    def static_estm(self, board):
        score = 0
        w, b = self.count(board)
        for i in range(len(board)):
            if board[i] == 'W':
                score += self.closemill(i, board)
                score += self.neighbours(i, board)

                if w > b:
                    score += w
                if w < b:
                    score -= b

            if board[i] == 'B':
                score -= (2 * self.neighbours(i, board))

        return score

    def generateADD(self, board):
        pos = []
        for i in range(len(board)):
            if board[i] == 'x':
                temp = board.copy()
                temp[i] = 'W'
                if self.closemill(i, temp) == 100:
                    pos = self.generateRem(temp, pos)
                else:
                    pos.append(temp)
        return pos

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if self.closemill(i, board) != 100:
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def generateBlackMove(self, board):
```



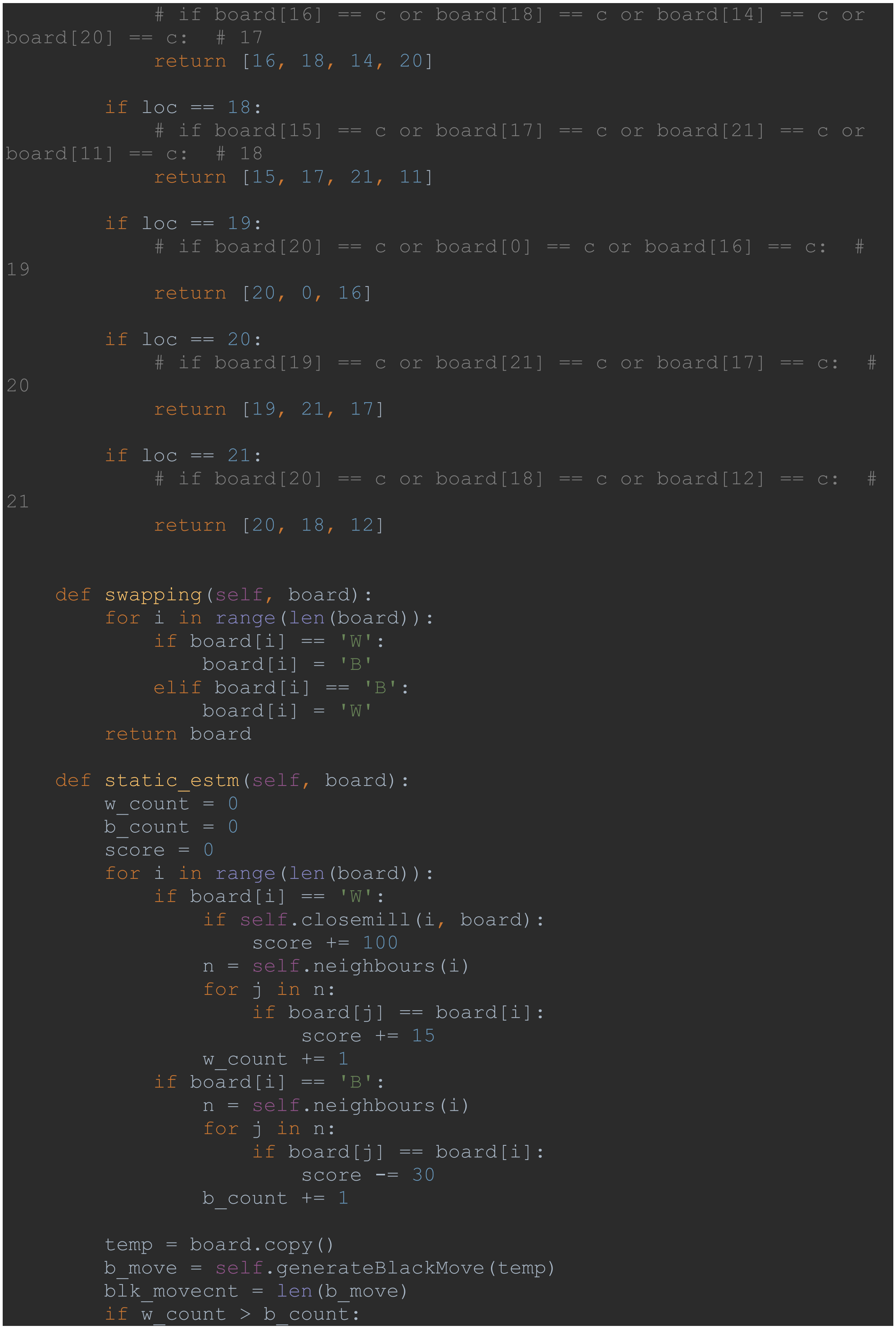
```
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.generateADD(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.generateADD(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.static_estm(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
            return max_board
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.static_estm(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxOpening()
        new_moves = obj.max_min(board, depth)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: '+new_s)
        print('Positions Evaluated: '+str(obj.pos_eval))
        print('MiniMax evaluation: '+str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
            # f2.write('Positions Evaluated: '+str(obj.pos_eval)+'\n')
            # f2.write('MiniMax evaluation:
'+str(obj.minimax_est)+'\n')
```

8. MiniMaxGameImproved

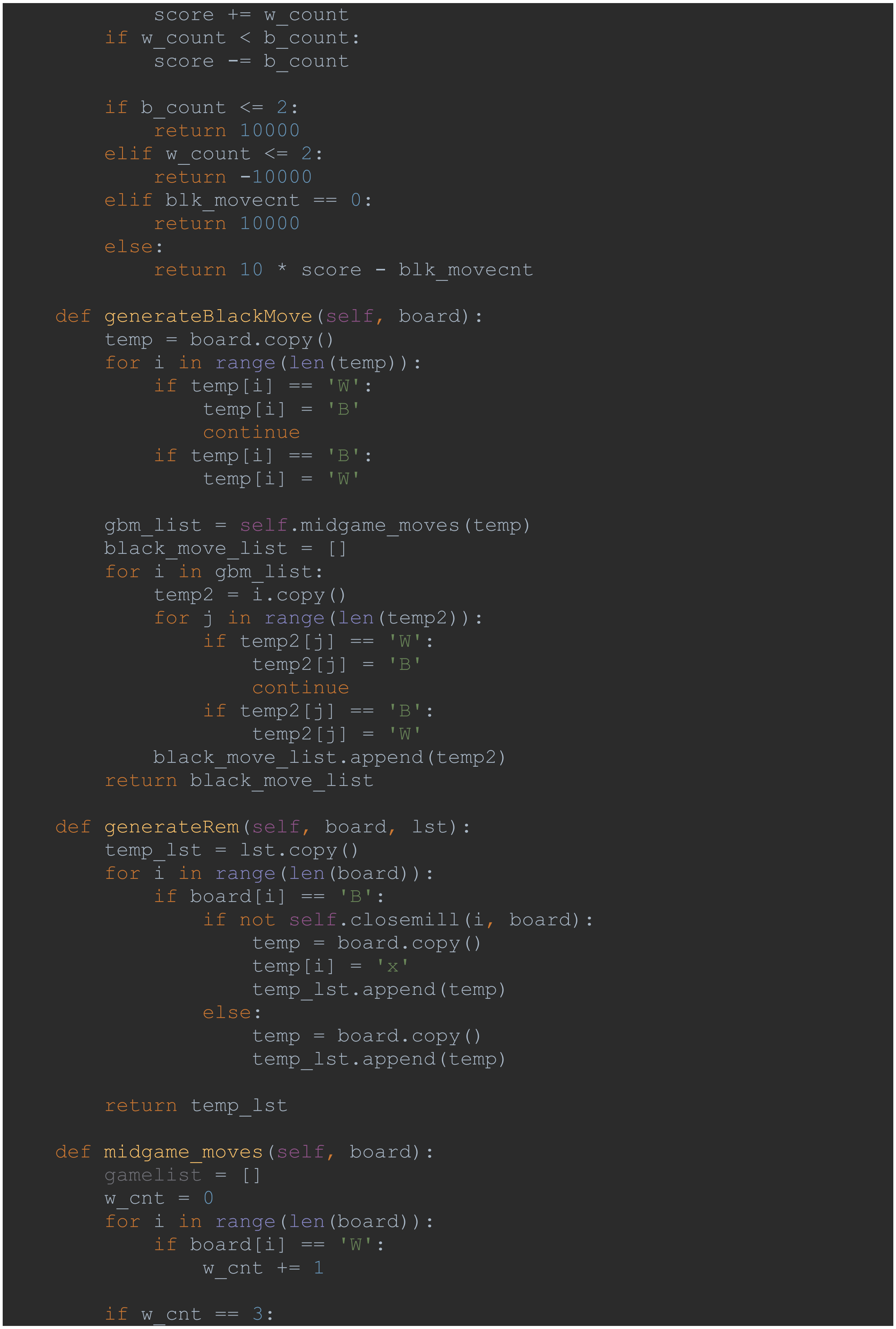
```
import sys


class MiniMaxGame:

    def __init__(self):
        self.minimax_est = None
        self.pos_eval = 0

    def closemill(self, loc, board):
        c = board[loc]
        if c == 'W' or c == 'B':
            if loc == 0:
                if board[1] == c and board[2] == c:  # 0
                    return True

            if loc == 1:
                if board[0] == c and board[2] == c:  # 1
                    return True

            if loc == 2:
                if (board[0] == c and board[1] == c) or (board[12] == c
and board[21] == c):  # 2
                    return True
```



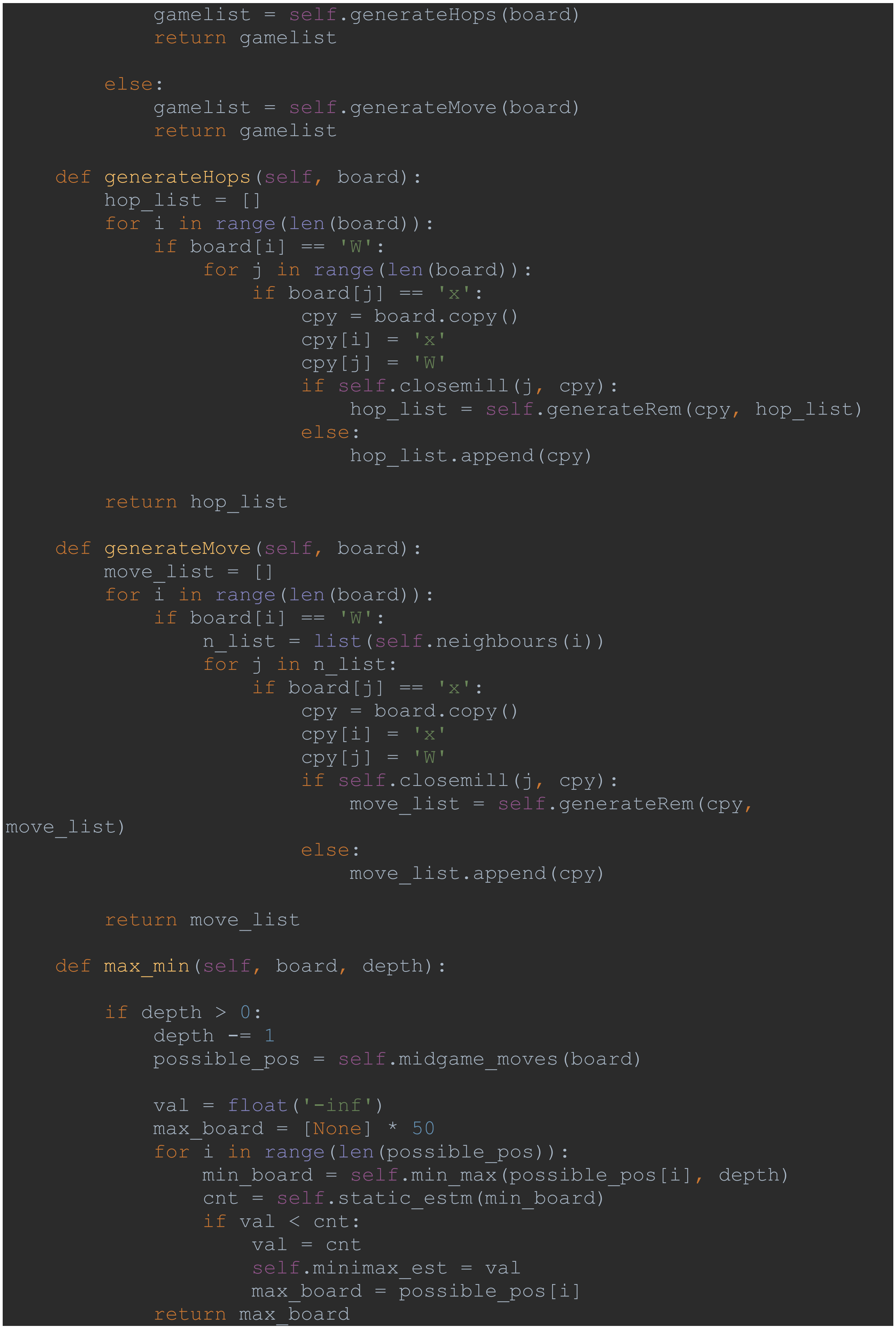
```
            if loc == 3:
                if (board[4] == c and board[5] == c) or (board[8] == c
and board[16] == c):  # 3
                    return True

            if loc == 4:
                if board[3] == c and board[5] == c:  # 4
                    return True

            if loc == 5:
                if (board[3] == c and board[4] == c) or (board[11] == c
and board[18] == c):  # 5
                    return True

            if loc == 6:
                if board[9] == c and board[13] == c:  # 6
                    return True

            if loc == 7:
                if board[10] == c and board[15] == c:  # 7
                    return True

            if loc == 8:
                if board[3] == c and board[16] == c:  # 8
                    return True

            if loc == 9:
                if board[6] == c and board[13] == c:  # 9
                    return True

            if loc == 10:
                if (board[7] == c and board[15] == c) or (board[11] ==
c and board[12] == c):  # 10
                    return True

            if loc == 11:
                if (board[10] == c and board[12] == c) or (board[5] ==
c and board[18] == c):  # 11
                    return True

            if loc == 12:
                if (board[10] == c and board[11] == c) or (board[2] ==
c and board[21] == c):  # 12
                    return True

            if loc == 13:
                if (board[14] == c and board[15] == c) or (board[6] ==
c and board[9] == c):  # 13
                    return True

            if loc == 14:
                if (board[13] == c and board[15] == c) or (board[17] ==
c and board[20] == c):  # 14
                    return True

            if loc == 15:
                if (board[13] == c and board[14] == c) or (board[7] ==
c and board[10] == c):  # 15
                    return True

            if loc == 16:
                if (board[17] == c and board[18] == c) or (board[3] ==
c and board[8] == c):  # 16
                    return True

            if loc == 17:
                if (board[16] == c and board[18] == c) or (board[14] ==
c and board[20] == c):  # 17
                    return True

            if loc == 18:
                if (board[16] == c and board[17] == c) or (board[5] ==
c and board[11] == c):  # 18
                    return True

            if loc == 19:
                if board[20] == c and board[21] == c:  # 19
                    return True

            if loc == 20:
                if (board[19] == c and board[21] == c) or (board[14] ==
c and board[17] == c):  # 20
                    return True

            if loc == 21:
                if (board[19] == c and board[20] == c) or (board[2] ==
c and board[12] == c):  # 21
                    return True

        return False

    def neighbours(self, loc):
        if loc == 0:
            # if board[1] == c or board[19] == c or board[3] == c:  # 0
            return [1, 3, 19]

        if loc == 1:
            # if board[0] == c or board[2] == c or board[4] == c:  # 1
            return [0, 4, 2]

        if loc == 2:
            # if board[1] == c or board[5] == c or board[12] == c:  # 2
            return [1, 5, 12]

        if loc == 3:
            # if board[0] == c or board[8] == c or board[6] == c or
board[4] == c:  # 3
            return [0, 8, 4, 6]

        if loc == 4:
            # if board[3] == c or board[5] == c or board[1] == c:  # 4
            return [3, 5, 1]
        if loc == 5:
            # if board[4] == c or board[7] == c or board[2] == c or
board[11] == c:  # 5
            return [4, 7, 2, 11]

        if loc == 6:
            # if board[7] == c or board[9] == c or board[3] == c:  # 6
            return [7, 9, 3]

        if loc == 7:
            # if board[6] == c or board[10] == c or board[5] == c:  # 7
            return [6, 10, 5]

        if loc == 8:
            # if board[3] == c or board[9] == c or board[16] == c:  # 8
            return [3, 9, 16]

        if loc == 9:
            # if board[8] == c or board[13] == c or board[6] == c:  # 9
            return [8, 13, 6]

        if loc == 10:
            # if board[7] == c or board[15] == c or board[11] == c:  #
10

            return [7, 15, 11]

        if loc == 11:
            # if board[10] == c or board[12] == c or board[5] == c or
board[18] == c:  # 11
            return [10, 12, 5, 18]

        if loc == 12:
            # if board[11] == c or board[2] == c or board[21] == c:  #
12

            return [11, 2, 21]

        if loc == 13:
            # if board[14] == c or board[16] == c or board[9] == c:  #
13

            return [14, 16, 9]

        if loc == 14:
            # if board[13] == c or board[15] == c or board[17] == c:  #
14

            return [13, 15, 17]

        if loc == 15:
            # if board[14] == c or board[18] == c or board[10] == c:  #
15

            return [14, 18, 10]

        if loc == 16:
            # if board[17] == c or board[19] == c or board[13] == c or
board[8] == c:  # 16
            return [17, 19, 13, 8]

        if loc == 17:
```



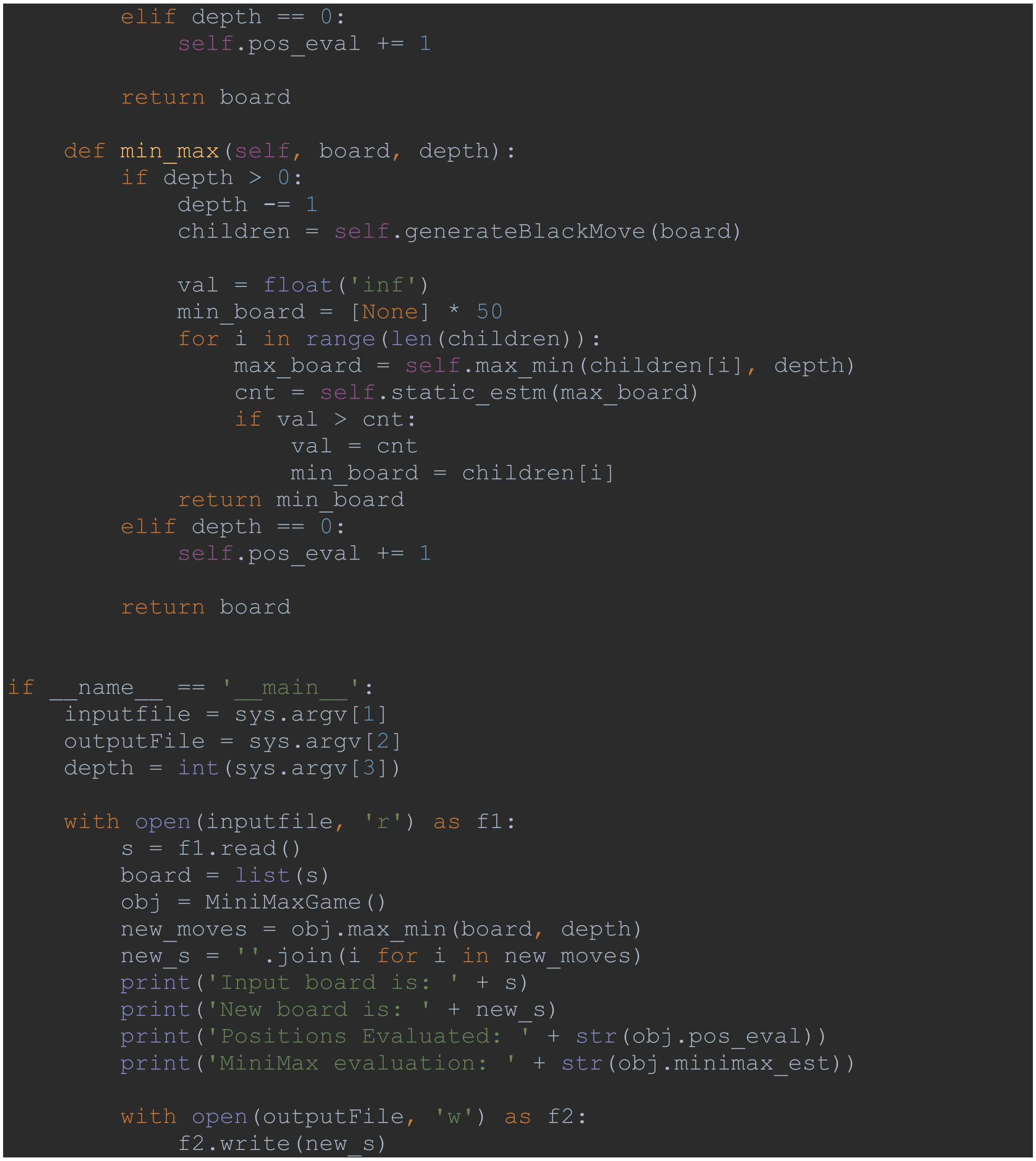
```
            # if board[16] == c or board[18] == c or board[14] == c or
board[20] == c:  # 17
            return [16, 18, 14, 20]

        if loc == 18:
            # if board[15] == c or board[17] == c or board[21] == c or
board[11] == c:  # 18
            return [15, 17, 21, 11]

        if loc == 19:
            # if board[20] == c or board[0] == c or board[16] == c:  #
19
            return [20, 0, 16]

        if loc == 20:
            # if board[19] == c or board[21] == c or board[17] == c:  #
20
            return [19, 21, 17]

        if loc == 21:
            # if board[20] == c or board[18] == c or board[12] == c:  #
21
            return [20, 18, 12]


    def swapping(self, board):
        for i in range(len(board)):
            if board[i] == 'W':
                board[i] = 'B'
            elif board[i] == 'B':
                board[i] = 'W'
        return board

    def static_estm(self, board):
        w_count = 0
        b_count = 0
        score = 0
        for i in range(len(board)):
            if board[i] == 'W':
                if self.closemill(i, board):
                    score += 100
                n = self.neighbours(i)
                for j in n:
                    if board[j] == board[i]:
                        score += 15
                w_count += 1
            if board[i] == 'B':
                n = self.neighbours(i)
                for j in n:
                    if board[j] == board[i]:
                        score -= 30
                b_count += 1

        temp = board.copy()
        b_move = self.generateBlackMove(temp)
        blk_movecnt = len(b_move)
        if w_count > b_count:
```



```
            score += w_count
        if w_count < b_count:
            score -= b_count

        if b_count <= 2:
            return 10000
        elif w_count <= 2:
            return -10000
        elif blk_movecnt == 0:
            return 10000
        else:
            return 10 * score - blk_movecnt

    def generateBlackMove(self, board):
        temp = board.copy()
        for i in range(len(temp)):
            if temp[i] == 'W':
                temp[i] = 'B'
                continue
            if temp[i] == 'B':
                temp[i] = 'W'

        gbm_list = self.midgame_moves(temp)
        black_move_list = []
        for i in gbm_list:
            temp2 = i.copy()
            for j in range(len(temp2)):
                if temp2[j] == 'W':
                    temp2[j] = 'B'
                    continue
                if temp2[j] == 'B':
                    temp2[j] = 'W'
            black_move_list.append(temp2)
        return black_move_list

    def generateRem(self, board, lst):
        temp_lst = lst.copy()
        for i in range(len(board)):
            if board[i] == 'B':
                if not self.closemill(i, board):
                    temp = board.copy()
                    temp[i] = 'x'
                    temp_lst.append(temp)
                else:
                    temp = board.copy()
                    temp_lst.append(temp)

        return temp_lst

    def midgame_moves(self, board):
        gamelist = []
        w_cnt = 0
        for i in range(len(board)):
            if board[i] == 'W':
                w_cnt += 1

        if w_cnt == 3:
```



```
            gamelist = self.generateHops(board)
            return gamelist

        else:
            gamelist = self.generateMove(board)
            return gamelist

    def generateHops(self, board):
        hop_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                for j in range(len(board)):
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            hop_list = self.generateRem(cpy, hop_list)
                        else:
                            hop_list.append(cpy)

        return hop_list

    def generateMove(self, board):
        move_list = []
        for i in range(len(board)):
            if board[i] == 'W':
                n_list = list(self.neighbours(i))
                for j in n_list:
                    if board[j] == 'x':
                        cpy = board.copy()
                        cpy[i] = 'x'
                        cpy[j] = 'W'
                        if self.closemill(j, cpy):
                            move_list = self.generateRem(cpy,
move_list)
                        else:
                            move_list.append(cpy)

        return move_list

    def max_min(self, board, depth):

        if depth > 0:
            depth -= 1
            possible_pos = self.midgame_moves(board)

            val = float('-inf')
            max_board = [None] * 50
            for i in range(len(possible_pos)):
                min_board = self.min_max(possible_pos[i], depth)
                cnt = self.static_estm(min_board)
                if val < cnt:
                    val = cnt
                    self.minimax_est = val
                    max_board = possible_pos[i]
            return max_board
```



```python
        elif depth == 0:
            self.pos_eval += 1

        return board

    def min_max(self, board, depth):
        if depth > 0:
            depth -= 1
            children = self.generateBlackMove(board)

            val = float('inf')
            min_board = [None] * 50
            for i in range(len(children)):
                max_board = self.max_min(children[i], depth)
                cnt = self.static_estm(max_board)
                if val > cnt:
                    val = cnt
                    min_board = children[i]
            return min_board
        elif depth == 0:
            self.pos_eval += 1

        return board


if __name__ == '__main__':
    inputfile = sys.argv[1]
    outputFile = sys.argv[2]
    depth = int(sys.argv[3])

    with open(inputfile, 'r') as f1:
        s = f1.read()
        board = list(s)
        obj = MiniMaxGame()
        new_moves = obj.max_min(board, depth)
        new_s = ''.join(i for i in new_moves)
        print('Input board is: ' + s)
        print('New board is: ' + new_s)
        print('Positions Evaluated: ' + str(obj.pos_eval))
        print('MiniMax evaluation: ' + str(obj.minimax_est))

        with open(outputFile, 'w') as f2:
            f2.write(new_s)
```