In [10]:

```
import numpy as np
import random
from time import sleep
def create_board():
    return(np.array([[0, 0, 0],
                     [0, 0, 0],
                     [0, 0, 0]]))
def possibilities(board):
    l = []

    for i in range(len(board)):
        for j in range(len(board)):

            if board[i][j] == 0:
                l.append((i, j))
    return(l)
def random_place(board, player):
    selection = possibilities(board)
    current_loc = random.choice(selection)
    board[current_loc] = player
    return(board)
def row_win(board, player):
    for x in range(len(board)):
        win = True
        for y in range(len(board)):
            if board[x, y] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)
def col_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)
def diag_win(board, player):
    win = True
    y = 0
    for x in range(len(board)):
        if board[x, x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win
```

```
def evaluate(board):
    winner = 0
    for player in [1, 2]:
        if (row_win(board, player) or
                col_win(board, player) or
                diag_win(board, player)):

            winner = player

    if np.all(board != 0) and winner == 0:
        winner = -1
    return winner
def play_game():
    board, winner, counter = create_board(), 0, 1
    print(board)
    sleep(2)
    while winner == 0:
        for player in [1, 2]:
            board = random_place(board, player)
            print("Board after " + str(counter) + " move")
            print(board)
            sleep(2)
            counter += 1
            winner = evaluate(board)
            if winner != 0:
                break
    return(winner)
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]:

```
MAX, MIN = 1000, -1000
def minimax(depth, nodeIndex, maximizingPlayer,
            values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]
    if maximizingPlayer:
        best = MIN
         for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = MAX
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best
if __name__ == "__main__":

    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

```

```
The optimal value is : 5
```

In [ ]: