For basic part, I implement IsFiveInLine(x, y) function. In this function, it will check 5 continuous points in horizontal line (right direction), vertical line (up direction), and 2 diagonal (right-up & right-down). Moreover, I do deep copy when clone the state and copy the board.

```
def Clone(self):
        """ Create a deep clone of this game state.
        """
        size=self.size
        st = five_in_line(size)
        st.board = copy.deepcopy(self.board)
        st.playerJustMoved = self.playerJustMoved
        return st
```

```
def IsFiveInLine(self, x, y):
    """ Speeds up GetMoves by only considering which are connected five in line of the playerjm.
    """
    # THis is the part you need to implement to complete the five_in_line game program
    # you need to specify the concept to detect whether a board consists of five in a line in one player ot
    # other to decide which player is the winner
    # The predicate function is called by GetResult(self,playerJustMoved)
    checksum = 0
    # detect horizontal
    if (self.IsOnBoard(x+4, y)):
        if (self.board[x][y] == self.board[x+1][y] and self.board[x][y] == self.board[x+2][y] and self.board[x][y] == self.board[x+3][y] and self.board[x][y] == self.board[x+4][y]):
            checksum = 1
    #detect vertical
    if(self.IsOnBoard(x, y+4)):
        if (self.board[x][y] == self.board[x][y+1] and self.board[x][y] == self.board[x][y+2] and self.board[x][y] == self.board[x][y+3] and self.board[x][y] == self.board[x][y+4]):
            checksum = 1
    #detect diagonal (slash up)
    if(self.IsOnBoard(x+4, y+4)):
        if (self.board[x][y] == self.board[x+1][y+1] and self.board[x][y] == self.board[x+2][y+2] and self.board[x][y] == self.board[x+3][y+3] and self.board[x][y] == self.board[x+4][y+4]):
            checksum = 1
    # detect diagonal (slash down)
    if(self.IsOnBoard(x+4, y-4)):
        if (self.board[x][y] == self.board[x+1][y-1] and self.board[x][y] == self.board[x+2][y-2] and self.board[x][y] == self.board[x+3][y-3] and self.board[x][y] == self.board[x+4][y-4]):
            checksum = 1
    return checksum
```

For bonus part, I implement some functions to guarantee player will get better and more reasonable move without MCTS algorithm. In some pattern, player will win the game however opponent block it such as four_in_line and three_in_line appear at the same time. In contrast, there are some pattern let opponent win the game if player don't block it as opponent get free_three, four_in_line or jump_four.

```
Specific Pattern:
  five_in_line = {ooooo}
  free_four = {_oooo_}
  free_three = {_ooo_}
  four_in_line (one_side_four) = {oooo_, _oooo, xoooo_, _ooox}
  jump_four = {oo_oo, o_ooo, ooo_o}
```

Thus, I return the move before do MCTS if there is a way to win the game or lose the game if choose other points. At first, it will judge whether exist the point let player win the game (five_in_line). Next, it will check if opponent exists four_in_line after player do the action or not, and it'll remove the moves which make player lose the game. If the opponent doesn't have chance to win the game in next step, it will check player have chance to get free_four, 2 four_in_line, 1 four_in_line and 1 free_three which let player win the game and return the move. Then, it will check if opponent have free_three on the board and remove this moves because free_three

```python
# Specific Select
    for move in node.untriedMoves:
        # five in raw - attack
        copystate = state.Clone()
        copystate.DoMove(move)
        if copystate.GetResult(player)==1.0:
            return move
        # four in raw - defense
    for move in node.untriedMoves:
        copystate = state.Clone()
        copystate.DoMove(move)
            # four in raw - defense
        if copystate.GetOneSideFour(op)>=1 or copystate.GetFreeFour(op)>=1 or copystate.GetJumpFour(op)>=1:
            deleteMove.append(move)
            continue
        else:
            # four in raw - attack
            if copystate.GetFreeFour(player) >= 1:
                return move
            # double four cross - attack
            elif copystate.GetOneSideFour(player) >= 2:
                return move
            # Three and four cross - attack
            elif copystate.GetOneSideFour(player)+copystate.GetFreeThree(player) >= 2:
                return move
            else:
                if copystate.GetFreeThree(op) >= 1:
                    deleteMove.append(move)
                    continue
            # double three cross - attack
                # elif copystate.GetFreeThree(player) >= 2:
                #     return move
    if deleteMove != node.untriedMoves:
        for delmove in deleteMove:
            node.untriedMoves.remove(delmove)
```

let opponent will get free_four at next and let player lose the game. Finally, if there isn't exist these specific pattern on the board, it will do MCTS and return the best move by ucb value.