```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <Keypad.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_LEDBackpack.h>
#include <mosquitto.h>

// Replace these with your WiFi credentials
const char* ssid = "dany_5G";
const char* password = "Adms1965&";

// Replace this with the IP address of your MQTT broker
const char* mqtt_server = "195.27.52.255";
#define BROKER_PORT 1883

WiFiClient espClient;
PubSubClient client(espClient);
Adafruit_8x8matrix matrix = Adafruit_8x8matrix();

// Matrix keypad pins
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};

byte rowPins[ROWS] = {16, 17, 18, 19};
byte colPins[COLS] = {22, 23, 21};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// Game state variables
char board[3][3] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}};
char turn = 'X';
bool gameOver = false;

void setup() {
  Serial.begin(115200);

  // Connect to WiFi
  Serial.print("Connecting to WiFi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("Connected to WiFi!");

  // Connect to MQTT broker
  client.setServer(mqtt_server, 1883);
  while (!client.connected()) {
    Serial.print("Connecting to MQTT broker: ");
    Serial.println(mqtt_server);
    if (client.connect("ESP32Client")) {
      Serial.println("Connected to MQTT broker!");
    } else {
      Serial.print("Failed to connect to MQTT broker. Error code: ");
      Serial.println(client.state());
      delay(2000);
    }
  }

  // Initialize matrix
  matrix.begin(0x70);

  // Subscribe to MQTT topics
  client.subscribe("led_matrix");
  client.subscribe("game_over");
}

void loop() {
```

```
  // Check for incoming MQTT messages
  client.loop();

  // Check for key press on matrix keypad
  char key = keypad.getKey();
  if (key != NO_KEY && !gameOver) {
    // Convert key press to board coordinates
    int x, y;
    switch (key) {
      case '1': x = 0; y = 0; break;
      case '2': x = 0; y = 1; break;
      case '3': x = 0; y = 2; break;
      case '4': x = 1; y = 0; break;
      case '5': x = 1; y = 1; break;
      case '6': x = 1; y = 2; break;
      case '7': x = 2; y = 0; break;
      case '8': x = 2; y = 1; break;
      case '9': x = 2; y = 2; break;
      default: return;
    }

    // Make move if spot is empty
    if (board[x][y] == ' ') {
      board[x][y] = turn;
      turn = (turn == 'X') ? 'O' : 'X';
    }

    // Update LED matrix and check for game over
    updateMatrix();
    checkGameOver();

    client.publish("Play");
  }
}

// MQTT callback function
void callback(char* topic, byte* payload, unsigned int length) {
  // Convert payload to string
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  // Handle game over message
  if (String(topic) == "game_over") {
    gameOver = true;
  }
   // Convert payload to integer
  int move = atoi((char*)message->payload);

  // Make move if spot is empty and game is not over
  if (move >= 0 && move <= 8 && board[move/3][move%3] == ' ' && !gameOver) {
    board[move/3][move%3] = turn;
    turn = (turn == 'X') ? 'O' : 'X';
  }
}

// Update LED matrix based on current game board
void updateMatrix() {
  matrix.clear();
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      if (board[i][j] == 'X') {
        matrix.drawPixel(i * 2, j * 2, LED_ON);
        matrix.drawPixel(i * 2 + 1, j * 2, LED_ON);
        matrix.drawPixel(i * 2, j * 2 + 1, LED_ON);
        matrix.drawPixel(i * 2 + 1, j * 2 + 1, LED_ON);
      } else if (board[i][j] == 'O') {
        matrix.drawCircle(i * 2 + 1, j * 2 + 1, 1, LED_ON);
      }
    }
  }
  matrix.writeDisplay();
}

void checkGameOver() {
  // Check rows
  for (int i = 0; i < 3; i++) {
```

```
    if (board[i][0] != ' ' && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {
      client.publish("game_over", (board[i][0] == 'X') ? "X wins!" : "O wins!");
      return;
    }
  }

  // Check columns
  for (int i = 0; i < 3; i++) {
    if (board[0][i] != ' ' && board[0][i] == board[1][i] && board[1][i] == board[2][i]) {
      client.publish("game_over", (board[0][i] == 'X') ? "X wins!" : "O wins!");
      return;
    }
  }

  // Check diagonals
  if (board[0][0] != ' ' && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {
    client.publish("game_over", (board[0][0] == 'X') ? "X wins!" : "O wins!");
    return;
  }
  if (board[0][2] != ' ' && board[0][2] == board[1][1] && board[1][1] == board[2][0]) {
    client.publish("game_over", (board[0][2] == 'X') ? "X wins!" : "O wins!");
    return;
  }

  // Check for draw
  bool draw = true;
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      if (board[i][j] == ' ') {
        draw = false;
        break;
      }
    }
  }
  if (draw) {
    client.publish("game_over", "Draw!");
  }
}
```