# Rocket Game

Konstantinos Zefkilis, Filippo Maffei, Federico Iop, Luca Pio Pierno

# Introduction

Introduction to the development of the game

# Challenges

### Smooth Joystick Control

Implementing smooth joystick control for the rocket.

### Real-Time Screen Updates

Ensuring real-time updates on the screen with minimal latency.

### Intuitive User Interface

Designing an intuitive user interface on the TFT LCD.

### Feedback Mechanisms Integration

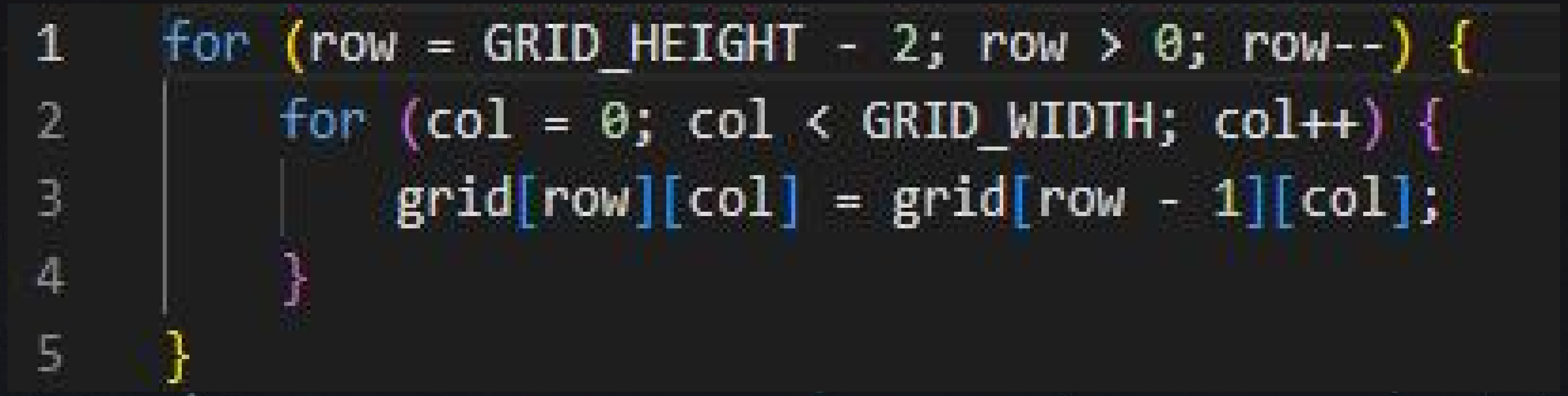
Integrating feedback mechanisms like LEDs and a buzzer.

### Reliable Communication Protocol

Establishing a reliable communication protocol between the MSP432 and the ESP32.

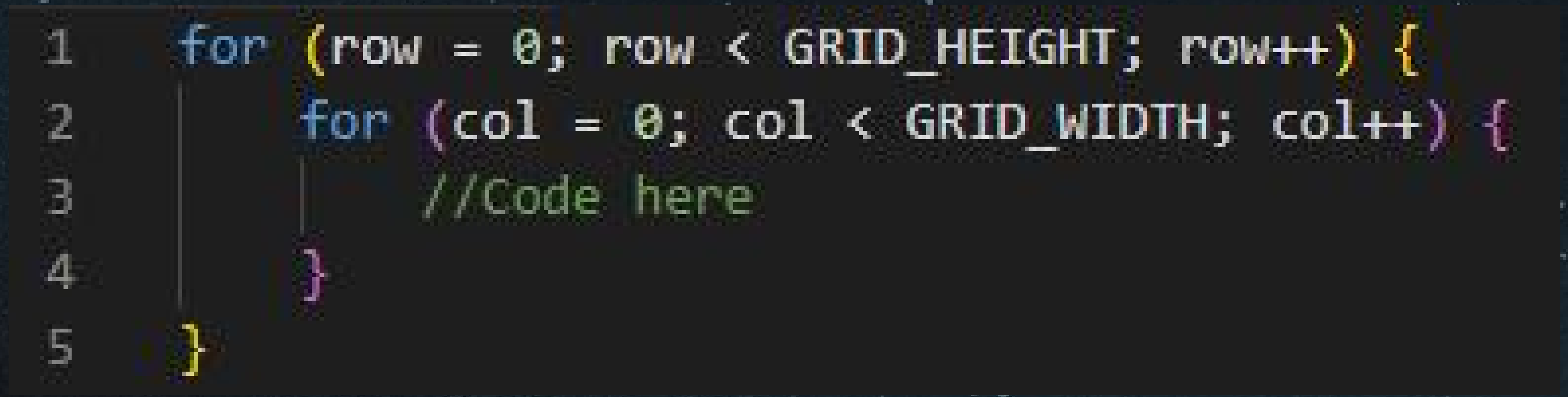
# Real time screen update

In order to have a screen that updates quickly, we implemented the updateGrid() function. These are the most important parts of the function:



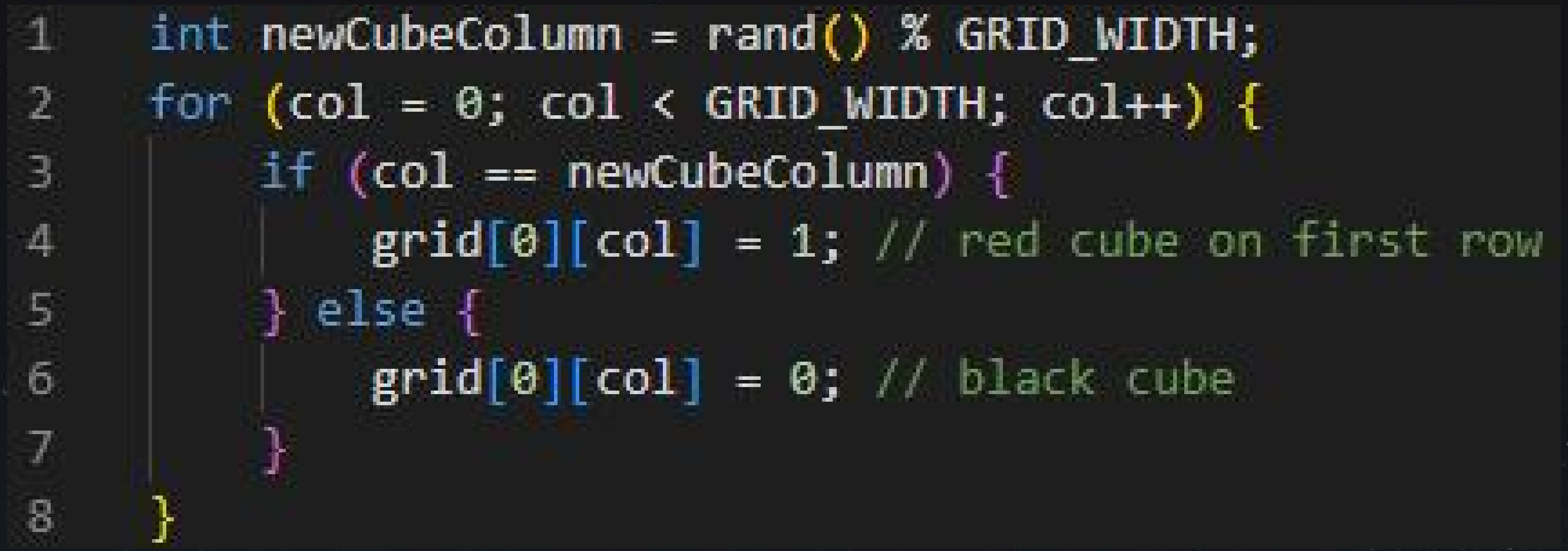
```c
for (row = GRID_HEIGHT - 2; row > 0; row--) {
    for (col = 0; col < GRID_WIDTH; col++) {
        grid[row][col] = grid[row - 1][col];
    }
}
```

Copies the squares of the previous row to the current row. So that the grid is updated.



```c
for (row = 0; row < GRID_HEIGHT; row++) {
    for (col = 0; col < GRID_WIDTH; col++) {
        //Code here
    }
}
```

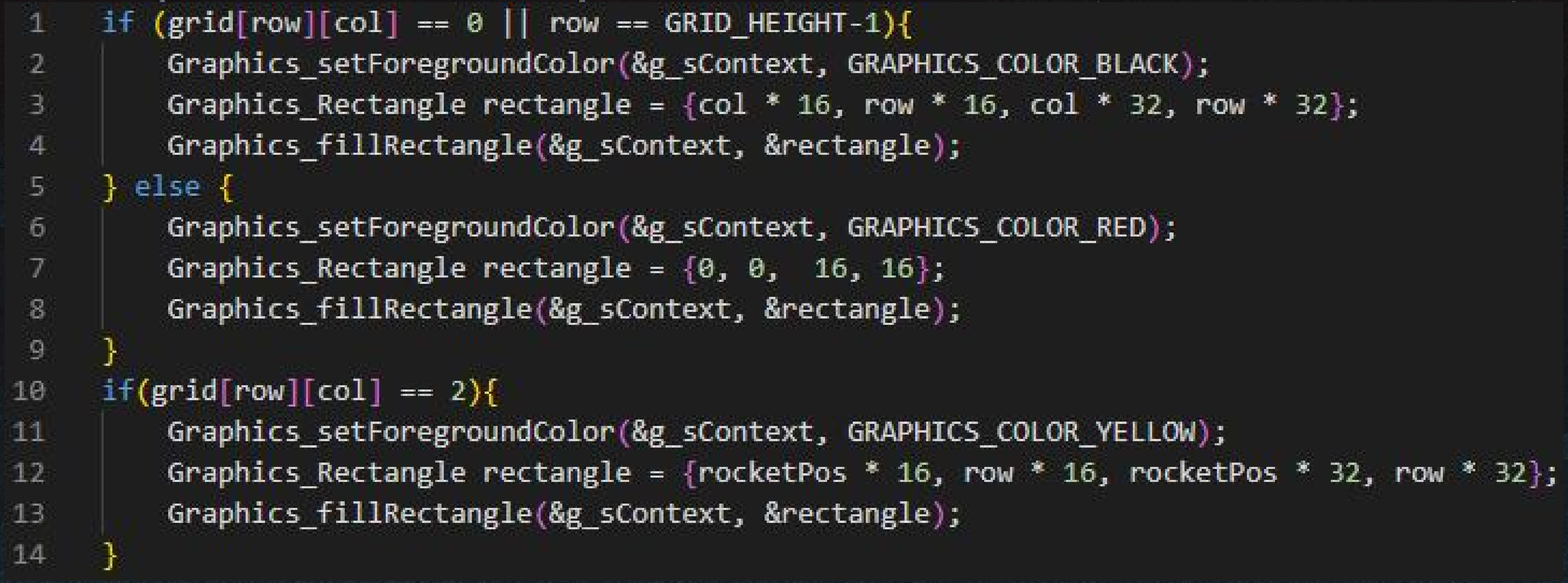
Iterate each array element, then colour it black (free space), red (obstacle) or green (rocket)



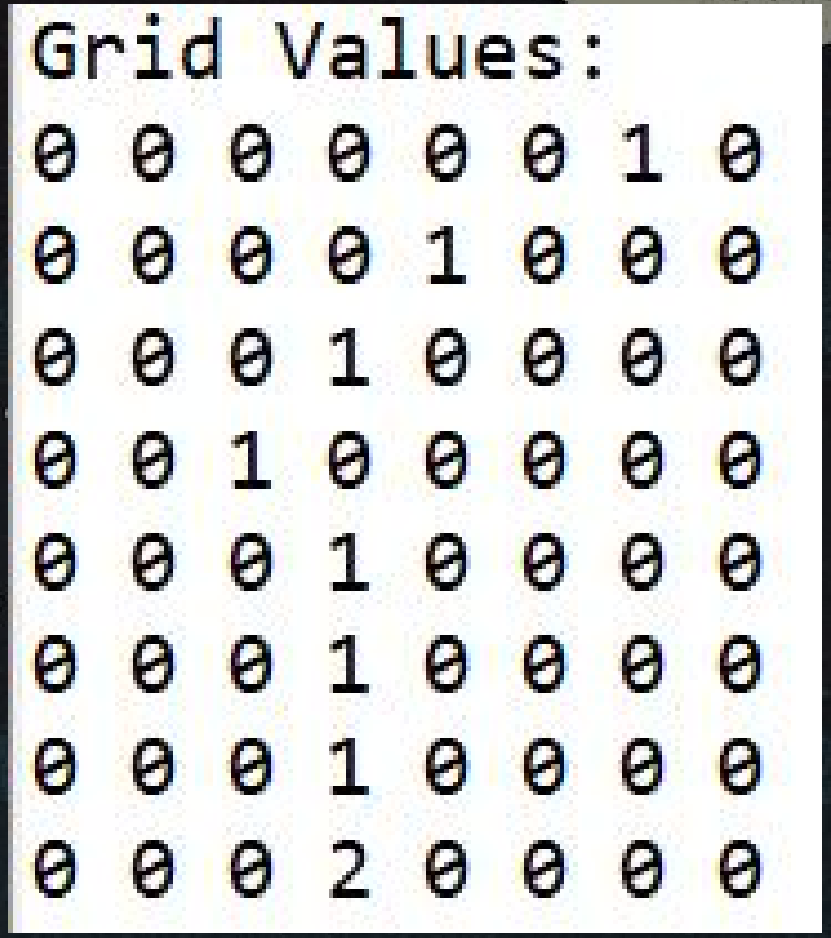
```c
int newCubeColumn = rand() % GRID_WIDTH;
for (col = 0; col < GRID_WIDTH; col++) {
    if (col == newCubeColumn) {
        grid[0][col] = 1; // red cube on first row
    } else {
        grid[0][col] = 0; // black cube
    }
}
```

Initially it generates a random number in which to place the red square (obstacle) and assigns the value 1 (obstacle) or 0 (free) to each element of the array represented by the grid

# Real time screen update



```
if (grid[row][col] == 0 || row == GRID_HEIGHT-1){
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
    Graphics_Rectangle rectangle = {col * 16, row * 16, col * 32, row * 32};
    Graphics_fillRectangle(&g_sContext, &rectangle);
} else {
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_Rectangle rectangle = {0, 0,  16, 16};
    Graphics_fillRectangle(&g_sContext, &rectangle);
}
if(grid[row][col] == 2){
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_YELLOW);
    Graphics_Rectangle rectangle = {rocketPos * 16, row * 16, rocketPos * 32, row * 32};
    Graphics_fillRectangle(&g_sContext, &rectangle);
}
```



```
Grid Values:
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 2 0 0 0 0
```

If the current array element is 0 (free square) or it is the last row (GRID_HEIGHT - 1) which must be all black except the rocket square it will be coloured black, otherwise it will be coloured red. If, on the other hand, the value of the current array element is 2, the square will be coloured green (rocket).

***Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);*** //All graphic operations will use the specified colour (black)

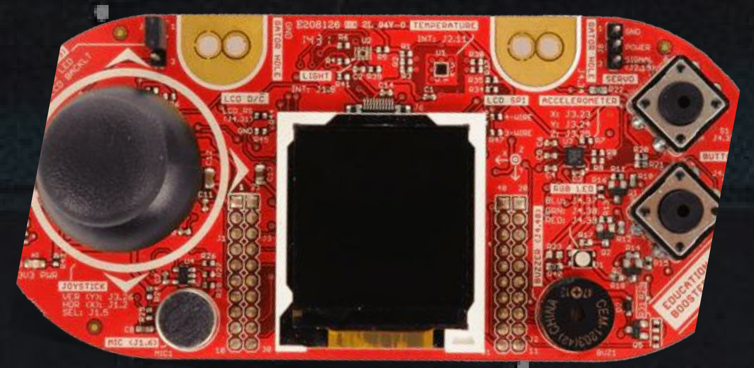
***Graphics_Rectangle rectangle = {0, 0, 16, 16};*** // Defines a rectangle with respective angle coordinates

***Graphics_fillRectangle(&g_sContext, &rectangle);*** // Fills the square with the previously chosen colour

# Hardware component

### MSP432P401R BoosterPackMKII

Development board with high performance and low power MSP432P401R microcontroller.

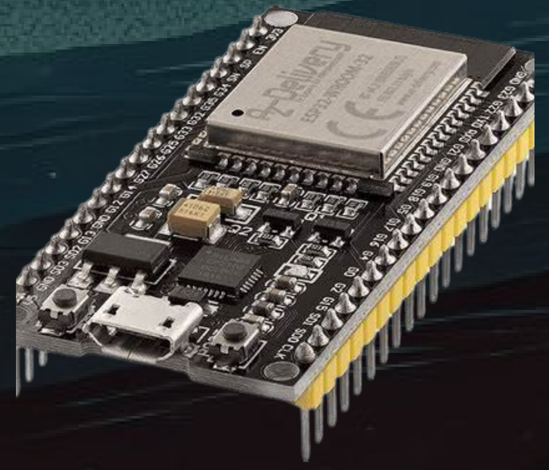


### MSP432P401R Launchpad

Expansion module with joystick, buttons, accelerometer, and color LCD.

### ESP32

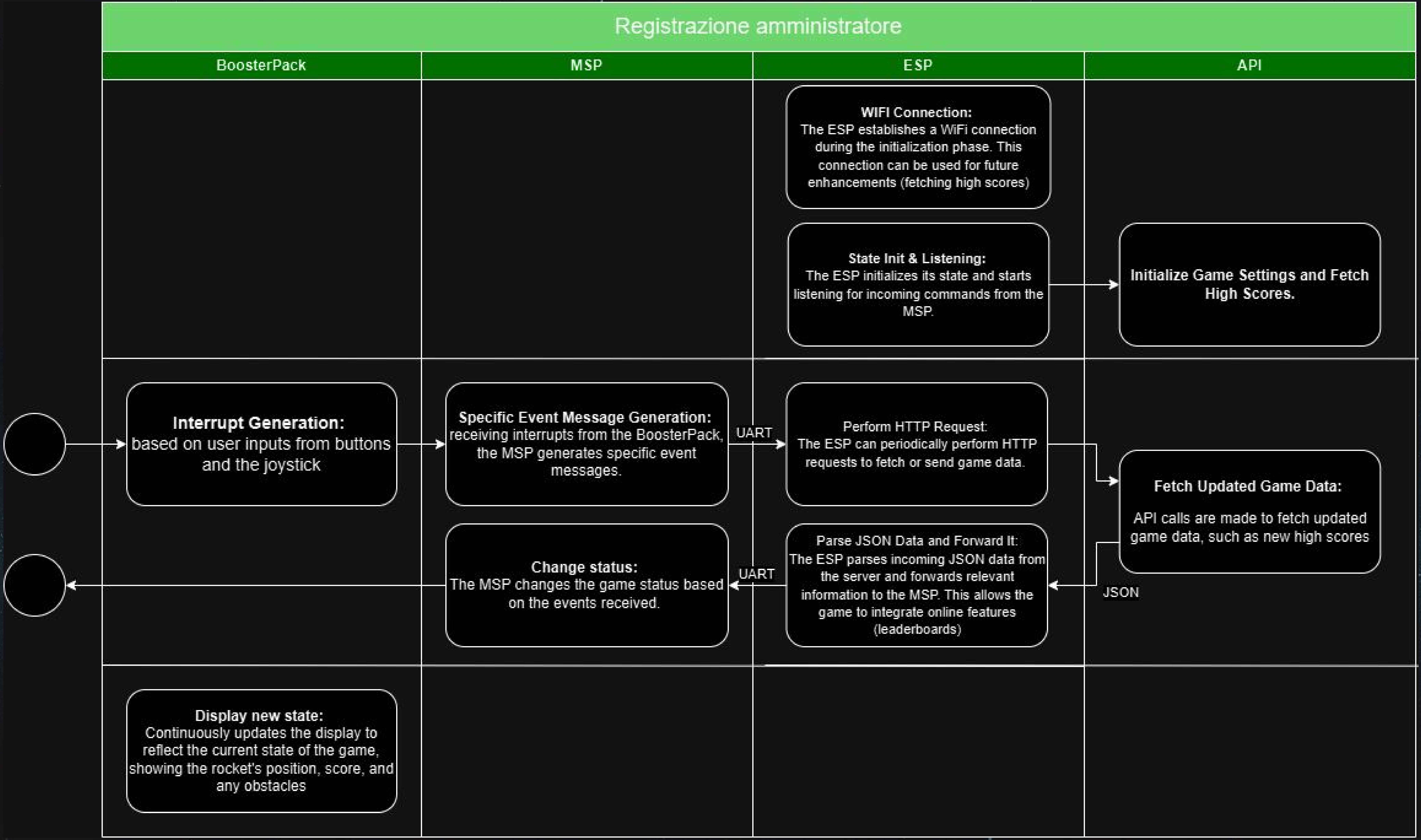
Wi-Fi and Bluetooth-enabled microcontroller for IoT projects.



# Development

Let’s talk about the development

# Working Flow



Registrazione amministratore

BoosterPack

MSP

ESP

API

**WIFI Connection:**
The ESP establishes a WiFi connection during the initialization phase. This connection can be used for future enhancements (fetching high scores)

**State Init & Listening:**
The ESP initializes its state and starts listening for incoming commands from the MSP.

**Initialize Game Settings and Fetch High Scores.**

**Interrupt Generation:**
based on user inputs from buttons and the joystick

**Specific Event Message Generation:**
receiving interrupts from the BoosterPack, the MSP generates specific event messages.

UART

Perform HTTP Request:
The ESP can periodically perform HTTP requests to fetch or send game data.

**Fetch Updated Game Data:**

API calls are made to fetch updated game data, such as new high scores

JSON

Parse JSON Data and Forward It:
The ESP parses incoming JSON data from the server and forwards relevant information to the MSP. This allows the game to integrate online features (leaderboards)

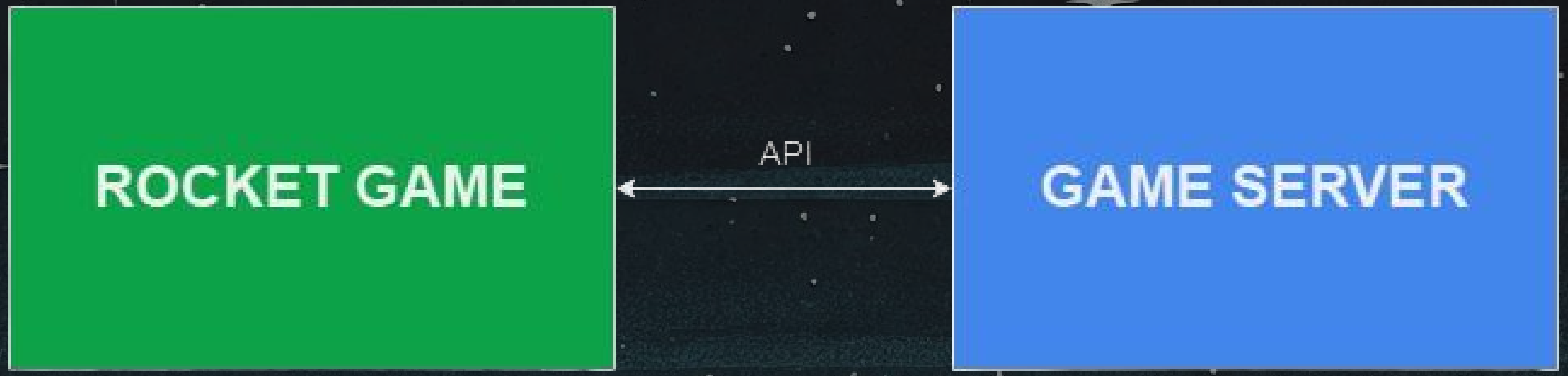
UART

**Change status:**
The MSP changes the game status based on the events received.

**Display new state:**
Continuously updates the display to reflect the current state of the game, showing the rocket's position, score, and any obstacles

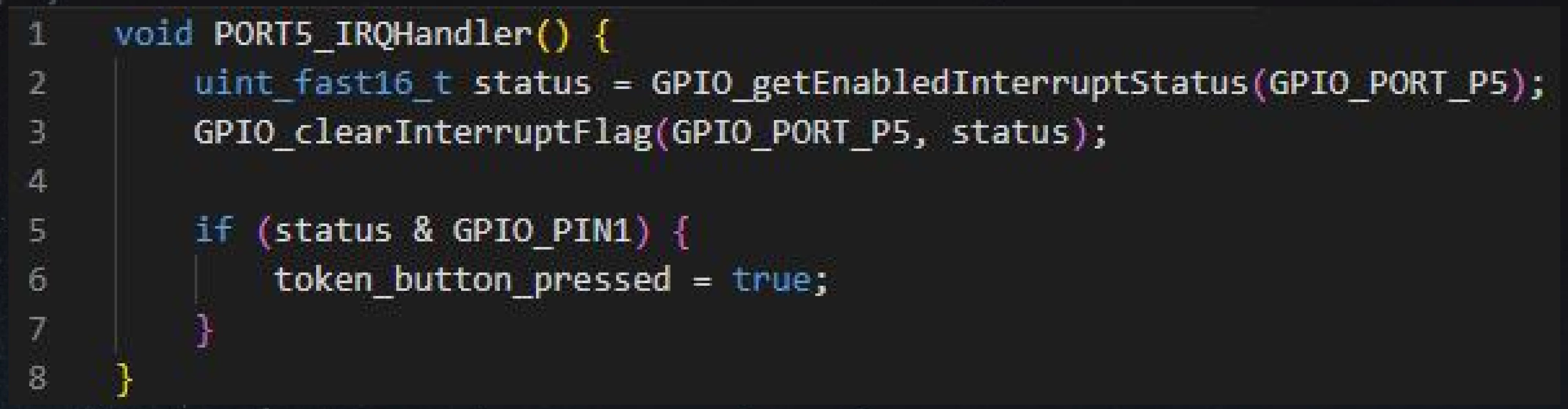
# Rocket game API

- Game Server Dev
- Fetching High Scores
- HTTP to use the APIs
- Parsing JSON Responses



# Interrupts used

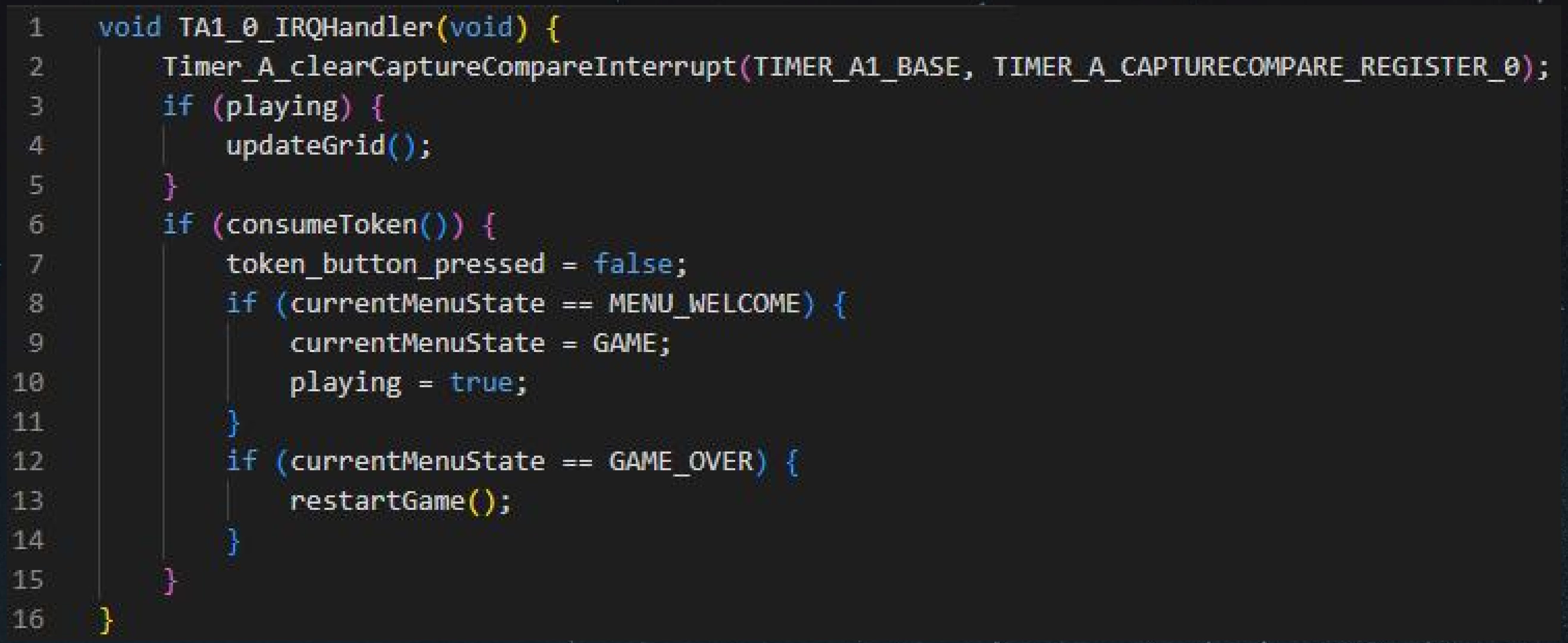
## Button pressure management



```c
void PORT5_IRQHandler() {
    uint_fast16_t status = GPIO_getEnabledInterruptStatus(GPIO_PORT_P5);
    GPIO_clearInterruptFlag(GPIO_PORT_P5, status);

    if (status & GPIO_PIN1) {
        token_button_pressed = true;
    }
}
```

Handles interrupts for GPIO port 5 of the MSP432, used to detect the pressing of physical buttons connected to the microcontroller. Port and pins used: Port 5, Pin 1.

## Game Logic Update

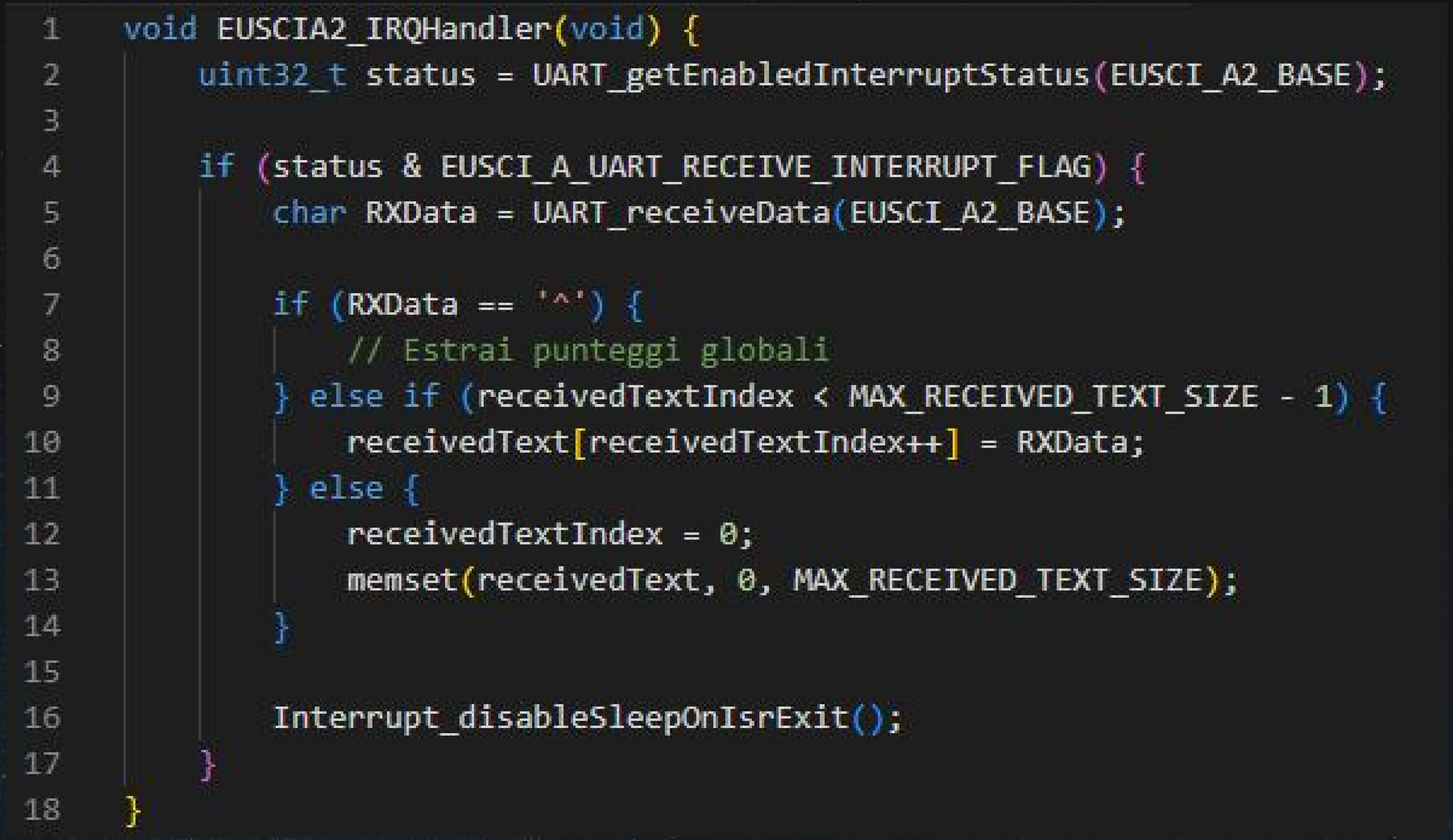


```c
void TA1_0_IRQHandler(void) {
    Timer_A_clearCaptureCompareInterrupt(TIMER_A1_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_0);
    if (playing) {
        updateGrid();
    }
    if (consumeToken()) {
        token_button_pressed = false;
        if (currentMenuState == MENU_WELCOME) {
            currentMenuState = GAME;
            playing = true;
        }
        if (currentMenuState == GAME_OVER) {
            restartGame();
        }
    }
}
```

Handler for Timer A. It is called when the timer reaches a specific value (every second and a half), indicating that it is time to update the game logic.

# Interrupts used

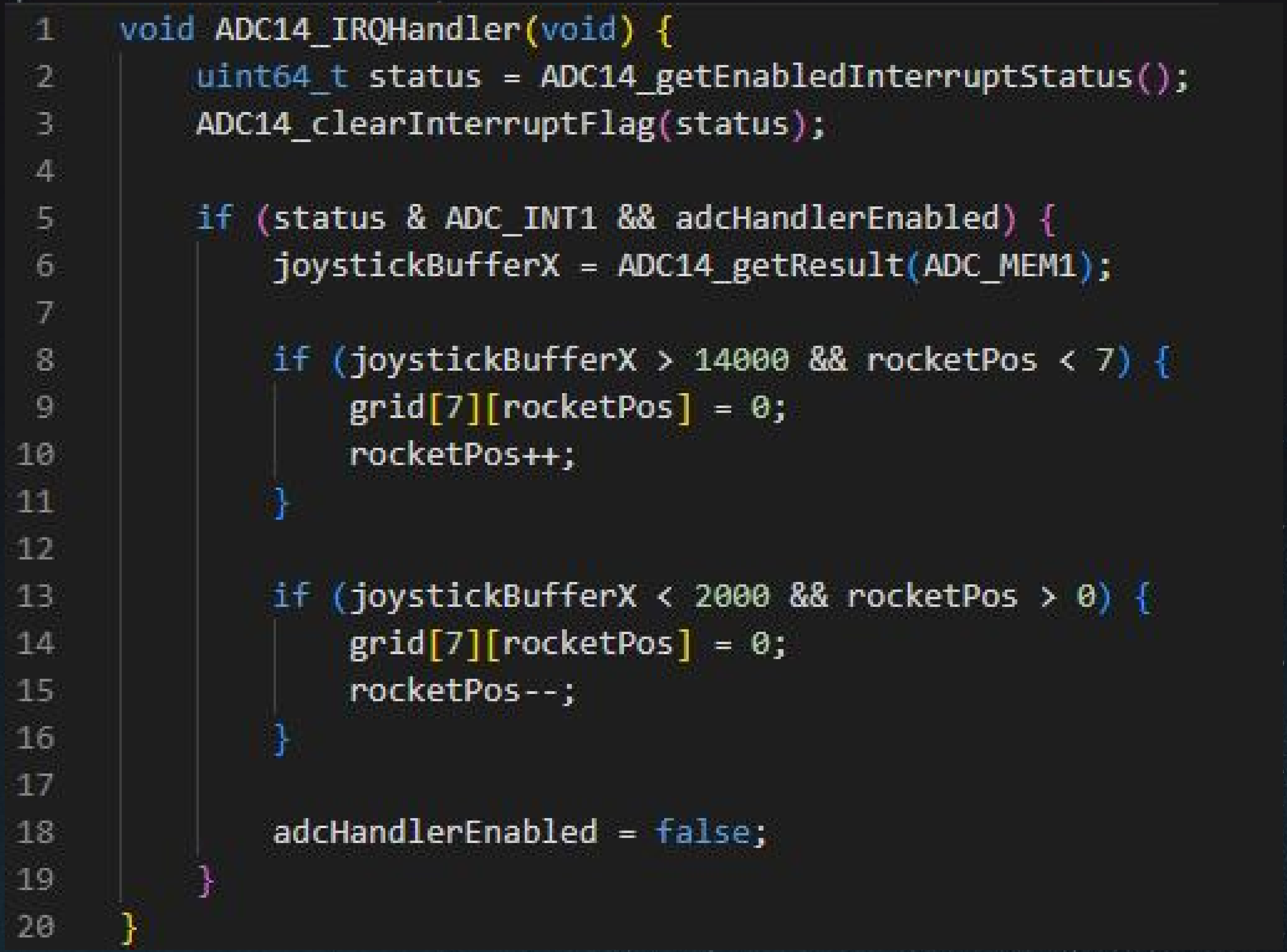
## UART Reception and Data Processing



```c
void EUSCIA2_IRQHandler(void) {
    uint32_t status = UART_getEnabledInterruptStatus(EUSCI_A2_BASE);

    if (status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG) {
        char RXData = UART_receiveData(EUSCI_A2_BASE);

        if (RXData == '^') {
            // Estrai punteggi globali
        } else if (receivedTextIndex < MAX_RECEIVED_TEXT_SIZE - 1) {
            receivedText[receivedTextIndex++] = RXData;
        } else {
            receivedTextIndex = 0;
            memset(receivedText, 0, MAX_RECEIVED_TEXT_SIZE);
        }

        Interrupt_disableSleepOnIsrExit();
    }
}
```

Handles UART interruptions for receiving data. Processes the received data to extract global scores and update the display with these scores.

## Rocket Position Update via Joystick



```c
void ADC14_IRQHandler(void) {
    uint64_t status = ADC14_getEnabledInterruptStatus();
    ADC14_clearInterruptFlag(status);

    if (status & ADC_INT1 && adcHandlerEnabled) {
        joystickBufferX = ADC14_getResult(ADC_MEM1);

        if (joystickBufferX > 14000 && rocketPos < 7) {
            grid[7][rocketPos] = 0;
            rocketPos++;
        }

        if (joystickBufferX < 2000 && rocketPos > 0) {
            grid[7][rocketPos] = 0;
            rocketPos--;
        }

        adcHandlerEnabled = false;
    }
}
```

Handles interrupts from the microcontroller's ADC module. It reads the value of the joystick and updates the position of the rocket in the game grid according to the joystick movements.

# ESP Development

### Connection

Connecting to WiFi.

### Score tracking

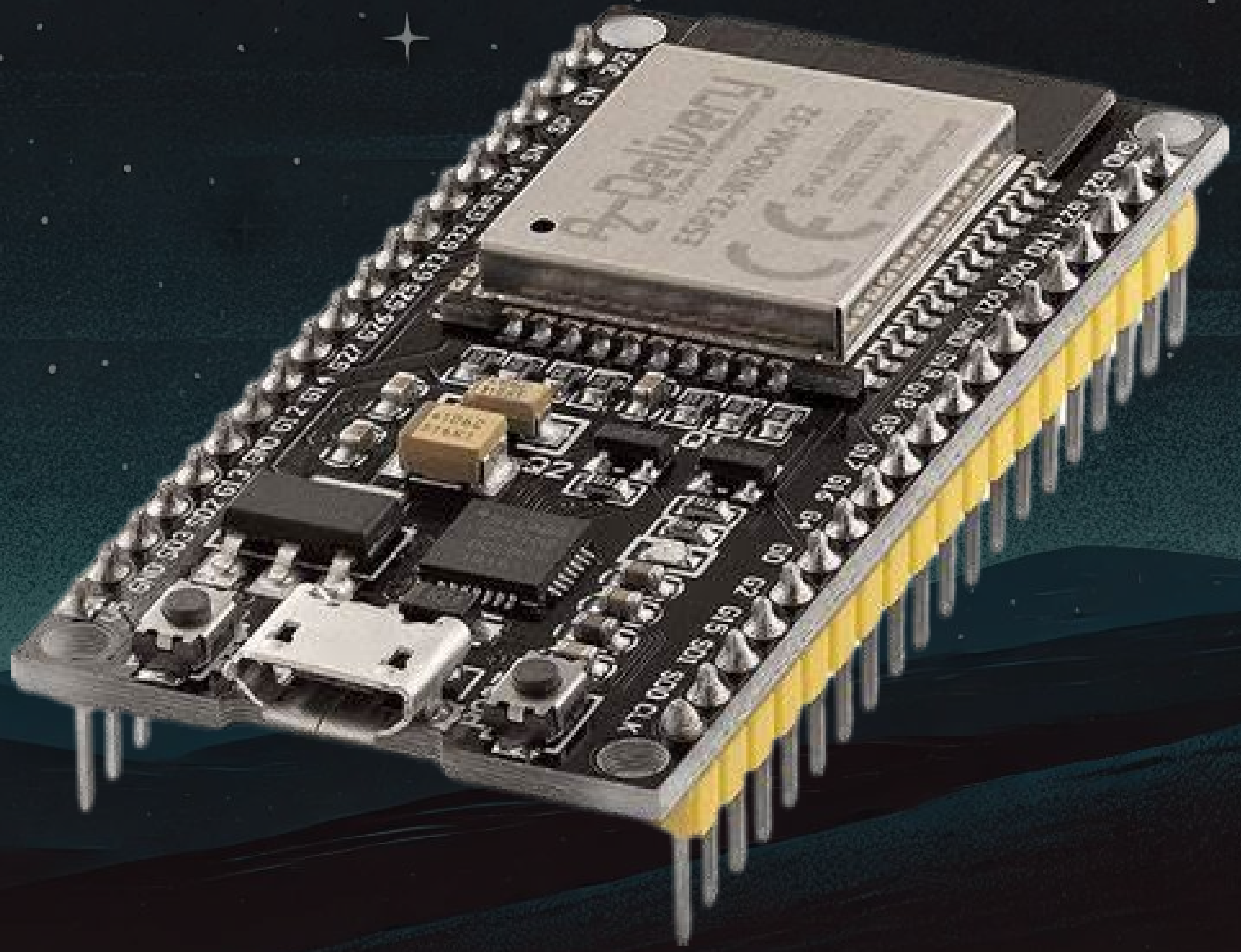
Making HTTP requests for potential score tracking.

### JSON responses

Parsing and filtering JSON responses.

### Sending data

Sending data through UART to MSP.



# MSP & Boosterpack Development

### ADC Interrupts

Setting up interrupts with ADC.

### Joystick

Implementing joystick control.

### buttons

Handling user push buttons.

### LCD Color

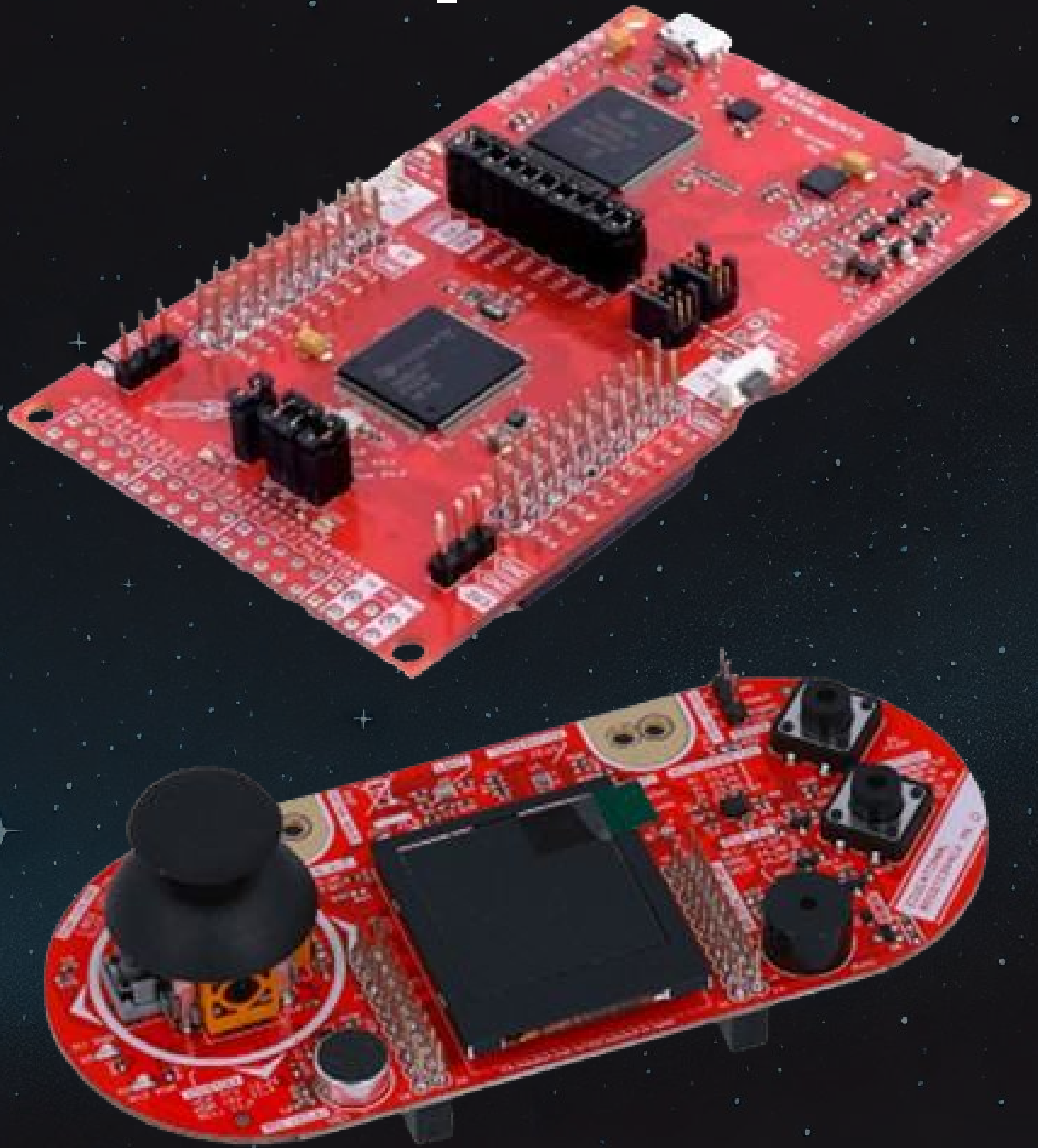
Managing the color TFT LCD.

### UI Design

Designing UI on the LCD screen.

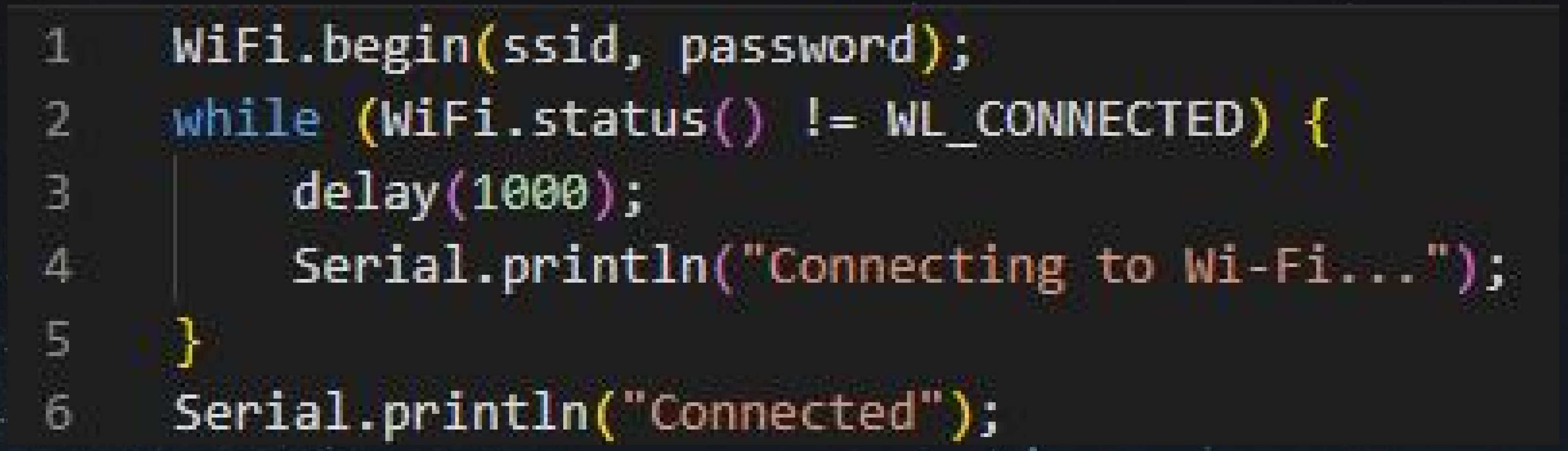
### ESP Integration

Integrating responses from ESP.



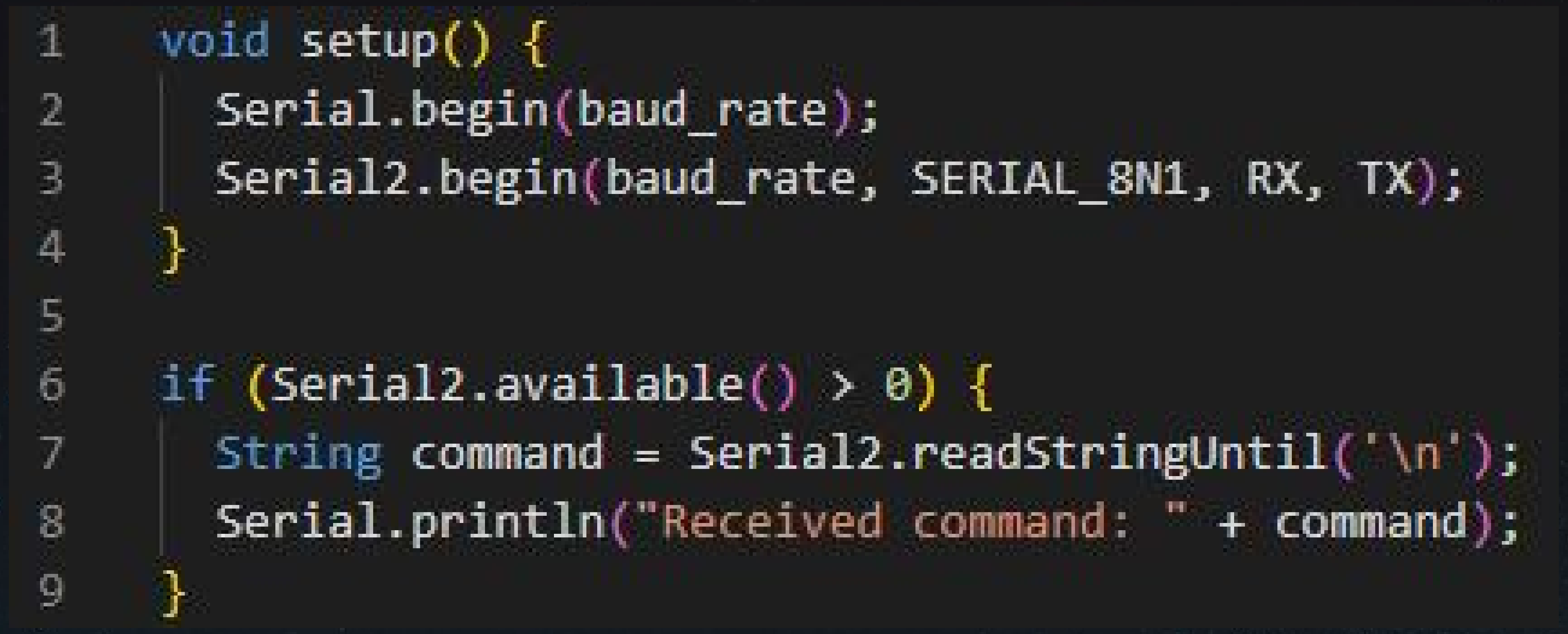
# Functionality Testing

# Wi-fi Test & UART Test



```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to Wi-Fi...");
}
Serial.println("Connected");
```

This test is essential to verify that the device can successfully connect to a Wi-Fi network. It ensures that the device is ready to communicate with other devices or servers via the network.



```
void setup() {
  Serial.begin(baud_rate);
  Serial2.begin(baud_rate, SERIAL_8N1, RX, TX);
}

if (Serial2.available() > 0) {
  String command = Serial2.readStringUntil('\n');
  Serial.println("Received command: " + command);
}
```

This test verifies the device's ability to communicate via the UART. It ensures that data can be correctly received by another device and displayed for debugging.

# Features

The features of the game

# Actions in the game

### Joystick

Left/Right Movement:

- Function: Moves the rocket left or right to avoid obstacles.

### Button S1

- Function: Starts the game from the welcome menu or restarts the game after a Game Over and pause the game

### Buzzer

- Function: Provides auditory feedback during important events, such as the start of the game, game over, or obstacle avoidance.

### Leds

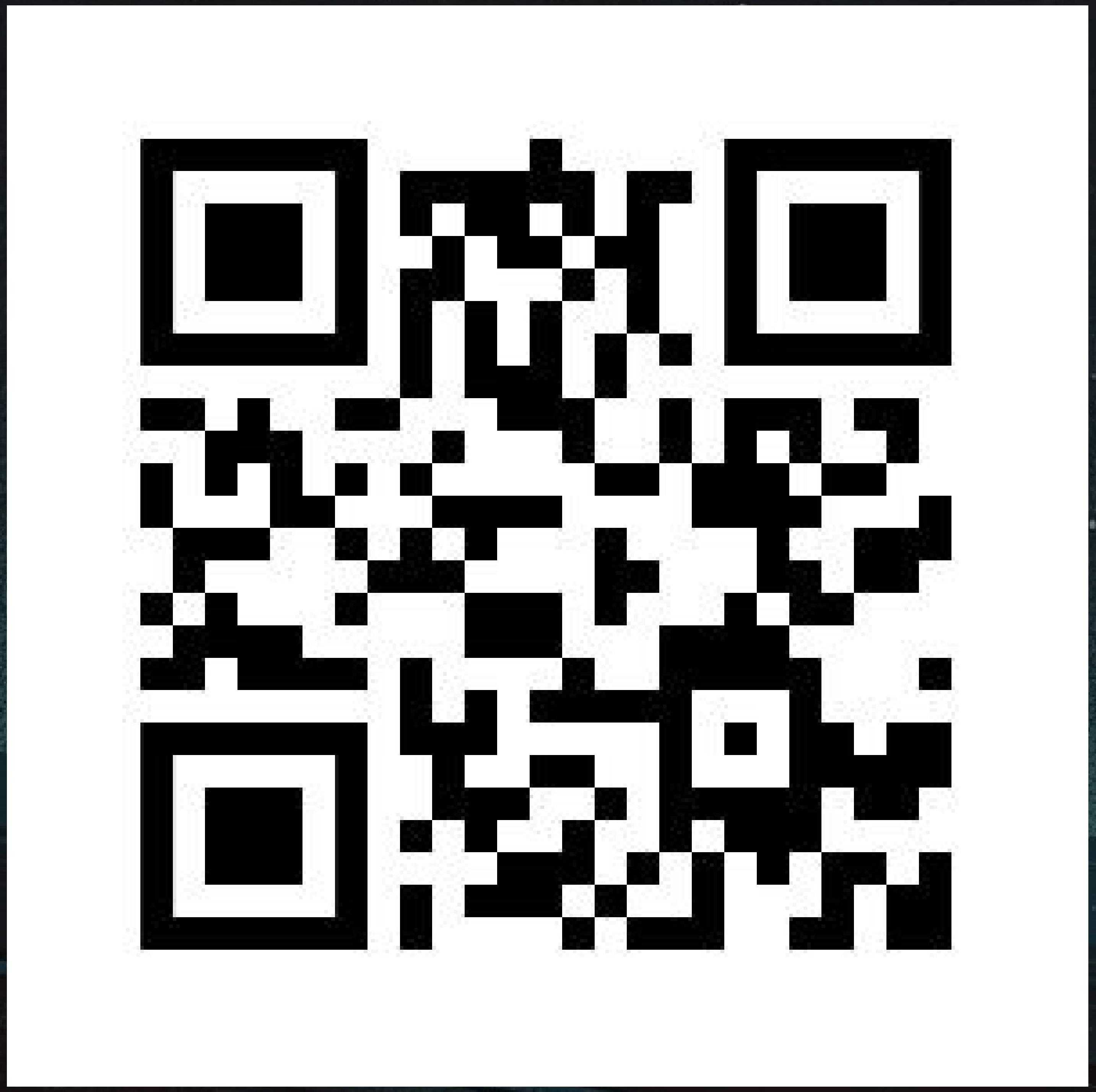
Red LED:

- Function: Indicates Game Over.

Green LED:

- Function: Indicates that the game is running and the system is ready.

# Qr-code with the video



# Future features

- **Improve graphics:** Implement advanced textures and smooth animations to improve the visual appearance of the game.
- **Use accelerometer to move the rocket:** Use the device's accelerometer to control the rocket's movements, providing a more intuitive and immersive gaming experience.
- **Receive more information through ESP with other controls:** Integrate ESP modules to receive external data, allowing new features and advanced controls to be added.