



In [10]:

```
1 import numpy as np
2 import random
3 from time import sleep
4 def create_board():
5     return(np.array([[0, 0, 0],
6                       [0, 0, 0],
7                       [0, 0, 0]]))
8 def possibilities(board):
9     l = []
10
11     for i in range(len(board)):
12         for j in range(len(board)):
13
14             if board[i][j] == 0:
15                 l.append((i, j))
16     return(l)
17 def random_place(board, player):
18     selection = possibilities(board)
19     current_loc = random.choice(selection)
20     board[current_loc] = player
21     return(board)
22 def row_win(board, player):
23     for x in range(len(board)):
24         win = True
25         for y in range(len(board)):
26             if board[x, y] != player:
27                 win = False
28                 continue
29
30         if win == True:
31             return(win)
32     return(win)
33 def col_win(board, player):
34     for x in range(len(board)):
35         win = True
36
37         for y in range(len(board)):
38             if board[y][x] != player:
39                 win = False
40                 continue
41
42         if win == True:
43             return(win)
44     return(win)
45 def diag_win(board, player):
46     win = True
47     y = 0
48     for x in range(len(board)):
49         if board[x, x] != player:
50             win = False
51     if win:
52         return win
53     win = True
54     if win:
55         for x in range(len(board)):
56             y = len(board) - 1 - x
57             if board[x, y] != player:
58                 win = False
59     return win
```

```

60 def evaluate(board):
61     winner = 0
62     for player in [1, 2]:
63         if (row_win(board, player) or
64             col_win(board, player) or
65             diag_win(board, player)):
66
67             winner = player
68
69     if np.all(board != 0) and winner == 0:
70         winner = -1
71     return winner
72 def play_game():
73     board, winner, counter = create_board(), 0, 1
74     print(board)
75     sleep(2)
76     while winner == 0:
77         for player in [1, 2]:
78             board = random_place(board, player)
79             print("Board after " + str(counter) + " move")
80             print(board)
81             sleep(2)
82             counter += 1
83             winner = evaluate(board)
84             if winner != 0:
85                 break
86     return(winner)
87 print("Winner is: " + str(play_game()))

```

```

[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 1 0]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 1 0]
 [2 0 0]
 [0 0 0]]
Board after 3 move
[[0 1 0]
 [2 0 0]
 [0 1 0]]
Board after 4 move
[[0 1 0]
 [2 2 0]
 [0 1 0]]
Board after 5 move
[[1 1 0]
 [2 2 0]
 [0 1 0]]
Board after 6 move
[[1 1 0]
 [2 2 0]
 [0 1 2]]
Board after 7 move
[[1 1 1]
 [2 2 0]
 [0 1 2]]
Winner is: 1

```

In [18]:

```
1 MAX, MIN = 1000, -1000
2 def minimax(depth, nodeIndex, maximizingPlayer,
3             values, alpha, beta):
4     if depth == 3:
5         return values[nodeIndex]
6     if maximizingPlayer:
7         best = MIN
8         for i in range(0, 2):
9             val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
10            best = max(best, val)
11            alpha = max(alpha, best)
12            if beta <= alpha:
13                break
14        return best
15     else:
16         best = MAX
17         for i in range(0, 2):
18             val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
19             best = min(best, val)
20             beta = min(beta, best)
21             if beta <= alpha:
22                 break
23        return best
24 if __name__ == "__main__":
25
26     values = [3, 5, 6, 9, 1, 2, 0, -1]
27     print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
28
```

The optimal value is : 5

In [ ]: