```
import random
import os.path
import json
random.seed()

def draw_board(board):
    # develop code to draw the board
    print(" {} | {} | {} ".format(board[0][0], board[0][1], board[0][2]))
   # print("---+---+---")
    print(" {} | {} | {} ".format(board[1][0], board[1][1], board[1][2]))
    #print("---+---+---")
    print(" {} | {} | {} ".format(board[2][0], board[2][1], board[2][2]))
    print("---+---+---")
    pass

def welcome(board):
    # prints the welcome message
    # display the board by calling draw_board(board)
    print("Welcome to Tic-Tac-Toe!")
    print("Here is the current state of the board: ")
    draw_board(board)
    pass

def initialise_board(board):
    # develop code to set all elements of the board to one space ' '
    for i in range(3):
        for j in range(3):
            board[i][j] = ' '
    return board
```

```
def get_player_move(board):
    while True:
        #move = input("Enter your move (1-9): ")
        try:
            move = int(input("Enter your move (1-9): "))
            #move = int(move)
        except ValueError:
            print("Invalid move, try again.")
            continue
        if move < 1 or move > 9:
            print("Invalid move, try again.")
            continue
        row, col = (move-1) // 3, (move-1) % 3
        if board[row][col] != ' ':
            print("Cell already occupied, try again.")
            continue
        break
    return row, col

def choose_computer_move(board):
    available_spaces = []
    for i in range(len(board)):
        for j in range(len(board[i])):
            if board[i][j] == ' ':
                available_spaces.append((i, j))
    random.seed()
    row, col = random.choice(available_spaces)
    return row, col
```

```
def check_for_win(board, mark):
    # develop code to check if either the player or the computer has won
    # return True if someone won, False otherwise
    for row in board:
        if row == [mark, mark, mark]:
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == mark and board[1][col] == mark and board[2][col] == mark:
            return True

    # Check diagonals
    if board[0][0] == mark and board[1][1] == mark and board[2][2] == mark:
        return True
    if board[0][2] == mark and board[1][1] == mark and board[2][0] == mark:
        return True

    return False

def check_for_draw(board):
    # develop cope to check if all cells are occupied
    # return True if it is, False otherwise
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True
```

```
def play_game(board):
    initialise_board(board)
    draw_board(board)

    while True:
        player_move = get_player_move(board)
        board[player_move[0]][player_move[1]] = 'X'
        draw_board(board)
        if check_for_win(board, 'X'):
            return 1
        if check_for_draw(board):
            return 0

        computer_move = choose_computer_move(board)
        board[computer_move[0]][computer_move[1]] = 'O'
        draw_board(board)
        if check_for_win(board, 'O'):
            return -1
        if check_for_draw(board):
            return 0



def menu():
    # get user input of either '1', '2', '3' or 'q'
    # 1 - Play the game
    # 2 - Save score in file 'leaderboard.txt'
    # 3 - Load and display the scores from the 'leaderboard.txt'
    # q - End the program
```

```
    print("\n")
    print("1 - Play the game")
    print("2 - Save score in file 'leaderboard.txt'")
    print("3 - Load and display the scores from the 'leaderboard.txt'")
    print("q - End the program")
    choice = input("Enter your choice: ")
    return choice
#filename='leaderboard.txt'
def load_scores():
    # develop code to load the leaderboard scores
    # from the file 'leaderboard.txt'
    # return the scores in a Python dictionary
    # with the player names as key and the scores as values
    # return the dictionary in leaders
    leaders = {}
    try:
        if os.path.exists("leaderboard.txt"):
            with open("leaderboard.txt", "r") as f:
                leaders = json.load(f)
    except:
        print("Error loading scores from the leaderboard.txt file")
    return leaders

def save_score(score):
    # develop code to ask the player for their name
    # and then save the current score to the file 'leaderboard.txt'

    leaders = {}
    if os.path.exists('leaderboard.txt'):
```

```
        with open('leaderboard.txt', 'r') as file:
            leaders = json.load(file)
    player_name = input("Enter your name: ")
    leaders[player_name] = score
    with open('leaderboard.txt', 'w') as file:
        json.dump(leaders, file)
    return


def display_leaderboard(leaders):
    # develop code to display the leaderboard scores
    # passed in the Python dictionary parameter leader

    print("LEADERBOARD")
    print("-----------")
    for i, (name, score) in enumerate(leaders.items()):
        print(f"{i+1}. {name}: {score}")
    pass

def main():
    board = [ ['1','2','3'],\
              ['4','5','6'],\
              ['7','8','9']]

    welcome(board)
    total_score = 0
    while True:
        choice = menu()
        if choice == '1':
```

```
            score = play_game(board)
            total_score += score
            print('Your current score is:',total_score)
        if choice == '2':
            save_score(total_score)
        if choice == '3':
            leader_board = load_scores()
            display_leaderboard(leader_board)
        if choice == 'q':
            print('Thank you for playing the "Unbeatable Noughts and Crosses" game.')
            print('Good bye')
            return

# Program execution begins here
if __name__ == '__main__':
    main()
```