D4 – Individual Report

Kithmal Amarasinghe
akca1e22@soton.ac.uk
Team D – Demolition Man

Abstract: The purpose of this report is to present the implementation of audio, non-contact Hall Effect Sensors in the D4 System Design Exercise 2025 for a pinball table prototype. This report examines these components' design, selection, integration, and impact on the gameplay experience, accuracy, and system efficiency.

1. Contribution

My primary responsibilities during this design exercise were the implementation of the audio system and the use of Hall effect sensors to provide non-contact sensing. It was my responsibility to design and integrate an audio system that would provide theme-based background music and interactive sound effects while ensuring seamless playback and clear sound quality. In addition, I implemented Hall effect sensors to detect ball movements in real time and trigger events based on those movements.

2. Specification

The objective was to design an audio system that delivers clear, responsive, and theme-based sound effects to enhance gameplay. Additionally, a debugging mode was implemented to allow headphone output via an audio jack whilst disabling the speakers. Furthermore, non-contact Hall Effect Sensors were integrated for ball detection, enabling point deduction or increment based on the encountered obstacle.

The audio system consisted of a DFPlayer Mini module, a 3.5 mm audio jack (STX-3120), two 4-ohm, 3W speakers (AS04004PO-2-LW152-R), and a mechanical switch to disable audio output to the speakers. The DFPlayer Mini was an ideal choice due to its feature-rich design, including a built-in MP3 decoder and SD card storage support. This allowed for the upload of multiple sound effects, which could be triggered based on specific in-game scenarios. Communication with the Arduino was achieved via UART, enabling precise selection and playback of MP3 files using an indexed structure within the SD card. Additionally, the module supports a 16-bit DAC, allowing it to directly drive speakers via the Arduino's PWM output without issues. The DFPlayer Mini also features built-in equalizer settings, allowing for audio customization based on different playback conditions. These equalizer settings help enhance sound clarity and adjust tonal balance to suit various audio output scenarios. Further details on how these settings were configured and optimized will be elaborated on in the next section. The 3.5 mm audio jack featured a high durability rating of 5,000 mating cycles, ensuring long-term reliability. It also included a detection pin for headphone insertion, which could be monitored via software to implement an effective debug mode. The speakers used had a frequency response of 200 Hz – 20 kHz, ensuring clear sound output. Additionally, with an output power of up to 3W, they provided speaker audio with minimal distortion.

The sensing system utilized Hall-effect sensors (A3214) for reliable, contactless detection of game elements. These sensors were paired with N52 Neodymium permanent magnets strategically placed beneath the playing field at each sensor location. The A3214 sensors were selected for their micropower operation and polarity independence, enabling them to detect both north and south magnetic poles without requiring a specific orientation. With an operating voltage range of 2.4V to 5.5V, the sensors seamlessly integrated with the microcontroller's power supply, simplifying the circuit design. Their chopper-stabilized architecture provided excellent temperature stability and eliminated offset drift caused by environmental variations. Additionally, each sensor featured a latched digital output, ensuring stable detection even with small/fast magnetic field exposure. The sensors had a typical operate point of ±48 Gauss, allowing precise activation when the pinball passed designated checkpoints. With a rapid response time of just 60 µs, they ensured real-time tracking of fast-moving objects on the playfield. The chosen N52 Neodymium magnets were well-suited for this application due to their proximity to the sensors' operating point and their durability. When the pinball approached a sensor's location, it disrupted the existing magnetic field, momentarily altering the Hall-effect sensor's constant "ON" state. This change was detected by the microcontroller, triggering the necessary point modifications after communication with the Arduino

3. Design & Simulation

In developing the audio system architecture, I handled speaker selection, integrated an audio playback module, and ensured smooth communication with the microcontroller while prioritizing clarity and reliability. Figure 1 presents the physical layout and connections enabling effective sound output. The circuit diagram shows an Arduino Mega linked to a DFPlayer Mini, which manages MP3 playback. Sound can be routed through speakers or a 3.5mm jack, with a mechanical switch allowing users to choose their preferred option. The Arduino supplies power and control signals to the DFPlayer Mini via TX and RX pins, while MP3 files are accessed from a microSD card. The module supports both speaker output (SPK_1 & SPK_2) and DAC output (DAC_R & DAC_L) for external audio connections. To maintain stability, several hardware enhancements were introduced: a 100µF capacitor smooths voltage fluctuations, a 10kΩ pull-down resistor steadies input signals, and a diode reduces voltage from 5V to 4.2V, preventing overheating. A 3.5mm audio jack enables external playback through headphones or additional speakers, sharing the DAC output with the DFPlayer Mini. Two speakers, connected to SPK_1 and SPK_2, can be controlled via a mechanical switch, which also facilitates mode switching between normal and debugging (headphone-only) operation. This design ensured crisp sound reproduction with minimal distortion, meeting the required speaker specifications.

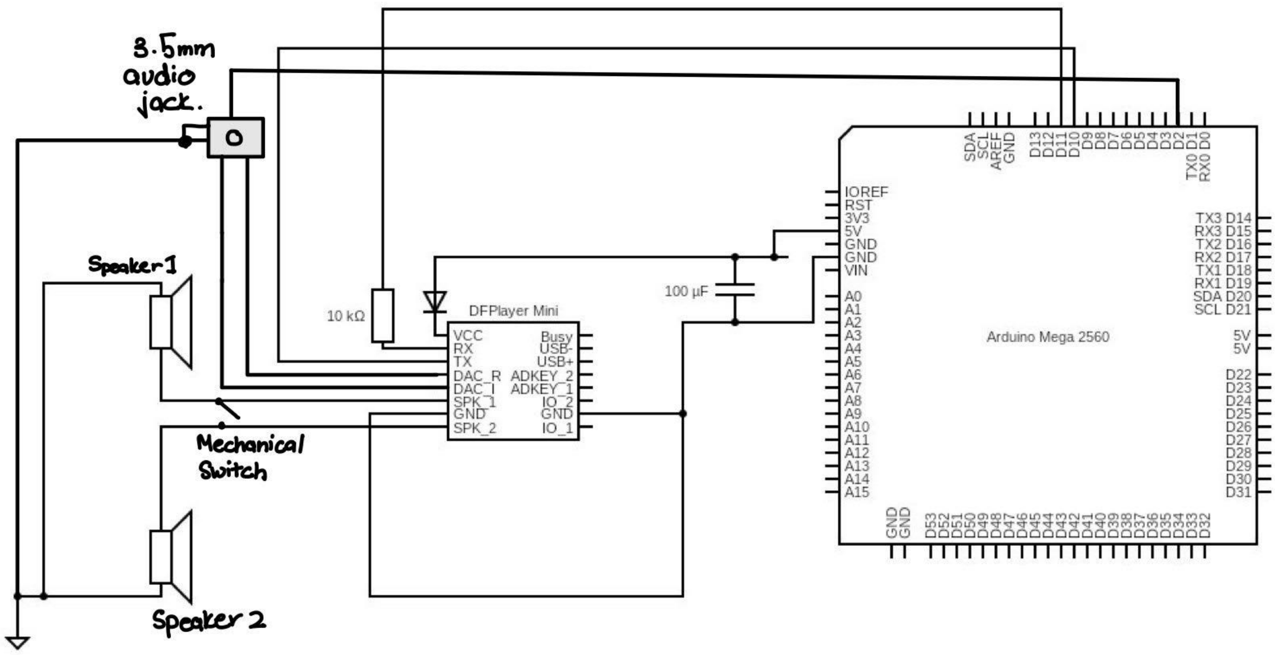
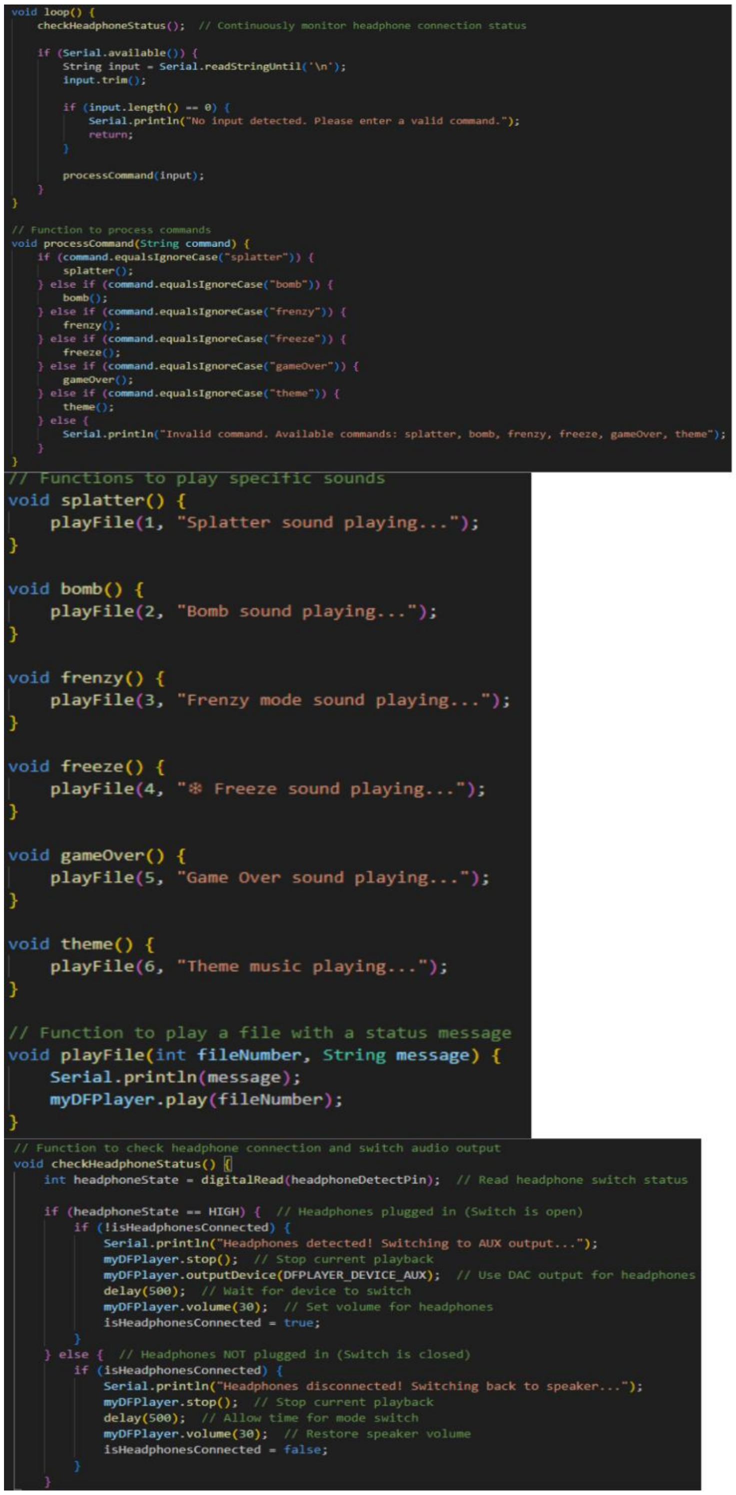


Figure 1 – Audio Architecture Circuit Diagram

While physical connections were established, software development was also a crucial aspect of the implementation. The main control loop was developed to manage audio playback, respond to user inputs, and handle communication between the Arduino and the DFPlayer Mini. This software ensured seamless operation, allowing the system to switch between audio sources, adjust playback settings, and synchronize sound effects with other system components. A visual representation of the main loop is provided in Figure 2, illustrating the logical flow of audio processing and control. Some of the code was inspired by the mentioned reference. The full code will be attached under Appendix A.



```
void loop() {
    checkHeadphoneStatus();  // Continuously monitor headphone connection status

    if (Serial.available()) {
        String input = Serial.readStringUntil('\n');
        input.trim();

        if (input.length() == 0) {
            Serial.println("No input detected. Please enter a valid command.");
            return;
        }

        processCommand(input);
    }
}

// Function to process commands
void processCommand(String command) {
    if (command.equalsIgnoreCase("splatter")) {
        splatter();
    } else if (command.equalsIgnoreCase("bomb")) {
        bomb();
    } else if (command.equalsIgnoreCase("frenzy")) {
        frenzy();
    } else if (command.equalsIgnoreCase("freeze")) {
        freeze();
    } else if (command.equalsIgnoreCase("gameOver")) {
        gameOver();
    } else if (command.equalsIgnoreCase("theme")) {
        theme();
    } else {
        Serial.println("Invalid command. Available commands: splatter, bomb, frenzy, freeze, gameOver, theme");
    }
}
// Functions to play specific sounds
void splatter() {
    playFile(1, "Splatter sound playing...");
}

void bomb() {
    playFile(2, "Bomb sound playing...");
}

void frenzy() {
    playFile(3, "Frenzy mode sound playing...");
}

void freeze() {
    playFile(4, "❄ Freeze sound playing...");
}

void gameOver() {
    playFile(5, "Game Over sound playing...");
}

void theme() {
    playFile(6, "Theme music playing...");
}

// Function to play a file with a status message
void playFile(int fileNumber, String message) {
    Serial.println(message);
    myDFPlayer.play(fileNumber);
}
// Function to check headphone connection and switch audio output
void checkHeadphoneStatus() {
    int headphoneState = digitalRead(headphoneDetectPin);  // Read headphone switch status

    if (headphoneState == HIGH) {  // Headphones plugged in (Switch is open)
        if (!isHeadphonesConnected) {
            Serial.println("Headphones detected! Switching to AUX output...");
            myDFPlayer.stop();  // Stop current playback
            myDFPlayer.outputDevice(DFPLAYER_DEVICE_AUX);  // Use DAC output for headphones
            delay(500);  // Wait for device to switch
            myDFPlayer.volume(30);  // Set volume for headphones
            isHeadphonesConnected = true;
        }
    } else {  // Headphones NOT plugged in (Switch is closed)
        if (isHeadphonesConnected) {
            Serial.println("Headphones disconnected! Switching back to speaker...");
            myDFPlayer.stop();  // Stop current playback
            delay(500);  // Allow time for mode switch
            myDFPlayer.volume(30);  // Restore speaker volume
            isHeadphonesConnected = false;
        }
    }
```

Figure 2 – Audio Architecture Code [1]

Implementing the above architecture required extensive testing and optimization. Equalization was effectively managed through the DFPlayer Mini and software-based adjustments. To improve audio quality, reverberation and delay effects were introduced, adding depth and realism to sound effects. For example, these enhancements made the sound of a ball impact more dramatic. Although the system initially used a mono speaker, a stereo expansion method was considered for future updates. Figure 3 illustrates the oscilloscope trace of the speaker's output after these improvements, representing the final waveform.

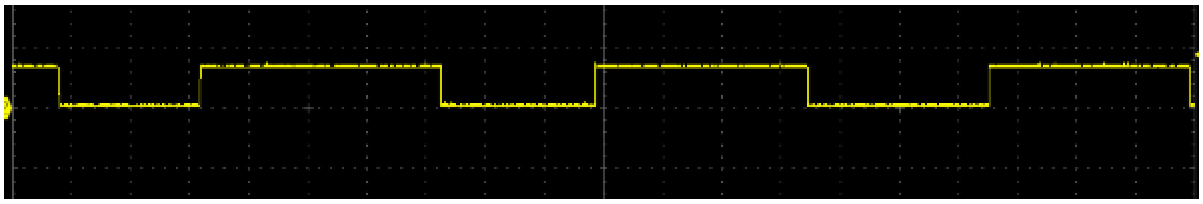
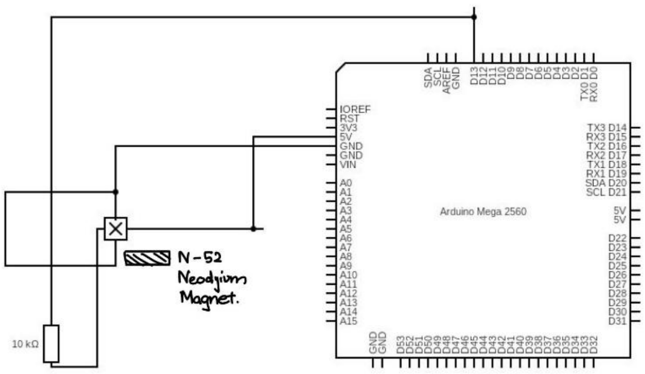


Figure 3 – Oscilloscope trace for Audio

To ensure clear, distortion-free, and noise-resistant audio output, thorough debugging and optimization were performed. Given that the system contained multiple electromechanical components, such as solenoids, LEDs, and microcontrollers, unwanted interference could impact the audio system. Several techniques were implemented to minimize these effects. First, a 100µF capacitor (seen in Figure 1) was introduced to smooth transient voltage spikes, preventing fluctuations that could disrupt audio playback. Next, due to interference from high-power components, a common ground was established for external components such as the speakers and audio jack, reducing audible noise, including the unwanted "hum" caused by grounding issues. Finally, load testing was conducted to ensure that the speakers could handle continuous music playback while the game was running. This helped verify that prolonged operation would not cause overheating or degradation in audio quality. Through these optimizations, the system achieved consistent, high-quality audio output, ensuring an immersive and interference-free experience.

As part of the non-contact sensing mechanism, I was responsible for selecting and integrating a suitable Hall effect sensor and a corresponding permanent magnet into the electronic system. After careful consideration, we chose the A3214 Hall effect sensor in combination with an N-52 neodymium magnet for optimal performance. Figure 4 illustrates the physical connections within the system, including a circuit diagram that depicts the Arduino interfaced with the Hall effect sensor and a 10kΩ pull-down resistor. The Hall effect sensor detects changes in the surrounding magnetic field, triggered by the presence of the N-52 neodymium magnet. In this setup, when the ferromagnetic pinball rolls over a specific region where the sensor is active, the constant "ON" state of the sensor is changed due to a change detected in the existing magnetic field. The sensor's changed output is then read by the Arduino. The 10kΩ pull-down resistor ensures that when no magnet is detected, the sensor's output remains at a stable LOW state rather than floating. The Arduino continuously monitors the sensor's output, enabling it to detect real-time changes in the magnetic field. Furthermore, Figure 5 illustrates the sensor's placement within the gameplay area, showing how it interacts with the pinball as part of the system.

Figure 4 – Hall Effect Sensing Architecture Circuit Diagram



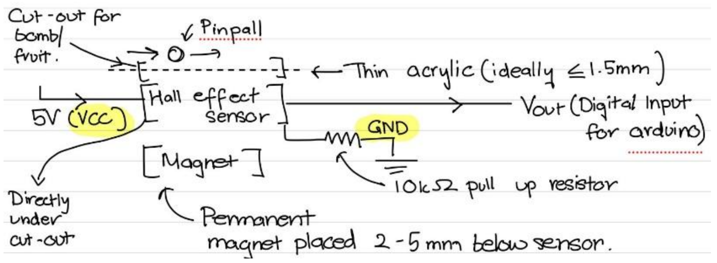
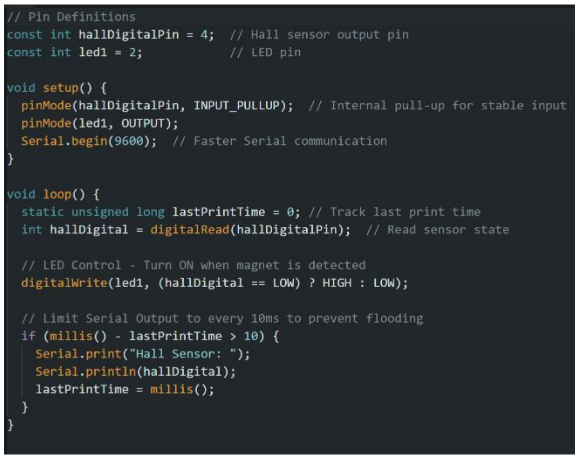


Figure 5 – Hall Effect Sensor Setup

While the physical connections between the Hall effect sensor and the Arduino were established, software development was equally crucial for the system's implementation. The main control loop was designed to continuously monitor the sensor's output, detect changes in the magnetic field, and respond accordingly. This software ensured seamless operation by processing sensor data, triggering appropriate actions based on magnetic field variations, and facilitating communication between the Arduino and other system components. A visual representation of the main loop is provided in Figure 6, illustrating the logical flow of sensor data processing and response. The complete code, which was adapted from the referenced source, is included in Appendix A for reference. During initial testing, an LED was connected to provide a clear visual indication of magnetic field changes. As shown in Figure 7 and referenced in the code in Figure 6, the LED remained on when no change in the magnetic field was detected. However, as soon as a change occurred, the LED turned off, alerting the user to the magnetic field variation.



```
// Pin Definitions
const int hallDigitalPin = 4;  // Hall sensor output pin
const int led1 = 2;            // LED pin

void setup() {
  pinMode(hallDigitalPin, INPUT_PULLUP);  // Internal pull-up for stable input
  pinMode(led1, OUTPUT);
  Serial.begin(9600);  // Faster Serial communication
}

void loop() {
  static unsigned long lastPrintTime = 0; // Track last print time
  int hallDigital = digitalRead(hallDigitalPin);  // Read sensor state

  // LED Control - Turn ON when magnet is detected
  digitalWrite(led1, (hallDigital == LOW) ? HIGH : LOW);

  // Limit Serial Output to every 10ms to prevent flooding
  if (millis() - lastPrintTime > 10) {
    Serial.print("Hall Sensor: ");
    Serial.println(hallDigital);
    lastPrintTime = millis();
  }
}
```

Figure 6 – Hall Effect Sensor Code [2]

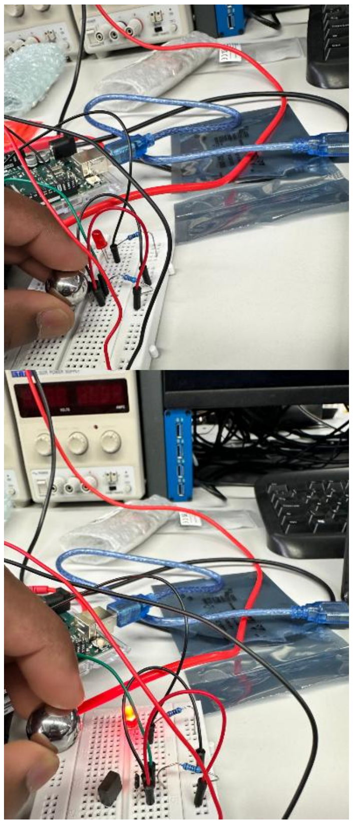
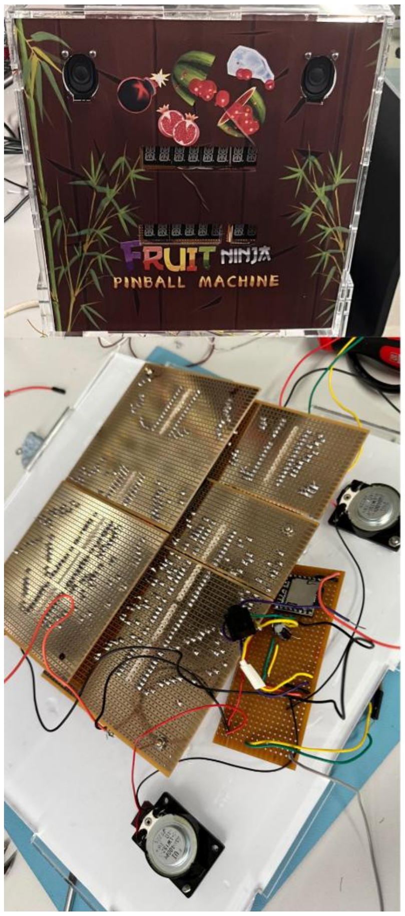


Figure 7 – Detection Example [2]

Other sensor designs were considered, including inductive sensors, since the ball can conduct electricity. While inductive sensors would have been well-suited for this application, they were ultimately not chosen due to their high cost. Optical sensors, such as IR break beam sensors, were also evaluated. However, they were deemed unsuitable because the ball rolls over a very small and specific area, leaving insufficient space to implement a cost-effective and reliable optical sensor. As a result, the Hall effect sensor was selected as the optimal choice, offering both a cost-effective and reliable solution for detecting ball movements.

4. Testing & Results



To evaluate audio performance, a frequency response analysis was conducted using an oscilloscope and a frequency generator, confirming a stable operating range from 200Hz to 20kHz. Output power measurements verified that the speakers consistently received a 3W output without noticeable distortion. Latency testing was performed to measure playback delay in response to game triggers, yielding an average response time of approximately 10ms. Additionally, interfacing tests ensured seamless communication between the DFPlayer Mini, amplifier, and microcontroller, confirming reliable integration. Once the basic functionality was verified, further testing was conducted, including stress testing and loudness testing. The stress test involved programming the microcontroller to instruct the DFPlayer Mini to output sound continuously at one-second intervals, which the system handled successfully without any issues. The loudness test, performed using the built-in software of the DFPlayer Mini, confirmed that the speakers produced clear and reliable sound, enhancing the immersive gaming experience. Figure 8 illustrates the system assembly and integration, depicting the overall connections and setup. As shown, the speakers were mounted on the headboard, and the circuit from Section 3 was soldered onto a stripboard to enhance modularity within the system. Audio was fully functional during submission time.

Figure 8 – Mounting & System Implementation of Audio

To evaluate the performance of the Hall Effect sensors used as non-contact detectors, various tests were conducted. Detection accuracy was assessed across different ball speeds and placements, achieving an accuracy of over 80%. Response time measurements, performed using an oscilloscope, confirmed reliable detection within 4–5 milliseconds. False trigger testing was carried out to determine misread rates, with results showing an error rate of less than 10% after calibration. Additionally, microcontroller integration testing ensured seamless sensor-to-microcontroller data transmission under real-world conditions. During initial mounting, the testing circuit from Section 3 was left to easily visualise sensor placement calibration, as shown in Figure 9. Once calibration and sensitivity checks were successfully completed, the sensors were integrated into the gameplay board, depicted in Figure 10. All fruits, modes, and bombs that sit flush with the board were implemented using these sensors. A final round of stress testing was conducted to verify the durability and effectiveness of the sensors. This included high-speed impact testing and mechanical wear assessment, as the heavy pinball (relative to the sensor component) repeatedly passed over the sensors. All tests were successfully passed, and the final product is shown in Figure 10. 5 out of 6 non-contact sensors used were functional during submission.

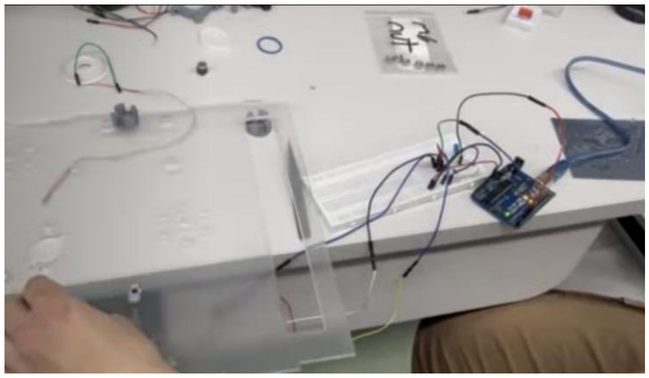


Figure 9 – Mounting & System Implementation of Non-contact Sensor

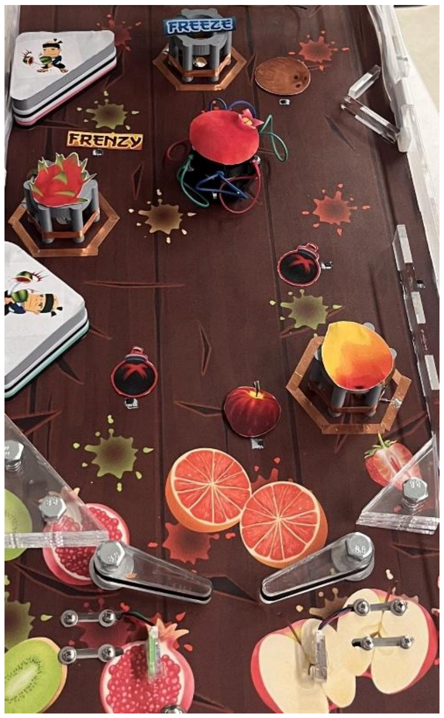


Figure 10 – Final Product

5. Management/Team Working

I managed my contributions by dividing tasks into milestones and following a structured workflow that included design, simulation, testing, and integration. Collaboration with my teammates allowed me to align my work with overall system requirements. Our team adopted a modular approach, dividing the project into multiple subsystems as outlined in Proposal Form 2. We followed the Scrum framework to enhance project management and adaptability. The team held weekly Scrum meetings (Sprint Planning and Stand-ups) to track progress, discuss challenges, and adjust plans as needed. During the project, tasks were organized into short sprints, ensuring timely delivery. We used Microsoft Teams to share documents, allowing each team member to access the necessary information from other subsystems. Additionally, to-do lists acted as our Sprint Backlog, printed for each member before every scheduled lab session, ensuring that everyone had clear objectives to achieve by the end of the session. Risk management played a key role in our project planning. We identified potential failure points such as sensor misalignment, audio distortion, and power supply fluctuations. To mitigate these risks, we began full subsystem integration well in advance of the final deadline. Additionally, alternative game modes were programmed in case of sensor failures as contingency plans. As part of our Scrum framework, debugging sessions serve as Sprint Reviews and Retrospectives for smooth systems functionality. The subsystems were tested independently before being integrated, ensuring smooth communication between them and reducing any last-minute issues. Team collaboration was further strengthened through peer reviews, where members tested and provided feedback on different subsystems. This structured Scrum-based approach helped us maintain efficiency, reduce potential errors, and ensure a well-integrated final product

6. Critical Evaluation & Reflection

The project successfully delivered a functional audio system and a reliable ball detection mechanism, significantly enhancing gameplay. However, challenges arose, including initial signal interference in the audio system, which was resolved through proper grounding and filtering, and Hall effect sensor misalignment in early tests, requiring recalibration for optimal detection. Furthermore, limited data collection during testing, where I could have gathered more oscilloscope trace captures and collated extensive data to create graphs showcasing system responsiveness and performance trends, significantly reduced proof of a working product. Although the full system was initially functional, a short between the ground and power supply, caused by poor cable management, resulted in the entire system failing. Unfortunately, due to time constraints, we were unable to locate and resolve the short before submission. Had more time been available, a fully functioning product could have been submitted. Despite this, the working system can still be seen in Figure 11. Key takeaways from the project include the importance of early integration testing, robust calibration methods, and detailed data collection for performance verification. Improved time management and earlier prototyping could have further enhanced system performance. Despite these challenges, the project showcased effective teamwork, problem-solving, and technical execution, resulting in a well-functioning and engaging pinball system.

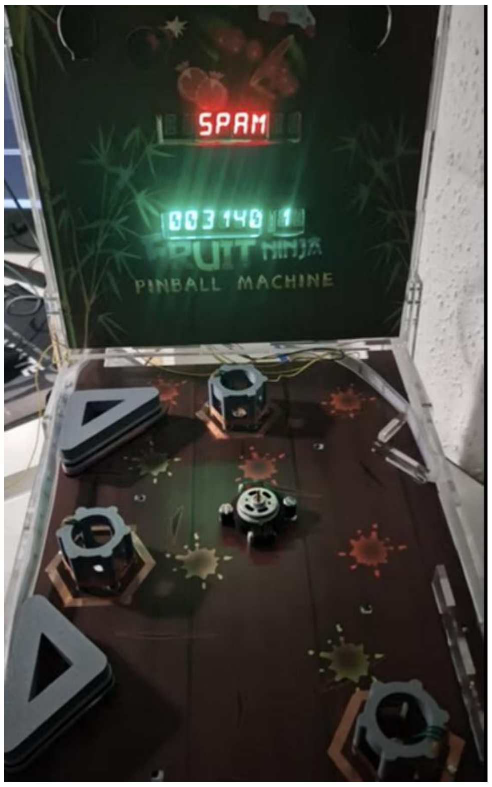


Figure 11 – Working Pinball Machine [3]

References

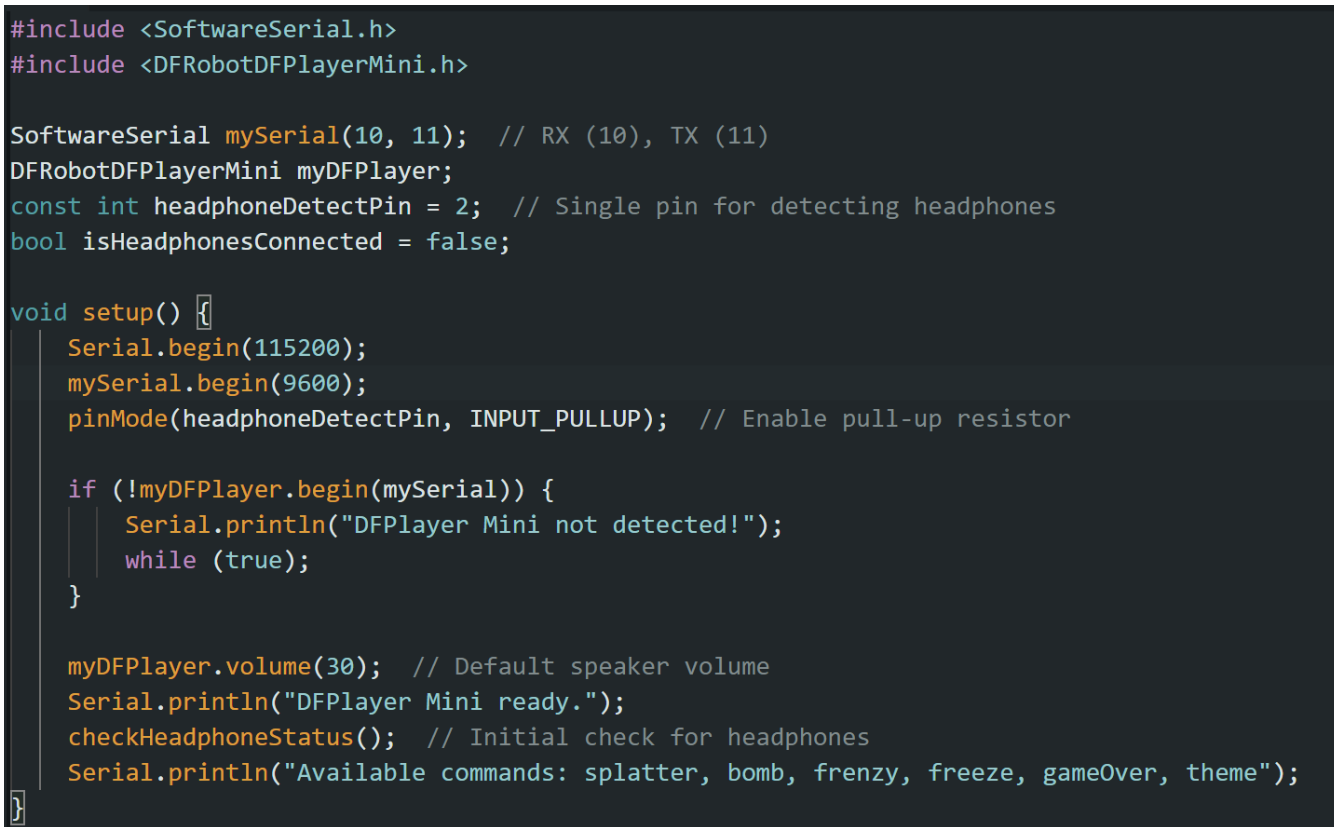
[1] Indrek, " DfPlayer Mini Module - Play MP3 Files With an Arduino (Step-by-step Guide)," YouTube, 2020. Available: https://www.youtube.com/watch?v=P42ICrgAtS4&t=79s. Referenced in February 2025

[2] Science Buddies, " How to Use a Hall Effect Sensor with Arduino (Lesson #31)," YouTube, 2020. Available: https://www.youtube.com/watch?v=q4BkhwoGzbM&t=24s. Referenced in February 2025

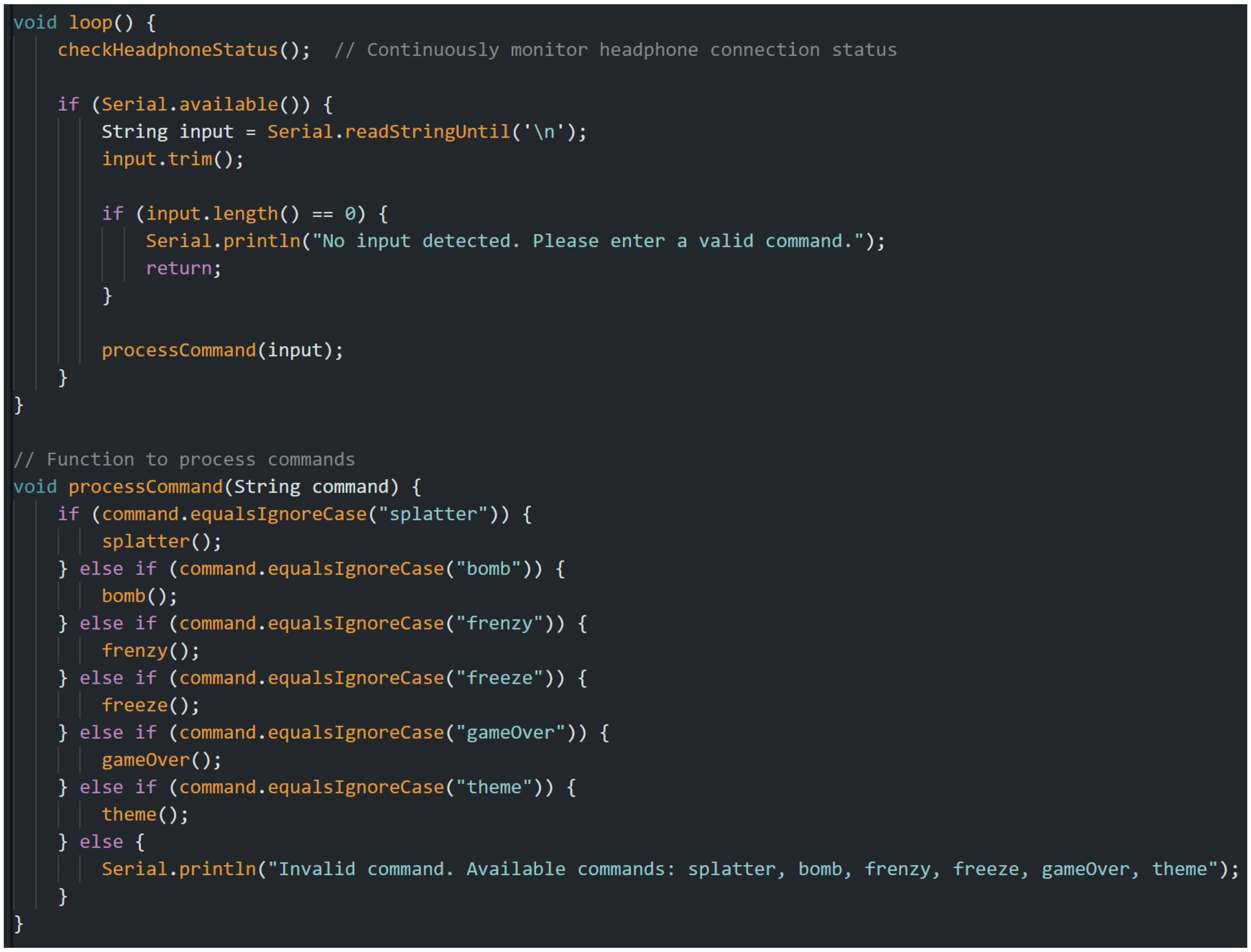
[3] Isaac Puffet, "Working Pinball Machine" Personal Collection, March 2025.

Appendix A

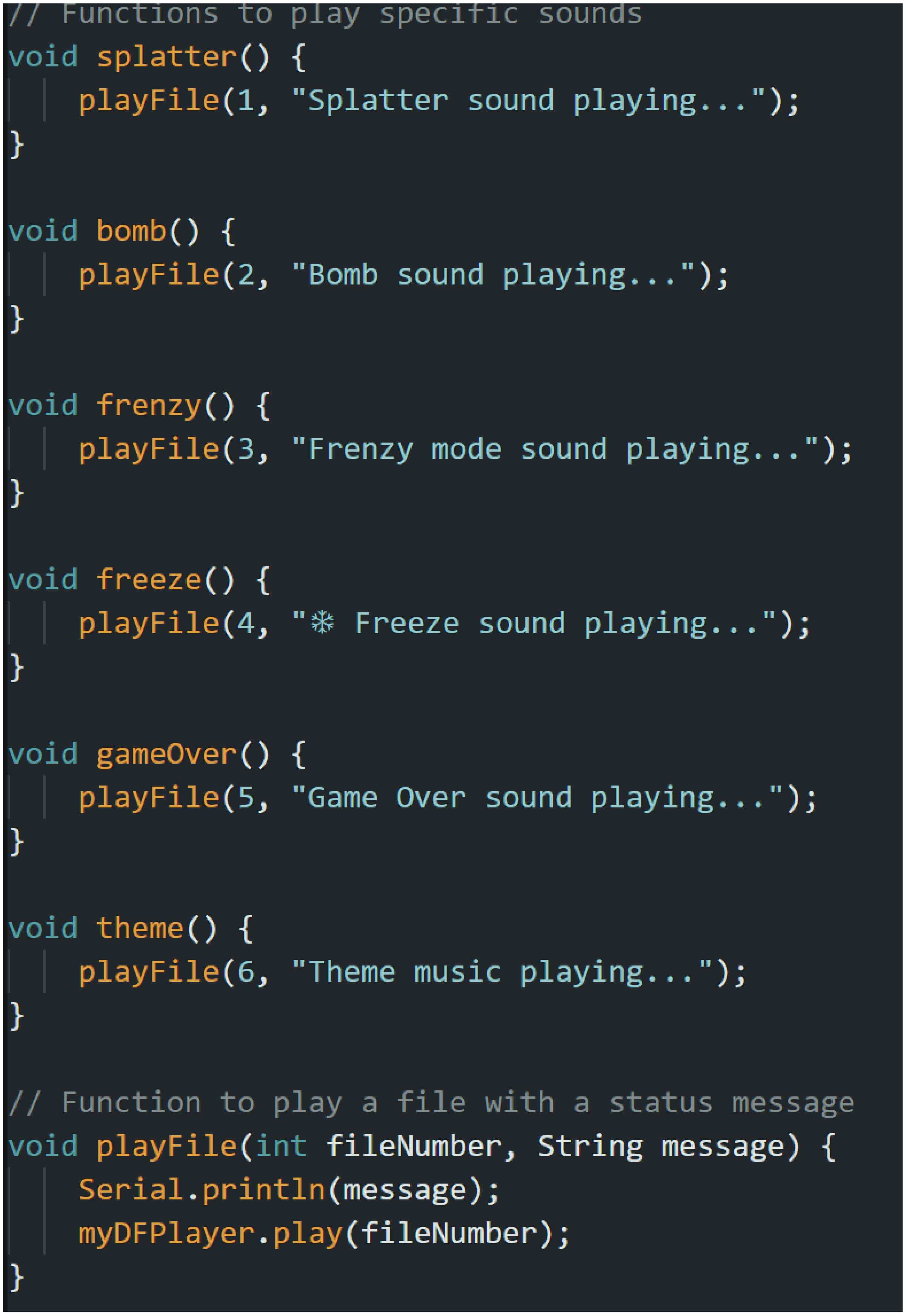
Code for Audio:



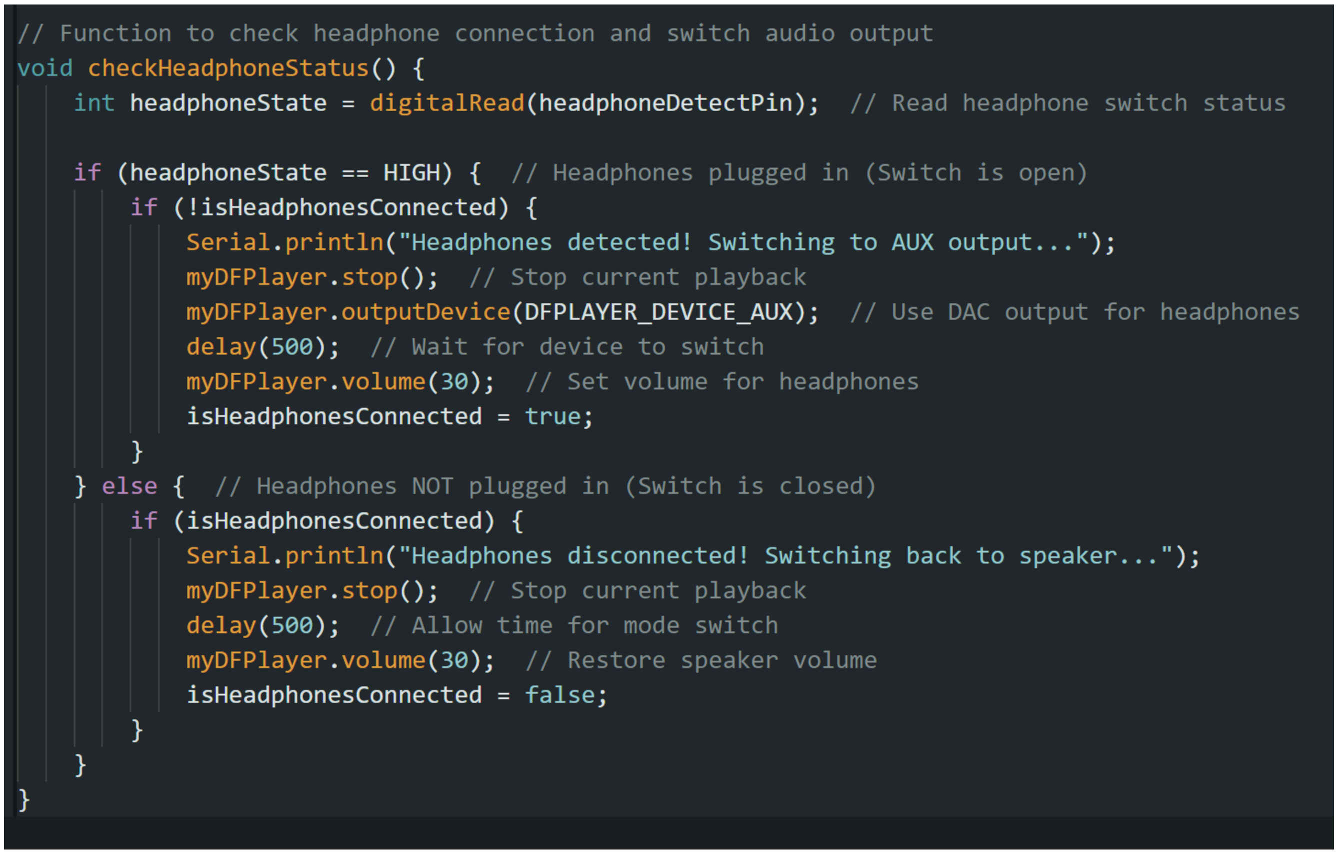
```
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>

SoftwareSerial mySerial(10, 11);  // RX (10), TX (11)
DFRobotDFPlayerMini myDFPlayer;
const int headphoneDetectPin = 2;  // Single pin for detecting headphones
bool isHeadphonesConnected = false;

void setup() {
    Serial.begin(115200);
    mySerial.begin(9600);
    pinMode(headphoneDetectPin, INPUT_PULLUP);  // Enable pull-up resistor

    if (!myDFPlayer.begin(mySerial)) {
        Serial.println("DFPlayer Mini not detected!");
        while (true);
    }

    myDFPlayer.volume(30);  // Default speaker volume
    Serial.println("DFPlayer Mini ready.");
    checkHeadphoneStatus();  // Initial check for headphones
    Serial.println("Available commands: splatter, bomb, frenzy, freeze, gameOver, theme");
}
```



```
void loop() {
    checkHeadphoneStatus();  // Continuously monitor headphone connection status

    if (Serial.available()) {
        String input = Serial.readStringUntil('\n');
        input.trim();

        if (input.length() == 0) {
            Serial.println("No input detected. Please enter a valid command.");
            return;
        }

        processCommand(input);
    }
}

// Function to process commands
void processCommand(String command) {
    if (command.equalsIgnoreCase("splatter")) {
        splatter();
    } else if (command.equalsIgnoreCase("bomb")) {
        bomb();
    } else if (command.equalsIgnoreCase("frenzy")) {
        frenzy();
    } else if (command.equalsIgnoreCase("freeze")) {
        freeze();
    } else if (command.equalsIgnoreCase("gameOver")) {
        gameOver();
    } else if (command.equalsIgnoreCase("theme")) {
        theme();
    } else {
        Serial.println("Invalid command. Available commands: splatter, bomb, frenzy, freeze, gameOver, theme");
    }
}
```

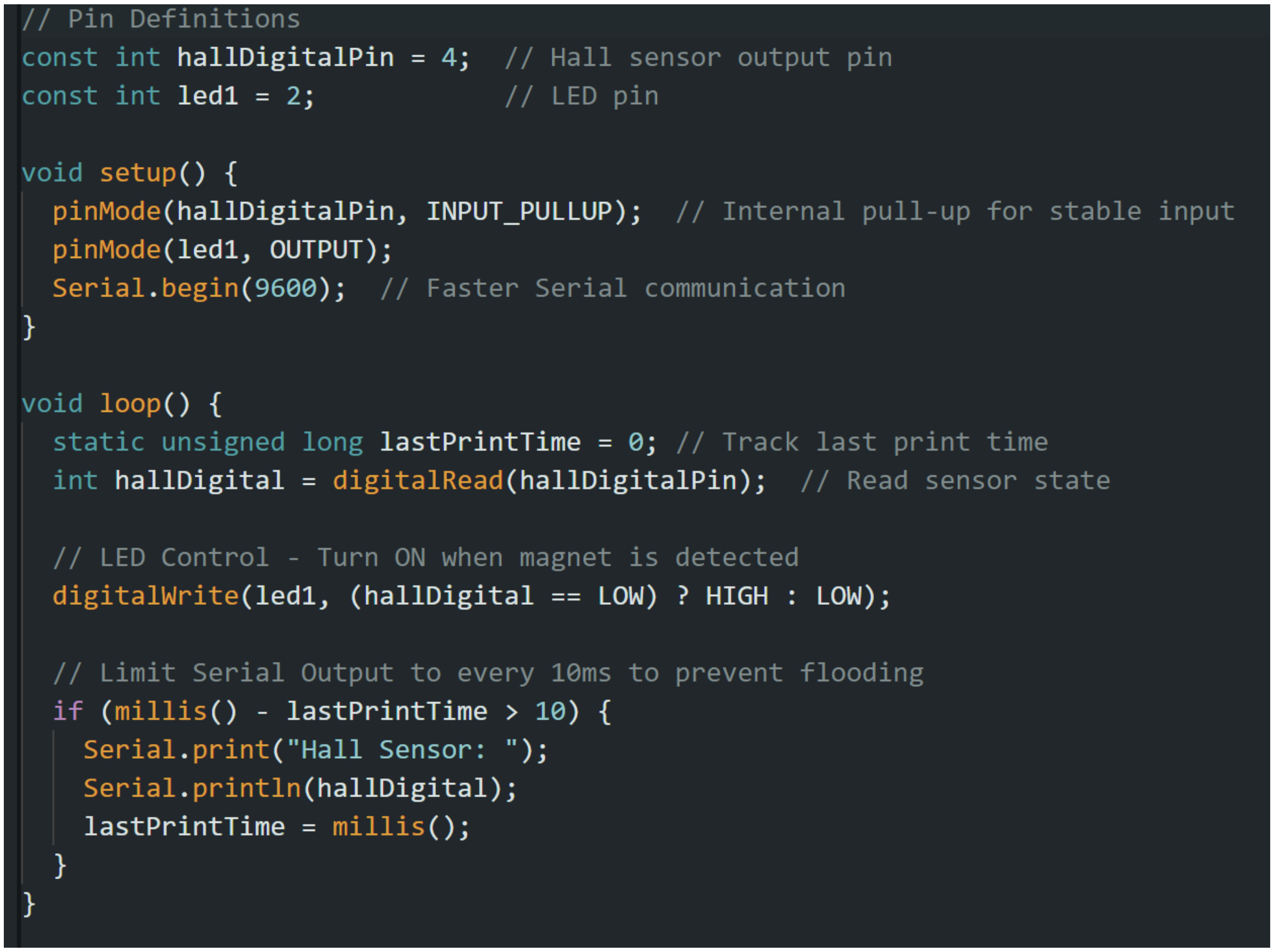


```
// Functions to play specific sounds
void splatter() {
    playFile(1, "Splatter sound playing...");
}

void bomb() {
    playFile(2, "Bomb sound playing...");
}

void frenzy() {
    playFile(3, "Frenzy mode sound playing...");
}

void freeze() {
    playFile(4, "❄ Freeze sound playing...");
}

void gameOver() {
    playFile(5, "Game Over sound playing...");
}

void theme() {
    playFile(6, "Theme music playing...");
}

// Function to play a file with a status message
void playFile(int fileNumber, String message) {
    Serial.println(message);
    myDFPlayer.play(fileNumber);
}
```



```
// Function to check headphone connection and switch audio output
void checkHeadphoneStatus() {
    int headphoneState = digitalRead(headphoneDetectPin);  // Read headphone switch status

    if (headphoneState == HIGH) {  // Headphones plugged in (Switch is open)
        if (!isHeadphonesConnected) {
            Serial.println("Headphones detected! Switching to AUX output...");
            myDFPlayer.stop();  // Stop current playback
            myDFPlayer.outputDevice(DFPLAYER_DEVICE_AUX);  // Use DAC output for headphones
            delay(500);  // Wait for device to switch
            myDFPlayer.volume(30);  // Set volume for headphones
            isHeadphonesConnected = true;
        }
    } else {  // Headphones NOT plugged in (Switch is closed)
        if (isHeadphonesConnected) {
            Serial.println("Headphones disconnected! Switching back to speaker...");
            myDFPlayer.stop();  // Stop current playback
            delay(500);  // Allow time for mode switch
            myDFPlayer.volume(30);  // Restore speaker volume
            isHeadphonesConnected = false;
        }
    }
}
```

Code for Hall Effect Sensor (Non-Contact)



```
// Pin Definitions
const int hallDigitalPin = 4;  // Hall sensor output pin
const int led1 = 2;            // LED pin

void setup() {
  pinMode(hallDigitalPin, INPUT_PULLUP);  // Internal pull-up for stable input
  pinMode(led1, OUTPUT);
  Serial.begin(9600);  // Faster Serial communication
}

void loop() {
  static unsigned long lastPrintTime = 0; // Track last print time
  int hallDigital = digitalRead(hallDigitalPin);  // Read sensor state

  // LED Control - Turn ON when magnet is detected
  digitalWrite(led1, (hallDigital == LOW) ? HIGH : LOW);

  // Limit Serial Output to every 10ms to prevent flooding
  if (millis() - lastPrintTime > 10) {
    Serial.print("Hall Sensor: ");
    Serial.println(hallDigital);
    lastPrintTime = millis();
  }
}
```