MIE438 - Final Project Report

Laser Tag System

April 12th, 2023

Group 12
Jasmine Chen - 1005024910
Nathalie Cristofaro - 1004838009
Kevon Seechan - 1004941708

Video: https://youtu.be/BiPX4E4-nns

Table of Contents

1.0 Introduction

Laser tag is an expensive game, and coordinating a game for a large group of people at an arcade can be difficult and time-consuming. An at-home laser tag system would be convenient for smaller groups of two or more people and provide a cheaper alternative.

The high-level objective of this project is to design and build a laser tag system that is capable of (i) emitting a unique IR signal to each individual player, (ii) receiving the IR signal, and (iii) using this concept, designing an embedded system with communications between multiple components in order to create an integrated gaming experience. (iv) The IR emitter will be triggered using push buttons. The unit should also be capable of (vi) lighting up, playing sound, and displaying points and player information to create audio and visual confirmation for the players.

2.0 Preliminary Goals and Plans

2.1 Concept and Idea Generation

2.1.1 IR Signal Modulation

Since infrared light is emitted continuously from natural/artificial light sources, there will be a lot of noise that might interfere with the system. To prevent noise interference, a carrier frequency of 38kHz has been chosen for the transmitter since the frequency is uncommon in nature and will therefore be distinguishable from any other IR sources. The IR signal will be modulated through an encoder or IC timer that generates pulsed signals. Figure 1 shows a simple circuit diagram for the IR pulse generator.

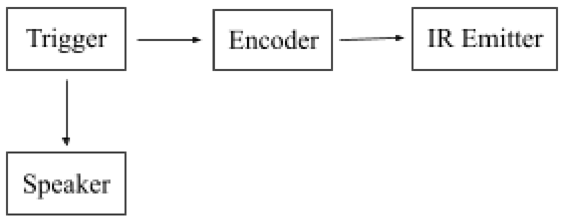

Figure 1: IR Pulse Generator Circuit Diagram.

2.1.2 Receiving the IR Signal

To filter out the noise, the receiver diode requires a band-pass filter that only allows an IR signal of 38kHz to pass through. The signal will then be amplified using a pre-amplifier and converted to a binary signal before being sent to the microcontroller. Figure 2 shows a simple circuit diagram for the receiver.

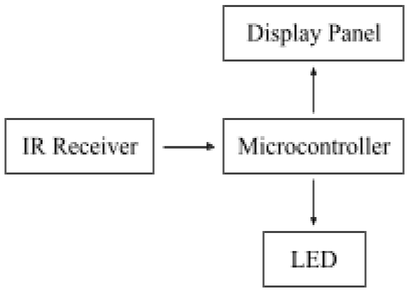

Figure 2: IR Receiver Circuit Diagram.

2.1.3 Player Identification and Trigger Confirmation

To ensure the system is able to identify the signals and associate them with the players accordingly, a display panel will be implemented to display the name of the tagger. LED lights and speakers will also be used as audio and visual confirmation for when the IR emitter and receiver are triggered.

2.2 Team Design Process and Scheduling

The general flow of the design process can be seen in Figure 3 below.

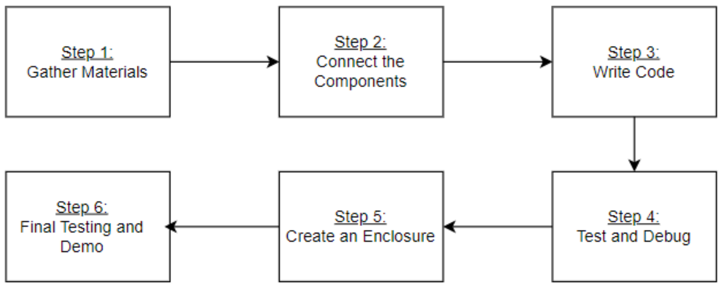

Figure 3: Design Flowchart.

To further streamline the design process, the team determined that the laser tag system can be broken into 4 subsections. The first subsection involves the components associated with the emitters. The second is made up of the receiver components. The third subsection consists of the code that controls the emitter and receivers. Finally, the fourth subsection deals with enclosures for all the parts.

The course syllabus states that the final report for the project is due on April 12th [1]. The project proposal was approved around February 6th, which gave the team approximately 9 weeks to complete the project. Additionally, all team members had a two hour gap on Fridays after

MIE438 lectures and as such, the time was used for team meetings and prototyping. See Appendix A for a breakdown of the weekly plan that was used for this project.

Research shows that the scope of work associated with this project is reasonable using the material taught in MIE438. For example, in ‘A wearable laser tag module powered by PGM-induced force feedback,’ the authors develop a complex laser tag game that uses an Arduino Mega microcontroller to control their emitter and receiver systems [2]. This is evidence that the team’s plan is viable. Additionally, there are multiple tutorials online which make use of different variations of the Arduino board to create laser tag systems of varying complexity [3, 4]. These tutorials further support that the laser tag system would be successful.

3.0 Design Changes

This section outlines the changes made throughout the project design phase and how they affected the overall system. The three main changes related to the IR receiver, encoded signal, and flex sensors.

IR Receiver

In the preliminary design, an IR receiver was chosen to receive the IR signal from the emitter. However, the parts received were faulty, and the team was forced to switch to a 2-pin phototransistor as it was the only option available in Myhal Fabrication. The IR receiver was initially chosen because the system would decode the received signal to help identify the player ID. Once the signal would be received, the IR receiver would convert the modulated signal into a binary signal and pass it along to the Arduino. However, a phototransistor does not have the demodulation circuit required to convert the encoded signal to binary. The photoresistor can only detect continuous light signals, thus it can only send an analog signal to the Arduino. Due to these hardware limitations, the team had to remove the function of player identification.

Encoded Signal

One of the objectives was player identification, which required a unique encoded signal for each player. However as mentioned above, due to faulty parts, the system is unable to interpret an encoded signal.

Flex Sensor

In addition to the design changes above, some resources from the proposed BOM were not available from the Myhal Fabrication Facility due to supply shortages. The flex sensors were not in stock and could not be incorporated into the system as originally planned, however these sensors were replaced with push buttons to trigger laser emission.

4.0 Final Design

The final design was successfully prototyped and demonstrated the laser tag system outlined in Sections 1.0 and 2.0. This section describes the function of the microcontroller, how users can interact with this embedded system, how the design operates, and also provides a revised bill of materials. Figure 4 shows an image of the final design.

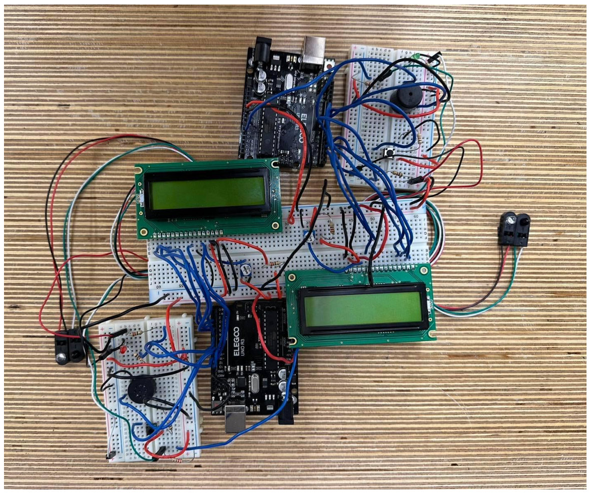

Figure 4: Final Design

4.1 Microcontroller Functionality Overview

Each laser tag player must carry their own laser transmitter, laser receiver, and point monitor, which are all controlled by an Arduino Uno microcontroller. An Arduino Uno was chosen for this project because Arduinos are typically low-cost, flexible, and easy-to-program microcontrollers that can be integrated into a variety of embedded system projects. As the main objectives for this design were to minimize cost and increase accessibility for players, Arduinos were best suited for this project.

Arduinos are compatible with a variety of different inputs and outputs, and this system requires the control and feedback of many types of components such as LEDs, speakers, IR sensors, LCDs, and more. These components are controlled via the Arduino which is programmed in C++. A limitation of many programming languages is their dependency on a programming software, the requirement of a compiler to upload/debug the program, and limited online resources to assist with programming functions. However, Arduinos do not have this issue, as one can use the Arduino Web Editor and code is readily available online, making it easy to troubleshoot program bugs for beginners if they wish to implement a similar system. In addition, the Arduino Web Editor and Arduino IDE portal come with integrated compilers, so a compiler does not need to be installed by the user. This makes the program accessible and customizable for anyone with a portable device, WiFi connection (if using the web portal), and USB-A cable.

The project required 12 digital pins for all its inputs and outputs, and an Arduino Uno offers 14 digital input/output (2 of which are the transmitting/receiving pins), thus this microcontroller contained the perfect amount of pins for this project. Other boards such as Arduino Megas and Raspberry Pis have more pins and many additional/complex features which were not necessary for this project, and these boards would only increase the overall cost for feature users. An Arduino Nano was not used for this project, as they typically require to be soldered to header pins so they can connect to inputs and outputs, which would not meet the team's objective of being user-friendly.

Arduinos run 16 million clock cycles per second (16MHz), and the most time sensitive component is the receiver which should be continuously checking for a shot to indicate if the player has been hit. To ensure that hits were detected, the IR transmitter is left on for 300ms, which easily falls within the minimum clock cycle time. The faster speeds of the Raspberry Pi and ESP32 were therefore not needed for this system.

4.2 User Interface

The final design has 5 different systems that all transmit, record, or display information for the user. As the user must interact with these different systems, there is audio-visual confirmation when a successful interaction is made. The various parts of the embedded system are described below.

System Components:

- **Laser transmitter:** emits an IR "laser" pulse from the player's transmitter when the trigger is pressed.
- **Laser receiver:** receives a shot/IR pulse from another player.
- **LCD game points monitor:** displays the player information (name), number of successful hits, number of hits taken, and gameplay sequences.
- **Main game system device:** this is the device that uploads the program to the Arduinos before the game begins. This device must be used to enter the player name before the game begins; however if such a device is not available, the Arduinos have already been programmed, and the game can still run without player names by pressing the trigger button.
- **LED/speaker:** these are the components that send audio visual confirmation to the player. When a shot is made or a hit is taken, the LED lights up and indicates a successful shot/hit to the player. The speaker emits sound for shots and hits, and also when gameplay changes (see Table 2).

These systems are all controlled by the Arduino Uno microcontroller. Figure 5 depicts a block diagram that describes the overall embedded system and the various points where user interaction can occur.

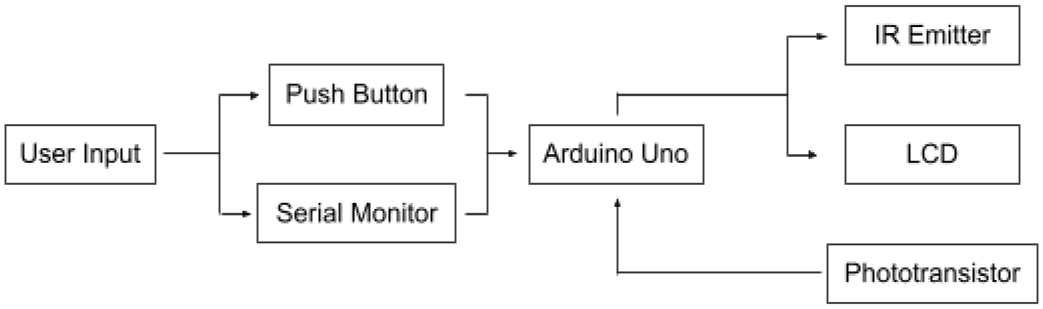

Figure 5: Embedded System Block Diagram

4.3 Embedded Firmware

Arduino Unos use a Harvard architecture memory structure, which consists of separate storage and pathways for programs and data. On an Arduino, there are 32K bytes of flash memory where programs are stored, and 2K bytes of static random access memory (SRAM) for data. Flash memory is a category of read only memory (ROM), and more specifically it is electrically erasable read only memory (EEPROM). When planning out the structure of the program for this embedded system, there were many different functions and function calls and therefore it was necessary to understand how much memory the program required, as taught in MIE438.

All function calls in the program make use of the stack automatically when passing variables to the called function and if the function itself declares new variables. Through MIE438, the team was able to understand how a high-level program uses the stack and CPU memory, and thus was equipped to optimize the code as needed if the program was too large or slow. The final program utilized 7150 bytes, or 22% of the total flash memory available and 474 bytes, or 23% of SRAM. Using a high-level programming language for this design was essential due to the many functions and inputs/outputs required, which would make using low-level code too complicated and lengthy. Since C++ is a high-level programming language, the user does not have to directly manipulate the stack. In addition, an interrupt-like sequence was used to receive IR signals while the player is making a shot, which is further described in Section 5.0. This sequence made use of the material taught during the interrupts and timers lecture in MIE438, as it interrupts the shot firing loop if the IR receiver senses a hit. The basics of the language C++ were taught in MIE438 lectures, labs, and practice problems, thus the team was prepared to use this language for the project.

The team also made use of the embedded Universal Asynchronous Receiver/Transmitter (UART) on the Arduino Uno module. At the beginning of the game, the user is required to enter their player name via the serial monitor. The serial monitor establishes a connection with the Arduino board using the USB cable and initializes UART communication at a baud rate of 9600. When the user enters their name and hits the ‘enter’ button, the UART module converts the serial data into parallel data that the microcontroller can process. Similarly, multiple aspects of the code print data onto the serial monitor to ensure systems are operating as expected. When such printing commands are called, the UART module converts the parallel data (the values that need to be printed) into serial stream data. This serial data is then passed to the serial monitor using a serial port.

State machine models were introduced in MIE438 and were recommended to be used as a design pattern to bring formal structure to complex tasks that lend themselves naturally to pre-defined states of execution. These models show the various actions of a system and the conditions that initiate them. Considering the number of states and transitions in this laser tag system, the team found that it would be beneficial to create such a model. Initially, the model helped the team identify any redundancies and dead-ends in the code, and then it was revised accordingly. The final diagram helped provide an overview of all the functions and conditions in the system, which greatly simplified the process of writing an implicit state machine program. Figure 6 shows the final state machine diagram used for this embedded system. There were a total of 3 inputs, 4 outputs, 6 transitions and 7 states. The code used for this system can be found in Appendix B.

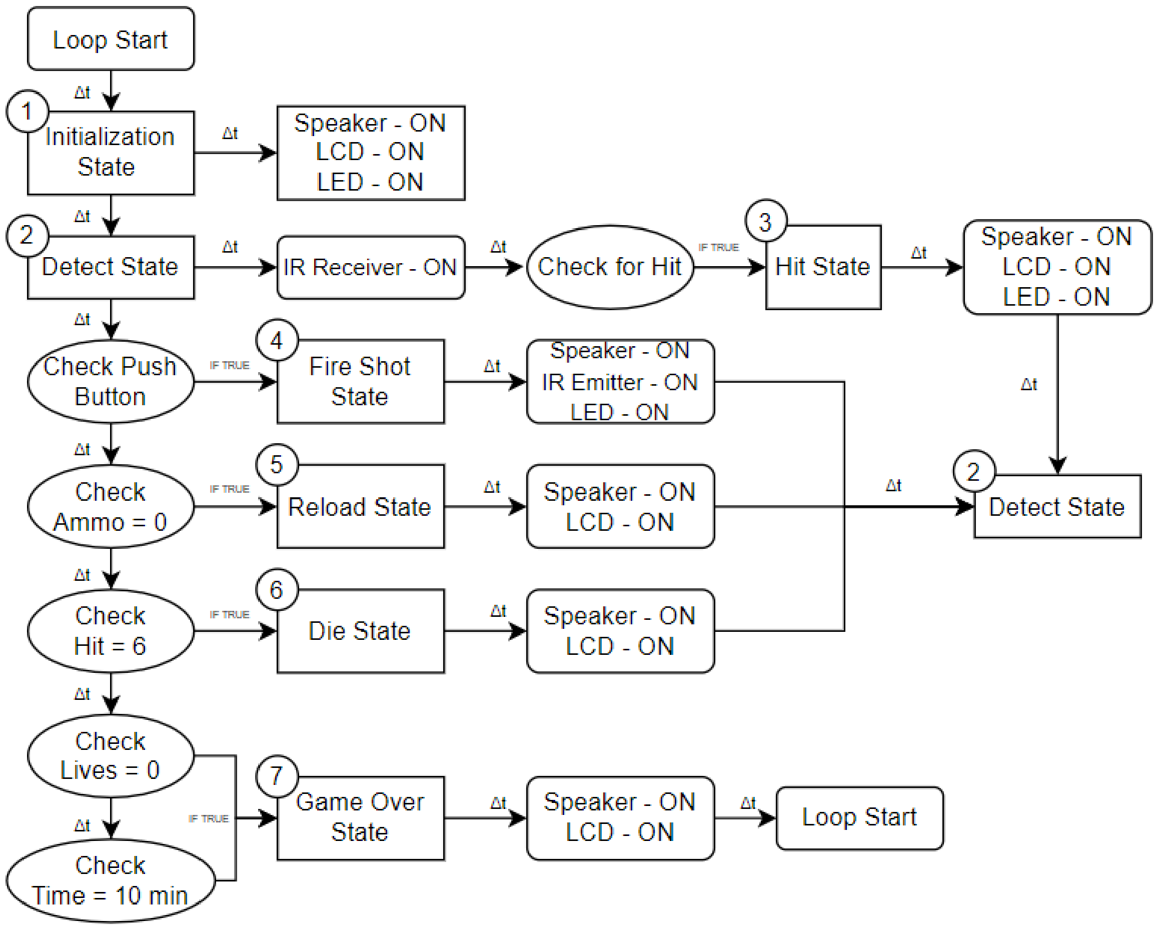

Figure 6: State Machine Diagram

4.4 Design Operation

This design offers a simple and straightforward user interface, as it should be easy for all ages to understand and use. A short summary of all the necessary functions, operation instructions, and their methods of confirmation in this laser tag system can be found in Table 2 below.

Table 2: Design Functions, and Operation Instructions.

Function	Operation Instruction	Action Confirmation
Enter player name	Use the main game system device to enter your name into the Arduino serial monitor.	When the MCU has been booted up, the LED will flash. When the game starts, the serial monitor and LCD screen prompt the player to enter their name. The hiding sequence begins once this action is complete.
Hide	Automatic.	The LCD will display the message “5s to hide!” and sound will play from the speaker.
Make a shot	Press the trigger button while aiming the IR sensor at another player’s receiver to make a shot.	The LED will flash and the speaker will play a shooting noise. The number of shots will be updated on the LCD.
Taking a hit	Automatic.	The LED will flash and the speaker will play a sound. The number of hits will be updated on the LCD.
Faint and revival	Automatic.	After 6 hits, the LCD will display “Life lost” and the total number of lives remaining. The speaker will play a sound for 10s while the player is reviving. The LCD will display “Reviving” during this time. Once revived, the LCD will display “Revived” and gameplay can continue.
Reloading	Automatic.	The LCD will display “AMMO RESETTING” and the speaker will play a sound for 5s. Gameplay can begin after the sound stops.
Gameover	Automatic.	Once a player has lost 3 lives, the game is over and the speaker will emit a sound. “GAMEOVER” will be displayed on the monitor along with the total number of shots and hits a player has.

4.5 Bill of Materials

The full list materials can be found in Appendix C. Overall, the total cost of the project was $17.68, which is below the course budget of $60.

5.0 Results

To evaluate the design, each feature of the design was first tested individually. The different features are as follows: play music, fire a shot, detect a hit, retrieve the players' name, reset the players' ammo, detect when a player has died, and detect when the game is over. To test these features, the code was run using the hardware multiple times. Additionally, the team simulated multiple different scenarios. For example, the shot firing done by the IR emitter and shot receiving done by the receiver were tested both in dark and bright conditions to determine the effect of light on the system. Furthermore, the range of the sensors was tested by placing them at different distances and angles. There were three main findings from the testing.

Firstly, it was found that the sensors function much better in the dark. This is because light sources such as lamps, device screens, and the sun all emit some degree of IR light which affects the receiver. To account for this, the team added a threshold value to the code. This value prevents the receivers from registering random hits caused by external light sources. The threshold value was varied until the system behaved desirably in both bright and dark environments.

The second finding was that the range of the emitter and receiver was smaller than what was stated in its datasheet. The sensors should be operable at a distance of roughly 0.9m, which is true if the emitter is set to be constantly high and placed in line with the receiver. However, in the case of a laser tag game, the player is constantly moving, and as such the emitter is placed at various angles. Additionally, the emitter is only briefly turned on when the player fires. Thus, the operable distance is significantly less. To solve this, the team set the emitter to be high for a set amount of time when the player fires, allowing the receiver to detect the signal before the player moves away. The team found that an emitter high time of 300ms allowed for hits to be detected and allowed for an operable distance of 0.05-0.1m. For longer range gameplay, the system would require better and faster IR emitters/receivers, which were out-of-budget for this project.

The final finding dealt with the Arduino's embedded environment and the coding style of the firing function. The team found that a player could not be hit when they were firing a shot. This is because the function was written in a 'blocking' style. This meant that the Arduino was incapable of receiving signals while executing a function within the main loop. To solve this issue, a break condition was introduced into the shooting while loop that allows for a shot to be received. This served as an interrupt sequence as taught in MIE438, thus allowing a player to be hit while they were firing a shot. See the 'void fireShot()' function in Appendix B below.

Once each feature was tested and debugged, the team instructed two users that were unfamiliar with the system to participate in a full game. The players were able to successfully play the game and each feature worked as expected.

6.0 Challenges and Reflection

The design performed as expected and successfully imitated a laser tag system as planned. The team made use of various features of the Arduino. These features included the Digital I/O pins, Analog I/O pins, Arduino IDE, memory, and embedded UART. The major challenges included structuring the code to simulate laser tag gameplay, debugging the code, and setting up the IR emitter and receivers.

Overall, using an Arduino Uno to simulate the laser tag game proved successful. The microcontroller had more than enough features and capabilities to satisfy the complexities of the game. The team would reselect this board or one of its counterparts such as the Arduino Mega if building the system again.

However, there are many other features of the board that the team could use in the future to better the design. For example, Arduino Unos are equipped with an Inter-Integrated Circuit (I2C) communication interface. This can be used to have many Arduino boards communicate with one another. Therefore, the team could use 12C in the future to incorporate more players and have each player's board send signals to others to improve game play. Moreover, it would be useful to have one 'master' board act as a referee that can send signals and instructions to other 'slave' boards via the I2C communication protocol.

Another feature that would benefit the design is the interrupt capability of pins 2 and 3 on the Arduino Uno. Currently, the code does not allow for multiple events to happen simultaneously. For example, when the player is reloading, the code does not allow for a hit to be detected. Interrupts could be added in to solve this problem. Specifically, interrupts could be used to improve the functionality of each of the program states by allowing each state to be interrupted if the player gets hit. Incorporating these features, using better IR emitters and receivers that can decode signals, and making an enclosure for the laser tag system would improve the design significantly.

7.0 References

[1] Sessional Dates | Faculty of Applied Science and Engineering. (n.d.). https://engineering.calendar.utoronto.ca/sessional-dates

[2] Kishishita, Y., Ramirez, A. J., Das, S., Thakur, C., Yanase, Y., & Kurita, Y. (2018). Muscleblazer. Proceedings of the First Superhuman Sports Design Challenge on First International Symposium on Amplifying Capabilities and Competing in Mixed Realities - SHS '18. https://doi.org/10.1145/3210299.3210302

[3] IEEE Xplore Full-Text PDF: (n.d.). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8039167

[4] Industries, I. (2017, November 6). *Arduino Laser Tag - Duino Tag*. Instructables. Retrieved February 4, 2023, from https://www.instructables.com/Duino-Tagger/

8.0 Appendices

Appendix A: Weekly Plan.

Week #	Date	Plan
1	10/02/2023	Order components
2	17/02/2023	Finalize design ideas for each subsection
3	24/02/2023	Wire emitter and receiver systems (subsections 1, 2, 3, & 5)
4	03/03/2023	Write code to control the emitter and receiver (subsections 1, 2, 3 & 5)
5	10/03/2023	Test and debug
6	17/03/2023	Develop LED, audio, and display systems (subsections 4, 5, 6)
7	24/03/2023	Develop an enclosure for the components
8	31/03/2023	Test and debug
9	07/04/2023	Begin writing final report
10	12/04/2023	Final report and demo due

Appendix B: Embedded System Code.

```
// MIE438 Group 12 Laser Tag Program

// Initialize LCD display
#include <LiquidCrystal.h>

//initialize LCD
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//initialize pins
int sensorPin = A0;  // IR receiver on pin 7
int senderPin = 6;   // IR emitter on pin 6
int triggerPin = 8;  // push button to fire shot on pin 8
int speakerPin = 9;  // speaker on pin 9
int blinkPin = 10;   // blinking LED used for visual effects on pin 10

//initialize boolean variables
bool hit = false;    //used to determine if player got hit
bool begin = true;   //used to initialize game play
bool fired = false;  //used to determine if player fired a shot
bool trigger;        //used to determine if player hit push button

//initialize string variables
String name;  //stores name of player

//initialize integer variables
int name_len;         //used to keep track of length of player's name
int maxShots = 6;     // maximum number of shots a player can fire;
int maxHits = 6;      // maximum number of hits a player can get;
int myShots = 0;      // keeps track of current shots fired;
int myHits = 0;       // keeps track of current hits received;
int totalHits = 0;    // keeps track of total hits during game play
int totalShots = 0;   // keeps track of total shots fired during game play
int numLives = 3;     // player has a maximum of 3 lives



void setup() {
  //regular setup
  lcd.begin(16, 2);
  pinMode(blinkPin, OUTPUT);
  pinMode(speakerPin, OUTPUT);
  pinMode(senderPin, OUTPUT);
  pinMode(triggerPin, INPUT);
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);

  //play sound at beginning
  startUpMode();
}


void loop() {
  while (millis() < 600000) {
    //check to see if the player has fired a shot
    senseFire();

    //check to see if the player has been hit
    detectHit();

    //get player name
    if (begin == true) {
      getPlayerName();
      //give the player time to hide
      hiding();
      begin = false;
    }

    //display current shots and hits
    displayShotsHits();

    //reset ammo if max shots reached
    if (myShots == maxShots) {
      ammoReset();
    }

    //decrement lives if max hits reached
    if (myHits == maxHits) {
      playerDied();
    }
  }
  //if time runs out
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("Times Up!");
  delay(3000);
  gameOver();
}

//FUNCTIONS:

//function that plays music on the speaker
void playTone(int tone, int duration) {
  for (long i = 0; i < duration * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(tone);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(tone);
  }
}

//function that determines if a shot has been fired
void senseFire() {
  //check the push button state
  trigger = digitalRead(triggerPin);

  if (trigger == HIGH && fired == false) {
    fired = true;
    myShots++;
    if (myHits <= maxHits) {
      Serial.print("Firing Shot : ");
      Serial.println(myShots);
      fireShot();
    }
  } else if (trigger == LOW) {
    if (fired == true) {
    }
    // reset the fired variable
    fired = false;
  }
}

//function that sets the senderPin high to fire a shot
void fireShot() {
  //set blinker pin high to indicate a shot is being fired
  digitalWrite(blinkPin, HIGH);

  // play firing sound
  playTone(100, 5);
  playTone(200, 5);
  playTone(300, 5);
  playTone(400, 5);
  playTone(500, 5);
  playTone(600, 5);

  //set senderPin high, this fires a beam of IR
  unsigned long senderStartTime = millis();
  while (millis() - senderStartTime < 500) {
    digitalWrite(senderPin, HIGH);
    if (detectHit()) {
      break;
    }
  }

  //reset the blinking pin to low
  digitalWrite(blinkPin, LOW);
  //reset sender pin to low
  digitalWrite(senderPin, LOW);
}



bool detectHit() {
  //sensorPin returns a value with 0-1024 based on the IR signal it receives
  if (analogRead(sensorPin) > 100) {
    hit = true;
    myHits = myHits + 1;
    Serial.println("IR receiver triggered - Hit!");
    clearRow(2);
    lcd.setCursor(0, 1);
    lcd.print("HIT!");
    playTone(900, 150);
    playTone(500, 150);
    digitalWrite(blinkPin, HIGH);
    delay(500);
    digitalWrite(blinkPin, LOW);
    clearRow(2);
  } else {
    hit = false;
  }
  return hit;
}

//function to clear the LCD
void clearRow(byte rowToClear) {
  lcd.setCursor(0, rowToClear);
  lcd.print("                ");
}

//function to play sounds and light up LED at the beginning
void startUpMode() {
  for (int i = 1; i < 4; i++) {
    digitalWrite(blinkPin, HIGH);
    playTone(900 * i, 200);
    digitalWrite(blinkPin, LOW);
    delay(200);
  }
}

//function to get the player's name using the serial monitor
//player's name is then displayed on the LCD
void getPlayerName() {
  lcd.clear();
  lcd.setCursor(0, 0);

  Serial.println("Please enter your player name.");
  lcd.print("Enter name");

  while (Serial.available() == 0) {};

  name = Serial.readString();
  name_len = name.length();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(name);
  lcd.setCursor(name_len - 1, 0);
  lcd.print(" ");
  lcd.setCursor(0, 1);
  lcd.print("Lives: ");
  lcd.print(numLives);
  delay(5000);
}

//function to give the player 5 seconds to hide
void hiding() {
  //give the player 5 seconds to hide
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("5s to hide!");
  unsigned long startTime = millis();    // start time of the loop
  while (millis() - startTime < 5000) {  // repeat for 5 seconds
    playTone(1000, 50);
    playTone(750, 50);
    playTone(500, 50);
    playTone(250, 50);
    playTone(500, 50);
    playTone(750, 50);
    playTone(1000, 50);
  }
}

//function to display hits and shots
void displayShotsHits() {
  //display the number of shots and hits
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("Shots:");
  lcd.print(myShots);
  lcd.print(" Hits:");
  lcd.print(myHits);
}

//function to similar an ammo reset after 6 shots
void ammoReset() {
  //store the total number of shots
  int melody = 0;
  totalShots = totalShots + myShots;

  //clear LCD screen
  clearRow(2);
  lcd.setCursor(0, 1);
  //let player know ammo is resetting
  lcd.print("AMMO RESETTING!");

  //play resetting song
  while (melody < 10) {
    playTone(900, 150);
    playTone(450, 150);
    melody = melody + 1;
  }

  //clear the LCD again
  clearRow(2);

  //reset the number of shots back to 0 (ammo reset)
  myShots = 0;
}

//function that contains instructions for when a player dies
void playerDied() {
  int melody = 0;
  numLives = numLives - 1;

  //clear the LCD
  //clearRow(1);
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("Life Lost");
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("Num Lives:");
  lcd.print(numLives);
  delay(3000);

  //if no lives are left, the game is over
  if (numLives == 0) {
    gameOver();
  }

  //else, the player's life is decremented by 1 and they are revived.
  else {
    //clearRow(1);
    clearRow(2);
    lcd.setCursor(0, 1);
    lcd.print("Reviving...");
    totalHits = totalHits + myHits;
    totalShots = totalShots + myShots;
    myShots = 0;
    myHits = 0;
    while (melody < 10) {
      playTone(900, 50);
      playTone(1800, 50);
      playTone(900, 50);
      melody = melody + 1;
    }
    clearRow(2);
    lcd.setCursor(0, 1);
    lcd.print("Revived!");
    delay(1000);
    //clearRow(1);
  }
}

//function that contains instructions for when the game is over
void gameOver() {
  totalShots = totalShots + myShots;
  totalHits = totalHits + myHits;
  myShots = 0;
  myHits = 0;

  //clearRow(1);
  clearRow(2);
  playTone(9000, 100);
  playTone(8000, 100);
  playTone(7000, 100);
  playTone(6000, 100);
  playTone(5000, 100);
  playTone(4000, 100);
  playTone(3000, 100);
  playTone(2000, 100);
  playTone(1000, 100);
  lcd.setCursor(0, 1);
  lcd.print("GAME OVER");
  delay(2000);
  clearRow(2);
  lcd.setCursor(0, 1);
  lcd.print("Shots:");
  lcd.print(totalShots);
  lcd.print(" Hits:");
  lcd.print(totalHits);
  delay(20000);
}
```

Appendix C: Bill of Materials

Part	Purpose/Source	Quantity	Unit Cost
Arduino Uno	Component control source / Provided	2	$ 0.00
LEDs	Visual indicators / Myhal spare parts	2	$ 0.00
Alphanumeric display	Player displays / Myhal Fab.	2	$ 3.13
General purpose speaker (1568-PRT-20660-ND)	Audio confirmation / DigiKey	2	$ 4.90
100-220 ohm resistors	LED, receiver, transmitter, and display pull up/down resistors / Myhal spare parts	4	$ 0.00
Wires	Electrical connections / Myhal spare parts	As required	$ 0.00
Push button	Gun triggers / Myhal spare parts	2	$ 0.00
Infrared receiver/transmitter	IR shot receiver and transmitter / Mayhal Fab.	2	$ 2.7
Breadboards	Component backboards / Mayhal Fab.	3	$ 6.55