3D Tetris on a 3x3x12 LED Matrix

Welcome to the 3D Tetris project! This repository documents the creation of a 3x3x12 LED matrix that enables users to play a fully functional 3D Tetris game. The project leverages an ESP32-based development board (ESPduino) with integrated web server functionality for remote control.

Project Overview

This project was developed as part of the **"Applied Microcontroller Technology"** module and integrates hardware assembly, software programming, and web-based GUI design. Inspired by the game available at 3DTetris.de, the system enables interactive gameplay through a wireless web interface.

Key features include:

- A custom-built 3x3x12 RGB LED matrix.
- ESP32 microcontroller for controlling LEDs and hosting a local web server.
- A responsive web interface for gameplay and real-time interactions.
- Open-source software written in C, HTML, CSS, and JavaScript.

Features

Hardware

- **ESPduino Controller**: Combines the ESP32 Wi-Fi chip with Arduino-compatible GPIO layout.
- **3x3x12 LED Matrix**: Comprised of 108 individually addressable PL9823 RGB LEDs.
- **Custom Housing**: 3D-printed for modularity and ease of assembly.
- **Power Supply**: Powered via USB-C with an integrated on/off switch.

Software

- **LED Control**: Powered by the FastLED library for efficient matrix management.
- **Game Logic**: Implements Tetris gameplay mechanics including rotation, collision detection, and line clearing.
- **Web Interface**: HTML/CSS-based GUI hosted via an onboard web server, with multi-language support.

Gameplay

- Control Tetrominoes (Tetris blocks) through the GUI.
- Real-time rendering of blocks on the LED matrix.
- Score tracking and high-score leaderboard.
- Adjustable levels and difficulty scaling.
- High-score table to save and display the top 10 scores.
- Difficulty selection for custom gameplay experiences.

Getting Started

Prerequisites

Assembly

- Circuit: The LED matrix is powered and controlled through a single GPIO pin of the ESPduino. A 5V USB-C power supply is used for both the matrix and the ESPduino.
- LED Matrix: LEDs are arranged in a 3x3x12 grid and tested individually for functionality. Custom fixtures were used to ensure precision during soldering.
- 3D printed Case: The case, 3D-printed using PLA, features dedicated mounts for the ESPduino, USB ports, and optional counterweights to enhance stability.
- How the Code Works: The ESP32 Tetris game is explained using a flowchart, detailing initialization, web interface, user input processing, game logic, scoring, and game over handling.

Software

- **Arduino IDE**: Download and install the Arduino IDE.
- **ESP32 Board Manager**: Add the ESP32 board manager by following this guide.
- **Libraries**: Install the following libraries:
  - Via Arduino IDE Library Manager:
    - `FastLED`
    - `WiFi`
    - `Preferences`
  - **ESPForm Library**:
    - Download the library from the ESPForm GitHub repository.
    - Extract the downloaded ZIP file.
    - Copy the extracted folder into the `libraries` directory of your Arduino IDE sketchbook folder.

Installation

1. **Clone the Repository**
   Open a terminal and run the following command to clone the repository:

   ```
   git clone https://github.com/18Markus1984/3D-Tetris.git
   ```

2. **Open the Project in Arduino IDE**
   Navigate to the cloned folder and open the `.ino` file in the Arduino IDE.

3. **Configure the ESP32 Board**
   - Go to **Tools** > **Board** and select **ESP32 Dev Module**.
   - Enable **PSRAM** under **Tools** > **PSRAM**.

4. **Upload the Sketch**
   - Ensure the correct COM port is selected under **Tools** > **Port**.
   - Click the **Upload** button to flash the code onto the ESPduino.

5. **Connect to the ESP32's Wi-Fi Network**
   - On your computer or mobile device, connect to the Wi-Fi network:
     - **SSID**: `3D-Tetris`
     - **Password**: `12345678`

6. **Access the Web Interface**
   - Open a web browser and navigate to:
     - http://192.168.4.1

You can now enjoy 3D Tetris using the interactive web interface!

How It Works

Code Flow Chart

```
graph TD;
    Start["ESP32 starts"] --> InitWiFi["Initialize WiFi"];
    InitWiFi --> LoadWebPage["Load webpage from html.h"];
    LoadWebPage --> WaitForClient["Wait for client requests"];

    WaitForClient -->|Start button pressed| StartGame["Start game"];
    StartGame --> GameLoop["Game loop"];

    GameLoop -->|Movement input received| UpdatePosition["Update piece position"];
    UpdatePosition --> GameLoop;

    GameLoop -->|Rotation input received| RotatePiece["Rotate piece"];
    RotatePiece --> GameLoop;

    GameLoop -->|Piece falls| CheckCollision["Check collision"];
    CheckCollision -->|Collision detected| PlacePiece["Place piece"];
    PlacePiece --> CheckLines["Check for complete lines"];
    CheckLines -->|Lines found| RemoveLines["Remove lines"];
    RemoveLines --> UpdateScore["Update score"];
    UpdateScore --> GameLoop;

    CheckLines -->|No lines| GameLoop;

    GameLoop -->|Game over| GameOver["Game Over"];
    GameOver --> WaitForClient;
```

LED Matrix

- **Construction**: LEDs are arranged in a 3x3x12 format, connected using PL9823-compatible drivers.
- **Control**: Each LED is addressed individually using a mapped index for 3D coordinates.

Web Interface

- **Languages Supported**: English, German, French, Spanish, Italian.
- **Functions**:
  - Rotate blocks around X, Y, Z axes.
  - Shift blocks along the XY-plane.
  - Drop blocks into place.
- **Additional Features**:
  - Live preview of the next Tetromino.
  - Display of current score and level.
  - **High-Score Table**: Saves and displays the top 10 scores along with player names. Extra entries are shown but not stored persistently.
  - **Difficulty Selection**: Choose between multiple difficulty levels to adjust the gameplay experience.

Results

The project met its primary objectives, showcasing:

1. A functional 3D Tetris game on a custom LED matrix.
2. An intuitive and accessible web-based control interface.
3. High scalability for future enhancements, including battery power and improved stability.

Future Improvements

- Reduce input latency for smoother gameplay.
- Enhance matrix stability to minimize movement during transport.
- Add battery support for portability.
- Optimize collision detection and rotation algorithms to avoid edge-case errors.

Acknowledgments

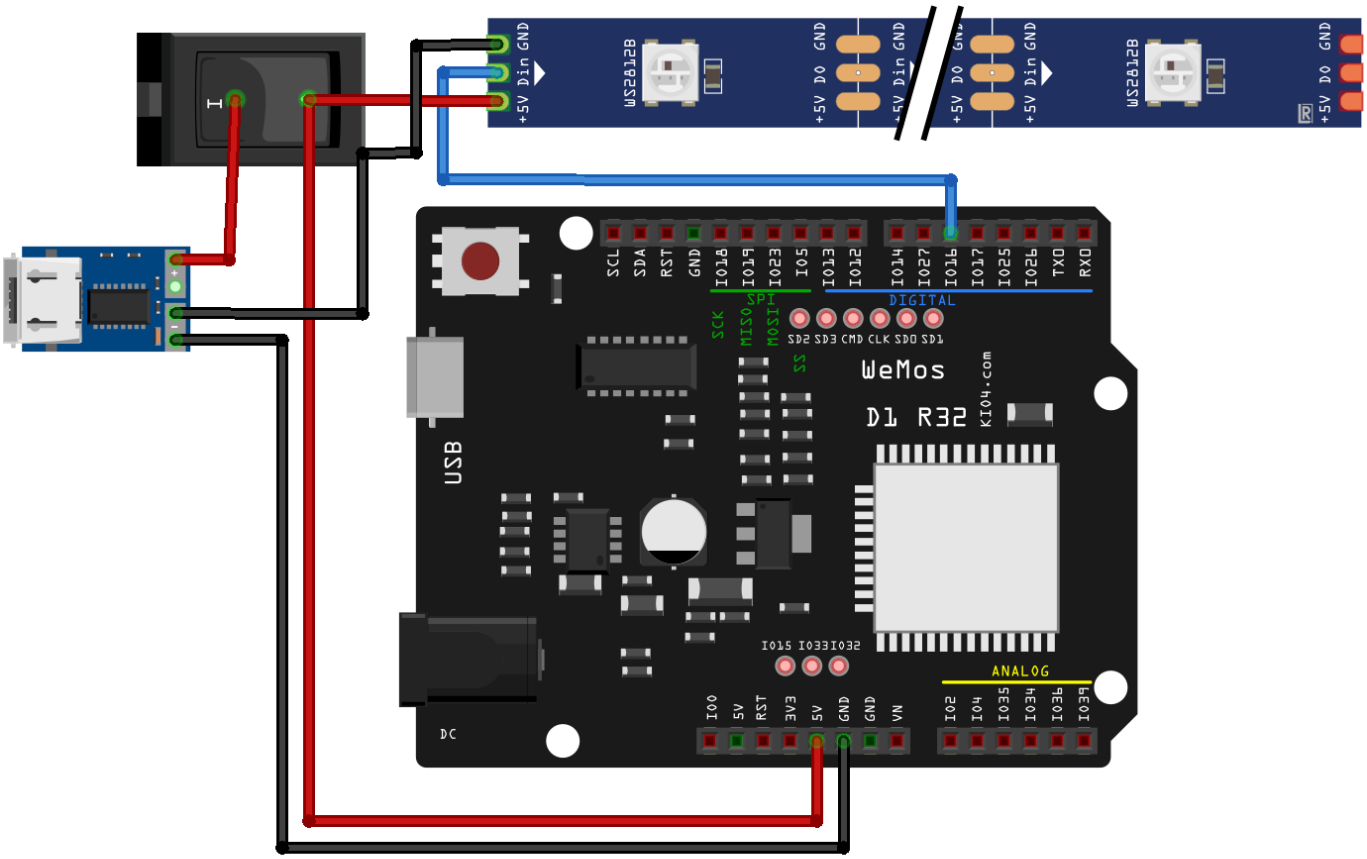
This project was developed by Marvin Heins, Markus Schmidt, and Leonard Holz-Diehl. Special thanks to the open-source community and the developers of the FastLED library.

License

Materials Required

- ESPduino (ESP32-based)
- Toggle switch
- USB-C power supply
- Heat shrink tubing for securing connections.
- Insulated wires for connections.

Wiring Diagram



fritzing

Step-by-Step Guide

1. Connecting the ESPduino

- Solder the **Digital Pin 16** of the ESPduino to the data input of the LED matrix.
- Connect the **5V** and **GND** lines from the matrix to the ESPduino.

2. Adding a Power Switch

- Install a toggle switch between the power source and the circuit.
- Use heat shrink tubing to insulate all connections.

3. Testing the Circuit

- Upload a simple animation program to the ESPduino.
- Verify that the entire matrix responds as expected.
- Check for loose or misaligned connections and fix them if necessary.

Troubleshooting Tips

- Signal continuity: Verify that the cabel for the Digital Pin 16 is connected to the correct pin and doesn't have a loose contact

Materials Required

To construct the 3x3x12 LED matrix, you will need the following materials:

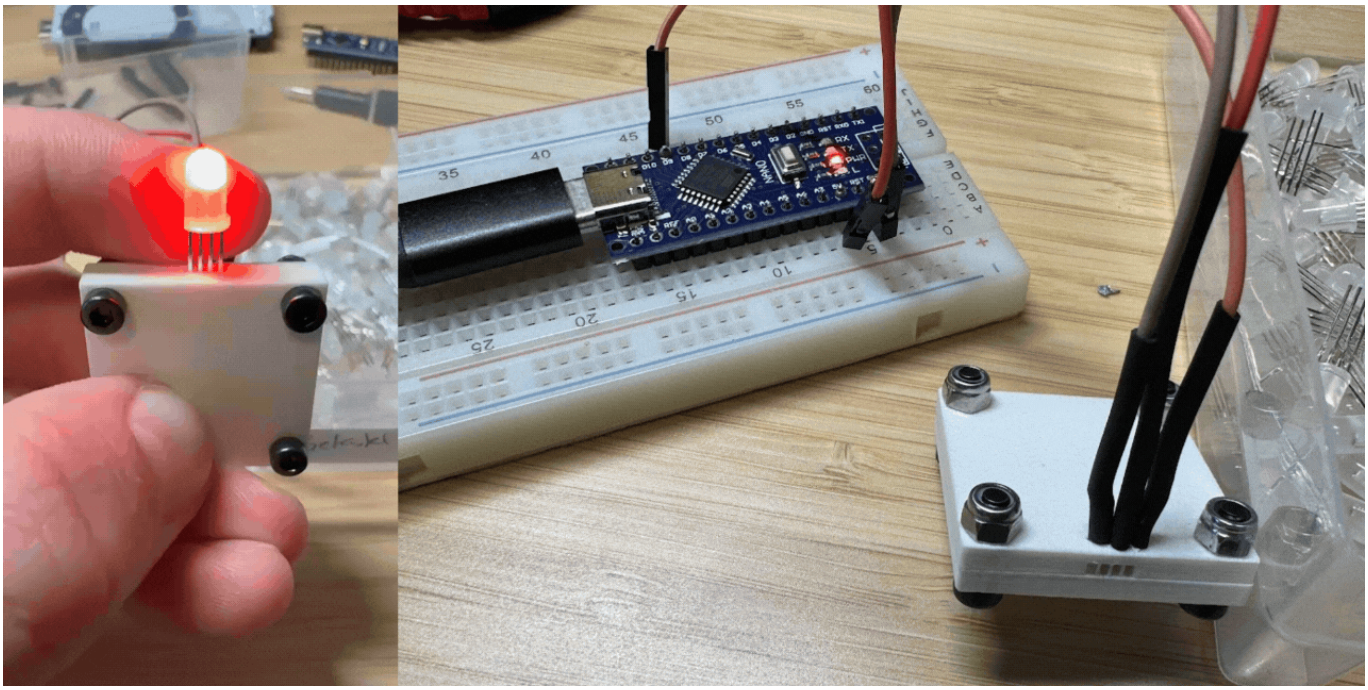
- 108 PL9823 RGB LEDs
- Soldering station with fine tips.
- Solder wire (preferably lead-free).
- Welding rods 0.8mm for connections.
- 3D-printed alignment guides and 8mm Steel rods.

Step-by-Step Assembly Guide

1. **Testing the LEDs**

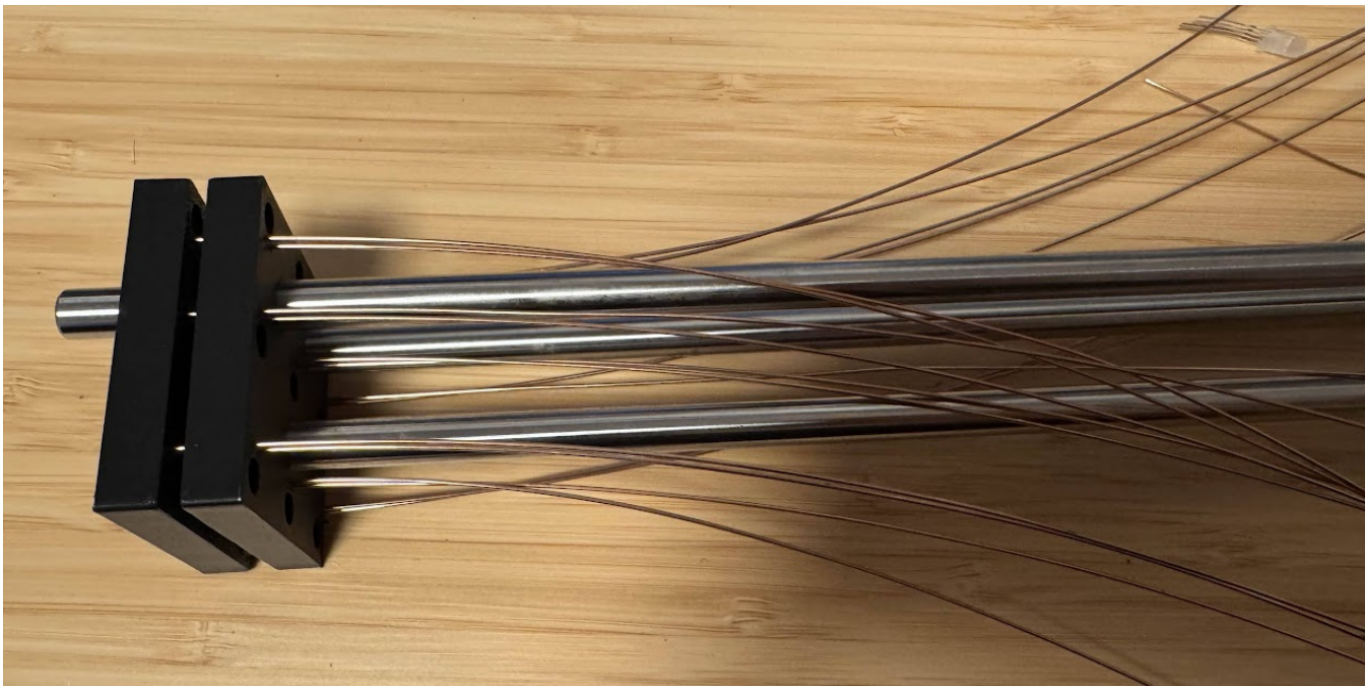
Before assembly, test each LED to ensure it functions correctly.

- Use a custom-built tester, connecting each LED to an Arduino and running a test program.
- Verify that the LEDs light up and change colors correctly.



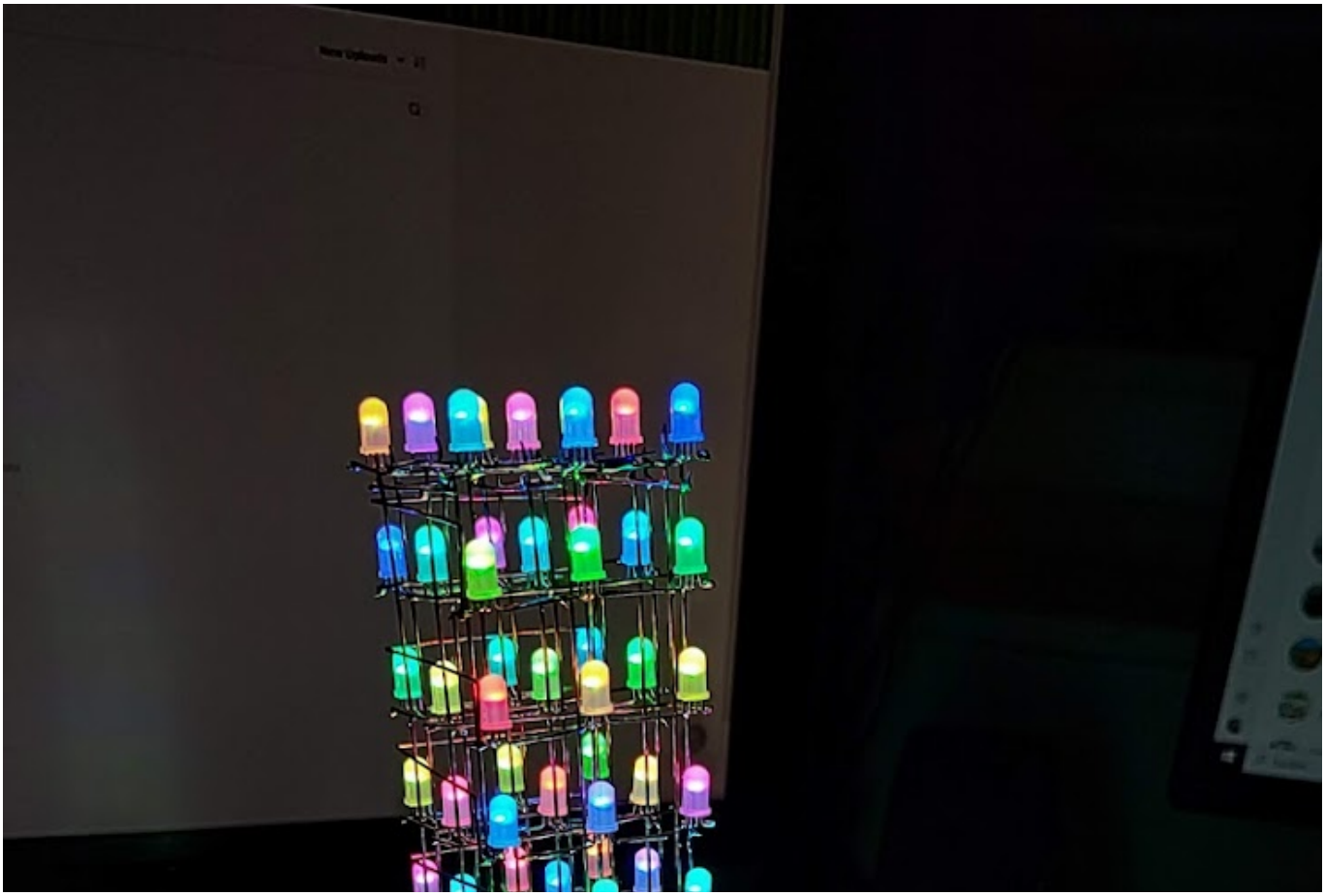
2. **Preparing the Matrix Frame**

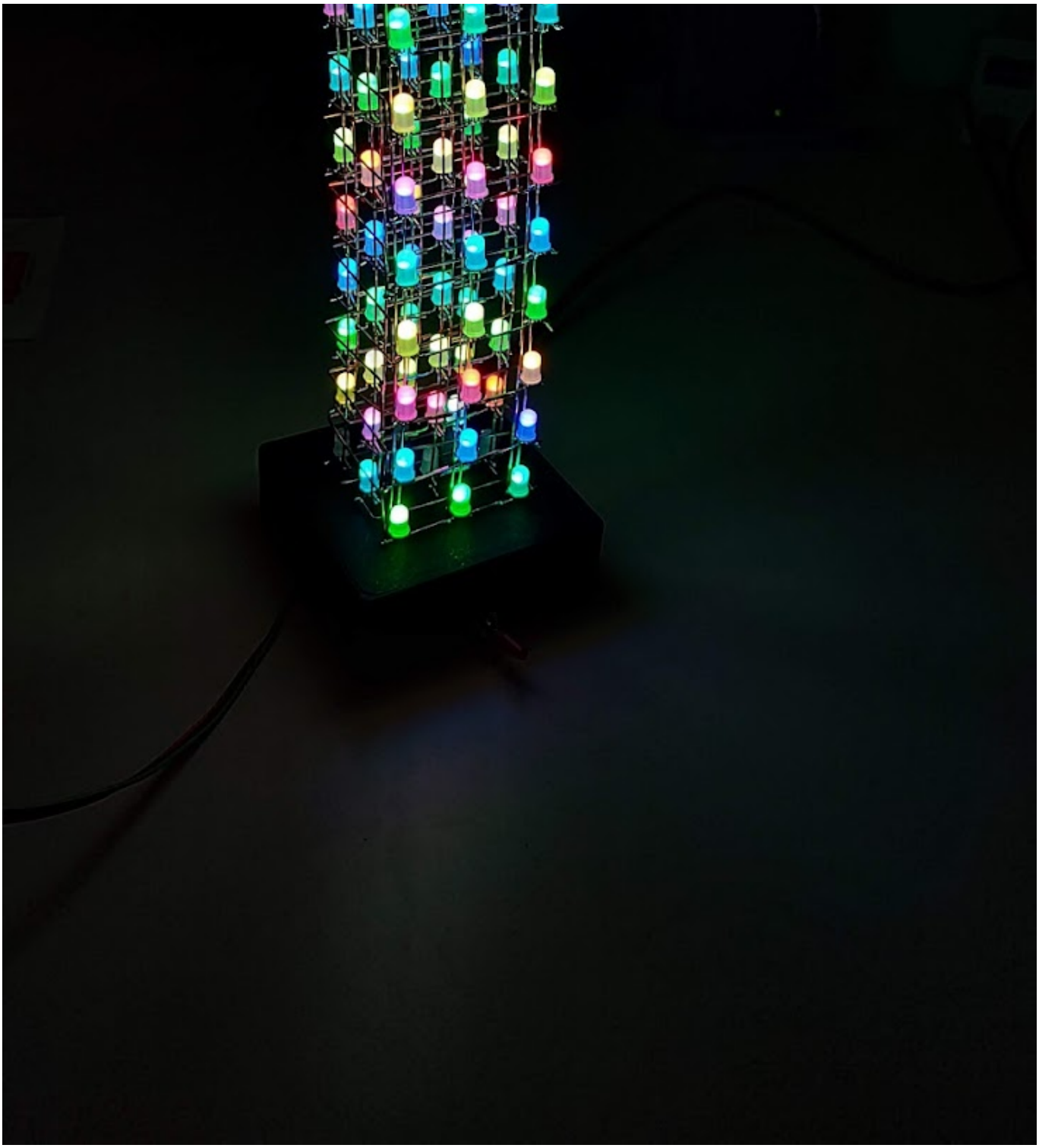
- Print the alignment guides using the provided 3D models.
- These guides ensure that all LEDs are positioned accurately for soldering.
- Insert each LED into the guide, aligning the pins consistently.



3. **Soldering the LEDs**

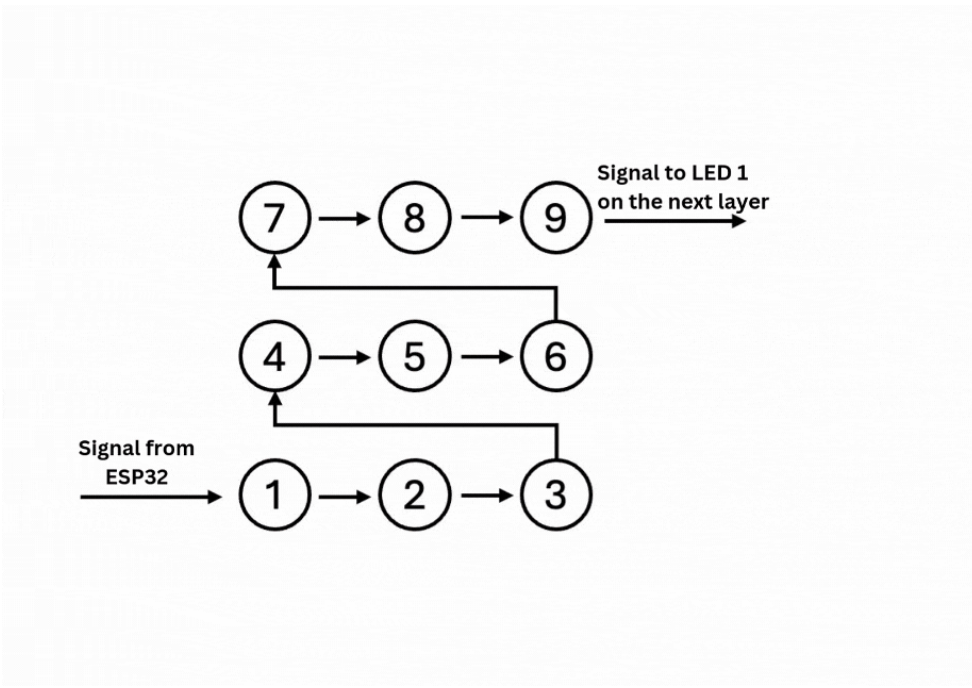
- Begin by soldering the power connections:
  - Pin 2 (5V) of each LED should connect to a common power bus.
  - Pin 3 (GND) connects to a ground bus.
- Use welding rods cut to size for the power and ground rails.
  - Straighten the rods using a drill and vice to ensure clean alignment.
  - Be sure the rods are straight, or your matrix will end up with a bend in it, like you can see in the picture.

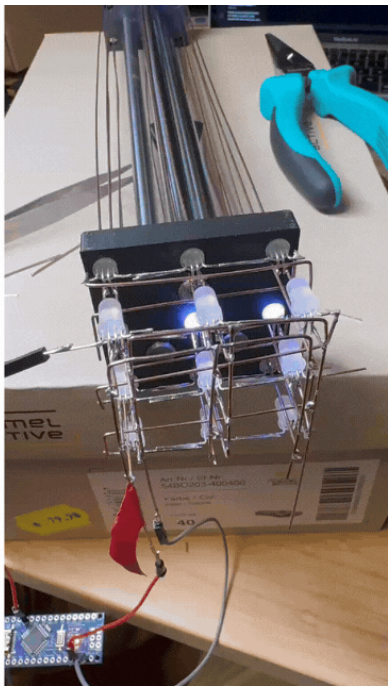




4. **Connecting Signal Lines**

- Solder the **DIN** (digital input) of each LED to the **DOUT** (digital output) of the previous LED in the chain.
- Follow the predefined order to ensure signal continuity.
- Test connections after completing each layer to catch errors early.



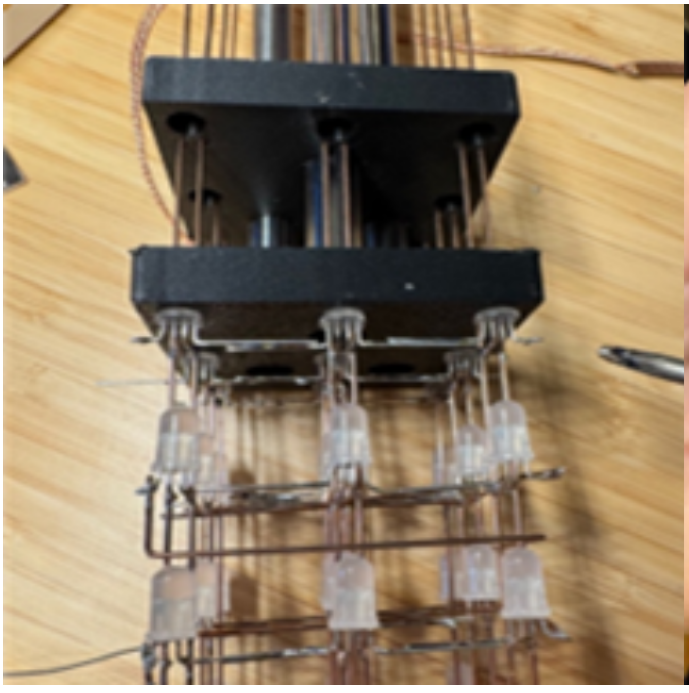


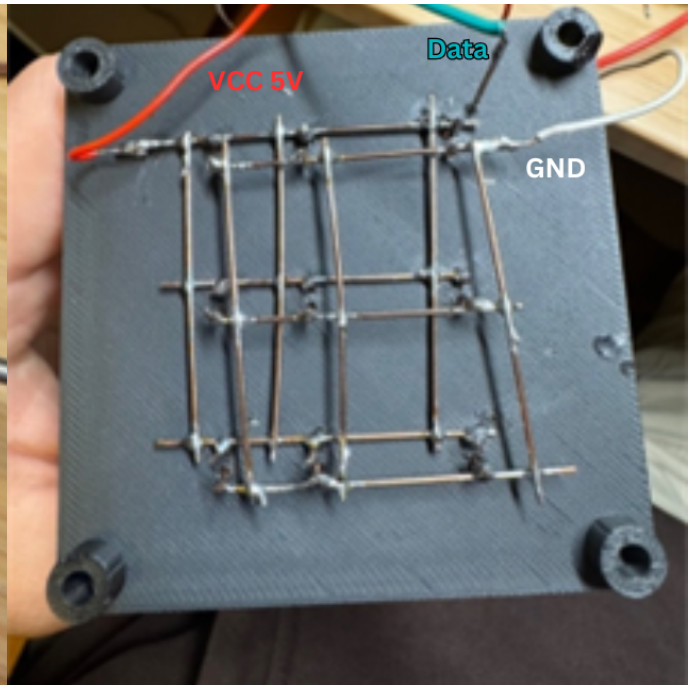
5. **Layer-by-Layer Assembly**

- Build the matrix layer by layer, starting from the bottom.
- After completing a layer, connect it to the previous one using small jumper wires or rods for the signal line.
- Ensure each layer is tested before proceeding to the next.

6. **Finalizing the Matrix**

- Once all 12 layers are complete, attach the matrix to its 3D-printed base.
- Solder the final connections for power, ground, and data lines to the ESPduino.
- Secure all connections with heat shrink tubing or electrical tape for durability.





Troubleshooting Tips

- **Non-functioning LEDs**: Double-check connections and test individual LEDs if needed.
- **Signal issues**: Ensure proper alignment and avoid crossing wires. Use a multimeter to test continuity.
- **Structural integrity**: Reinforce the matrix with additional supports if necessary to prevent wobbling.
- **Straighten the rods**: Make sure that the rods are properly straightened, so that your Matrix doesn't get a curved shape.

Tools and References

For further guidance on soldering techniques and matrix assembly, you can consult:

- The FastLED Library Documentation for programming and troubleshooting LEDs.
- The video tutorial "Make Your Own SIMPLE 5x5x5 RGB LED Cube" by GreatScott for inspiration.

Materials Required

- 3D-printed components (CAD files provided in NX12 format)
- Brass inserts with M3 threads (16 in total)
- M3 screws for assembly
- Plexiglas sheet for the outer cover
- Self-tapping screws for the USB-C port
- Counterweight (optional, for added stability)

Design Overview

The case was designed in NX12 to be compact and easy to 3D print, with minimal contact on the print bed. It includes:

1. **Matrix Holder**: Positioned to support the LED matrix during soldering and assembly. Pre-aligned holes ensure proper orientation of the matrix wires.
2. **Main Body**: Contains slots for the ESPduino mount, USB-C port, and toggle switch.
3. **Plexiglas Cover**: Attaches securely to the base with M3 screws.
4. **Counterweight Mount**: Located at the bottom of the case to prevent tipping.

Step-by-Step Assembly Guide

1. Preparing the Case

- Print the case components using PLA or a similar material. Ensure all dimensions are accurate for proper alignment.
- Insert brass M3-threaded inserts into the designated holes using a soldering iron.
  - **12 inserts** are used for securing internal components (e.g., ESPduino mount, Plexiglas cover).
  - **4 additional inserts** at the bottom allow for attaching a counterweight.

2. Attaching the Matrix

- Secure the LED matrix to its holder using M3 screws. Ensure the wires are aligned through the designated holes in the holder for clean routing.

3. Mounting the ESPduino

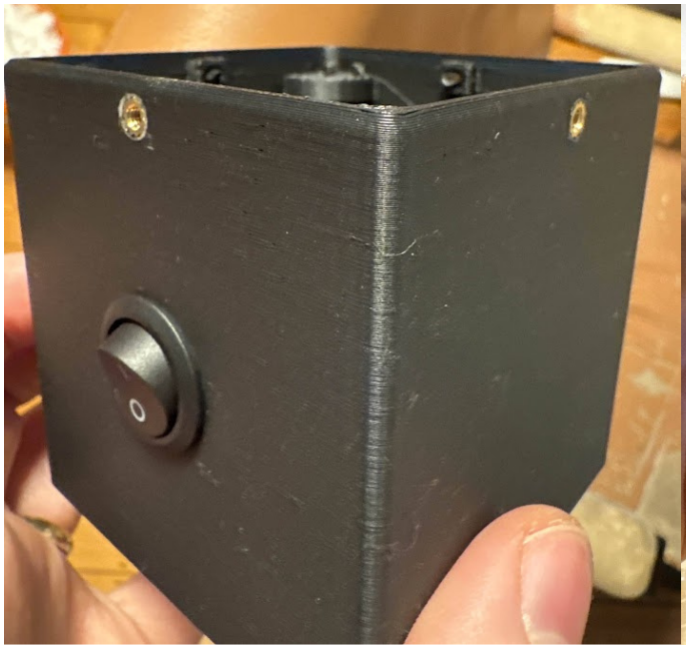
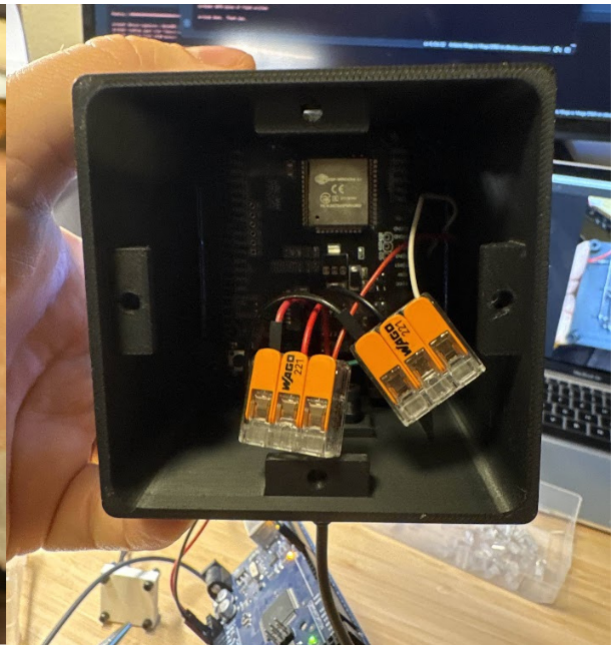
- Attach the ESPduino to its 3D-printed mount and fasten it inside the case.
- Align the ESPduino's USB and GPIO ports with the case openings for easy access.

4. Installing the USB-C Port

- Mount the USB-C port into its designated slot using self-tapping screws.
- Place heat-resistant washers on the screws to prevent damage to the case.

5. Adding the Toggle Switch

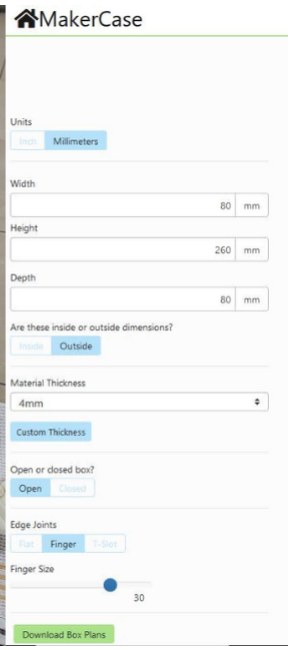
- Secure the toggle switch in its allocated opening.
- Connect the switch to the power lines using terminal connectors or soldering.

6. Making the Plexiglas cover

- Adjust the values on the box making side MakerCase. You can see the values in the picture. Modify the thickness for the kind of plexiglas you are using. Or use the manufacturing files we provided for 4mm thick plexiglas.
- Use some clamps and the holder to secure the cover while gluing.



7. Finalizing the Case

- Attach the Plexiglas cover to the main body using M3 screws.
- Optionally, screw a counterweight into the bottom mounts to enhance stability.

Assembly Order Notes

1. Insert brass inserts before assembling any components.
2. Mount the matrix first, followed by the ESPduino.
3. Install the USB-C port and toggle switch after mounting the main components.
4. Ensure all wires are routed neatly to avoid interference with the case or other parts.

Additional Tips

- Use thread-locking adhesive on screws to prevent loosening during operation.
- Test the electrical connections before sealing the case with the Plexiglas cover.
- For improved aesthetics, sand and paint the 3D-printed parts before assembly.

ESP32 Tetris Game Logic

```
flowchart LR;
    Start["ESP32 starts"] --> InitWiFi["Initialize WiFi"];
    InitWiFi --> LoadWebPage["Load webpage from html.h"];
    LoadWebPage --> WaitForClient["Wait for client requests"];

    WaitForClient -->|Start button pressed| StartGame["Start game"];
    StartGame --> GameLoop["Game loop"];

    GameLoop -->|Movement input received| UpdatePosition["Update piece position"];
    UpdatePosition --> GameLoop;

    GameLoop -->|Rotation input received| RotatePiece["Rotate piece"];
    RotatePiece --> GameLoop;

    GameLoop -->|Piece falls| CheckCollision["Check collision"];
    CheckCollision -->|Collision detected| PlacePiece["Place piece"];

    PlacePiece --> CheckLines["Check for complete lines"];

    CheckLines -->|Lines found| RemoveLines["Remove lines"] -->
UpdateScore["Update score"] --> GameLoop;
    CheckLines -->|No lines| GameLoop;

    GameLoop -->|Game over check| Decision{Game Over?};

    Decision -->|Yes| GameOver["Game Over"] --> WaitForClient;
    Decision -->|No| GameLoop;
```

1. ESP32 Initialization

(Steps: `ESP32 starts` → `Initialize WiFi`)

- When the ESP32 is powered on, it **initializes the WiFi connection**.
- The ESP32 can either **connect to an existing network** or **set up an access point (AP mode)**.
- This allows users to **connect via a web browser** and control the game remotely.

📌 **Code responsible for this step:**

```
WiFi.begin(ssid, password);  // Connect to WiFi network
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html); // Send HTML page
});
server.begin();  // Start web server
```

2. **Loading the Web Interface**

(Step: `Load webpage from html.h`)

- The ESP32 **hosts a local web page** stored in `html.h`.
- This **web page serves as the UI**, allowing players to interact with the game.
- The page includes:
  - A **start button** to begin the game.
  - **Control buttons** for moving and rotating the Tetris pieces.
  - A **canvas** to display the game board.

📌 **How the HTML is loaded:**

```
static const char *index_html = R"--espform--(
<!DOCTYPE html>
<html>
<head> <title>3D Tetris</title> </head>
<body>
    <button id="startButton">Start</button>
    <canvas id="gameCanvas"></canvas>
    <script> /* JavaScript to control game */ </script>
</body>
</html>
)--espform--";
```

---

3. **Waiting for Player Input**

(Step: `Wait for client requests`)

- The ESP32 is now **waiting for the player to interact** with the web interface.
- When the **"Start" button is pressed**, the ESP32 receives an HTTP request and starts the game.

📌 **JavaScript sends the start command:**

```
document.getElementById('startButton').addEventListener('click', function() {
    fetch('/startGame');  // Sends request to ESP32 to start the game
});
```

📌 **ESP32 processes the start request:**

```
server.on("/startGame", HTTP_GET, [](AsyncWebServerRequest *request) {
    isGameRunning = true;
    request->send(200, "text/plain", "Game Started");
});
```

---

4. **Game Loop Execution**

(Step: `Game loop`)

- The ESP32 runs a **loop** that continuously updates the game state.
- This includes **handling user inputs**, **moving pieces**, **checking collisions**, and **updating the display**.

📌 **Core game loop in ESP32:**

```
void loop() {
    if (isGameRunning) {
        checkInputs();  // Process movement/rotation inputs
        updateGame();   // Move the falling piece
        drawGame();     // Send updated game state to the webpage
    }
}
```

---

5. **Handling User Inputs**

(Steps: `Movement input received` → `Update piece position`)

- The ESP32 **receives input from the player** (e.g., move left, right, rotate).
- The piece position is updated accordingly.

📌 **JavaScript sends move request:**

```
document.getElementById('moveLeft').addEventListener('click', function() {
    fetch('/move?dir=left');  // Send request to ESP32
});
```

📌 **ESP32 processes movement:**

```
server.on("/move", HTTP_GET, [](AsyncWebServerRequest *request) {
    String direction = request->getParam("dir")->value();
    if (direction == "left") currentPiece.x--;
    else if (direction == "right") currentPiece.x++;
    request->send(200, "text/plain", "Moved");
});
```

---

6. **Checking Collision and Placing Pieces**

(Steps: `Piece falls` → `Check collision` → `Place piece`)

- The game **checks if the falling piece has hit the ground or another block**.
- If a collision is detected, the piece is **placed on the board**, and a new piece is generated.

📌 **Collision detection:**

```
bool checkCollision(Tetromino piece) {
    for (int i = 0; i < piece.blocks.length; i++) {
        int x = piece.blocks[i][0];
        int y = piece.blocks[i][1];
        if (board[y][x] != EMPTY) return true;  // Collision detected
    }
    return false;
}
```

---

7. **Clearing Lines and Updating Score**

(Steps: `Check for complete lines` → `Remove lines` → `Update score`**)**

- If a **row is completely filled**, it is removed, and the score is increased.
- The game **sends the updated score to the webpage**.

📌 **Checking and clearing lines:**

```
void checkLines() {
    for (int y = 0; y < BOARD_HEIGHT; y++) {
        if (isFullRow(y)) {
            removeRow(y);
            score += 10;  // Increase score
        }
    }
}
```

📌 **Updating the score on the webpage:**

```
fetch('/getScore').then(response => response.text()).then(score => {
    document.getElementById('scoreText').innerText = score;
});
```

---

8. **Game Over Check**

(Steps: `Game Over?` → `Yes` → `Game Over`**)**

- If a new piece **cannot be placed at the top**, the game **ends**.
- The ESP32 **sends a Game Over message to the webpage**.

📌 **Checking for Game Over:**

```
bool isGameOver() {
    return checkCollision(newPiece);
}
```

📌 **Displaying Game Over in JavaScript:**

```
if (gameOver) {
    document.getElementById('gameOverPopup').style.display = 'block';
}
```

Summary

- **ESP32 initializes WiFi** and **serves the game webpage**.
- **Player interacts with the web page**, and **ESP32 handles requests**.
- **Game loop runs**, processing **movement, rotation, collisions, and line clearing**.
- **Score updates dynamically**, and the **game ends when no more moves are possible**.