# Arduino Dot-Matrix Controllers

Jibran Somroo[1], Ezana Abebe[2]
[1,2]Department of Electrical and Software Engineering
[1]jibran.somroo@ucalgary.ca, [2]ezana.abebe@ucalgary.ca

***Abstract***— The goal of this lab was to design, simulate, and construct an embedded system that uses an IR remote and a joystick to control a 2x2 lighted square on an 8x8 dot matrix display. Important ideas like system operating modes, analog peripherals, device interfacing, and user interfacing design were introduced by the project. Initial development and testing were conducted in Wokwi, where the joystick and IR remote coded independently to optimize simulation performance. Both control techniques were merged in the final implementation, enable smooth transitions between joystick and remote operation. Challenges included unexpected IR remote hexadecimal values, which were resolved through debugging via the Arduino serial monitor. The completed circuit successfully maintained boundary constraints, responded to user inputs, and showed the current mode on a 16x2 LCD.

## I. Introduction

The focus of this lab is to design, simulate and build a system that employs an IR remote control and a joystick to place a 2x2 lighted square on an 8x8 dot matrix display. The project introduces key ideas such as system operation modes, analog peripherals, switch-case conditions, and interfacing with external devices. These abilities are fundamental for embedded systems and robotics applications. To simulate hardware and software interactions, we will first construct and test the system's code using the Wokwi online simulator. Once the simulation objectives are met, we will move on to building the physical system, assembling the elements, and uploading the code for real-world testing. The skills developed in this lab will be programming hardware (controlling the dot matrix and reading input from a joystick and IR remote), simulation and debugging (testing the system in a virtual environment before hardware implementation), device interfacing (manipulating the display's position via the joystick and remote control) and user interface design (displaying the current mode on the 16x2 LCD screen and ensuring the square stays within the boundaries of the matrix). Those who successfully complete this lab will have hands-on experience designing embedded systems, which will help them with more complex engineering projects and real-world applications.

## II. Simulations and Analysis

The design and simulation were created primarily in Wokwi using C code. When modelling the circuit in the simulation we used our previous model as a template as seen in Figure 1 as it contains the wiring for 16x2 LCD for the final model. To start however, the lab was split into two separate parts: the remote control and joystick. These two components were coded in separate Wokwi files due to a major problem we ran into. That being the compiling time of the simulation. Often during compilation of the simulation when the entire circuit contained all the components for the final version as seen in Figure 2 it caused a very noticeable increase of compilation time. What normally took around 5 seconds had started taking around 1 minute if the server even let it through. To combat this the lab was simply split into two parts as previously stated. During the process of creating the code for the joystick it was initially programmed so that whenever the user moves the joystick the 2x2 red lights in the 8x8 dot-matrix would simply move along with it. After adding basic boundaries and conditions once those boundaries were met the joystick control aspect of the lab was nearly completed. All that was left was creating a function that caused the 2x2 lights to move back in the middle once the joystick was let go. After the joystick coding component was completed the IR remote control component was created by recycling a majority of the components of the joystick model and code. To start with, the 8x8 dot-matrix wiring and the code that caused the 2x2 lights to return to the middle were borrowed from the joystick control code. During the coding of the IR remote it was very similar to that of the joystick but much more controllable if anything as when changing the location of the 2x2 lights in the dot matrix the fact that continuous movement didn’t have to be considered made it very easy. However, one problem that occurred was that when coding, though thankfully it was quickly found out due to the use of a serial monitor that when pressing a button, the IR remote it resulted in a different hexadecimal value being received then predicted in the sample codes. After fixing this issue by doing some basic debugging the IR remote control aspect of the circuit was fully functional. All that had to be done was integrating both circuits into one, which was especially easy as the IR remote control circuit was created by recycling the joystick model. The circuit that was created can be seen in Figure 2, once the circuit was completed all that had to be done was to create a simple function that allowed for switching modes between the joystick mode and IR remote control mode.

## III. Circuit Building

When fabricating the circuit, the design in Figure 2 was used as a blueprint to keep the process structured and organized. To start with, a GND and a VCC were set to the negative and positive parts of the terminal of the breadboard respectively. This was to ensure that each component could have access to ground and power as well as make it easier when integrating the power module. After that the most basic components were wired up first which included the 8x8 dot-matrix, the IR receiver and the joystick, this can be seen in Figure 3. During the process of fabricating the circuit, the components were tested using their individual component specific codes. During this time, we ran into a problem, a problem that we were very familiar with due to it having occurred in the simulation as well. That problem being that the remote resulted in various hexadecimal values being received that were not appropriate to the simulation. In other words, the remote was outputting values that didn’t weren’t being outputted in the simulation. However, as this problem was faced before it was relatively easy to fix as all that had to be done was opening the serial monitor in the Arduino IDE and figuring out which hexadecimal value corresponded with which buttons. Once the debugging of the IR remote was completed the rest of the circuit was built with relative ease due to prior experience of fabrication. When the entire circuit was built it was tested one final time as a whole with only the power module providing it power. And upon confirming that it worked in the way specified in the instructions the circuit was concluded as a success.

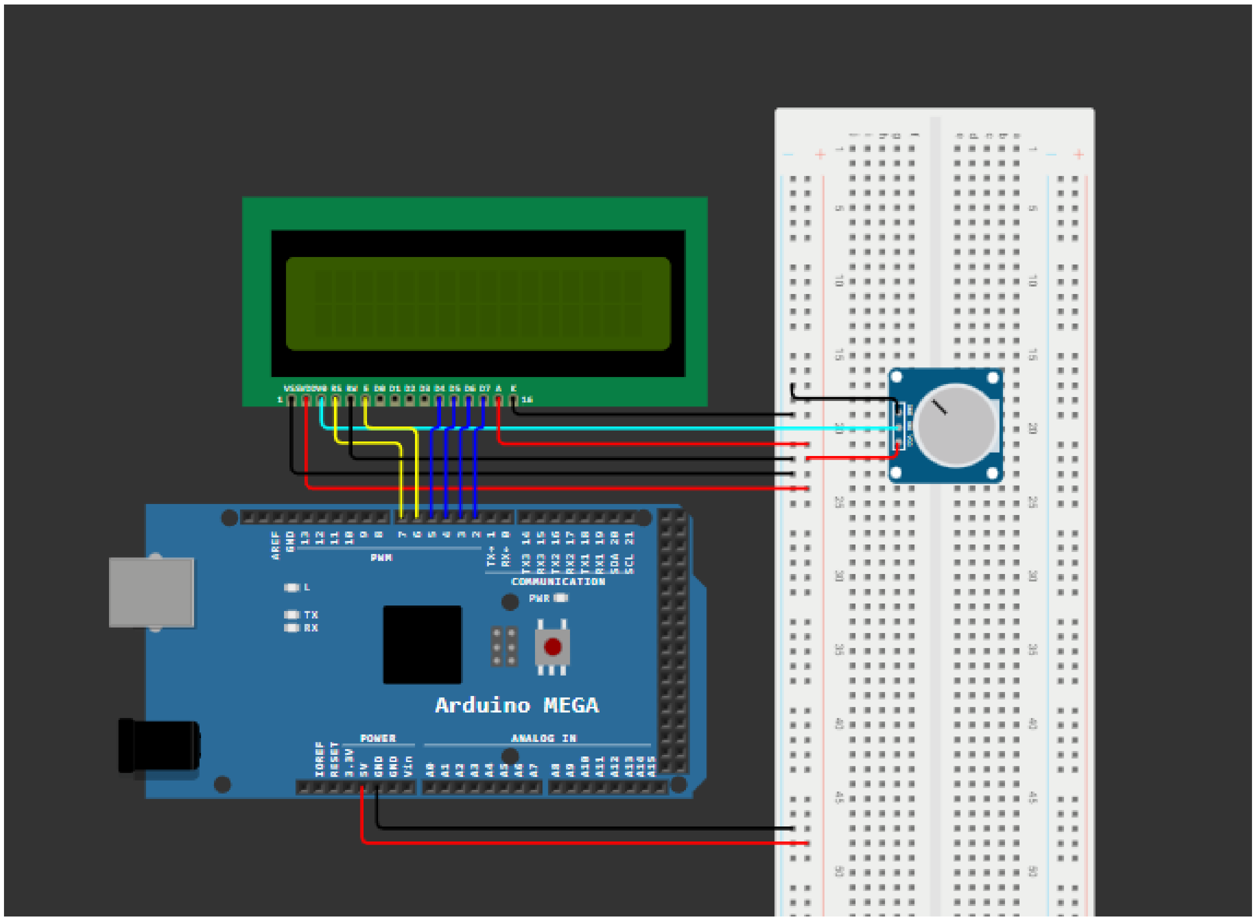


Fig. 1. An example of the initial template of the wiring design created in Wokwi

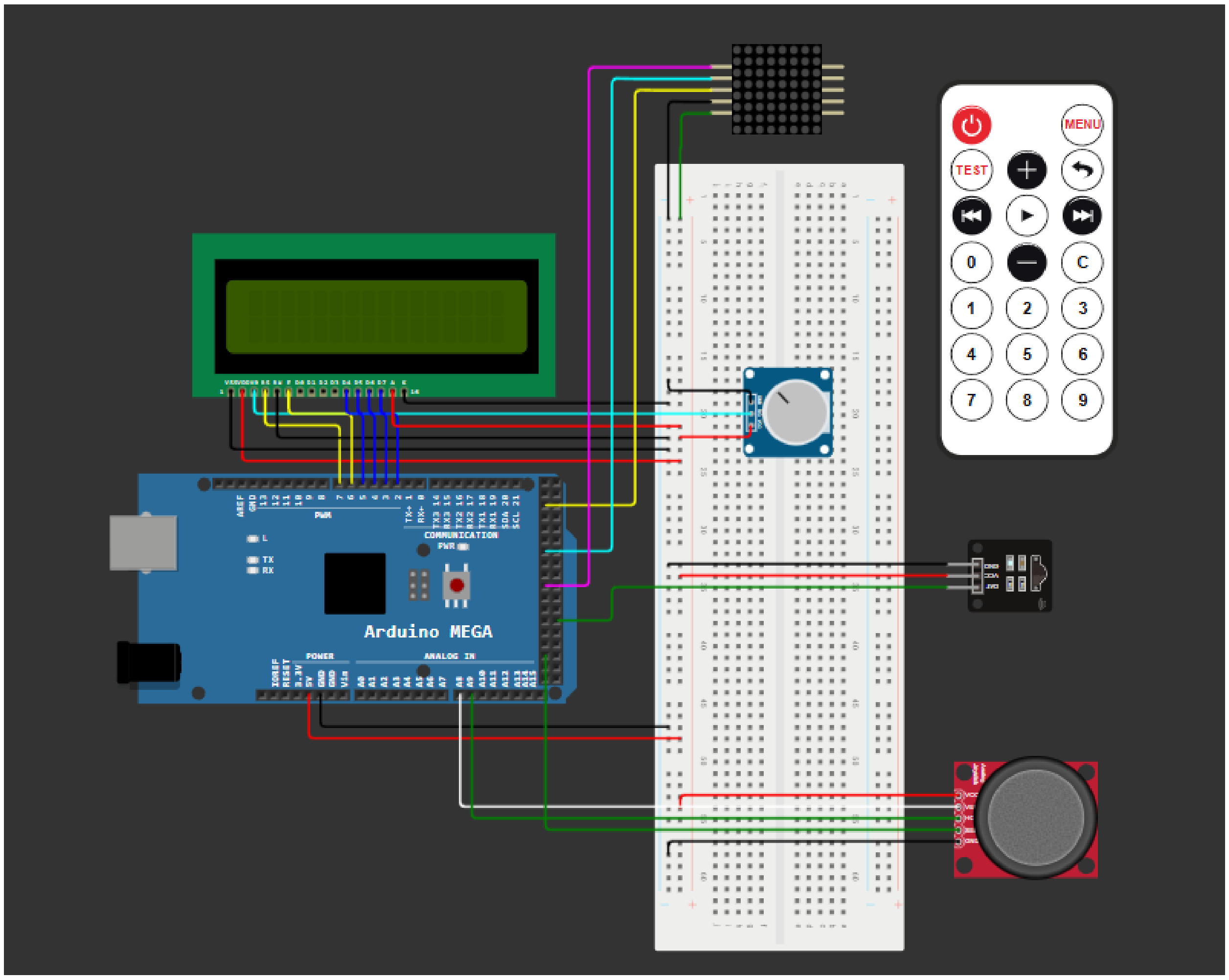


Fig. 2. The final circuit model created in Wokwi

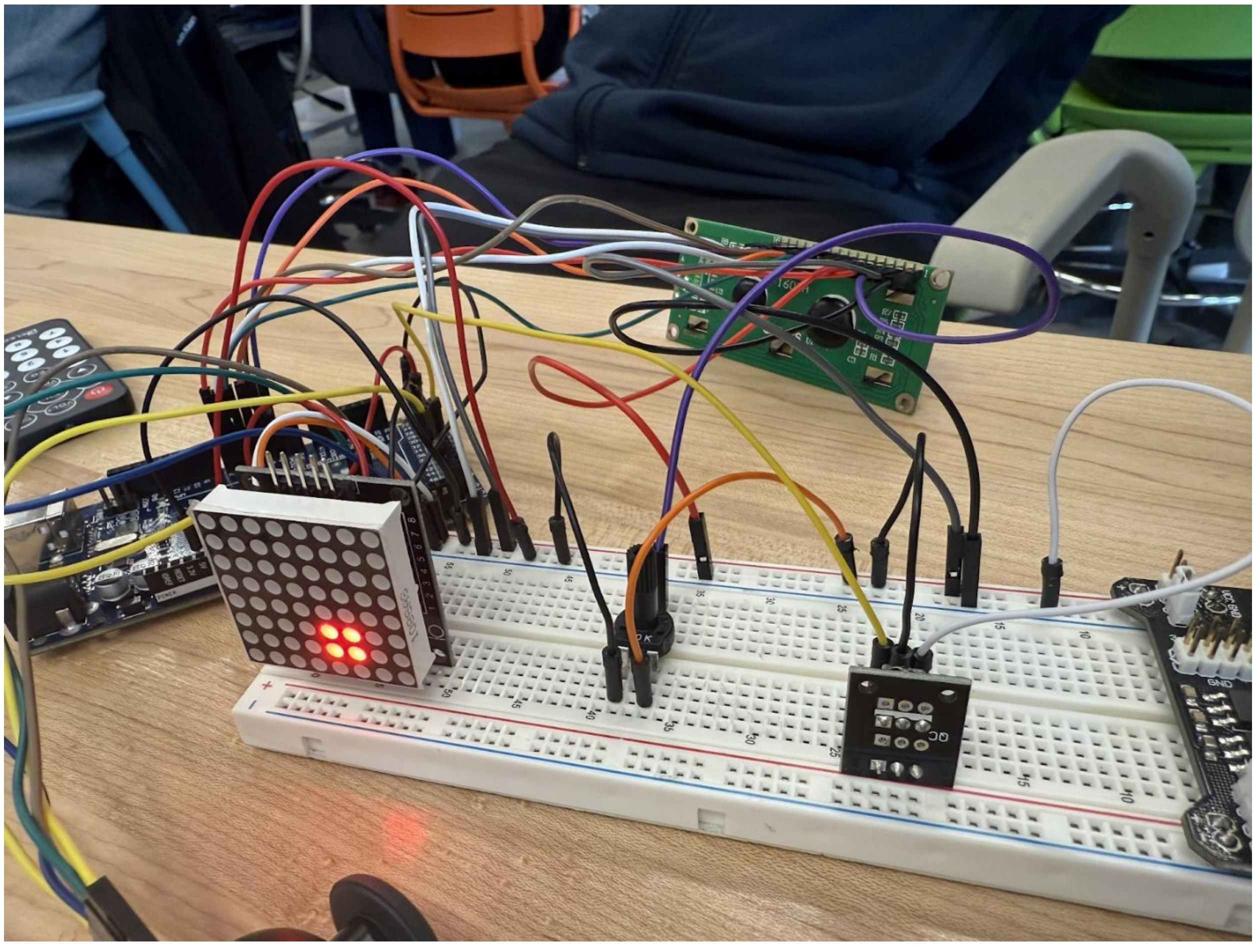


Fig. 3.  Part of the fabrication of the circuit for the lab

## IV. Discussion

1) We used the digitalRead function to determine if the joystick was pressed, a LOW state indicating pressed, while HIGH indicates released. If the function with the parameter of the joystick's "SELECT" pin returns LOW, the mode will be toggled from joystick to remote, and vice versa. We initialize a Boolean variable called joystickMode that holds the current mode and is initially set to True (meaning the default is joystick mode). If digitalRead returns LOW, it enacts the toggleMode function, which changes the current mode. This function sets the joystickMode variable to its complement state, changing the mode. However, for example, if the current mode is joystick, and we press on the joystick, the mode will flicker between remote and joystick a couple times but eventually ending up in remote mode. To fix this, various delays ranging from 25 to 200 milliseconds were implemented in the toggle mode function so that the mode would only change once after the button is pressed (refer to the Appendix, A.1 and A.2).
2) A 10-bit resolution provides values between 0 and 1023 ($2^{10} - 1$), which we used as the ranges for our 8x8 dot matrix. If an 8-bit resolution was implemented instead, the range of values would decrease to 0 and 255 ($2^{8} - 1$). The boundaries of our code would have to adjust to the new range, and the limited range would make the joystick's movement less precise.
3) The MAX7219 modules can combined by connecting the DOUT (Data Out) pin of the first module to the next module's DIN (Data In) pin. The clock, chip select, ground, and VCC all connect to their identical counterparts. The data is transferred by initially loading the data into the internal 16-bit shift register on the CLK's (clock pin) rising edge. The data is transmitted by the DOUT pin, and once all the data is sent, the CS (chip select) pin latched the data and updates the display.
4) The power supply module is needed to demonstrate that our circuit can function without requiring a connection to a computer. It provides power to the entire circuit, allowing for independent functionality.
5) When a button is pressed on the IR remote, a unique binary code is modulated into infrared light, which is emitted by the remote (the code indicates the start signal, a unique address, and the specific command). The IR receiver detects the modulated infrared light and decodes it, which then sends the appropriate command to the devices (in this case the position of the square dot matrix).
6) An IR remote can be hacked and is done by causing the original device to emit an IR signal (like pressing a button the remote), which is then decoded by the IR receiver like usual. Once the signal is captured, it can be replayed which allows the new device to capture this signal and emulate the command. The receiver does not care where the message comes from, if it is valid and decodable.
7) A method to making an "unhackable" TV remote is not known, but there are measures that could vastly improve remote security. An encryption protocol could be utilized, so that the signal sent by the remote could be encrypted and require a decryption key for it to be decoded. The remote signal could be intercepted but could not be deciphered without the decryption key. Another method that could be used is rolling codes, which makes it so that a distinct code is created each time a button on the remote is pressed. Using a shared method between the remote control and the TV, this code is updated with each press. When the receiver detects the code, it decrypts and validates it, ensuring that the signal is from an authorized source. An intercepted code is now only valid for a limited period. However, while these are robust measures for preventing a hack, any security system is susceptible to hacking, along with the increase in cost and complexity caused by the stated methods, preventing there from being a truly "unhackable" remote.

## V. Conclusion

The experiment explored the integration of multiple components that have the capability to control into a singular circuit. The objective of this lab was to create a circuit capable of having a joystick and an IR remote control that can manipulate a 2x2 light inside of an 8x8 dot-matrix in different manners. The circuit was also made capable of switching between the two without conflicting or causing the code to throw an error. This lab taught the applications of multiple components as well as the consideration of the possible implications that integrating multiple components may impose. Hence, this lab was a great stepping stone in learning embedded systems as it provided an opportunity to gain hands-on experience in using multiple input devices (sensors) as well as multiple output devices (indicators). Therefore, this lab can be considered very successful as it widened our understanding of embedded systems and taught us how to create a system where components interact with each other much more deeply than previous labs.

## CONTRIBUTIONS

Jibran Somroo: Responsible for a large portion of the coding and simulation while not having access to the Arduino kit (HC-SR04 ultrasonic sensor, servo motor and 16x2 LCD display); Contributed to the lab report and particularly the sections *Simulation and Analysis, Circuit Building and Conclusion.*

Ezana Abebe: Translated the simulation from Wokwi to a real-life breadboard by assembling the circuit; Fixed any post-simulation problems with code; Contributed to the lab report and particularly the sections *Abstraction, Introduction, Discussion, Appendix*.

## ACKNOWLEDGMENT

## REFERENCES

References should be listed in IEEE format, which is shown below. There are some reference managers you can find to help track and format your references. Some free options include Mendeley and Zotero. Mendeley can be used with MS Word.

[1] Makerguides, "How to use an IR receiver and remote with Arduino", *Makerguides*, https://www.makerguides.com/ir-receiver-remote-arduino-tutorial/

[2] Dziubym, "Controlling 8x8 Dot Matrix with Max7219 and Arduino", *Arduino Project Hub*, Nov 20, 2020, https://projecthub.arduino.cc/Dziubym/controlling-8x8-dot-matrix-with-max7219-and-arduino-0c417a

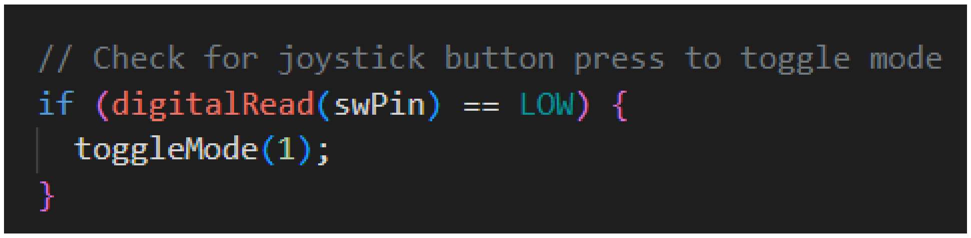
[3] Arduino, "Arduino – Joystick", *Arduino Get Started*, https://arduinogetstarted.com/tutorials/arduino-joystick

[4] Maxim, "MAX7219/MAX7221 – Serially Interfaced, 8-Digit LED Display Drivers", *Maxim*, 2021, https://www.analog.com/media/en/technical-documentation/data-sheets/max7219-max7221.pdf

[5] Alan Zucconi, "How to hack any IR remote controller", *Alan Zucconi*, Aug 19, 2015, https://www.alanzucconi.com/2015/08/19/how-to-hack-any-ir-remote-controller/

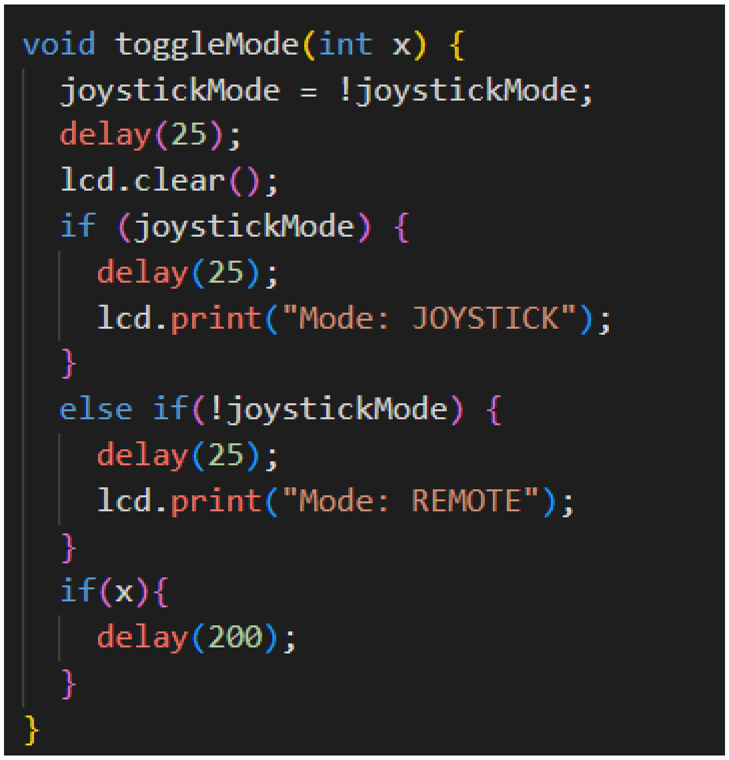
[6] Duncan Wilson, "How does a remote control work the TV", UCL, https://www.ucl.ac.uk/culture-online/case-studies/2022/sep/how-does-remote-control-work-tv#:~:text=TV%20remote%20controls%20work%20in,TV%20is%20called%20the%20receiver.

## APPENDIX



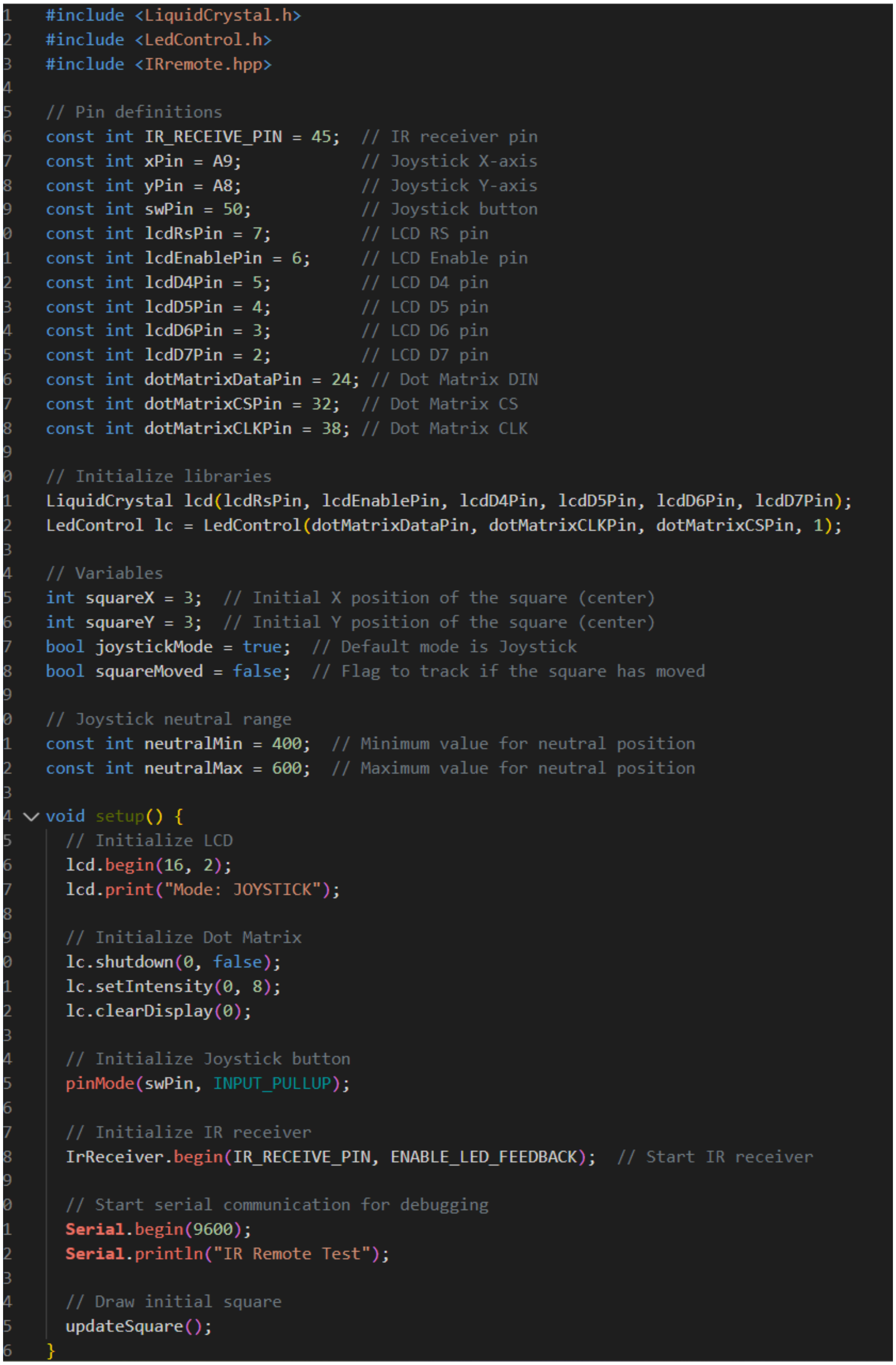
```
// Check for joystick button press to toggle mode
if (digitalRead(swPin) == LOW) {
  toggleMode(1);
}
```

A.1 – Checks if button is pressed and changes mode accordingly

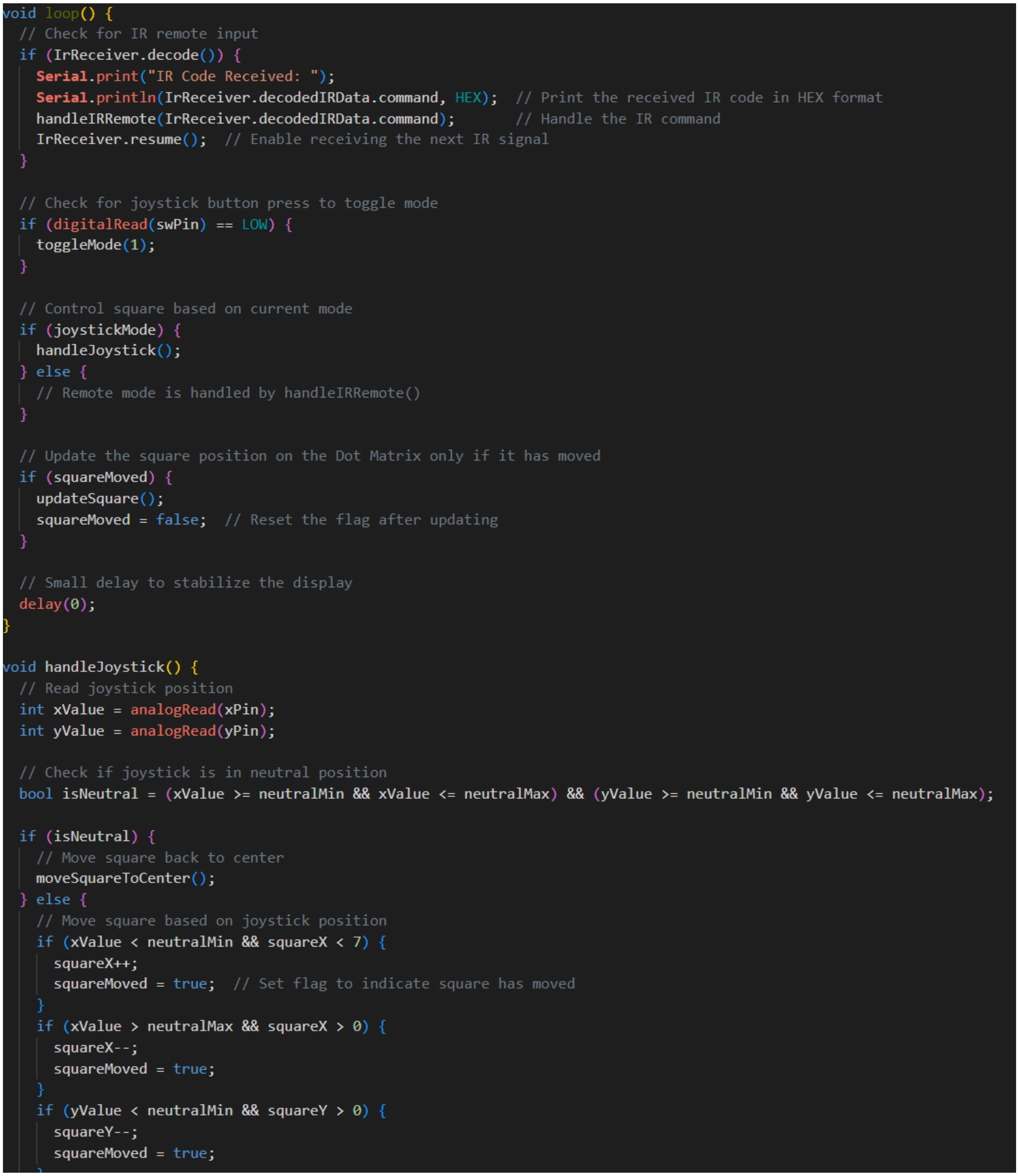


```
void toggleMode(int x) {
  joystickMode = !joystickMode;
  delay(25);
  lcd.clear();
  if (joystickMode) {
    delay(25);
    lcd.print("Mode: JOYSTICK");
  }
  else if(!joystickMode) {
    delay(25);
    lcd.print("Mode: REMOTE");
  }
  if(x){
    delay(200);
  }
}
```

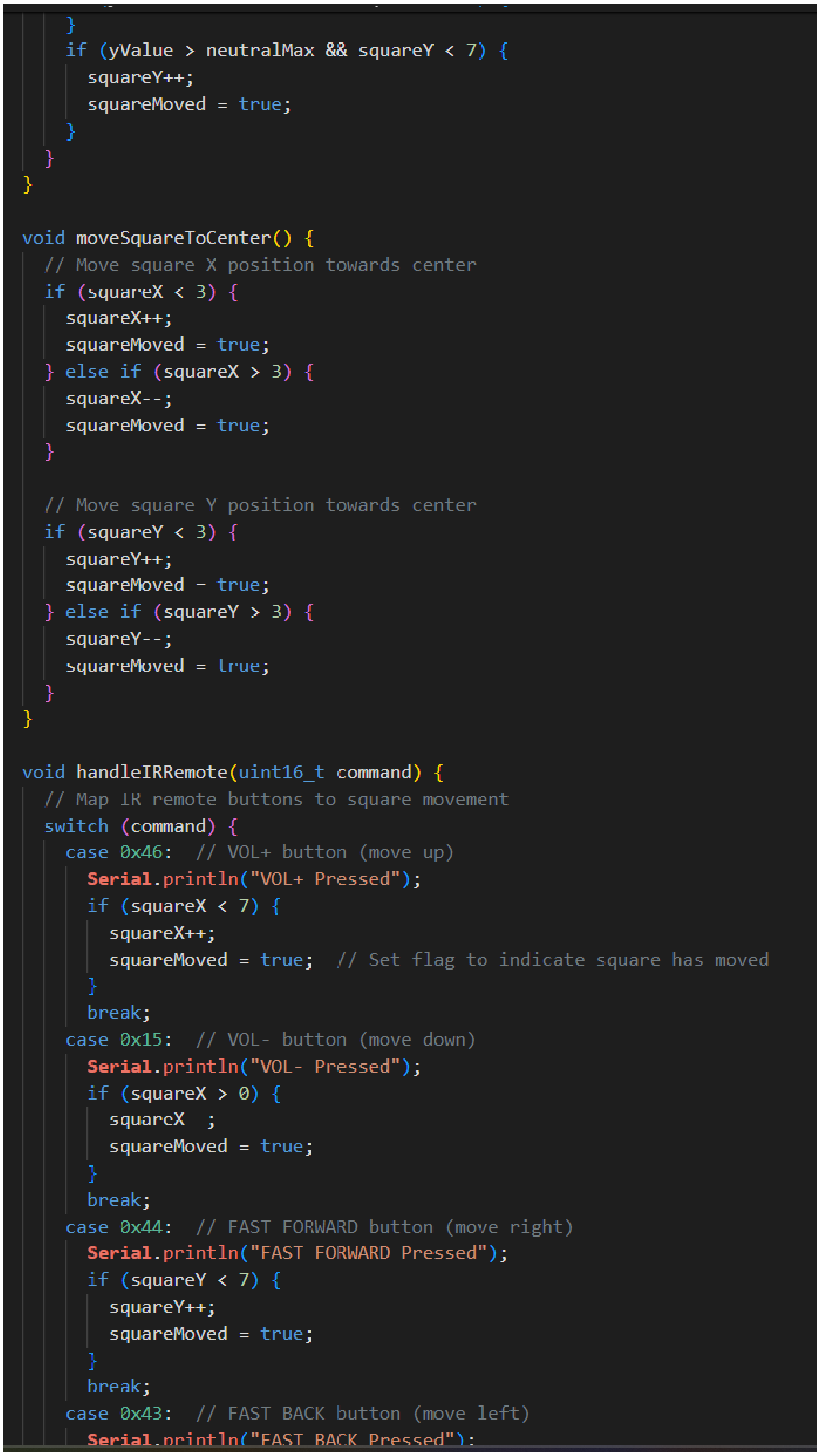
A.2 – toggleMode function which determines the current mode and changes it accordingly with delays to prevent the mode from changing more than once.



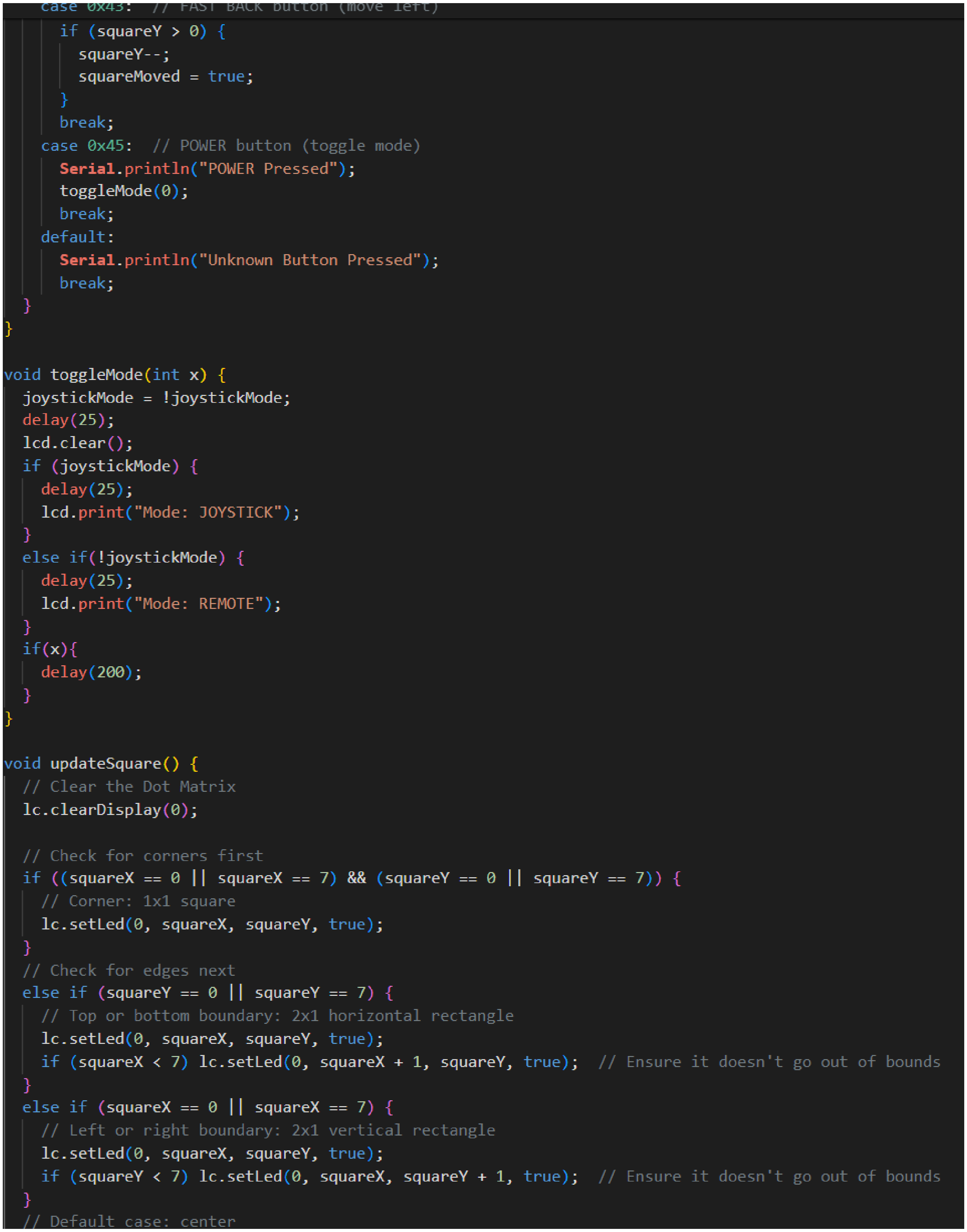
```cpp
#include <LiquidCrystal.h>
#include <LedControl.h>
#include <IRremote.hpp>

// Pin definitions
const int IR_RECEIVE_PIN = 45;  // IR receiver pin
const int xPin = A9;            // Joystick X-axis
const int yPin = A8;            // Joystick Y-axis
const int swPin = 50;           // Joystick button
const int lcdRsPin = 7;         // LCD RS pin
const int lcdEnablePin = 6;     // LCD Enable pin
const int lcdD4Pin = 5;         // LCD D4 pin
const int lcdD5Pin = 4;         // LCD D5 pin
const int lcdD6Pin = 3;         // LCD D6 pin
const int lcdD7Pin = 2;         // LCD D7 pin
const int dotMatrixDataPin = 24; // Dot Matrix DIN
const int dotMatrixCSPin = 32;  // Dot Matrix CS
const int dotMatrixCLKPin = 38; // Dot Matrix CLK

// Initialize libraries
LiquidCrystal lcd(lcdRsPin, lcdEnablePin, lcdD4Pin, lcdD5Pin, lcdD6Pin, lcdD7Pin);
LedControl lc = LedControl(dotMatrixDataPin, dotMatrixCLKPin, dotMatrixCSPin, 1);

// Variables
int squareX = 3;  // Initial X position of the square (center)
int squareY = 3;  // Initial Y position of the square (center)
bool joystickMode = true;  // Default mode is Joystick
bool squareMoved = false;  // Flag to track if the square has moved

// Joystick neutral range
const int neutralMin = 400;  // Minimum value for neutral position
const int neutralMax = 600;  // Maximum value for neutral position

void setup() {
  // Initialize LCD
  lcd.begin(16, 2);
  lcd.print("Mode: JOYSTICK");

  // Initialize Dot Matrix
  lc.shutdown(0, false);
  lc.setIntensity(0, 8);
  lc.clearDisplay(0);

  // Initialize Joystick button
  pinMode(swPin, INPUT_PULLUP);

  // Initialize IR receiver
  IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK);  // Start IR receiver

  // Start serial communication for debugging
  Serial.begin(9600);
  Serial.println("IR Remote Test");

  // Draw initial square
  updateSquare();
}
```



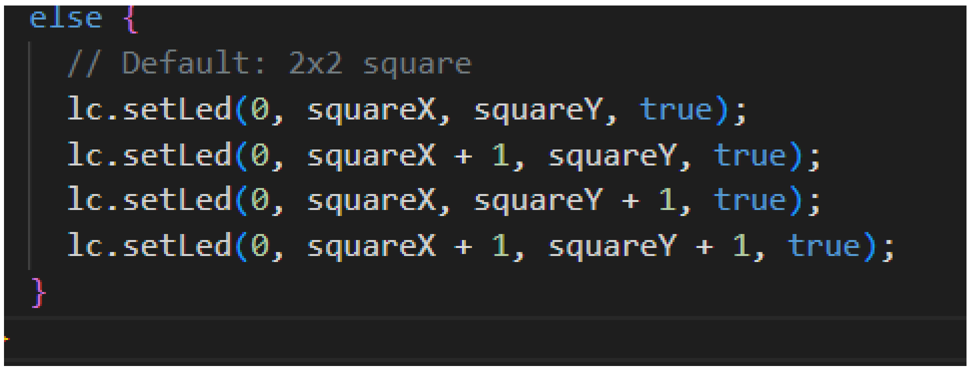
```
void loop() {
  // Check for IR remote input
  if (IrReceiver.decode()) {
    Serial.print("IR Code Received: ");
    Serial.println(IrReceiver.decodedIRData.command, HEX);  // Print the received IR code in HEX format
    handleIRRemote(IrReceiver.decodedIRData.command);       // Handle the IR command
    IrReceiver.resume();  // Enable receiving the next IR signal
  }

  // Check for joystick button press to toggle mode
  if (digitalRead(swPin) == LOW) {
    toggleMode(1);
  }

  // Control square based on current mode
  if (joystickMode) {
    handleJoystick();
  } else {
    // Remote mode is handled by handleIRRemote()
  }

  // Update the square position on the Dot Matrix only if it has moved
  if (squareMoved) {
    updateSquare();
    squareMoved = false;  // Reset the flag after updating
  }

  // Small delay to stabilize the display
  delay(0);
}

void handleJoystick() {
  // Read joystick position
  int xValue = analogRead(xPin);
  int yValue = analogRead(yPin);

  // Check if joystick is in neutral position
  bool isNeutral = (xValue >= neutralMin && xValue <= neutralMax) && (yValue >= neutralMin && yValue <= neutralMax);

  if (isNeutral) {
    // Move square back to center
    moveSquareToCenter();
  } else {
    // Move square based on joystick position
    if (xValue < neutralMin && squareX < 7) {
      squareX++;
      squareMoved = true;  // Set flag to indicate square has moved
    }
    if (xValue > neutralMax && squareX > 0) {
      squareX--;
      squareMoved = true;
    }
    if (yValue < neutralMin && squareY > 0) {
      squareY--;
      squareMoved = true;
```



```
    }
    if (yValue > neutralMax && squareY < 7) {
      squareY++;
      squareMoved = true;
    }
  }
}

void moveSquareToCenter() {
  // Move square X position towards center
  if (squareX < 3) {
    squareX++;
    squareMoved = true;
  } else if (squareX > 3) {
    squareX--;
    squareMoved = true;
  }

  // Move square Y position towards center
  if (squareY < 3) {
    squareY++;
    squareMoved = true;
  } else if (squareY > 3) {
    squareY--;
    squareMoved = true;
  }
}

void handleIRRemote(uint16_t command) {
  // Map IR remote buttons to square movement
  switch (command) {
    case 0x46:  // VOL+ button (move up)
      Serial.println("VOL+ Pressed");
      if (squareX < 7) {
        squareX++;
        squareMoved = true;  // Set flag to indicate square has moved
      }
      break;
    case 0x15:  // VOL- button (move down)
      Serial.println("VOL- Pressed");
      if (squareX > 0) {
        squareX--;
        squareMoved = true;
      }
      break;
    case 0x44:  // FAST FORWARD button (move right)
      Serial.println("FAST FORWARD Pressed");
      if (squareY < 7) {
        squareY++;
        squareMoved = true;
      }
      break;
    case 0x43:  // FAST BACK button (move left)
      Serial.println("FAST BACK Pressed");
```



```
    case 0x43:  // FAST BACK button (move left)
      if (squareY > 0) {
        squareY--;
        squareMoved = true;
      }
      break;
    case 0x45:  // POWER button (toggle mode)
      Serial.println("POWER Pressed");
      toggleMode(0);
      break;
    default:
      Serial.println("Unknown Button Pressed");
      break;
  }
}

void toggleMode(int x) {
  joystickMode = !joystickMode;
  delay(25);
  lcd.clear();
  if (joystickMode) {
    delay(25);
    lcd.print("Mode: JOYSTICK");
  }
  else if(!joystickMode) {
    delay(25);
    lcd.print("Mode: REMOTE");
  }
  if(x){
    delay(200);
  }
}

void updateSquare() {
  // Clear the Dot Matrix
  lc.clearDisplay(0);

  // Check for corners first
  if ((squareX == 0 || squareX == 7) && (squareY == 0 || squareY == 7)) {
    // Corner: 1x1 square
    lc.setLed(0, squareX, squareY, true);
  }
  // Check for edges next
  else if (squareY == 0 || squareY == 7) {
    // Top or bottom boundary: 2x1 horizontal rectangle
    lc.setLed(0, squareX, squareY, true);
    if (squareX < 7) lc.setLed(0, squareX + 1, squareY, true);  // Ensure it doesn't go out of bounds
  }
  else if (squareX == 0 || squareX == 7) {
    // Left or right boundary: 2x1 vertical rectangle
    lc.setLed(0, squareX, squareY, true);
    if (squareY < 7) lc.setLed(0, squareX, squareY + 1, true);  // Ensure it doesn't go out of bounds
  }
  // Default case: center
```



```
else {
  // Default: 2x2 square
  lc.setLed(0, squareX, squareY, true);
  lc.setLed(0, squareX + 1, squareY, true);
  lc.setLed(0, squareX, squareY + 1, true);
  lc.setLed(0, squareX + 1, squareY + 1, true);
}
```

A.3 – Overall code