

✓ Lab 2- Tic Tac Toe

In this lab your will build a $n \times n$ Tic Tac Toe game. As you do the exercises, make sure your solutions work for any size Tic Tac Toe game.

Exercise 1: Write a function that creates an n by n matrix (of list of lists) which will represent the state of a Tie Tac Toe game. Let 0, 1, and 2 represent empty, "X", and "O", respectively.

```
# Write you solution heredef create_tic_tac_toe_matrix(n):
    """
    Create an n by n matrix for representing the state of a Tic Tac Toe game.

    Parameters:
    n (int): The size of the matrix (n x n).

    Returns:
    list of lists: An n by n matrix initialized with 0s.
    """
    # Create an n x n matrix filled with 0s
    matrix = [[0 for _ in range(n)] for _ in range(n)]
    return matrix

# Example usage:
n = 4 # Size of the matrix
matrix = create_tic_tac_toe_matrix(n)
for row in matrix:
    print(row)
```

```
def create_tic_tac_toe_matrix(n):
    """
    Create an n by n matrix for representing the state of a Tic Tac Toe game.

    Parameters:
    n (int): The size of the matrix (n x n).

    Returns:
    list of lists: An n by n matrix initialized with 0s.
    """
    # Create an n x n matrix filled with 0s
    matrix = [[0 for _ in range(n)] for _ in range(n)]
    return matrix

# Example usage:
n = 4 # Size of the matrix
matrix = create_tic_tac_toe_matrix(n)
for row in matrix:
    print(row)
```



Show hidden output

Exercise 2: Write a function that takes 2 integers n and m as input and draws a n by m game board. For example the following is a 3x3 board:

```
--- --- ---
|   |   |   |
--- --- ---
|   |   |   |
--- --- ---
|   |   |   |
--- --- ---
```

```
def draw_board(n, m):
    # Print the top border
    print(' ' + ' --- ' * m)

    for _ in range(n):
        # Print the middle part with vertical bars
        print(' | ' + ' | '.join(' ' * 3 for _ in range(m)) + ' | ')

        # Print the horizontal border after each row
        print(' ' + ' --- ' * m)

# Example usage
draw_board(3, 3)
```

```
def draw_board(n, m):
    # Print the top border
    print(' ' + ' --- ' * m)

    for _ in range(n):
        # Print the middle part with vertical bars
        print(' | ' + ' | '.join(' ' * 3 for _ in range(m)) + ' | ')

        # Print the horizontal border after each row
        print(' ' + ' --- ' * m)

# Example usage
draw_board(3, 3)
```



```

    ---  ---  ---
|      |      |      |
    ---  ---  ---
|      |      |      |
    ---  ---  ---
|      |      |      |
    ---  ---  ---
```

Exercise 3: Modify exercise 2, so that it takes a matrix of the form from exercise 1 and draws a tic-tac-toe board with "X"s and "O"s.

```
def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    m = len(matrix[0])

    def draw_line():
        # Draw the horizontal border
        print(' ' + ' --- ' * m)

    def draw_row(row):
        # Draw a row with the content
        print(' | ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for cell in row) + ' |')

    # Draw the board
    draw_line() # Draw the top border

    for row in matrix:
        draw_row(row) # Draw the current row
        draw_line() # Draw the horizontal border after the row

# Example usage
matrix = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['X', 'O', 'X']
]

draw_tic_tac_toe_board(matrix)
```

```

def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    m = len(matrix[0])

    def draw_line():
        # Draw the horizontal border
        print(' ' + ' --- ' * m)

    def draw_row(row):
        # Draw a row with the content
        print(' | ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for cell in row) + ' |')

    # Draw the board
    draw_line() # Draw the top border

    for row in matrix:
        draw_row(row) # Draw the current row
        draw_line() # Draw the horizontal border after the row

# Example usage
matrix = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['X', 'O', 'X']
]

draw_tic_tac_toe_board(matrix)

```



```

    ---  ---  ---
    |  X  |  O  |  X  |
    ---  ---  ---
    |  O  |  X  |  O  |
    ---  ---  ---
    |  X  |  O  |  X  |
    ---  ---  ---

```

Exercise 4: Write a function that takes a n by n matrix representing a tic-tac-toe game, and returns -1, 0, 1, or 2 indicating the game is incomplete, the game is a draw, player 1 has won, or player 2 has one, respectively. Here are some example inputs you can use to test your code:

```

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
                all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
            all(matrix[i][n - 1 - i] == player for i in range(n)):
            return True

        return False

    # Check for winner
    player1_wins = check_winner('X')
    player2_wins = check_winner('O')

    if player1_wins:
        return 1
    if player2_wins:
        return 2

    # Check for incomplete game
    if any(matrix[i][j] == ' ' for i in range(n) for j in range(n)):
        return -1

    # If no winner and no empty cells, it's a draw
    return 0

# Example usage
matrix1 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', 'X', 'X']
]

matrix2 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', 'O', 'X']
]

matrix3 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', ' ', 'X']
]

```

```
matrix4 = [  
    ['X', 'O', 'X'],  
    ['O', 'O', 'O'],  
    ['X', 'X', 'X']  
]  
  
print(check_tic_tac_toe(matrix1)) # Output: 1 (Player 1 wins)  
print(check_tic_tac_toe(matrix2)) # Output: 2 (Player 2 wins)  
print(check_tic_tac_toe(matrix3)) # Output: -1 (Incomplete)  
print(check_tic_tac_toe(matrix4)) # Output: 2 (Player 2 wins)
```

```

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
                all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
            all(matrix[i][n - 1 - i] == player for i in range(n)):
            return True

        return False

    # Check for winner
    player1_wins = check_winner('X')
    player2_wins = check_winner('O')

    if player1_wins:
        return 1
    if player2_wins:
        return 2

    # Check for incomplete game
    if any(matrix[i][j] == ' ' for i in range(n) for j in range(n)):
        return -1

    # If no winner and no empty cells, it's a draw
    return 0

# Example usage
matrix1 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', 'X', 'X']
]

matrix2 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', 'O', 'X']
]

matrix3 = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['O', ' ', 'X']
]

```



```

matrix4 = [
    ['X', 'O', 'X'],
    ['O', 'O', 'O'],
    ['X', 'X', 'X']
]

print(check_tic_tac_toe(matrix1)) # Output: 1 (Player 1 wins)
print(check_tic_tac_toe(matrix2)) # Output: 2 (Player 2 wins)
print(check_tic_tac_toe(matrix3)) # Output: -1 (Incomplete)
print(check_tic_tac_toe(matrix4)) # Output: 2 (Player 2 wins)

```

```

⇒ 1
   1
   1
   1

```

```

winner_is_2 = [[2, 2, 0],
               [2, 1, 0],
               [2, 1, 1]]

winner_is_1 = [[1, 2, 0],
               [2, 1, 0],
               [2, 1, 1]]

winner_is_also_1 = [[0, 1, 0],
                   [2, 1, 0],
                   [2, 1, 1]]

no_winner = [[1, 2, 0],
             [2, 1, 0],
             [2, 1, 2]]

also_no_winner = [[1, 2, 0],
                  [2, 1, 0],
                  [2, 1, 0]]

```

Exercise 5: Write a function that takes a game board, player number, and (x,y) coordinates and places "X" or "O" in the correct location of the game board. Make sure that you only allow filling previously empty locations. Return `True` or `False` to indicate successful placement of "X" or "O".

```
def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print(place_marker(board, 1, 0, 0)) # Output: True (Valid placement)
print(place_marker(board, 2, 0, 0)) # Output: False (Cell already occupied)
print(place_marker(board, 1, 3, 3)) # Output: False (Out of bounds)
print(place_marker(board, 2, 1, 1)) # Output: True (Valid placement)

# Board state after operations
for row in board:
    print(row)
```

```

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True


    # If the location is not empty, return False
    return False

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print(place_marker(board, 1, 0, 0)) # Output: True (Valid placement)
print(place_marker(board, 2, 0, 0)) # Output: False (Cell already occupied)
print(place_marker(board, 1, 3, 3)) # Output: False (Out of bounds)
print(place_marker(board, 2, 1, 1)) # Output: True (Valid placement)

# Board state after operations
for row in board:
    print(row)

```

 True
 False
 False
 True
 ['X', '', '']
 ['', 'O', '']
 ['', '', '']

Exercise 6: Modify Exercise 3 to show column and row labels so that players can specify location using "A2" or "C1".

```

def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    row_labels = [chr(i) for i in range(ord('A'), ord('A') + n)]

    def draw_line():
        # Draw the horizontal border with column labels
        print(' ' + '---' * n)
        print(' ' + ' '.join(f' {i+1} ' for i in range(n)))
        print(' ' + ' ' + '---' * n)

    def draw_row(index, row):
        # Draw a row with row label and the content
        print(f'{row_labels[index]} ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for

    # Draw the board
    draw_line() # Draw the top border
    for i, row in enumerate(matrix):
        draw_row(i, row) # Draw the current row with the row label
        draw_line() # Draw the horizontal border after the row

def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
                all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
            all(matrix[i][n - 1 - i] == player for i in range(n)):
            return True

        return False

    # Check for winner
    player1_wins = check_winner('X')
    player2_wins = check_winner('O')

    if player1_wins:

```

```
        return 1
    if player2_wins:
        return 2

    # Check for incomplete game
    if any(matrix[i][j] == '' for i in range(n) for j in range(n)):
        return -1

    # If no winner and no empty cells, it's a draw
    return 0

# Example usage
matrix = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['X', 'O', 'X']
]

print("Current Board:")
draw_tic_tac_toe_board(matrix)
result = check_tic_tac_toe(matrix)
print("Game Result:", result) # Expected Output: 1 (Player 1 wins)

# Testing the label to index conversion
print(convert_label_to_index("A1")) # Output: (0, 0)
print(convert_label_to_index("C3")) # Output: (2, 2)
```

```

def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    row_labels = [chr(i) for i in range(ord('A'), ord('A') + n)]

    def draw_line():
        # Draw the horizontal border with column labels
        print(' ' + '---' * n)
        print(' ' + ' '.join(f' {i+1} ' for i in range(n)))
        print(' ' + ' ' + '---' * n)

    def draw_row(index, row):
        # Draw a row with row label and the content
        print(f'{row_labels[index]} ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for

    # Draw the board
    draw_line() # Draw the top border
    for i, row in enumerate(matrix):
        draw_row(i, row) # Draw the current row with the row label
        draw_line() # Draw the horizontal border after the row

def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
                all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
            all(matrix[i][n - 1 - i] == player for i in range(n)):
            return True

        return False

    # Check for winner
    player1_wins = check_winner('X')
    player2_wins = check_winner('O')

    if player1_wins:

```

```

        return 1
    if player2_wins:
        return 2

    # Check for incomplete game
    if any(matrix[i][j] == '' for i in range(n) for j in range(n)):
        return -1

    # If no winner and no empty cells, it's a draw
    return 0

# Example usage
matrix = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['X', 'O', 'X']
]

print("Current Board:")
draw_tic_tac_toe_board(matrix)
result = check_tic_tac_toe(matrix)
print("Game Result:", result) # Expected Output: 1 (Player 1 wins)

# Testing the label to index conversion
print(convert_label_to_index("A1")) # Output: (0, 0)
print(convert_label_to_index("C3")) # Output: (2, 2)

```



Current Board:

```

-----
 1   2   3
-----
A X  |  O  |  X  |
-----
 1   2   3
-----
B O  |  X  |  O  |
-----
 1   2   3
-----
C X  |  O  |  X  |
-----
 1   2   3
-----

```

```

Game Result: 1
(0, 0)
(2, 2)

```

Exercise 7: Write a function that takes a board, player number, and location specified as in exercise 6 and then calls exercise 5 to correctly modify the board.

```

def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print("Initial Board:")
draw_tic_tac_toe_board(board)

# Place markers
print(update_board(board, 1, "A1")) # Output: True (Valid placement)
print(update_board(board, 2, "B2")) # Output: True (Valid placement)

```



```
print(update_board(board, 1, "A1")) # Output: False (Cell already occupied)
print(update_board(board, 2, "D4")) # Output: False (Out of bounds)

print("Updated Board:")
draw_tic_tac_toe_board(board)
```

```
def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print("Initial Board:")
draw_tic_tac_toe_board(board)

# Place markers
print(update_board(board, 1, "A1")) # Output: True (Valid placement)
print(update_board(board, 2, "B2")) # Output: True (Valid placement)
```

```

print(update_board(board, 1, "A1")) # Output: False (Cell already occupied)
print(update_board(board, 2, "D4")) # Output: False (Out of bounds)

print("Updated Board:")
draw_tic_tac_toe_board(board)

```



Initial Board:

```

-----
 1   2   3
-----
A   |   |   |
-----
 1   2   3
-----
B   |   |   |
-----
 1   2   3
-----
C   |   |   |
-----
 1   2   3
-----

```

True

True

False

False

Updated Board:

```

-----
 1   2   3
-----
A  X  |   |   |
-----
 1   2   3
-----
B   |  O  |   |
-----
 1   2   3
-----
C   |   |   |
-----
 1   2   3
-----

```

Exercise 8: Write a function is called with a board and player number, takes input from the player using python's `input`, and modifies the board using your function from exercise 7. Note that you should keep asking for input until you have gotten a valid input that results in a valid move.

```
def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

def get_valid_move(board, player):
    while True:
        # Get input from the player
        move = input(f"Player {player} (enter move in format 'A1'): ").strip().upper()

        # Check if the input is in the correct format
        if len(move) < 2:
            print("Invalid input format. Please use format 'A1'.")
            continue

        row_label = move[0]
        col_label = move[1:]
```



```

# Check if row_label is valid
if not row_label.isalpha() or not col_label.isdigit():
    print("Invalid input format. Row should be a letter and column should be a number")
    continue

# Check if col_label is a valid number
if not col_label.isdigit() or int(col_label) < 1:
    print("Invalid column number. It should be a positive integer.")
    continue

# Check if the move is valid and place the marker
if update_board(board, player, move):
    print(f"Move {move} placed successfully.")
    break
else:
    print("Invalid move. Cell is already occupied or out of bounds.")

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print("Initial Board:")
draw_tic_tac_toe_board(board)

# Simulate input for Player 1 and Player 2
get_valid_move(board, 1) # Player 1 makes a move
print("Updated Board:")
draw_tic_tac_toe_board(board)

get_valid_move(board, 2) # Player 2 makes a move
print("Updated Board:")
draw_tic_tac_toe_board(board)
Explanation:
convert_label_to_index(label):

Converts a location label like "A2" into matrix indices.
place_marker(board, player, x, y):

Places the player's marker on the board if the cell is empty and the coordinates are valid.
update_board(board, player, location):

Uses the label to place the marker on the board.
get_valid_move(board, player):

Continuously prompts the user for input until a valid move is entered.
Validates the input format and checks if the location is valid and empty.
Calls update_board to place the marker if the move is valid.
This implementation ensures that the board is updated correctly and handles invalid input gracefully.

```



```
def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

def get_valid_move(board, player):
    while True:
        # Get input from the player
        move = input(f"Player {player} (enter move in format 'A1'): ").strip().upper()

        # Check if the input is in the correct format
        if len(move) < 2:
            print("Invalid input format. Please use format 'A1'.")
            continue

        row_label = move[0]
        col_label = move[1:]
```



```

# Check if row_label is valid
if not row_label.isalpha() or not col_label.isdigit():
    print("Invalid input format. Row should be a letter and column should be a number")
    continue

# Check if col_label is a valid number
if not col_label.isdigit() or int(col_label) < 1:
    print("Invalid column number. It should be a positive integer.")
    continue

# Check if the move is valid and place the marker
if update_board(board, player, move):
    print(f"Move {move} placed successfully.")
    break
else:
    print("Invalid move. Cell is already occupied or out of bounds.")

# Example usage
board = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
]

print("Initial Board:")
draw_tic_tac_toe_board(board)

# Simulate input for Player 1 and Player 2
get_valid_move(board, 1) # Player 1 makes a move
print("Updated Board:")
draw_tic_tac_toe_board(board)

get_valid_move(board, 2) # Player 2 makes a move
print("Updated Board:")
draw_tic_tac_toe_board(board)

```



Initial Board:

```

-----
1   2   3
-----
A   |   |   |
-----
1   2   3
-----
B   |   |   |
-----

```

```

      1   2   3
      -----
C      |   |   |
      -----
      1   2   3
      -----
Player 1 (enter move in format 'A1'): 1
Invalid input format. Please use format 'A1'.
Player 1 (enter move in format 'A1'): A1
Move A1 placed successfully.
Updated Board:

```

```

      1   2   3
      -----
A  X   |   |   |
      -----

```

```

      1   2   3
      -----
B      |   |   |
      -----

```

```

      1   2   3
      -----
C      |   |   |
      -----

```

```

      1   2   3
      -----
Player 2 (enter move in format 'A1'): C3
Move C3 placed successfully.
Updated Board:

```

```

      1   2   3
      -----
A  X   |   |   |
      -----

```

```

      1   2   3
      -----
B      |   |   |
      -----

```

```

      1   2   3
      -----
C      |   |  O   |
      -----

```

```

      1   2   3
      -----

```

Exercise 9: Use all of the previous exercises to implement a full tic-tac-toe game, where an appropriate board is drawn, 2 players are repeatedly asked for a location coordinates of where they wish to place a mark, and the game status is checked until a player wins or a draw occurs.

```

def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    row_labels = [chr(i) for i in range(ord('A'), ord('A') + n)]

    def draw_line():
        # Draw the horizontal border with column labels
        print(' ' + '---' * n)
        print(' ' + ' '.join(f' {i+1} ' for i in range(n)))
        print(' ' + ' ' + '---' * n)

    def draw_row(index, row):
        # Draw a row with row label and the content
        print(f'{row_labels[index]} ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for

    # Draw the board
    draw_line() # Draw the top border
    for i, row in enumerate(matrix):
        draw_row(i, row) # Draw the current row with the row label
        draw_line() # Draw the horizontal border after the row

def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

```



```

def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
                all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
            all(matrix[i][n - 1 - i] == player for i in range(n)):
            return True

        return False

    # Check for winner
    player1_wins = check_winner('X')
    player2_wins = check_winner('O')

    if player1_wins:
        return 1
    if player2_wins:
        return 2

    # Check for incomplete game
    if any(matrix[i][j] == ' ' for i in range(n) for j in range(n)):
        return -1

    # If no winner and no empty cells, it's a draw
    return 0

def get_valid_move(board, player):
    while True:
        # Get input from the player
        move = input(f"Player {player} (enter move in format 'A1'): ").strip().upper()

        # Check if the input is in the correct format
        if len(move) < 2:
            print("Invalid input format. Please use format 'A1'.")
            continue

        row_label = move[0]

```

```

col_label = move[1:]

# Check if row_label is valid
if not row_label.isalpha() or not col_label.isdigit():
    print("Invalid input format. Row should be a letter and column should be a number")
    continue

# Check if col_label is a valid number
if not col_label.isdigit() or int(col_label) < 1:
    print("Invalid column number. It should be a positive integer.")
    continue

# Check if the move is valid and place the marker
if update_board(board, player, move):
    print(f"Move {move} placed successfully.")
    break
else:
    print("Invalid move. Cell is already occupied or out of bounds.")

def play_tic_tac_toe():
    n = 3 # Standard tic-tac-toe board size
    board = [['' for _ in range(n)] for _ in range(n)]

    print("Initial Board:")
    draw_tic_tac_toe_board(board)

    current_player = 1

    while True:
        print(f"\nPlayer {current_player}'s turn:")
        get_valid_move(board, current_player)
        draw_tic_tac_toe_board(board)

        result = check_tic_tac_toe(board)

        if result == 1:
            print("Player 1 wins!")
            break
        elif result == 2:
            print("Player 2 wins!")
            break
        elif result == 0:
            print("It's a draw!")
            break

    # Switch player
    current_player = 2 if current_player == 1 else 1

```

```

def draw_tic_tac_toe_board(matrix):
    n = len(matrix)
    row_labels = [chr(i) for i in range(ord('A'), ord('A') + n)]

    def draw_line():
        # Draw the horizontal border with column labels
        print(' ' + '---' * n)
        print(' ' + ' '.join(f' {i+1} ' for i in range(n)))
        print(' ' + ' ' + '---' * n)

    def draw_row(index, row):
        # Draw a row with row label and the content
        print(f'{row_labels[index]} ' + ' | '.join(f' {cell} ' if cell != '' else ' ' for cell in row))

    # Draw the board
    draw_line() # Draw the top border
    for i, row in enumerate(matrix):
        draw_row(i, row) # Draw the current row with the row label
        draw_line() # Draw the horizontal border after the row

def convert_label_to_index(label):
    row_label = label[0]
    col_label = label[1:]

    row_index = ord(row_label.upper()) - ord('A')
    col_index = int(col_label) - 1

    return row_index, col_index

def place_marker(board, player, x, y):
    n = len(board)

    # Validate player number
    if player not in (1, 2):
        return False

    # Validate coordinates
    if not (0 <= x < n and 0 <= y < n):
        return False

    # Determine the marker to place
    marker = 'X' if player == 1 else 'O'

    # Check if the location is empty
    if board[x][y] == '':
        board[x][y] = marker
        return True

    # If the location is not empty, return False
    return False

```

```
def update_board(board, player, location):
    # Convert the label to indices
    x, y = convert_label_to_index(location)

    # Place the marker on the board
    return place_marker(board, player, x, y)

def check_tic_tac_toe(matrix):
    n = len(matrix)

    def check_winner(player):
        # Check rows and columns
        for i in range(n):
            if all(matrix[i][j] == player for j in range(n)) or \
               all(matrix[j][i] == player for j in range(n)):
                return True

        # Check diagonals
        if all(matrix[i][i] == player for i in range(n)) or \
```