Red Light – Green Light Game

CPE 4010: Sensors, Actuators and Integration

Anindita Deb

12/06/2024

Description of the Problem

The objective of this project is to create a dynamic and interactive "Red Light - Green Light" game environment utilizing Arduino as well as a multitude of sensors and actuators that were introduced in the labs over the course of the semester. For a solo individual completing the project, the primary mechanism involves moving the player manually by hand or with a magnet under the field, requiring the player's base to be constructed from a light metal. The game's timing dynamics are controlled by defining Green Light and Red-Light periods as 2 seconds and 3 seconds, respectively. During the Red-Light period, the servo holding the ultrasonic sensor completes a rotation towards the player and continuously measures the range to detect any motion, discarding small changes due to measurement error during consecutive sensor readings. External and timer interrupts play a crucial role in fulfilling the final project as detecting push button inputs is necessary to start the game, and managing a down-counting timer set from 60 seconds to 0 seconds. The overall goal is to successfully navigate an object, token, or figurine whether by hand or magnet across a playing field during Green Light segments from the starting point to the other side of the field within the 60 second time limit. If the player can do so then they will win; however, if their movement is detected during a Red-Light phase or the 60 second timer is reached then the player will be eliminated.

Component List

1. **Arduino Uno**
2. **Ultrasonic Range Sensor (HC-SR04)**
3. **Servo Motors (2 units)**
4. **LEDs (Red and Green)**
5. **Buzzer**
6. **16x2 LCD Display**
7. **Push Button**
8. **Cardboard/Plywood (game field)**
9. **Lightweight player object** (lego)
10. **Cardboard pieces** for servo and height for the player piece

Implementation Steps

Requirements

To reiterate the minimum requirements for the project. A single player must be able to be moved either by hand or by magnet underneath the playing field. Game play will begin after a push button is pressed which allow the game start. The game is broken into two main phases: green light and red-light phase. The phases will last for each 2 seconds and 3 seconds, respectively. During the 3 second red-light phase, a servo motor with a HC-SR04 ultrasonic motion sensor attached to the top of it will rotate 180° to face the playing field. For the remainder of the red-light phase, the ultrasonic sensor will detect for movement. If movement is recorded, then it will trigger a second servo motor which will rotate 180 degrees across the playing field and knock the player off; therefore eliminating. If no motion is detected during a red light, then the game will continue with the ultrasonic attached servo spinning back to its original position- away from the playing field and the next green light will commence. Players are allowed to freely move during the 2 second green light phase. An LED with the appropriate color will illuminate during the various phases of gameplay, and a passive buzzer will play a varying frequency tone as well. An LCD screen will also inform the player which color phase the game is in as well as how many seconds are remaining in the game. During the game, the segments of red and green light will flip flop back and forth until 60 seconds have been reached. After that, the player will be informed that time is up on the LCD screen and all LEDs will be unilluminated and the buzzer will no longer play a tone. The servo containing the elimination arm will be swept across the playing field eliminating the player unless they have reached the safety line. After the servo sweeps across the field, the game will be over and players may play again by hitting the push button that was originally used to start the game.

Hardware

To perform the requirements for this project the following hardware will be needed: 1 breadboard to allow all logical connections. This will be done by utilizing a numerous number of wires. Beyond the basics, 2 servo motors will be used with one controlling an arm either made of a dowel rod or cardboard. The second servo will house the ultrasonic sensor which will be used to monitor for motion detection during the red-light period. Also, during the red-light period, a red LED will be needed to be illuminated to inform the player not to move. After the ultrasonic sensor returns to facing away and the green light period begins then a second, green LED will be used and lit up. With both LEDs, a 220-ohm resistor will be needed to ensure proper current flow to avoid burnout. During both phases, a single passive buzzer will be implemented with varying frequencies between the two game phases. All the information including what color light phase the game is in and how much time left will be displayed on an LCD monitor. A push button will be also be needed to begin the game. The push button will need a 10KΩ resistor to ensure the current flows properly throughout that part of the system. Finally, the entire project is coded on to the Arduino Mega 2560 microcontroller board.

Addendum to needed hardware, a potentiometer was added to the LCD display so that the proper brightness could be reach for the message to display. Without the potentiometer, the LCD screen was blank. A potentiometer was used during the lab, so the troubleshooting for this issue was not a problem.

Workload Distribution

As this is a solo project, I will be performing all the coding myself. Coding will be on the Arduino IDE. I set up the board with all the components and integrated each sensor accordingly. Lastly, I wrote up the report myself as well.

Implementation

The project will utilize two interrupts as the basis of the code. The first is a push button interrupt. To do so, a falling push button ISR will be placed in the setup function, which will change a Boolean variable from "not started" to "started." In the setup function, the necessary variables will also be initialized to begin any gameplay, as the push button must be able to start a new game even once a previous game has just ended. The setup function will include the integration of the LCD display and setting the serial baud rate to 9600 bits per second. The two servos will be attached to their appropriate pins, chosen based on where they fit best according to the playing field. Next, the pin modes for each hardware component will be set, including turning the ultrasonic trigger, red LED, green LED, and passive buzzer into various outputs. The push button and ultrasonic echo will become inputs. After everything is set up, the code will inform the player that they can begin the game by displaying the message “Press to Start.”

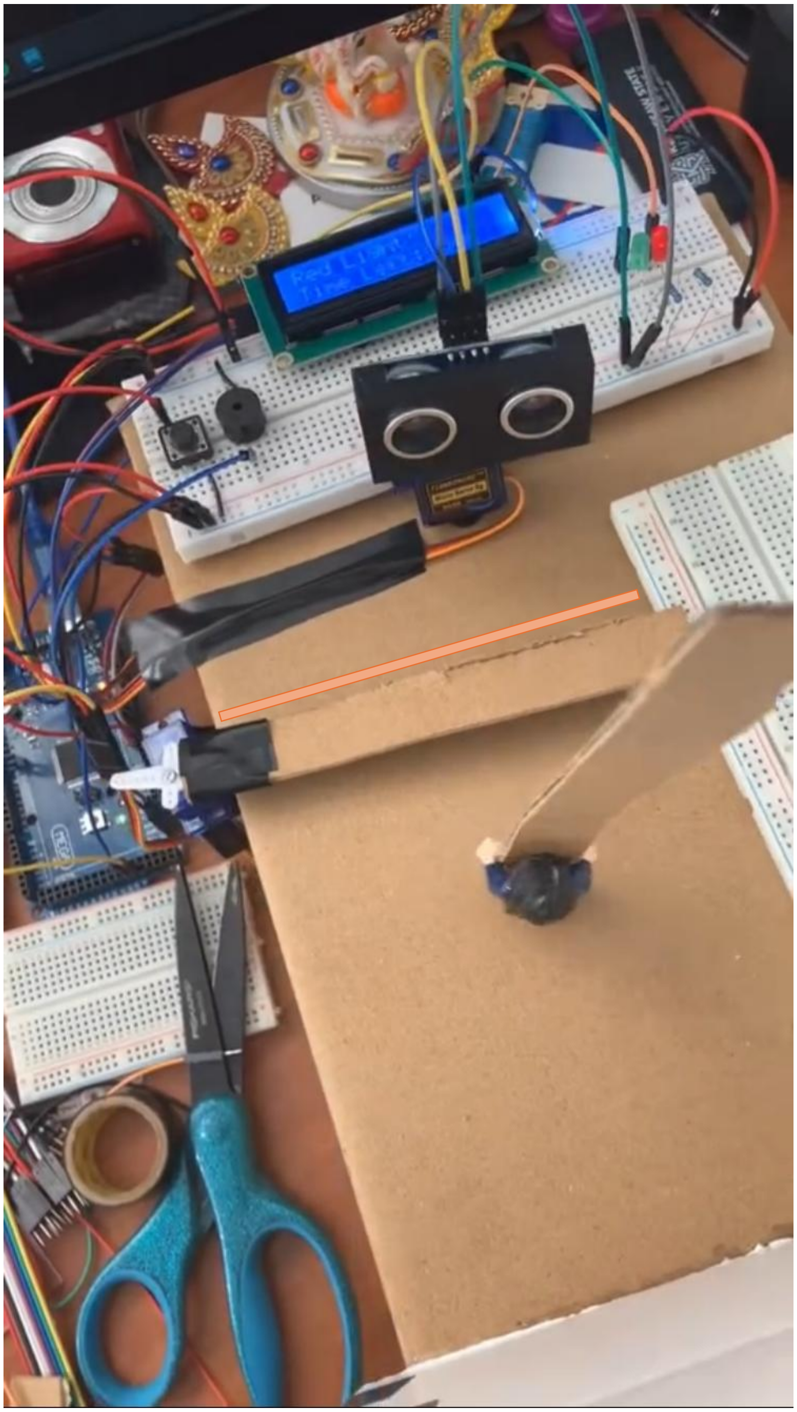
Pressing the button will cause the game to begin. A message will appear on the LCD stating “Game Started.” This will start a timer that will begin counting down from 60 to 0 seconds. During that time, the game will bounce back and forth between two different phases. These phases will be implemented in the form of separate functions. In the Green Light function, the green LED’s output will be set to high while the red LED’s output will be set to low. A buzzer will play a tone, the LCD will display the time counting down, and the appropriate delays will be implemented. This phase will last for a total of 2 seconds. Afterwards, a Boolean variable will flip from green to red, switching the game to the Red Light phase. In the Red Light function, the red LED will have its output set to high while the green LED will be set to low. The passive buzzer will play a slightly different frequency than during the green-light phase, and the LCD will continue counting down while displaying the message "Red Light!" This gameplay will last for 3 seconds before the Boolean variable switches back to green, flipping the game back to the green-light phase.

During the Red Light phase, the servo holding the ultrasonic motion sensor will rotate 180 degrees to face the game field. Motion will be constantly recorded during this phase, and if motion is detected, it will change a Boolean variable from false to true, causing the elimination servo to activate, thereby eliminating the player. If no motion is detected during the red-light phase, the variable will remain false, preventing the servo from activating. The player elimination will be handled by its own separate function, which will be called if the motion-detected variable becomes true. In this function, a message will be displayed stating "Motion Detected! – Player Eliminated." The angle of 180 degrees will be written to the elimination servo, causing the arm to move across the field and eliminate the player's token. Afterward, the elimination servo will return to its original position by setting the angle to 0 degrees. The original setup function will then be called so the player can play again.

The game will continue to flip between red light and green light phases until the timer reaches 60 seconds. The timer will be set up utilizing Timer2, which manages 1-second intervals for updating the game time (note: Timer1 was not used in the code). After the timer reaches 60 seconds, the game will automatically end with a message displaying "Time's Up!" and the elimination servo will be activated by writing the angle of 180 degrees to the servo. If the player's token has not reached beyond the safety line in that time, it will be knocked off the playing field, signifying the player has been eliminated. Afterward, the servo will have an angle of 0 degrees written to it, returning it to its original orientation. The setup function will then be called to return the game to its initial state, allowing the player to press the push button to play again.

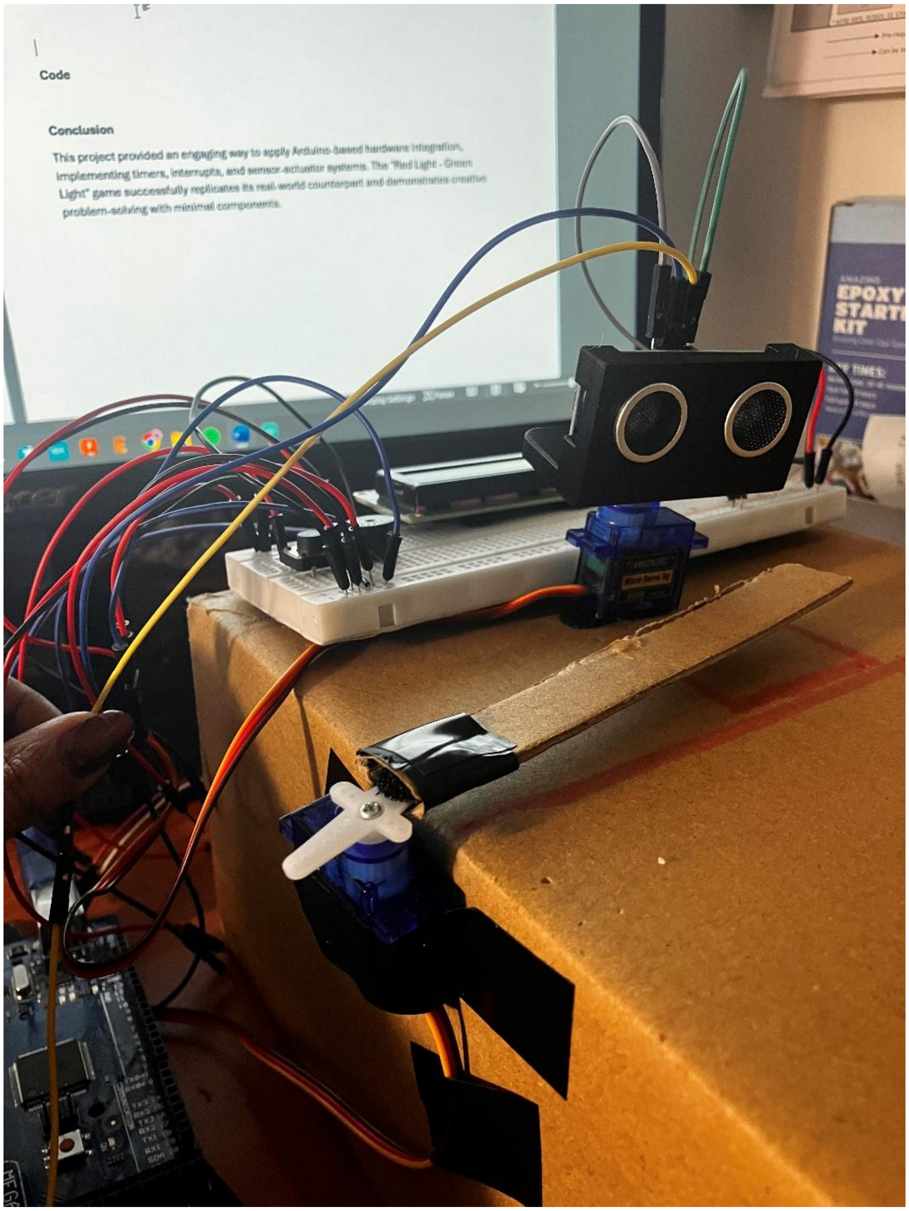
Picture of Setup and Game Field

Picture of playing field



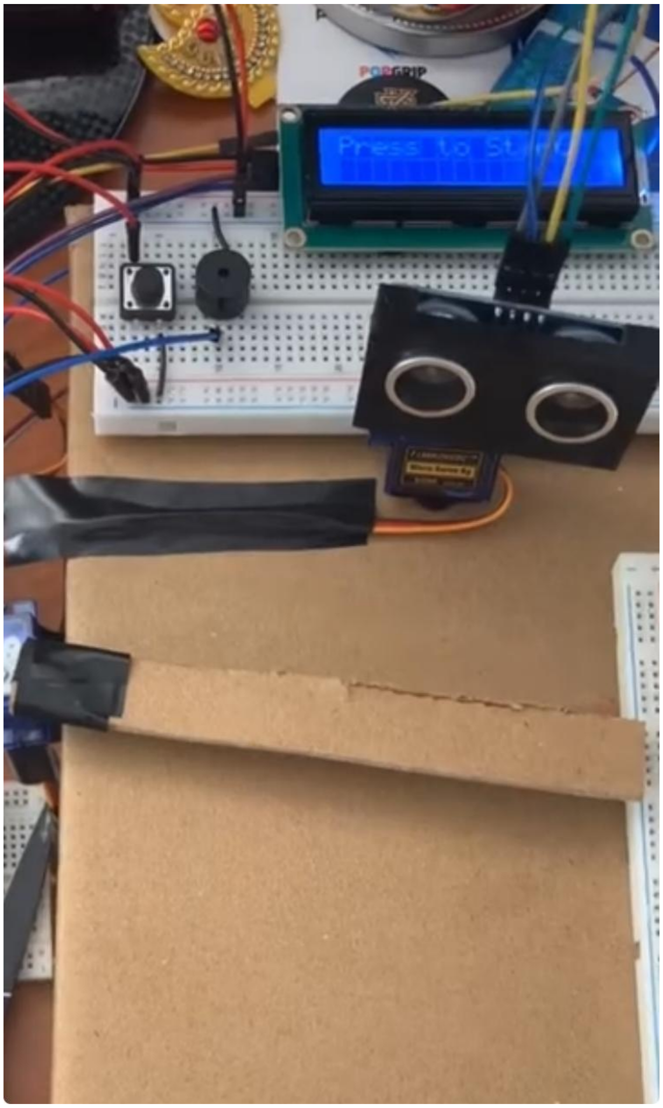
The above photo shows the final layout of the game with the lego holding an extra piece of cardboard to get more height.

Servo motors


Code
Conclusion
This project provided an engaging way to apply Arduino-based hardware integration, implementing timers, interrupts, and sensor-actuator systems. The "Red Light - Green Light" game successfully replicates its real-world counterpart and demonstrates creative problem-solving with minimal components.
EPOXY
STARTE
KIT

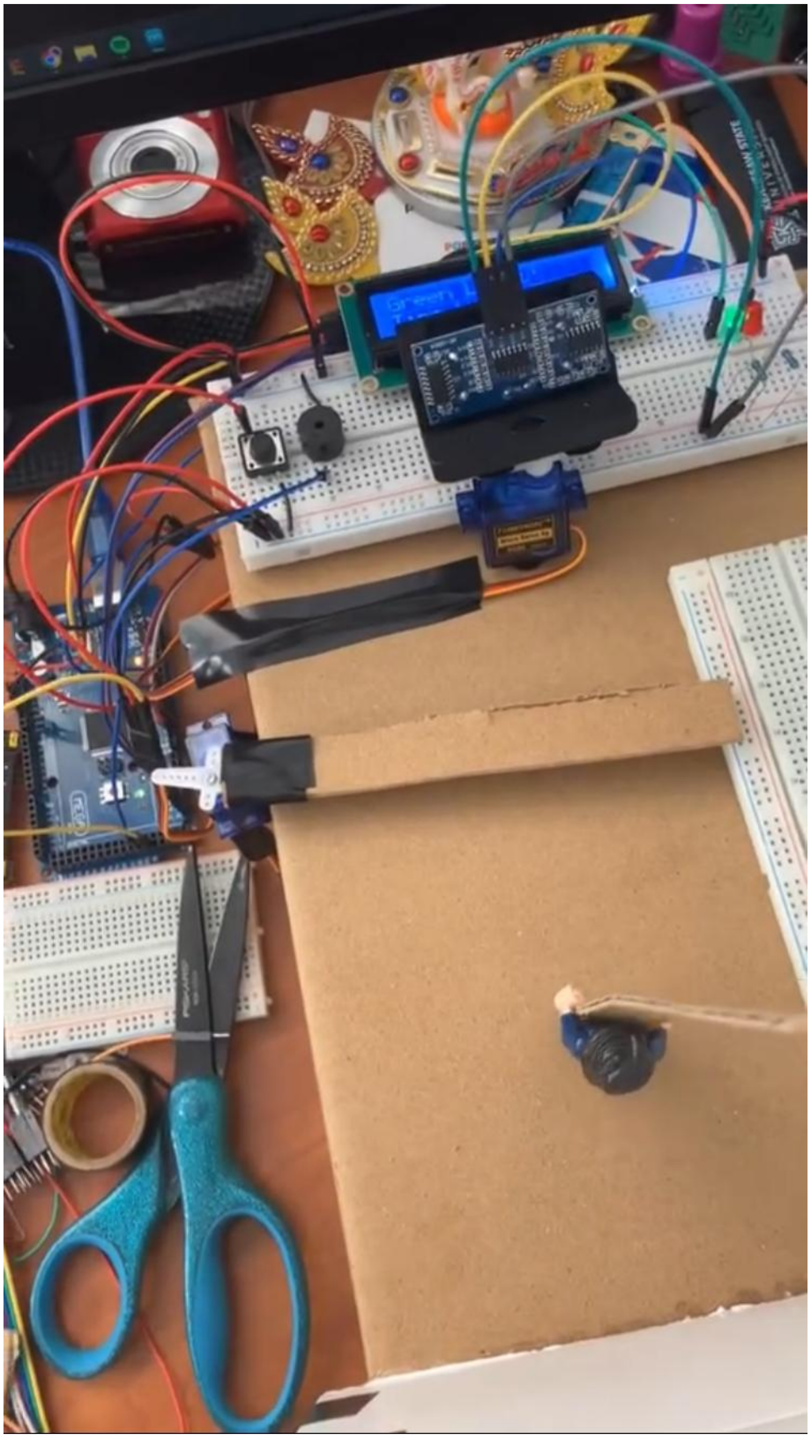
Servo motors are off to the left and center of the game. The elimination arm is just slightly in front of the finish line/target area the player must reach to win.

Press to start Mechanic display screen



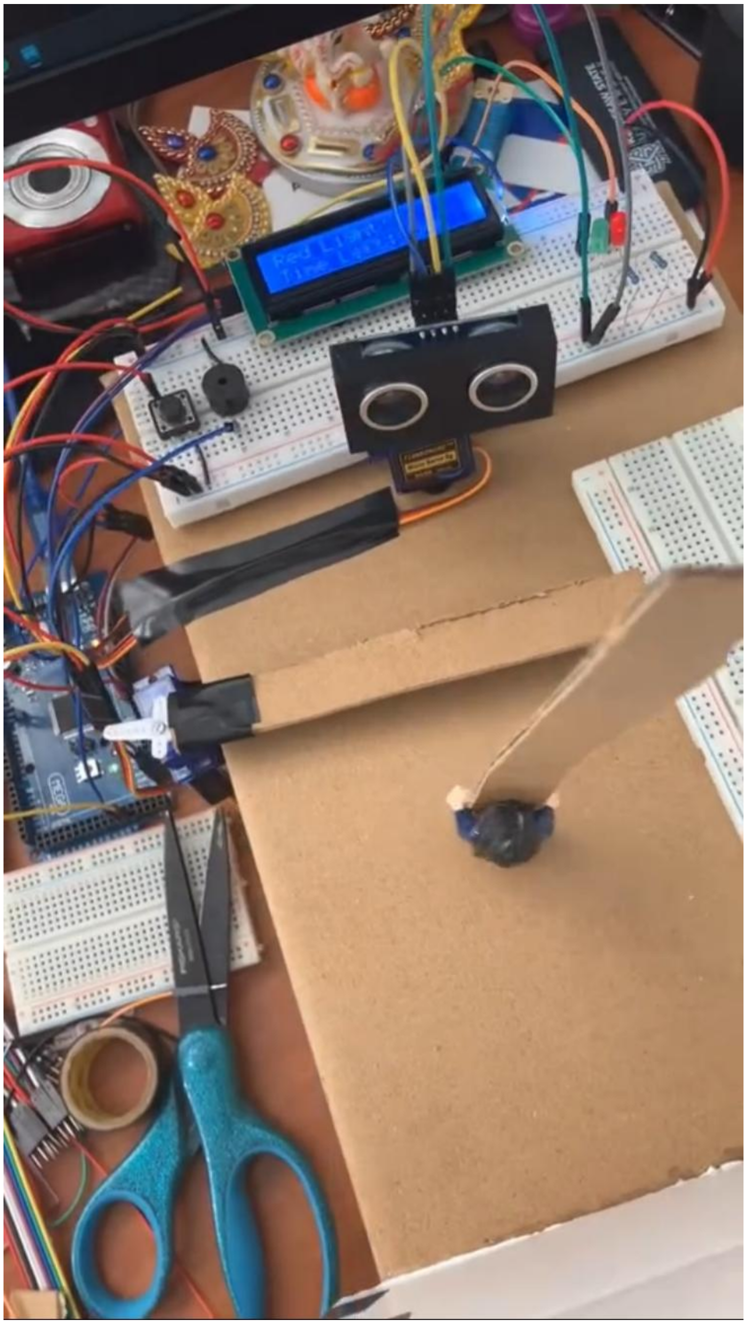
The game boots up with a prompt to press the button on the left side of the bread board to start the game.

Green Light



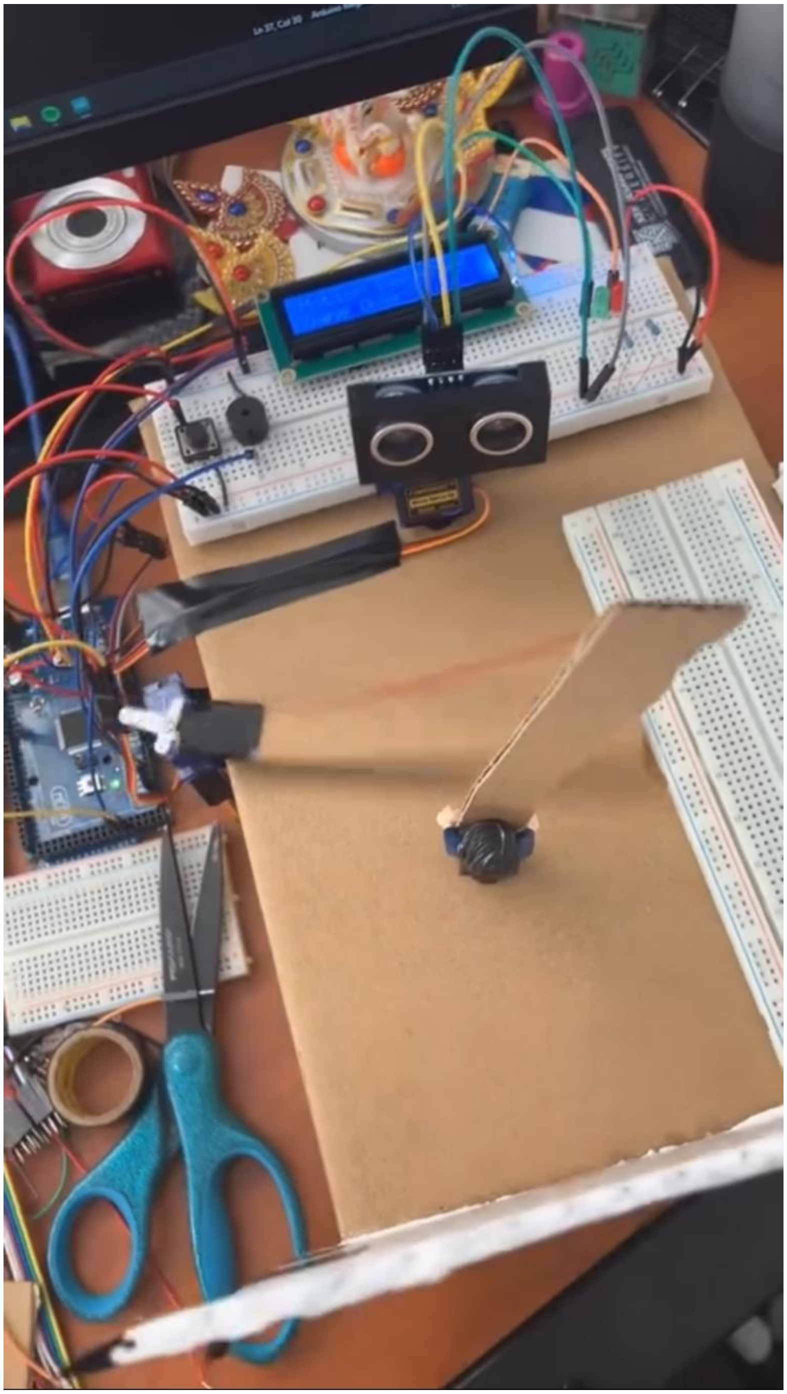
Green light mode prompts onto the LCD screen the mode and time left. The led on the right sided of the breadboard displays the green LED. The servo motor holding the ultrasonic sensor turns away from the player to allow them time to move without triggering the elimination servo arm.

Red Light (no motion detected)



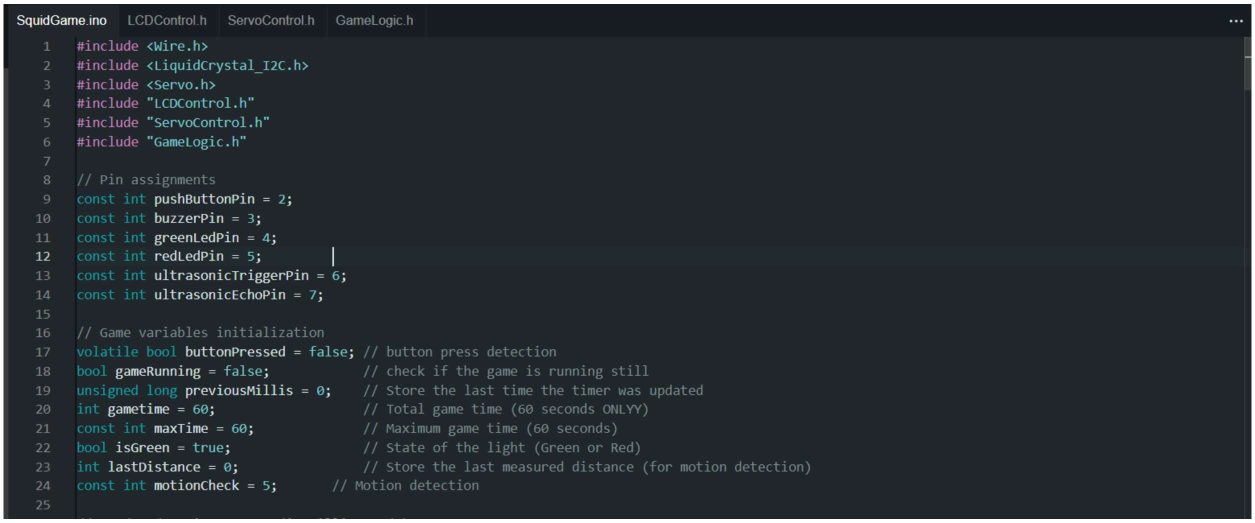
During red light mode there are two separate responses to the player. The features they share are the prompt of the Red Light Mode on the LCD screen and the red LED with turn on the right side of the breadboard.

Red Light (Motion Detected and servo arm swings)

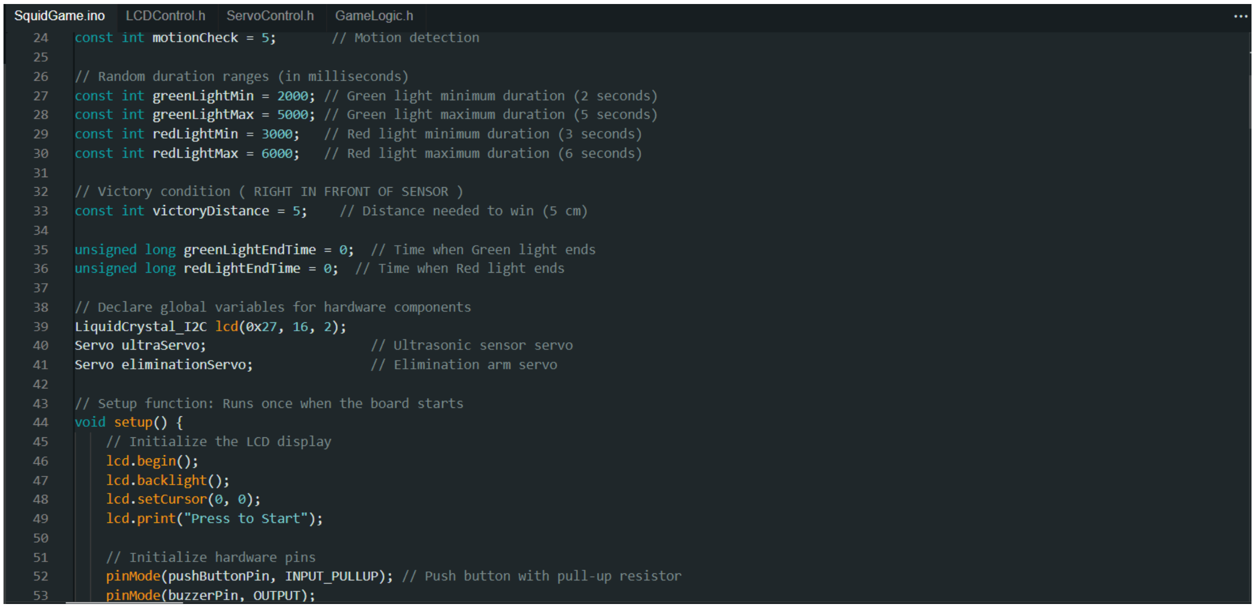


During the red light mode and the ultrasonic sensor senses movement it will display a prompt on the LCD saying “Motion Detected! – Player Eliminated” and proceed to use the elimination servo arm to eliminate the player from the board.

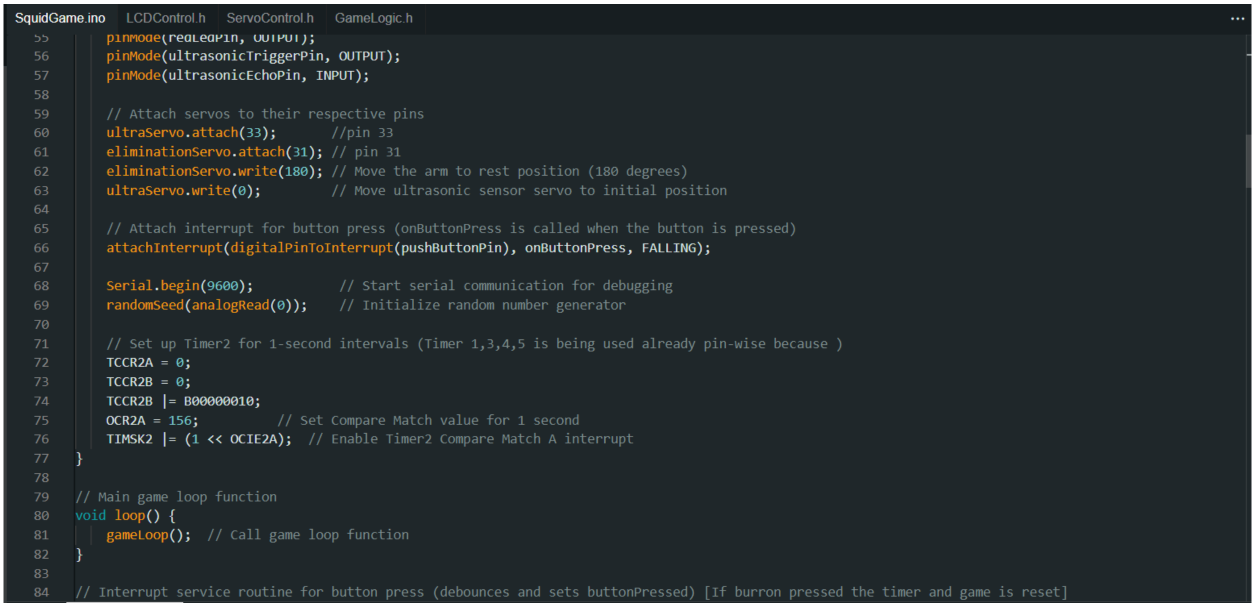
Code



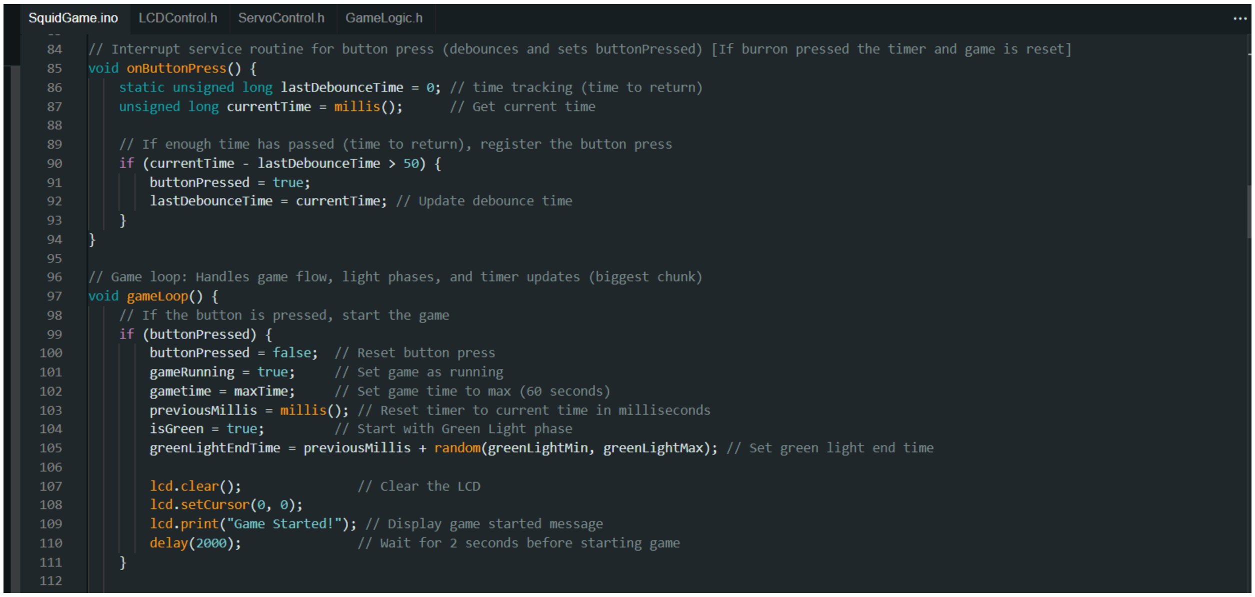
```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include "LCDControl.h"
#include "ServoControl.h"
#include "GameLogic.h"

// Pin assignments
const int pushButtonPin = 2;
const int buzzerPin = 3;
const int greenLedPin = 4;
const int redLedPin = 5;
const int ultrasonicTriggerPin = 6;
const int ultrasonicEchoPin = 7;

// Game variables initialization
volatile bool buttonPressed = false; // button press detection
bool gameRunning = false;            // check if the game is running still
unsigned long previousMillis = 0;    // Store the last time the timer was updated
int gametime = 60;                   // Total game time (60 seconds ONLYY)
const int maxTime = 60;              // Maximum game time (60 seconds)
bool isGreen = true;                 // State of the light (Green or Red)
int lastDistance = 0;                // Store the last measured distance (for motion detection)
const int motionCheck = 5;       // Motion detection

```



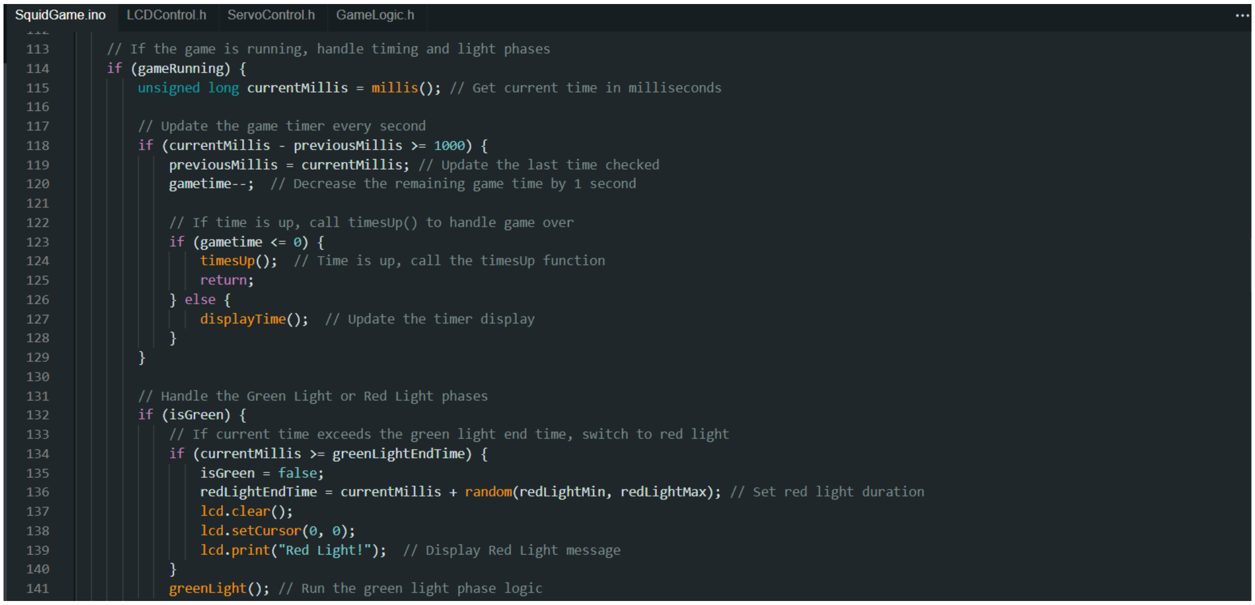
```
const int motionCheck = 5;       // Motion detection

// Random duration ranges (in milliseconds)
const int greenLightMin = 2000; // Green light minimum duration (2 seconds)
const int greenLightMax = 5000; // Green light maximum duration (5 seconds)
const int redLightMin = 3000;   // Red light minimum duration (3 seconds)
const int redLightMax = 6000;   // Red light maximum duration (6 seconds)

// Victory condition ( RIGHT IN FRFONT OF SENSOR )
const int victoryDistance = 5;    // Distance needed to win (5 cm)

unsigned long greenLightEndTime = 0;  // Time when Green light ends
unsigned long redLightEndTime = 0;  // Time when Red light ends

// Declare global variables for hardware components
LiquidCrystal_I2C lcd(0x27, 16, 2);
Servo ultraServo;                     // Ultrasonic sensor servo
Servo eliminationServo;               // Elimination arm servo

// Setup function: Runs once when the board starts
void setup() {
    // Initialize the LCD display
    lcd.begin();
    lcd.backlight();
    lcd.setCursor(0, 0);
    lcd.print("Press to Start");

    // Initialize hardware pins
    pinMode(pushButtonPin, INPUT_PULLUP); // Push button with pull-up resistor
    pinMode(buzzerPin, OUTPUT);
```



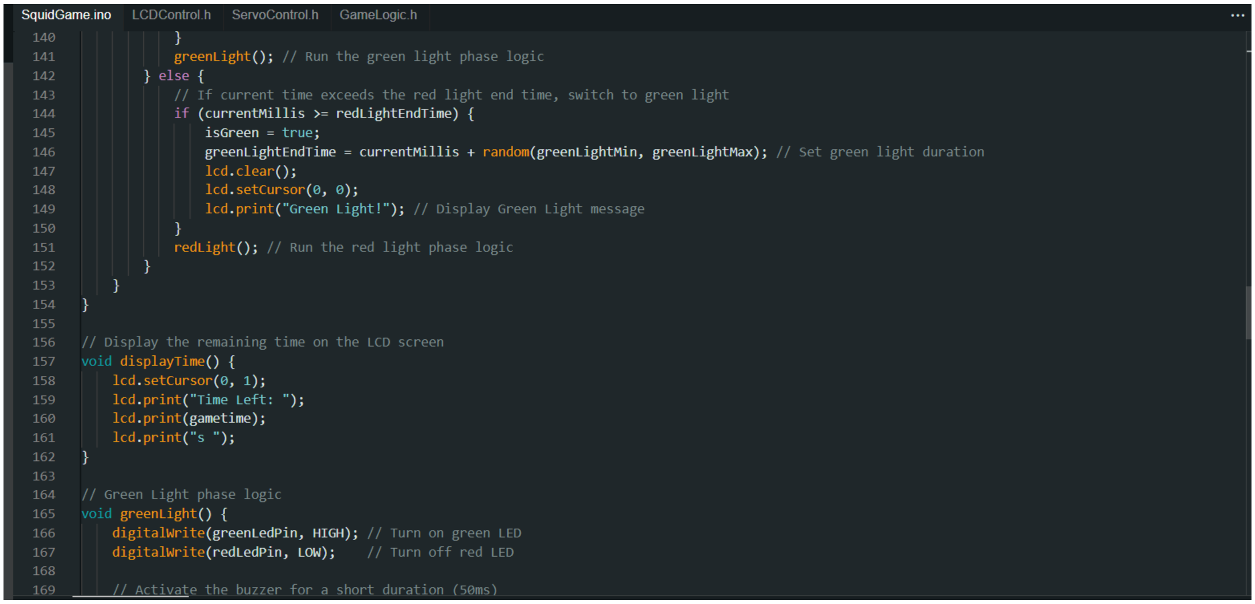
```
    pinMode(redLedPin, OUTPUT);
    pinMode(ultrasonicTriggerPin, OUTPUT);
    pinMode(ultrasonicEchoPin, INPUT);

    // Attach servos to their respective pins
    ultraServo.attach(33);       //pin 33
    eliminationServo.attach(31); // pin 31
    eliminationServo.write(180); // Move the arm to rest position (180 degrees)
    ultraServo.write(0);         // Move ultrasonic sensor servo to initial position

    // Attach interrupt for button press (onButtonPress is called when the button is pressed)
    attachInterrupt(digitalPinToInterrupt(pushButtonPin), onButtonPress, FALLING);

    Serial.begin(9600);           // Start serial communication for debugging
    randomSeed(analogRead(0));    // Initialize random number generator

    // Set up Timer2 for 1-second intervals (Timer 1,3,4,5 is being used already pin-wise because )
    TCCR2A = 0;
    TCCR2B = 0;
    TCCR2B |= B00000010;
    OCR2A = 156;          // Set Compare Match value for 1 second
    TIMSK2 |= (1 << OCIE2A);  // Enable Timer2 Compare Match A interrupt
}

// Main game loop function
void loop() {
    gameLoop();  // Call game loop function
}

// Interrupt service routine for button press (debounces and sets buttonPressed) [If burron pressed the timer and game is reset]
```



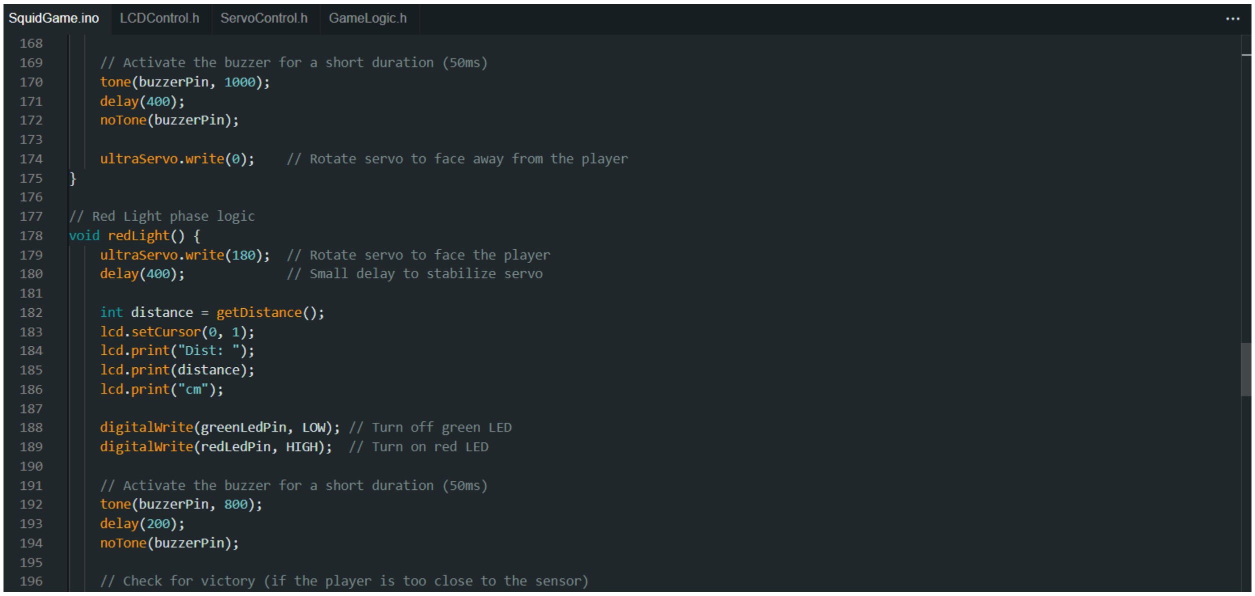
```
// Interrupt service routine for button press (debounces and sets buttonPressed) [If burron pressed the timer and game is reset]
void onButtonPress() {
    static unsigned long lastDebounceTime = 0; // time tracking (time to return)
    unsigned long currentTime = millis();      // Get current time

    // If enough time has passed (time to return), register the button press
    if (currentTime - lastDebounceTime > 50) {
        buttonPressed = true;
        lastDebounceTime = currentTime; // Update debounce time
    }
}

// Game loop: Handles game flow, light phases, and timer updates (biggest chunk)
void gameLoop() {
    // If the button is pressed, start the game
    if (buttonPressed) {
        buttonPressed = false;  // Reset button press
        gameRunning = true;     // Set game as running
        gametime = maxTime;     // Set game time to max (60 seconds)
        previousMillis = millis(); // Reset timer to current time in milliseconds
        isGreen = true;         // Start with Green Light phase
        greenLightEndTime = previousMillis + random(greenLightMin, greenLightMax); // Set green light end time

        lcd.clear();                // Clear the LCD
        lcd.setCursor(0, 0);
        lcd.print("Game Started!"); // Display game started message
        delay(2000);                // Wait for 2 seconds before starting game
    }

```



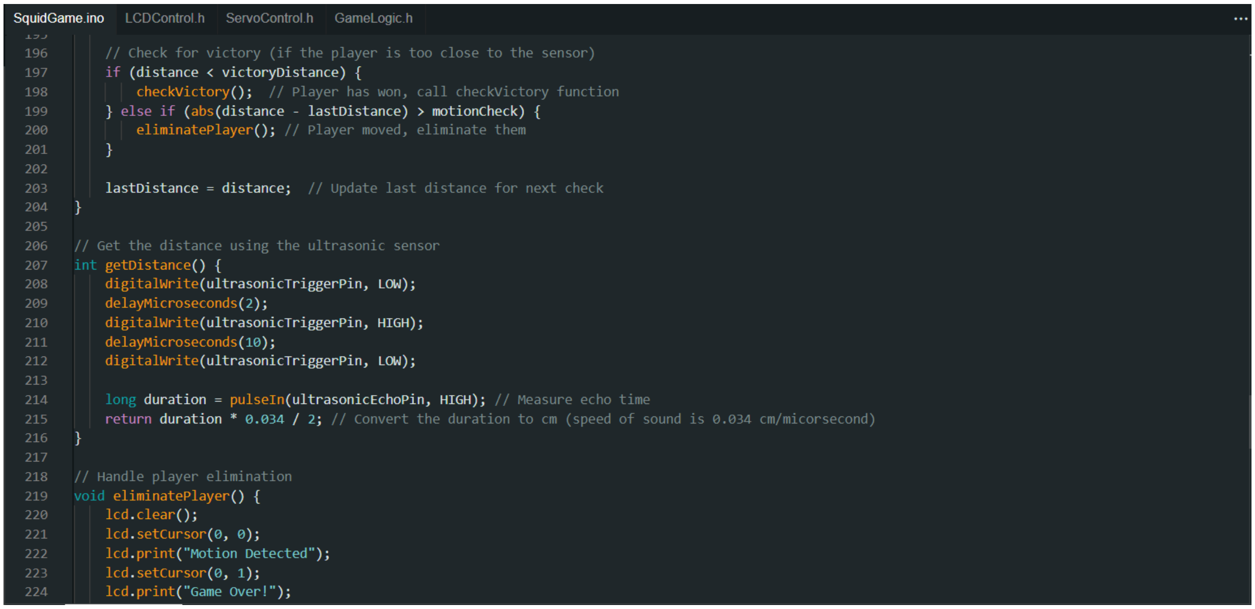
```
    // If the game is running, handle timing and light phases
    if (gameRunning) {
        unsigned long currentMillis = millis(); // Get current time in milliseconds

        // Update the game timer every second
        if (currentMillis - previousMillis >= 1000) {
            previousMillis = currentMillis; // Update the last time checked
            gametime--;  // Decrease the remaining game time by 1 second

            // If time is up, call timesUp() to handle game over
            if (gametime <= 0) {
                timesUp();  // Time is up, call the timesUp function
                return;
            } else {
                displayTime();  // Update the timer display
            }
        }

        // Handle the Green Light or Red Light phases
        if (isGreen) {
            // If current time exceeds the green light end time, switch to red light
            if (currentMillis >= greenLightEndTime) {
                isGreen = false;
                redLightEndTime = currentMillis + random(redLightMin, redLightMax); // Set red light duration
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print("Red Light!");  // Display Red Light message
            }
            greenLight(); // Run the green light phase logic
```



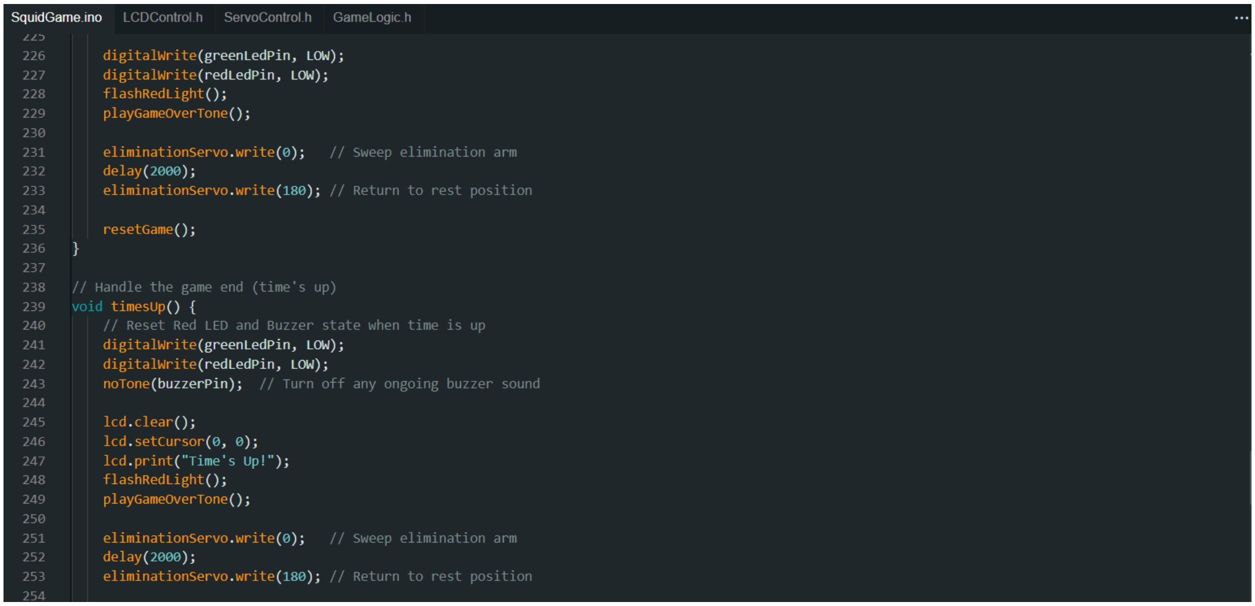
```
            }
            greenLight(); // Run the green light phase logic
        } else {
            // If current time exceeds the red light end time, switch to green light
            if (currentMillis >= redLightEndTime) {
                isGreen = true;
                greenLightEndTime = currentMillis + random(greenLightMin, greenLightMax); // Set green light duration
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print("Green Light!"); // Display Green Light message
            }
            redLight(); // Run the red light phase logic
        }
    }
}

// Display the remaining time on the LCD screen
void displayTime() {
    lcd.setCursor(0, 1);
    lcd.print("Time Left: ");
    lcd.print(gametime);
    lcd.print("s ");
}

// Green Light phase logic
void greenLight() {
    digitalWrite(greenLedPin, HIGH); // Turn on green LED
    digitalWrite(redLedPin, LOW);    // Turn off red LED

    // Activate the buzzer for a short duration (50ms)
```



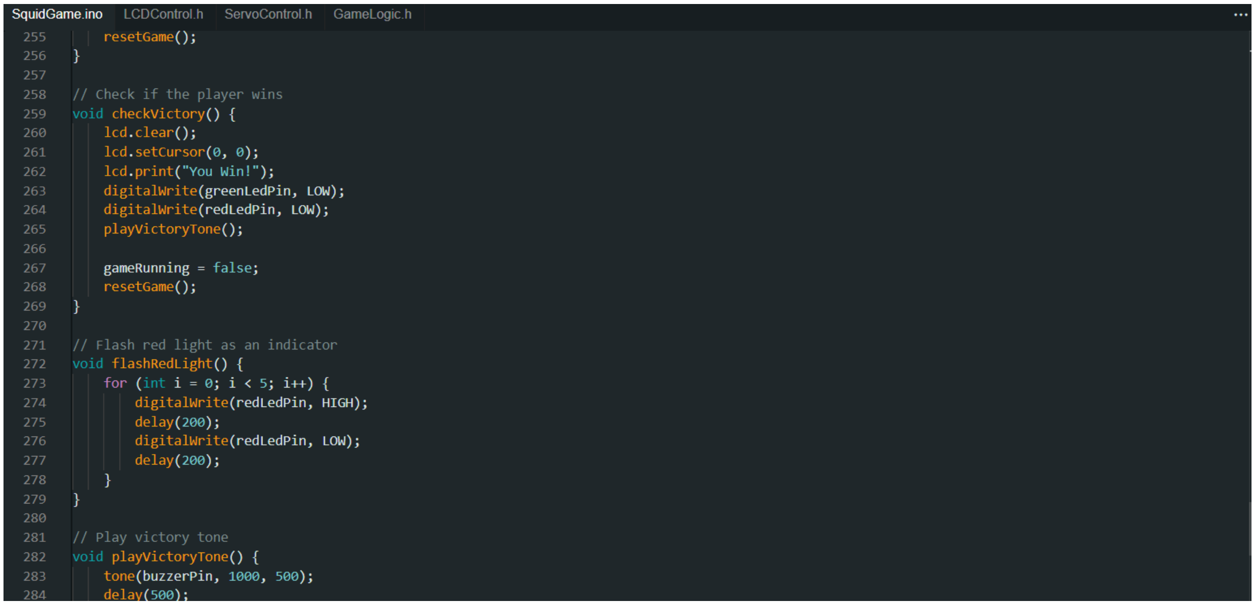
```

    // Activate the buzzer for a short duration (50ms)
    tone(buzzerPin, 1000);
    delay(400);
    noTone(buzzerPin);

    ultraServo.write(0);    // Rotate servo to face away from the player
}

// Red Light phase logic
void redLight() {
    ultraServo.write(180);  // Rotate servo to face the player
    delay(400);             // Small delay to stabilize servo

    int distance = getDistance();
    lcd.setCursor(0, 1);
    lcd.print("Dist: ");
    lcd.print(distance);
    lcd.print("cm");

    digitalWrite(greenLedPin, LOW); // Turn off green LED
    digitalWrite(redLedPin, HIGH);  // Turn on red LED

    // Activate the buzzer for a short duration (50ms)
    tone(buzzerPin, 800);
    delay(200);
    noTone(buzzerPin);

    // Check for victory (if the player is too close to the sensor)
```



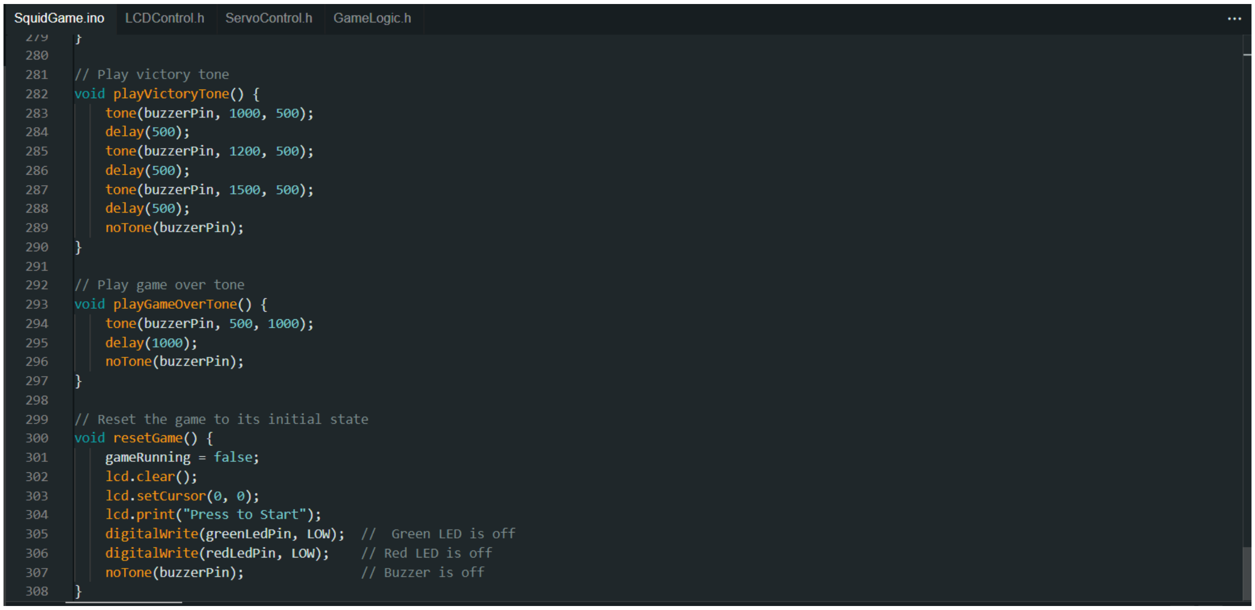
```
    // Check for victory (if the player is too close to the sensor)
    if (distance < victoryDistance) {
        checkVictory();  // Player has won, call checkVictory function
    } else if (abs(distance - lastDistance) > motionCheck) {
        eliminatePlayer(); // Player moved, eliminate them
    }

    lastDistance = distance;  // Update last distance for next check
}

// Get the distance using the ultrasonic sensor
int getDistance() {
    digitalWrite(ultrasonicTriggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(ultrasonicTriggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(ultrasonicTriggerPin, LOW);

    long duration = pulseIn(ultrasonicEchoPin, HIGH); // Measure echo time
    return duration * 0.034 / 2; // Convert the duration to cm (speed of sound is 0.034 cm/micorsecond)
}

// Handle player elimination
void eliminatePlayer() {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Motion Detected");
    lcd.setCursor(0, 1);
    lcd.print("Game Over!");
```



```
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, LOW);
    flashRedLight();
    playGameOverTone();

    eliminationServo.write(0);   // Sweep elimination arm
    delay(2000);
    eliminationServo.write(180); // Return to rest position

    resetGame();
}

// Handle the game end (time's up)
void timesUp() {
    // Reset Red LED and Buzzer state when time is up
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, LOW);
    noTone(buzzerPin);  // Turn off any ongoing buzzer sound

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Time's Up!");
    flashRedLight();
    playGameOverTone();

    eliminationServo.write(0);   // Sweep elimination arm
    delay(2000);
    eliminationServo.write(180); // Return to rest position

```



```
    resetGame();
}

// Check if the player wins
void checkVictory() {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("You Win!");
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, LOW);
    playVictoryTone();

    gameRunning = false;
    resetGame();
}

// Flash red light as an indicator
void flashRedLight() {
    for (int i = 0; i < 5; i++) {
        digitalWrite(redLedPin, HIGH);
        delay(200);
        digitalWrite(redLedPin, LOW);
        delay(200);
    }
}

// Play victory tone
void playVictoryTone() {
    tone(buzzerPin, 1000, 500);
    delay(500);
```



```
}

// Play victory tone
void playVictoryTone() {
    tone(buzzerPin, 1000, 500);
    delay(500);
    tone(buzzerPin, 1200, 500);
    delay(500);
    tone(buzzerPin, 1500, 500);
    delay(500);
    noTone(buzzerPin);
}

// Play game over tone
void playGameOverTone() {
    tone(buzzerPin, 500, 1000);
    delay(1000);
    noTone(buzzerPin);
}

// Reset the game to its initial state
void resetGame() {
    gameRunning = false;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Press to Start");
    digitalWrite(greenLedPin, LOW);  //  Green LED is off
    digitalWrite(redLedPin, LOW);    // Red LED is off
    noTone(buzzerPin);               // Buzzer is off
}
```

Conclusion

This project presented both challenges and learning opportunities as I worked to bring the "Red Light - Green Light" game to life using Arduino. One of the most significant adjustments was designing a playing field that fit the project constraints, utilizing a small cardboard box with a hollow side for moving the player magnetically underneath. This design was lightweight, simple, and effective, but required careful alignment of the player's metal base with the magnet for smooth operation. I've needed to make adjustment to the player pieces as problems arose by adding more height.

Debugging was another key aspect of the process. Ensuring that the ultrasonic sensor could accurately detect movement without false positives required fine-tuning the sensor's positioning and filtering out noise from minor environmental vibrations. The integration of the LCD display, LEDs, and buzzer to provide visual and auditory feedback added complexity but made the game more immersive. Proper synchronization of the hardware components through timers and interrupts was critical to the game's functionality and fairness.

These challenges deepened my understanding of how to combine hardware and software in practical applications. Although troubleshooting and refining each element took time, it was rewarding to see the final system operate as intended. Completing this project has enhanced my confidence in working with sensors, actuators, and microcontrollers, preparing me for more ambitious projects in the future.

Code additional:

SquidGame.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include "LCDControl.h"
#include "ServoControl.h"
#include "GameLogic.h"

// Pin assignments
const int pushButtonPin = 2;
const int buzzerPin = 3;
const int greenLedPin = 4;
const int redLedPin = 5;
const int ultrasonicTriggerPin = 6;
const int ultrasonicEchoPin = 7;

// Game variables initialization
volatile bool buttonPressed = false; // button press detection
bool gameRunning = false;          // check if the game is running still
unsigned long previousMillis = 0;   // Store the last time the timer was updated
int gametime = 60;                // Total game time (60 seconds ONLYY)
const int maxTime = 60;           // Maximum game time (60 seconds)
bool isGreen = true;              // State of the light (Green or Red)
int lastDistance = 0;             // Store the last measured distance (for motion detection)
const int motionCheck = 5;      // Motion detection

// Random duration ranges (in milliseconds)
const int greenLightMin = 2000; // Green light minimum duration (2 seconds)
const int greenLightMax = 5000; // Green light maximum duration (5 seconds)
const int redLightMin = 3000;   // Red light minimum duration (3 seconds)
const int redLightMax = 6000;   // Red light maximum duration (6 seconds)

// Victory condition ( RIGHT IN FRFONT OF SENSOR )
const int victoryDistance = 5;    // Distance needed to win (5 cm)

unsigned long greenLightEndTime = 0;  // Time when Green light ends
unsigned long redLightEndTime = 0;  // Time when Red light ends

// Declare global variables for hardware components
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
Servo ultraServo;                // Ultrasonic sensor servo
Servo eliminationServo;          // Elimination arm servo

// Setup function: Runs once when the board starts
void setup() {
  // Initialize the LCD display
  lcd.begin();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Press to Start");

  // Initialize hardware pins
  pinMode(pushButtonPin, INPUT_PULLUP); // Push button with pull-up resistor
  pinMode(buzzerPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
  pinMode(ultrasonicTriggerPin, OUTPUT);
  pinMode(ultrasonicEchoPin, INPUT);

  // Attach servos to their respective pins
  ultraServo.attach(33);     //pin 33
  eliminationServo.attach(31); // pin 31
  eliminationServo.write(180); // Move the arm to rest position (180 degrees)
  ultraServo.write(0);       // Move ultrasonic sensor servo to initial position

  // Attach interrupt for button press (onButtonPress is called when the button is pressed)
  attachInterrupt(digitalPinToInterrupt(pushButtonPin), onButtonPress, FALLING);

  Serial.begin(9600);        // Start serial communication for debugging
  randomSeed(analogRead(0));   // Initialize random number generator

  // Set up Timer2 for 1-second intervals (Timer 1,3,4,5 is being used already pin-wise because
)
  TCCR2A = 0;
  TCCR2B = 0;
  TCCR2B |= B00000010;
  OCR2A = 156;        // Set Compare Match value for 1 second
  TIMSK2 |= (1 << OCIE2A);  // Enable Timer2 Compare Match A interrupt
}
```

```
// Main game loop function
void loop() {
  gameLoop();  // Call game loop function
}

// Interrupt service routine for button press (debounces and sets buttonPressed) [If burron
pressed the timer and game is reset]
void onButtonPress() {
  static unsigned long lastDebounceTime = 0; // time tracking (time to return)
  unsigned long currentTime = millis();     // Get current time

  // If enough time has passed (time to return), register the button press
  if (currentTime - lastDebounceTime > 50) {
    buttonPressed = true;
    lastDebounceTime = currentTime; // Update debounce time
  }
}

// Game loop: Handles game flow, light phases, and timer updates (biggest chunk)
void gameLoop() {
  // If the button is pressed, start the game
  if (buttonPressed) {
    buttonPressed = false;  // Reset button press
    gameRunning = true;     // Set game as running
    gametime = maxTime;     // Set game time to max (60 seconds)
    previousMillis = millis(); // Reset timer to current time in milliseconds
    isGreen = true;        // Start with Green Light phase
    greenLightEndTime = previousMillis + random(greenLightMin, greenLightMax); // Set green
light end time

    lcd.clear();           // Clear the LCD
    lcd.setCursor(0, 0);
    lcd.print("Game Started!"); // Display game started message
    delay(2000);           // Wait for 2 seconds before starting game
  }

  // If the game is running, handle timing and light phases
  if (gameRunning) {
    unsigned long currentMillis = millis(); // Get current time in milliseconds

    // Update the game timer every second
    if (currentMillis - previousMillis >= 1000) {
      previousMillis = currentMillis; // Update the last time checked
      gametime--;  // Decrease the remaining game time by 1 second

      // If time is up, call timesUp() to handle game over
      if (gametime <= 0) {
        timesUp();  // Time is up, call the timesUp function
        return;
      } else {
        displayTime();  // Update the timer display
      }
    }

    // Handle the Green Light or Red Light phases
    if (isGreen) {
      // If current time exceeds the green light end time, switch to red light
      if (currentMillis >= greenLightEndTime) {
        isGreen = false;
        redLightEndTime = currentMillis + random(redLightMin, redLightMax); // Set red light
duration
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Red Light!");  // Display Red Light message
      }
      greenLight(); // Run the green light phase logic
    } else {
      // If current time exceeds the red light end time, switch to green light
      if (currentMillis >= redLightEndTime) {
        isGreen = true;
        greenLightEndTime = currentMillis + random(greenLightMin, greenLightMax); // Set
green light duration
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Green Light!"); // Display Green Light message
      }
      redLight(); // Run the red light phase logic
    }
  }
}

// Display the remaining time on the LCD screen
```

```
void displayTime() {
  lcd.setCursor(0, 1);
  lcd.print("Time Left: ");
  lcd.print(gametime);
  lcd.print("s ");
}

// Green Light phase logic
void greenLight() {
  digitalWrite(greenLedPin, HIGH); // Turn on green LED
  digitalWrite(redLedPin, LOW);    // Turn off red LED

  // Activate the buzzer for a short duration (50ms)
  tone(buzzerPin, 1000);
  delay(400);
  noTone(buzzerPin);

  ultraServo.write(0);    // Rotate servo to face away from the player
}

// Red Light phase logic
void redLight() {
  ultraServo.write(180);  // Rotate servo to face the player
  delay(400);           // Small delay to stabilize servo

  int distance = getDistance();
  lcd.setCursor(0, 1);
  lcd.print("Dist: ");
  lcd.print(distance);
  lcd.print("cm");

  digitalWrite(greenLedPin, LOW); // Turn off green LED
  digitalWrite(redLedPin, HIGH);  // Turn on red LED

  // Activate the buzzer for a short duration (50ms)
  tone(buzzerPin, 800);
  delay(200);
  noTone(buzzerPin);

  // Check for victory (if the player is too close to the sensor)
  if (distance < victoryDistance) {
    checkVictory();  // Player has won, call checkVictory function
  } else if (abs(distance - lastDistance) > motionCheck) {
    eliminatePlayer(); // Player moved, eliminate them
  }

  lastDistance = distance;  // Update last distance for next check
}

// Get the distance using the ultrasonic sensor
int getDistance() {
  digitalWrite(ultrasonicTriggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(ultrasonicTriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(ultrasonicTriggerPin, LOW);

  long duration = pulseIn(ultrasonicEchoPin, HIGH); // Measure echo time
  return duration * 0.034 / 2; // Convert the duration to cm (speed of sound is 0.034
cm/micorsecond)
}

// Handle player elimination
void eliminatePlayer() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Motion Detected");
  lcd.setCursor(0, 1);
  lcd.print("Game Over!");

  digitalWrite(greenLedPin, LOW);
  digitalWrite(redLedPin, LOW);
  flashRedLight();
  playGameOverTone();

  eliminationServo.write(0);   // Sweep elimination arm
  delay(2000);
  eliminationServo.write(180); // Return to rest position

  resetGame();
}
```

```
// Handle the game end (time's up)
void timesUp() {
  // Reset Red LED and Buzzer state when time is up
  digitalWrite(greenLedPin, LOW);
  digitalWrite(redLedPin, LOW);
  noTone(buzzerPin);  // Turn off any ongoing buzzer sound

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Time's Up!");
  flashRedLight();
  playGameOverTone();

  eliminationServo.write(0);   // Sweep elimination arm
  delay(2000);
  eliminationServo.write(180); // Return to rest position

  resetGame();
}

// Check if the player wins
void checkVictory() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("You Win!");
  digitalWrite(greenLedPin, LOW);
  digitalWrite(redLedPin, LOW);
  playVictoryTone();

  gameRunning = false;
  resetGame();
}

// Flash red light as an indicator
void flashRedLight() {
  for (int i = 0; i < 5; i++) {
    digitalWrite(redLedPin, HIGH);
    delay(200);
    digitalWrite(redLedPin, LOW);
    delay(200);
  }
}

// Play victory tone
void playVictoryTone() {
  tone(buzzerPin, 1000, 500);
  delay(500);
  tone(buzzerPin, 1200, 500);
  delay(500);
  tone(buzzerPin, 1500, 500);
  delay(500);
  noTone(buzzerPin);
}

// Play game over tone
void playGameOverTone() {
  tone(buzzerPin, 500, 1000);
  delay(1000);
  noTone(buzzerPin);
}

// Reset the game to its initial state
void resetGame() {
  gameRunning = false;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Press to Start");
  digitalWrite(greenLedPin, LOW);  //  Green LED is off
  digitalWrite(redLedPin, LOW);    // Red LED is off
  noTone(buzzerPin);             // Buzzer is off
}
```

LCDControl.h

```
#ifndef LCDControl_h
#define LCDControl_h

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

extern LiquidCrystal_I2C lcd;
```

```
//using the previous lab to set up the LCD screen
void setupLCD() {
  lcd.begin();
  lcd.setBacklight(1);
  lcd.setCursor(0, 0);
  lcd.print("Press to Start"); //reset or default text statement
}

void displayTime(int gametime) {
  lcd.setCursor(0, 1);
  lcd.print("Time Left: ");
  lcd.print(gametime);
  lcd.print("s ");
}

#endif
```

ServoControl.h

```
#ifndef ServoControl_h
#define ServoControl_h

#include <Servo.h>

extern Servo ultraServo;      // Declare servos globally
extern Servo eliminationServo;

void setupServo() {
  ultraServo.attach(33);
  eliminationServo.attach(31);
  eliminationServo.write(180); // Set elimination servo to rest
  ultraServo.write(0);        // Point ultrasonic sensor away initially
}

void moveServo(int position) {
  ultraServo.write(position);  // Move servo to specified position
}

#endif
```

GameLogic.h

```
#ifndef GAMELOGIC_H
#define GAMELOGIC_H

// External variable declarations
extern LiquidCrystal_I2C lcd;
extern Servo ultraServo;
extern Servo eliminationServo;

extern const int greenLedPin;
extern const int redLedPin;
extern const int buzzerPin;
extern const int ultrasonicTriggerPin;
extern const int ultrasonicEchoPin;
extern bool gameRunning;
extern int gametime;
extern bool isGreen;
extern int lastDistance;
extern const int motionTolerance;

// Function Declarations
void gameLoop();  //rules of the red light and green light usage to make full fleshed game
void setupGame(); //
void greenLight();  //green light face away from player
void redLight();    //red light face the player
int getDistance();  //how distance will be caluclated
void checkVictory();  //less than 5?? i made an actual box on the board to help show the wins
void eliminatePlayer(); //elimservo rules
void timesUp();
void resetGame();
void playVictoryTone();
void playGameOverTone();
void flashRedLight();

#endif
```