```python
"""
MATH20621 - Coursework 3
Student name: <Jieyi Huang>
Student id:   <11108970>
Student mail: <jieyi.huang@student.manchester.ac.uk>

Do not change any part of this string except to replace
the <tags> with your name, id and university email address.
"""

def request_location(question_str):
    """
    Prompt the user for a board location, and return that location.

    Takes a string parameter, which is displayed to the user as a prompt.

    Raises ValueError if input is not a valid integer,
    or RuntimeError if the location typed is not in the valid range.

    **************************************************************
    DO NOT change this function in any way
    You MUST use this function for ALL user input in your program
    **************************************************************
    """
    loc = int(input(question_str))
    if loc<0 or loc>=24:
        raise RuntimeError("Not a valid location")
    return loc


def draw_board(g):
    """
    Display the board corresponding to the board state g to console.
    Also displays the numbering for each point on the board, and the
    number of counters left in each players hand, if any.
    A reference to remind players of the number of each point is also displayed.

    You may use this function in your program to display the board
    to the user, but you may also use your own similar function, or
    improve this one, to customise the display of the game as you choose
    """
    def colored(r, g, b, text):
        """
        Spyder supports coloured text! This function creates coloured
        version of the text 'text' that can be printed to the console.
        The colour is specified with red (r), green (g), blue (b) components,
        each of which has a range 0-255.
        """
        return f"\033[38;2;{r};{g};{b}m{text}\033[0m"

    def piece_char(i):
        """
        Return the (coloured) character corresponding to player i's counter,
        or a + to indicate an unoccupied point
        """
        if i==0:
            return colored(100,100,100,'+')
        elif i==1:
            return colored(255,60,60,'X')
        elif i==2:
            return colored(60,120,255,'O')


    board = '''
x--------x--------x  0--------1--------2
|        |        |  |        |        |
|  x-----x-----x  |  |  3-----4-----5  |
|  |     |     |  |  |  |     |     |  |
|  |  x--x--x  |  |  |  |  6--7--8  |  |
|  |  |     |  |  |  |  |  |     |  |  |
x--x--x     x--x--x  9-10-11    12-13-14
|  |  |     |  |  |  |  |  |     |  |  |
|  |  x--x--x  |  |  |  | 15-16-17  |  |
|  |     |     |  |  |  |     |     |  |
|  x-----x-----x  |  |  18---19----20  |
|        |        |  |        |        |
x--------x--------x  21------22-------23
'''
    boardstr = ''
    i = 0
    for c in board:
        if c=='x':
            boardstr += piece_char(g[0][i])
            i += 1
        else:
            boardstr += colored(100,100,100,c)
    if g[1]>0 or g[2]>0:
        boardstr += '\nPlayer 1: ' + (piece_char(1)*g[1])
        boardstr += '\nPlayer 2: ' + (piece_char(2)*g[2])
    print(boardstr)

############################
# The functions for each task

def is_adjacent(i, j):
    # TODO: implement function here
    # Define the connections between points on the board
    adjacent_list = {
    0: [1, 9], 1: [0, 2, 4], 2: [1, 14],
    3: [4, 10], 4: [1, 3, 5, 7], 5: [4, 13],
    6: [7, 11], 7: [4, 6, 8], 8: [7, 12],
    9: [0, 10, 21], 10: [3, 9, 11, 18], 11: [6, 10, 15],
    12: [8, 13, 17], 13: [5, 12, 14, 20], 14: [2, 13, 23],
    15: [11, 16], 16: [15, 17, 19], 17: [12, 16],
    18: [10, 19], 19: [16, 18, 20, 22], 20: [13, 19],
    21: [9, 22], 22: [19, 21, 23], 23: [14, 22]
                                               }
    # Check that neither point is the same, and that the second point is in the adjacency list of the first
    return i != j and j in adjacent_list.get(i, [])

def new_game():
    # TODO: implement function here
    # Initialize the game board with all zeros, 9 counters for each player, and set current player as player 1
    g = [[0] * 24, 9, 9, 1]
    return g

def remaining_counters(g):
    # TODO: implement function here
    # Extract the active player
    active_player= g[3]
    # Count the active player's counters on the board
    counters_on_board = g[active_player]
    # Count the active player's counters in hand
    counters_in_hand = g[0].count(active_player)
    # The total available counters is the sum of counters on the board and in hand
    total_counters = counters_on_board + counters_in_hand
    return total_counters

def is_in_mill(g, i):
    # TODO: implement function here
    # Define all mills for each point
    mills = {
        0: [[1, 2], [9, 21]],
        1: [[0, 2],[4,7]],
        2: [[0, 1], [14, 23]],
        3: [[4, 5], [10, 18]],
        4: [[3, 5],[1,7]],
        5: [[3, 4], [13, 20]],
        6: [[7, 8], [11, 15]],
        7: [[6, 8],[1,4]],
        8: [[6, 7], [12, 17]],
        9: [[0, 21], [10, 11]],
        10: [[3, 18], [9, 11]],
        11: [[6, 15], [9, 10]],
        12: [[8, 17], [13, 14]],
        13: [[5, 20], [12, 14]],
        14: [[2, 23], [12, 13]],
        15: [[6, 11], [16, 17]],
        16: [[15, 17], [19, 22]],
        17: [[8, 12], [15, 16]],
        18: [[3, 10], [19, 20]],
        19: [[16, 22], [18, 20]],
        20: [[5, 13], [18, 19]],
        21: [[0, 9], [22, 23]],
        22: [[16, 19], [21, 23]],
        23: [[2, 14], [21, 22]]
    }
    if i < 0 or i > 23 or g[0][i] == 0:
        return -1
    # Check if the point is part of any mill
    player = g[0][i]
    for mill in mills[i]:
        if all(g[0][j] == player for j in mill):
            return player
    return 0

def player_can_move(g):
    # TODO: implement function here
    current_player = g[3]
    # Player can move if they have counters in hand
    if g[current_player] > 0:
        return True
    for i, player in enumerate(g[0]):
        if player == current_player:
            adjacent_points = [index for index, value in enumerate(g[0]) if is_adjacent(i, index)]
            if any(g[0][adj_point] == 0 for adj_point in adjacent_points):
                return True
    return False

def place_counter(g, i):
    # TODO: implement function here
    # Check if the point i is already occupied
    if g[0][i] != 0:
        raise RuntimeError("Point already occupied.")
    # Determine the current player
    current_player = g[3]
    # Place the player's counter at point i
    g[0][i] = current_player
    # Decrement the number of counters the current player has in hand
    if current_player == 1:
        g[1] -= 1
    else:  # current_player == 2
        g[2] -= 1

def move_counter(g, i, j):
    # TODO: implement function here
    active_player = g[3]
    # Check if points i and j are adjacent
    if not is_adjacent(i, j):
        raise RuntimeError("Points are not adjacent.")
    # Check whether point i contains a counter of the current player
    if g[0][i] != active_player:
        raise RuntimeError("No counter of the current player at the point.")
    # Check if point j is unoccupied
    if g[0][j] != 0:
        raise RuntimeError("Point already occupied.")
    # Move the counter
    g[0][i] = 0
    g[0][j] = active_player

def remove_opponent_counter(g, i):
    # TODO: implement function here
    # Get the opponent player
    opponent = 3-g[3]
    # Check if point i is occupied by the opponent's counter
    if g[0][i] != opponent:
        raise RuntimeError("No opponent's counter on this point.")
    # Remove the counter from point i
    g[0][i] = 0

def turn(g):
    # TODO: implement function here
    # Check if the current player can move
    if not player_can_move(g):
        return False
    draw_board(g)
    # Current player
    current_player = g[3]
    counters_in_hand = g[1] if current_player == 1 else g[2]
    # Handle counter placement or movement
    if counters_in_hand > 0:
        while True:
            try:
                location = request_location("Player" + str(current_player) + ":Enter a location to place your counter: ")
                place_counter(g, location)
                break
            except RuntimeError as e:
                print(f"Invalid placement: {e}")
            except ValueError:
                print(" Enter a number between 0 and 23.")
    else:
        while True:
            try:
                from_location = request_location("Player" + str(current_player) + ":Enter the location of your counter to move: ")
                to_location = request_location("Player" + str(current_player) + ":Enter the new location to move your counter to: ")
                move_counter(g, from_location, to_location)
                break
            except RuntimeError as e:
                print(f"Invalid move: {e}")
            except ValueError:
                print("Enter a number between 0 and 23.")
    draw_board(g)
    if is_in_mill(g, location if counters_in_hand > 0 else to_location):
        while True:
            try:
                remove_location = request_location("Player" + str(current_player) + ":Enter the location of an opponent's counter to remove: ")
                remove_opponent_counter(g, remove_location)
                break
            except RuntimeError as e:
                print(f"Invalid location: {e}")
```

```python
            except ValueError:
                print("Enter a number between 0 and 23.")
    # Update the game state to switch the current player
    g[3] = 3-g[3]
    # Return True to indicate the game continues
    return True

def save_state(g, filename):

    try:
        with open(filename, 'w') as file:
            # Write the board state
            file.write(','.join(map(str, g[0])) + '\n')
            # Write the number of counters left for each player
            file.write(str(g[1]) + '\n')
            file.write(str(g[2]) + '\n')
            # Write the current player
            file.write(str(g[3]) + '\n')
    except Exception as e:
        raise RuntimeError(f"Error saving game state: {e}")

def load_state(filename):

    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            board_state = list(map(int, lines[0].strip().split(',')))
            counters_p1 = int(lines[1].strip())
            counters_p2 = int(lines[2].strip())
            current_player = int(lines[3].strip())
            return [board_state, counters_p1, counters_p2, current_player]
    except Exception as e:
        raise RuntimeError(f"Error loading game state: {e}")

def play_game():
    # Initialize a new game
    g = new_game()
    while True:
        # Take a turn for the current player
        if not turn(g):
            # If a player cannot make a valid move or have less than 3 counters, the game ends
            break
    # Determine the winner
    winner = 3-g[3]
    print(f"Congratulations to Player {winner}!You won the game")

def main():
    # You could add some tests to main()
    # to check your functions are working as expected

    # The main function will not be assessed. All code to
    # play the game should be in the play_game() function,
    # and so your main function should simply call this.
    play_game()
main()
```