```python
import random


# The starter function, lets the user decide how to play the game
def gamePick():
    global gamePick
    print('\tTic-Tac-Toe')

    options = ['2 Players local play', 'Unbeatable Computer']
    for i in range(2):
        print(f"{i + 1}) {options[i]}")

    while True:
        # Validating the input
        gamePick = str(input('Please pick your game type: '))

        # These conditions are seperated to check first if the input not a number
        # and only after that check if its withing the range of options
        if not gamePick.isdigit():
            print('❌: Wrong input for gamePick, try again')
        elif int(gamePick) > len(options) or int(gamePick) < 1:
            print('❌: Wrong input for gamePick, try again')
        else:
            break

    gamePick = int(gamePick) - 1
    if gamePick == 0:
        localPlay()
    elif gamePick == 1:
        vsComp()


# Switching the currently playing player
def switchPlayer():
    if player['sign'] == 'X':
        player['sign'] = 'O'

        # Checks if you're playing against ai
        if(gamePick == 1):
            player['name'] = 'Computer'
        else:
            player['name'] = secPlayerName
    else:
        player['sign'] = 'X'
        player['name'] = firstPlayerName


# Reset the board, get the players names and flip a coin to see who's playing first
def reset(isFirstTime=False):
    global board, player, firstPlayerName, secPlayerName, score
    if isFirstTime:
        firstPlayerName = str(input('Please enter player1 name: '))

        # gamePick 0 = [2 players local play] => need a second players name
        if gamePick == 0:
            secPlayerName = str(input('Please enter player2 name: '))
            score = {
                firstPlayerName: 0,
                secPlayerName: 0
            }
        else:
            score = {
                firstPlayerName: 0
            }

        player = {
            'name': firstPlayerName,
            'sign': 'X',
        }
    board = [['_', '_', '_'],
             ['_', '_', '_'],
             ['_', '_', '_']]

    coinFlipPick = str(input('Please enter a coin flip pick(0/1): '))
    randomCoinFlip = str(random.randint(0, 1))
    print(f"Flipped the coin: {randomCoinFlip}")
    if coinFlipPick != randomCoinFlip:
        switchPlayer()

    if not isFirstTime:
        if gamePick == 0:
            localPlay(isFirstTime)
        elif gamePick == 1:
            vsComp(isFirstTime)


def printBoard():
    print('    0    1    2')
    col = 0
    for row in range(3):
        print(f"{col} {board[row]}")
        col += 1


# Returns the winner sign if there's a winner, if not it will return None to continue playing
def getWinner():
    # Row win
    for row in range(3):
        if '_' not in board[row]:
            if len(set(board[row])) == 1:
                return board[row][0]

    # Column win
    for row in range(3):
        column = []
        for col in range(3):
            if board[col][row] != '_':
                column.append(board[col][row])
            if len(set(column)) == 1 and len(column) == 3:
                return column[0]

    # Crosses win
    cross1 = []
    cross2 = []
    for i in range(3):
        if board[i][i] != '_':
            cross1.append(board[i][i])
            if len(set(cross1)) == 1 and len(cross1) == 3:
                return board[i][i]
        if board[i][2 - i] != '_':
            cross2.append(board[i][2 - i])
            if len(set(cross2)) == 1 and len(cross2) == 3:
                return board[i][2 - i]

    # No win, board is not full and its checking this condition
    for row in range(0, 3):
        for col in range(0, 3):
            if (board[row][col] == '_'):
                return None

    # Draw
    return '_'


def isValidFormatInput():
    while True:
        move = str(
            input('Please enter the indexes for your move(x,y): '))
        try:
            [x, y] = [int(x) for x in move.split(',')]
            return [x, y]
        except Exception:
            print('❌: Wrong format, try again using the correct format: x,y')


def isValidMove():
    while True:
        [x, y] = isValidFormatInput()

        # Check if one of the indexes is out of range
        # or longer than the accepted format
        if (
            x > 2 or x < 0 or
            y > 2 or y < 0 or
            board[x][y] != '_'
        ):
            print('❌: Not a valid move, enter the indexes again please: ')
            # return False
        else:
            return [x, y]


'''
    One of two functions,
    this function will try to minimize the opponents score.
    Using recoursion in order to test every board possibillity, the function check
    if the possibility tree is making the opponent lose.
    it will return a value by this terms:
     1: win
     0: draw
    -1: loss
'''


def minCompMove():

    # Initialize the outcome of the move with the lowest possible value
    outcome = 2

    x = None
    y = None
    result = getWinner()

    # X wins = computer loss
    if result == 'X':
        return (-1, 0, 0)

    # O wins = computer wins
    elif result == 'O':
        return (1, 0, 0)

    # _ = draw
    elif result == '_':
        return (0, 0, 0)

    # Find the best outcome for all unused cells
    for row in range(3):
        for col in range(3):
            if board[row][col] == '_':
                # Creating a possibility recourse tree
                board[row][col] = 'X'
                [m, maxRow, maxCol] = compMove()
                if m < outcome:
                    outcome = m
                    x = row
                    y = col
                # Revert Changes
                board[row][col] = '_'

    return [outcome, x, y]


'''
    One of two functions,
    This function will try to maximize the computers score.
    Using recoursion in order to test every board possibillity, the function check
    if the possibility tree is getting us to win with the least moves.
    it will return a value by this terms:
     1: win
     0: draw
    -1: loss
'''


def compMove():

    # Initialize the outcome of the move with the lowest possible value
    outcome = -2

    x = None
    y = None
    result = getWinner()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '_':
        return (0, 0, 0)

    for row in range(3):
        for col in range(3):
            if board[row][col] == '_':
                # Creating a possibility recourse tree
                board[row][col] = 'O'
                [m, minRow, minCol] = minCompMove()
                if m > outcome:
                    outcome = m
                    x = row
                    y = col
                # Revert Changes
                board[row][col] = '_'
    return [outcome, x, y]


# Checks if the board is empty
def isFirstMove():
    for row in range(0, 3):
        for col in range(0, 3):
            if (board[row][col] != '_'):
                return False
    return True


def printScores():
    sortedScores = sorted(
        score.items(), key=lambda scoreData: scoreData[1], reverse=True)
    for index, scoreTuple in enumerate(sortedScores):
        print(f"{index + 1}) {scoreTuple[1]}: {scoreTuple[0]}")


def updateScore(winnerSign):
    if winnerSign == 'draw':
        if not player['name'] == 'Computer':
            score[player['name']] += 1

        # Check if theres only one player(happens when playing vs computer)
        if len(score.keys()) > 1:
            switchPlayer()
            if not player['name'] == 'Computer':
                score[player['name']] += 1
    else:

        # A short condition to check if the winner is the currently active player.
        if player['sign'] != winnerSign[0]:
            switchPlayer()
        if not player['name'] == 'Computer':
            score[player['name']] += 2


def humanMove(playerSign):
    [x, y] = isValidMove()
    board[x][y] = playerSign
    switchPlayer()


def restart():
    # Optional restart after the game has finished
    print('Type showScores to see the score board')
    answer = str(
        input('Would you like to restart(Y/N/showScores): ')).lower()
    if answer == 'showscores':
        printScores()
        restart()
    elif answer == 'y':
        reset()
    else:
        print('Thank you for playing!')
        return False


def vsComp(isFirstTime=True):
    if isFirstTime:
        reset(True)

    while True:
        printBoard()
        winner = getWinner()

        if winner != None:
            if winner == 'X':
                print('X Wins.')
                updateScore('X')
            elif winner == 'O':
                print('O Wins.')
                updateScore('O')
            elif winner == '_':
                print('Draw!')
                updateScore('draw')

            # Checking if the user wants to end the game or not
            if not restart():
                return

        print(player['name'] + '\'s turn as ' + player['sign'])

        # User's turn
        if player['sign'] == 'X':
            humanMove('X')

        # Computer's turn
        else:
            if isFirstMove():
                board[random.randint(0, 2)][random.randint(0, 2)] = 'O'
            else:
                [status, x, y] = compMove()
                board[x][y] = 'O'
            switchPlayer()


def localPlay(isFirstTime=True):
    if isFirstTime:
        reset(True)

    while True:
        printBoard()
        winner = getWinner()

        if winner != None:
            if winner == 'X':
                print('X Wins.')
                updateScore('X')
            elif winner == 'O':
                print('O Wins.')
                updateScore('O')
            elif winner == '_':
                print('Draw!')
                updateScore('draw')

            # Checking if the user wants to end the game or not
            if not restart():
                return

        print(player['name'] + '\'s turn as ' + player['sign'])

        humanMove(player['sign'])


gamePick()

```