Open field tic tac toe(4 * 4)

we can change board size by changing Height and width and k value in the code

Here my code shows width=4, height=4, k=4

In [32]:

```
from collections import namedtuple, Counter, defaultdict
import math
import functools
import random
cache = functools.lru_cache(10**6)
```

In [33]:

```
class Game:

    def actions(self, states):
        """allowable moves from this states."""
        raise NotImplementedError

    def result(self, states, move):
        """from making a move from a states."""
        raise NotImplementedError

    def is_terminal(self, states):
        """final states for the game."""
        return not self.actions(states)

    def utility(self, states, player):
        """final states to player."""
        raise NotImplementedError
```

In [34]:

```
def play_game(game, strategies: dict, verbose=False):
    states = game.initial
    while not game.is_terminal(states):
        player = states.to_move
        move = strategies[player](game, states)
        states = game.result(states, move)
        if verbose:
            print('Player', player, 'move:', move)
            print(states)
    return states
```

In [35]:

```
def minimax_search(game, states):
    player = states.to_move

    def max_value(states):
        if game.is_terminal(states):
            return game.utility(states, player), None
        v, move = -infinity, None
        for a in game.actions(states):
            v2, _ = min_value(game.result(states, a))
            if v2 > v:
                v, move = v2, a
        return v, move

    def min_value(states):
        if game.is_terminal(states):
            return game.utility(states, player), None
        v, move = +infinity, None
        for a in game.actions(states):
            v2, _ = max_value(game.result(states, a))
```

```
            if v2 < v:
                v, move = v2, a
        return v, move

    return max_value(states)

infinity = math.inf
```

In [36]:

```
def alphabeta_search(game, states):

    player = states.to_move

    def max_value(states, alpha, beta):
        if game.is_terminal(states):
            return game.utility(states, player), None
        v, move = -infinity, None
        for a in game.actions(states):
            v2, _ = min_value(game.result(states, a), alpha, beta)
            if v2 > v:
                v, move = v2, a
                alpha = max(alpha, v)
            if v >= beta:
                return v, move
        return v, move

    def min_value(states, alpha, beta):
        if game.is_terminal(states):
            return game.utility(states, player), None
        v, move = +infinity, None
        for a in game.actions(states):
            v2, _ = max_value(game.result(states, a), alpha, beta)
            if v2 < v:
                v, move = v2, a
                beta = min(beta, v)
            if v <= alpha:
                return v, move
        return v, move

    return max_value(states, -infinity, +infinity)
```

In [37]:

```
class TicTacToe(Game):
    def __init__(self, height=3, width=3, k=3):
        self.k = k # k in a row
        self.squares = {(x, y) for x in range(width) for y in range(height)}
        self.initial = Board(height=height, width=width, to_move='X', utility=0)

    def actions(self, board):
        """Legal moves are any square not yet taken."""
        return self.squares - set(board)

    def result(self, board, square):
        """Place a marker for current player on square."""
        player = board.to_move
        board = board.new({square: player}, to_move=('O' if player == 'X' else 'X'))
        win = k_in_row(board, player, square, self.k)
        board.utility = (0 if not win else +1 if player == 'X' else -1)
        return board

    def utility(self, board, player):
        """Return the value to player; 1 for win, -1 for loss, 0 otherwise."""
        return board.utility if player == 'X' else -board.utility

    def is_terminal(self, board):
        """A board is a terminal state if it is won or there are no empty squares."""
        return board.utility != 0 or len(self.squares) == len(board)

    def display(self, board): print(board)
```

In [38]:

```
def k_in_row(board, player, square, k):
    def in_row(x, y, dx, dy):
        return 0 if board[x, y] != player else 1 + in_row(x + dx, y + dy, dx, dy)
    return any(in_row(*square, dx, dy) + in_row(*square, -dx, -dy) - 1 >= k
               for (dx, dy) in ((0, 1), (1, 0), (1, 1), (1, -1)))
```

In [39]:

```
class ConnectFour(TicTacToe):
    def __init__(self): super().__init__(width=4, height=4, k=4)

    def actions(self, board):
        """In each column you can play only the lowest empty square in the column."""
        return {(x, y) for (x, y) in self.squares - set(board)
                if y == board.height - 1 or (x, y + 1) in board}
```

In [40]:

```
class Board(defaultdict):
    empty = '.'
    off = '#'

    def __init__(self, width=8, height=8, to_move=None, **kwds):
        self.__dict__.update(width=width, height=height, to_move=to_move, **kwds)

    def new(self, changes: dict, **kwds) -> 'Board':
        board = Board(width=self.width, height=self.height, **kwds)
        board.update(self)
        board.update(changes)
        return board

    def __missing__(self, loc):
        x, y = loc
        if 0 <= x < self.width and 0 <= y < self.height:
            return self.empty
        else:
            return self.off

    def __hash__(self):
        return hash(tuple(sorted(self.items()))) + hash(self.to_move)

    def __repr__(self):
        def row(y): return ' '.join(self[x, y] for x in range(self.width))
        return '\n'.join(map(row, range(self.height))) +  '\n'
```

In [41]:

```
def random_player(game, state): return random.choice(list(game.actions(state)))
```

In [42]:

```
def player(search_algorithm):
    return lambda game, state: search_algorithm(game, state)[1]
```

In [43]:

```
play_game(ConnectFour(), dict(X=random_player, O=random_player), verbose=True).utility
#if the output is 1 then the player whose move is X,won the game
#if the output is 0 then the match was draw
#if the output is -1 then the player whose move is O,won the game
```

```
Player X move: (1, 3)
. . . .
. . . .
. . . .
. X . .

Player O move: (1, 2)
. . . .
. . . .
. O . .
```

```
. X . .

Player X move: (2, 3)
. . . .
. . . .
. O . .
. X X .

Player O move: (0, 3)
. . . .
. . . .
. O . .
O X X .

Player X move: (3, 3)
. . . .
. . . .
. O . .
O X X X

Player O move: (3, 2)
. . . .
. . . .
. O . O
O X X X

Player X move: (2, 2)
. . . .
. . . .
. O X O
O X X X

Player O move: (1, 1)
. . . .
. O . .
. O X O
O X X X

Player X move: (1, 0)
. X . .
. O . .
. O X O
O X X X

Player O move: (2, 1)
. X . .
. O O .
. O X O
O X X X

Player X move: (0, 2)
. X . .
. O O .
X O X O
O X X X

Player O move: (0, 1)
. X . .
O O O .
X O X O
O X X X

Player X move: (0, 0)
X X . .
O O O .
X O X O
O X X X

Player O move: (3, 1)
X X . .
O O O O
X O X O
O X X X
```

Out[43]:

```
-1
```

In [ ]:

```
```