Hangman Game

EE120A Final Project

Cristian Molano
cmola002@ucr.edu
862202594

Part 1. Introduction

The Hangman project is aimed at developing an interactive and educational hangman game. This project leverages programming fundamentals and circuit design principles to create an enjoyable game experience. The hangman game is designed similar to the classic word-guessing game. Players must deduce a hidden word by guessing individual letters within a limited number of attempts. Get a guess right, get a bonus lifepoint, get a guess wrong and you will lose a life point. The game will keep going on until your life points go to zero. The further you get, the more challenging it becomes. This project not only focuses on implementing core gameplay mechanics but also enhances user interaction through visual and auditory feedback using LEDs and a buzzer.

Complexities

- Linear Progression: The linear progression aspect of the game worked properly when you beat a level, you would move onto the next level while maintaining your current lives and get either a 3 letter, 4 letter, or 5 letter word depending on the level you were in.
- RNG: The RNG did randomize the words perfectly almost every time the game was refreshed in every level of the game until the player lost or won.
- Servo Motor: The servo operated according to the pulses I gave it and only went off when it was going into a new level and only rotated once.

Extra Parts

- LCD Display: Shows parameters including lives remaining, correct letters guessed, current level, and more.
- 90dB Buzzer: Enhances the user experience by adding sound to the game. This also serves as additional feedback, playing unique sounds for incorrect/correct guesses.
- Red and Green LEDs: Provide additional user feedback for guess correctness. It also adds a cool effect, enhancing the overall experience.

By incorporating these complexities and additional components, the hangman project felt more complete and improved the overall user experience. Adding the LEDs and buzzer serve two great purposes. The first being to provide another source of user feedback during gameplay. On the other hand, the LEDs and audio make the project more enjoyable by adding visuals and sound.

Part 2. System Design

Figure 1 is a picture of the completed system circuit. The display uses up most of the pins in the Arduino. Opting for a smaller display may resolve this issue at the cost of less screen space, which will be important when displaying all the game information. The servo motor is connected to pin 11. The LEDs and buzzers are wired next to each other and have the required resistors.

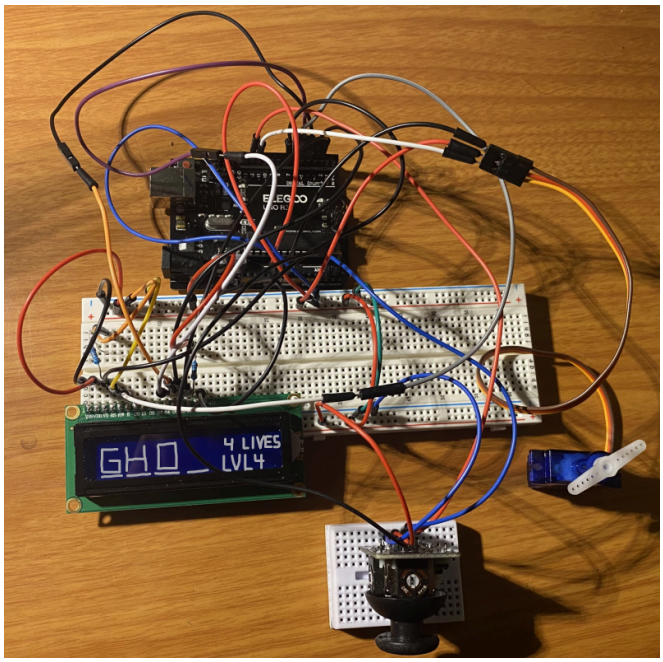

Figure 1. Photo of Setup During Development

Implementation Methods

Hangman Game Logic and User Interface

The Hangman Game Project employs an Arduino Uno microcontroller interfaced with various input and output components to create an interactive gaming experience.

- Game Logic: Showcases C++ algorithms for word selection, letter checking, and game progression.
- User Interface: Utilizes a 16x2 LCD display to show game status, including guessed letters and remaining attempts.
- Input Handling: Implements a joystick for user interaction, enabling user control for letter selection and gameplay.
- Scoring and Feedback: Provides visual feedback through the display for incorrect and correct guesses, level status, and lives remaining.

Real-time Monitoring and Data Processing

- Monitors game states and updates display constantly.
- Processes user inputs and letter being guessed, then compares with the word

Wiring Diagrams and Flowcharts

Figure 2 shows the wiring diagram for the project and its components. Important to note here is that the LCD screen uses most of the pins in this project. This display can be replaced by a more pin efficient display such as an OLED display. The servo motor is directly connected to pin 11. The joystick serves as the main user control method and is using analog pins A3, A4 and digital pin 12.

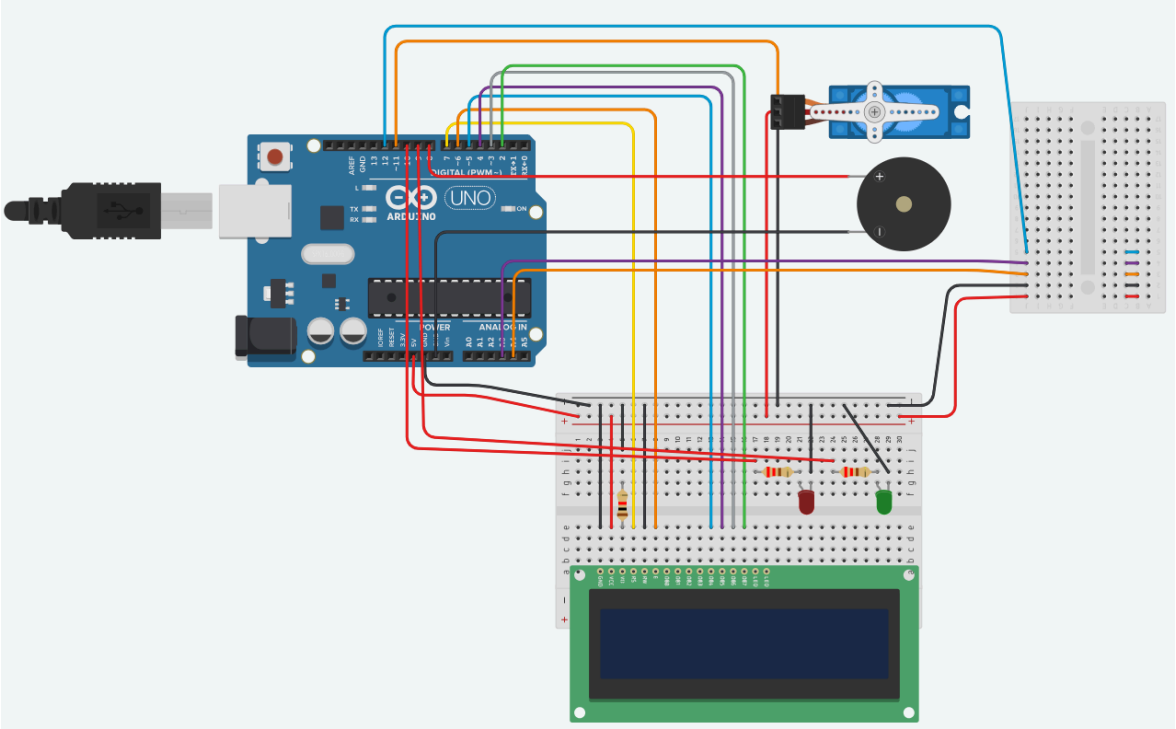

Figure 2. Wiring Diagram

Task Diagram of the System

1. Initialization:
    - Set up level parameters, motor values, and state machines
    - Initialize variables to specified values
2. Game Logic:
    - Continuously read user guesses from the joystick and check if correct. Update current level and current state according to guesses and remaining live variables.

- System will flow through the states until either lives are depleted or the user beats all levels.
- Once a level is beat, the servo values will also update accordingly.

3. Servo Motor:
   - When the user advances to the next level, the servo motor will provide feedback and activate once.
   - The servo motor has states of its own, constantly checking the current level variable and comparing with a specified value in order to move to the next level.

Figure 3 shows the system task diagram in full detail. First off is the initialization of all variables and states to the desired initial conditions. This includes the current level, servo values, guesses, and display values. The program will always check the current level and go through the level changing procedure before going on to the game logic to play the game according to the level it identified. The program will run through the game logic and constantly read user input to check if the user guessed correctly. Depending on the result, the win/loss evaluation will update the amount of lives accordingly. If the user guessed correctly, the display will be updated and provide feedback. The user will then advance to the next level and update the servo values to activate it. The servo motor provides additional feedback to the user and a cool effect to enhance the user experience. This will then repeat until either the user depletes all of their lives or beats all three levels.

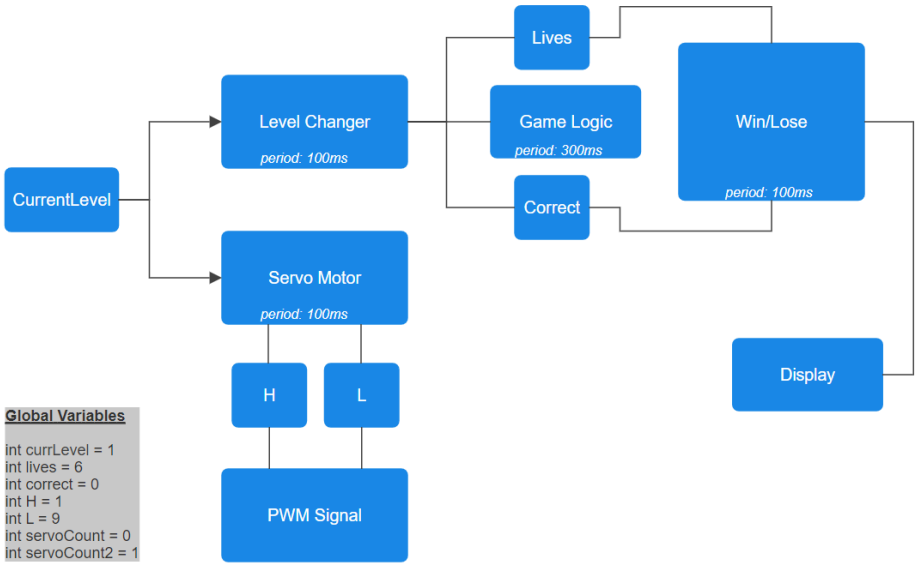

Figure 3. System Task Diagram

Figure 4 shows the synchronous state machine flowchart for the level progression in the game. The initial state will assign all initial conditions, then check the current level to assign a difficulty to the game. Starting from level 1, the user must get three correct guesses so they can move to

level 2. Seven correct guesses will take them to level 3. Finally, 12 correct guesses will take them to their results showing they have beaten all challenges. When in any of the level states, if the user depletes all of their six lives, the display will show a game over screen where the game can then be restarted. Overall, this implementation of state machines allows for a seamless transition between the progressively challenging levels.

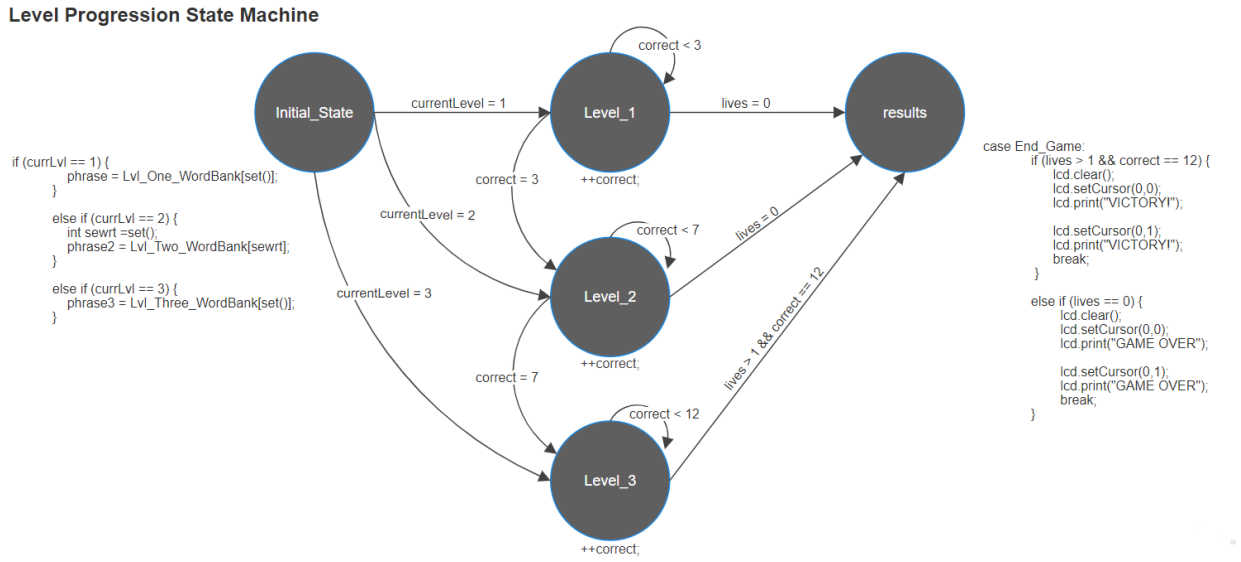

Figure 4. Sync SM Flowchart

Explanation

The Hangman project operates with the Arduino Uno as the central processing unit, managing game logic, user interaction, and feedback mechanisms. Upon initialization, the system sets up the necessary components and begins monitoring user inputs and game state. Based on the user inputs, the user makes a choice of letters to guess. The program will check the chosen letter with the string of letters defined. If wrong, the user will lose a “life” and continue to guess until all 6 lives are depleted. If the user guesses correctly, they will continue to guess and move on to the next level.

Automatic Game Operation:

- **Word Selection:** Randomly selects a word from a predefined list for each game session.
- **Guess Processing:** Accepts user inputs through push buttons to guess letters.
- **Game Progression:** State machines allow a seamless implementation of challenging levels that the user can progress through if they guess the words correctly.
- **Feedback Mechanisms:** Updates display and activates the servo motor based on user input.

Real-time Monitoring:

- **Game monitoring:** Updates the LCD display to show guessed letters, remaining attempts, and current game status.
- **Performance Optimization:** Ensures efficient use of system resources to maintain smooth gameplay, responsiveness and variable data.

Conclusion

The Hangman project demonstrates the integration of microcontroller technology with interactive gaming concepts, providing a user-friendly and engaging experience. Through meticulous design and implementation of game logic, user interface, and feedback mechanisms, this project shows how a blend of hardware and software can provide for great interactive projects like this. In the end, this project achieves its objective of showcasing Arduino Uno's versatility in educational and recreational applications.

Part 3. Implementation & Circuit Schematics

Schematics

The schematic shown in figure 5 details all wiring and component implementations for this project. The LCD display, servo motor and joystick are directly connected to the Arduino Uno. The servo is connected to pin 11, while the display uses up pins D2-D7. The joystick module uses D12 as well as analog A3 and A4. The necessary resistors are also shown.

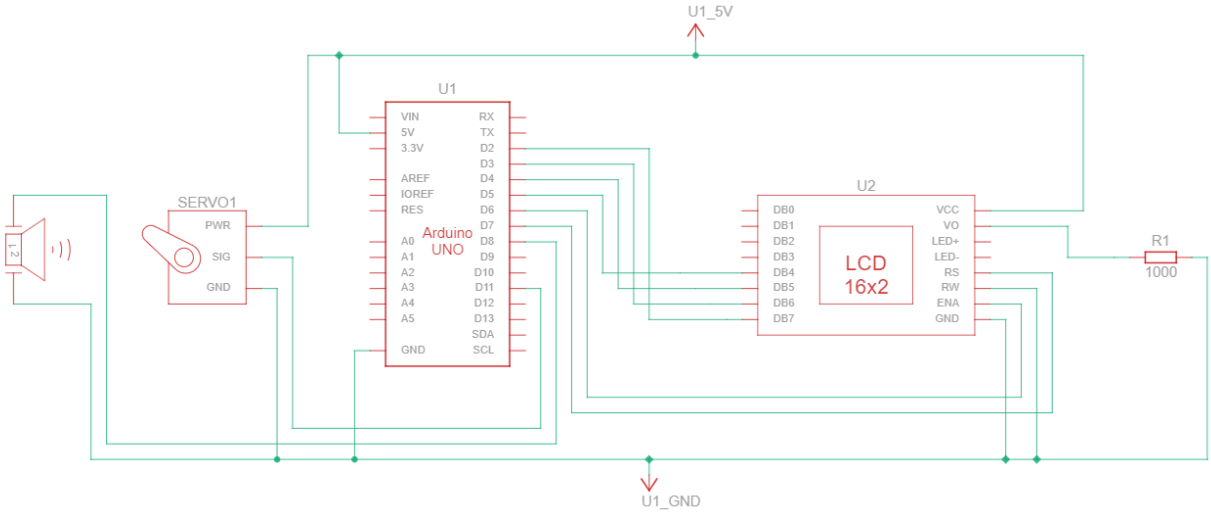

Figure 5. Circuit Schematic

Code

For programming the Arduino Uno, Arduino IDE was used. To program the game, sync state machines were used for their seamless implementation. Specifically, the separation of specific challenge levels translated well into state machines because the program can jump into the desired level and continue the rest of the program. Much of the code involved implementing the game logic and making separate states for each level. In total, there are 5 states used for the game status, and 2 additional states for the servo motor, specifically for the low and high values. The most challenging part of this program is keeping track of all the different states and when the system should jump to these states. As said, there were 7 different states that needed to be monitored, each with their own actions and updated variables. Below is the complete code for this project.

Arduino IDE Program (Arduino Uno Code):

```
#include<LiquidCrystal.h>
#define SERVO_PIN 11
LiquidCrystal lcd (7,6,5,4,3,2); //RS, E, D4, D5, D6, D7
```

```
//char Letters[26] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

 char Letters2[26] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

int Letter_Changer = 0;
int Letter_Changer2 = 0;
int Joystick_Btn = 13;
int Joystick_Btn2 = 12;
//int JoyStick_X1 = A1;
//int JoyStick_Y1 = A2;
int JoyStick_X2 = A3;
int JoyStick_Y2 = A4;

char guesses [250] = {'\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
'\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0' };

int arrayCounter = 0;
int guessNum = 0;
int lives = 6;
int correct = 0;
String phrase = "BARBB";
String phrase2 = "BBBA";
String phrase3 = "Work Please";
int k = 0;
int L = 9;
int H = 1;
int ServoCnt = 0;
int ServoCnt2 = 1;

int lenMicroSecondsOfPeriod = 20 * 1000; // 25 milliseconds (ms)
int lenMicroSecondsOfPulse = 1 * 1000; // 1 ms is 0 degrees
int first = 0.7 * 1000; //0.5ms is 0 degrees in HS-422 servo
int end = 3.7 * 1000;
int increment = 0.01 * 1000;
int current = 0;
```

```
unsigned long seed = 0;
char Display [250] = {'_',  '_', '_', '_', '_', '_', '_', '_', '_', '_', '_',
'_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_'};
int currLvl = 1;

const String Lvl_One_WordBank [21] = {" ", "BAR", "MOW", "MOR", "FIE", "YAG",
"DAP", "ORF", "SAL", "REV", "HOD", "ALT", "TAW", "ERE",
                                "NEF", "OBI", "NEB", "KEB", "ALB", "LUX", "BIS"};


const char* Lvl_Two_WordBank [21] = {" ", "DEME", "DAZE", "SEAH", "JILL", "ZION",
"SCOT", "YERN", "BAWD", "CAKE", "AIRY", "SPET",
                               "BIAS", "JOUK", "HALE", "DRAG", "BANC", "STOR",
"ROCK", "RHOB", "YOLL"};

const char* Lvl_Three_WordBank [21] = {" ", "RETCH", "SPONG", "APAIR", "PALPI",
"QUAIL", "SLICH", "WHELP", "BOSOM", "ALGOW", "GOUTY",
                                  "TITHE", "BUSBY", "URITH", "SHORE", "DIKER",
"BASSO", "OILER", "CATER", "HINNY", "EDEMA"};

byte Heart[8] = {  0b00000,
                   0b01010,
                   0b11111,
                   0b11111,
                   0b01110,
                   0b00100,
                   0b00000,
                   0b00000 };

typedef struct task {
  int state;
  unsigned long period;
  unsigned long elapsedTime;
  int (*TickFct)(int);

} task;
```

```
int delay_gcd;
const unsigned short tasksNum = 3; // Remember to change this as your testing
each SM
task tasks[tasksNum];


// ********************************************* Button Reader
*****************************************
/*int Button_Reader(int x) {
  if (x == 0 && digitalRead(x) == HIGH) {
      return 1;
  }
      else if (x == 13 && digitalRead(x) == LOW) {
        return 1;
      }
      else {
        return 0;
      }
}

*/

int Button_Reader2(int x) {
  if (x == 0 && digitalRead(x) == HIGH) {
      return 1;
  }
      else if (x == 12 && digitalRead(x) == LOW) {
        return 1;
      }
      else {
        return 0;
      }
}

// ********************************************* correctness
*****************************************
int correctness (char guess)
{
```

```
//Check if phrase contains char

if (currLvl == 1) {
    for (int i = 0; i < 250; i ++) {
      if (guess == guesses[i])//If previously entered
      {
        return 0;
      }
    }

    for (int i = 0; i < phrase.length(); i ++) {
      if (phrase.charAt(i) == guess)//If correct
      {
        if (lives < 6) {
          ++lives;
        }

        guessNum ++;
        guesses [guessNum-1] = guess;
        return 1;
      }
    }

    //If not correct
    lives --;
    guessNum ++;
    guesses [guessNum-1] = guess;
    Serial.println("WE ARE GOOD");

    return -1;
}

else if (currLvl == 2) {
      for (int i = 0; i < 250; i ++) {
        if (guess == guesses[i])//If previously entered
        {
          return 0;
        }
```

```
      }

      for (int i = 0; i < phrase2.length(); i ++) {
        if (phrase2.charAt(i) == guess)//If correct
        {
          if (lives < 6) {
            ++lives;
          }

          guessNum ++;
          guesses [guessNum-1] = guess;
          return 1;
        }
      }

      //If not correct
      lives --;
      guessNum ++;
      guesses [guessNum-1] = guess;
      Serial.println("WE ARE GOOD");

      return -1;
  }

  else if (currLvl == 3) {
      for (int i = 0; i < 250; i ++) {
        if (guess == guesses[i])//If previously entered
        {
          return 0;
        }
      }

      for (int i = 0; i < phrase3.length(); i ++) {
        if (phrase3.charAt(i) == guess)//If correct
        {
          if (lives < 6) {
            ++lives;
          }
```

```
            guessNum ++;
            guesses [guessNum-1] = guess;
            return 1;
          }
        }

        //If not correct
        lives --;
        guessNum ++;
        guesses [guessNum-1] = guess;
        Serial.println("WE ARE GOOD");

        return -1;
    }


}

// ********************************************* updateScreen
*********************************************
void updateScreen (int code)
{
  if (currLvl == 1) {
        if (code == 0)//Repeat of a character
        {
          for (int i = 0; i < 3; i ++)
          {
            lcd.setCursor(0,0);
            lcd.print("    REPEAT    ");
            delay(200);
            lcd.setCursor(0,0);
            lcd.print("              ");
            delay(200);
          }

        }
```

```
      else if (code == 1)//Correct character
      {
        for (int i = 0; i < phrase.length(); i ++)
        {
          if (phrase.charAt(i) == guesses [guessNum-1])
          {
            correct ++;
            Display[i] = guesses [guessNum-1];
            lcd.setCursor(0, 1);

            for (int i = 0; i < phrase.length(); i++) {
                lcd.print(Display[i]);
            }

          }
        }

      }

      else if (code == -1)//Incorrect character
      {
        Serial.println("WRONGGG");

      }
  }

  else if (currLvl == 2) {
      if (code == 0)//Repeat of a character
      {
        for (int i = 0; i < 3; i ++)
        {
          lcd.setCursor(0,0);
          lcd.print("    REPEAT    ");
          delay(200);
          lcd.setCursor(0,0);
          lcd.print("              ");
          delay(200);
        }
```

```
    }

    else if (code == 1)//Correct character
    {
      for (int i = 0; i < phrase2.length(); i ++)
      {
        if (phrase2.charAt(i) == guesses [guessNum-1])
        {
          correct ++;
          Display[i] = guesses [guessNum-1];
          lcd.setCursor(0, 1);

          for (int i = 0; i < phrase2.length(); i++) {
              lcd.print(Display[i]);
          }
        }
      }
    }

    else if (code == -1)//Incorrect character
    {
      Serial.println("WRONGGG");
    }
}

else if (currLvl == 3) {
    if (code == 0)//Repeat of a character
    {
      for (int i = 0; i < 3; i ++)
      {
        lcd.setCursor(0,0);
        lcd.print("    REPEAT    ");
        delay(200);
        lcd.setCursor(0,0);
        lcd.print("              ");
        delay(200);
      }
```

```
        }

        else if (code == 1)//Correct character
        {
          for (int i = 0; i < phrase3.length(); i ++)
          {
            if (phrase3.charAt(i) == guesses [guessNum-1])
            {
              correct ++;
              Display[i] = guesses [guessNum-1];
              lcd.setCursor(0, 1);

              for (int i = 0; i < phrase3.length(); i++) {
                  lcd.print(Display[i]);
              }
            }
          }
        }

        else if (code == -1)//Incorrect character
        {
          Serial.println("WRONGGG");
        }
    }
}
// ****************************************** Rand Function
*****************************************
int impRandom() {

          const unsigned long a = 16807;
          const unsigned long m = 21474893647;
          const unsigned long q = 127773;
          const double r = 2836.0;
          double test, lo;
          int hi;
          hi = seed / (double)q;
          lo = seed - (double)q * (double)hi;
```

```
          test = (double)a*lo - r*(double)hi;
          if (test > 0.0) {
            seed = test;
          }
          else {
            seed = test + (double)m;
          }
          int out = seed / m;
          return out;
}

int set() {

        int Number = impRandom() % 20 + 1;

        if ( Number < 0) {
            Serial.println("This was negative");
            Serial.println(Number);
            return Number * -1;
        }

        else if (Number > 0) {
            Serial.println("This was pos");
            Serial.println(Number);
            return Number; //1-16
        }
}


// ****************************************** Screen.  Function
*****************************************

void Screen() {
                if (currLvl == 1) {
                    lcd.clear(); // Change player one to "120bMan"
                    lcd.setCursor(0, 0);
                    lcd.print("P1<");
                    lcd.print(Letters2[Letter_Changer2]);
```

```
            lcd.print(">");
            lcd.print("      ");
            lcd.print("120B!");
            //lcd.print("P2<");
            //lcd.print(Letters2[Letter_Changer2]);
            //lcd.print(">");
            lcd.setCursor(13, 16);
            lcd.print("");
            lcd.print(lives);
            lcd.print("");
            lcd.setCursor(7, 0);
            lcd.print("L");
            lcd.print(currLvl);
            lcd.print("");

            lcd.setCursor(0, 1);


            for (int i = 0; i < phrase.length(); i++) {
                lcd.print(Display[i]);
            }

            Heart_Display();
        }


        else if (currLvl == 2) {
            //Serial.println("WE AT TWOOO");
            lcd.clear(); // Change player one to "120bMan"
            lcd.setCursor(0, 0);
            lcd.print("P1<");
            lcd.print(Letters2[Letter_Changer2]);
            lcd.print(">");
            lcd.print("      ");
            lcd.print("120B!");
            //lcd.print("P2<");
            //lcd.print(Letters2[Letter_Changer2]);
            //lcd.print(">");
```

```
            lcd.setCursor(13, 16);
            lcd.print("");
            lcd.print(lives);
            lcd.print("");
            lcd.setCursor(7, 0);
            lcd.print("L");
            lcd.print(currLvl);
            lcd.print("");

            lcd.setCursor(0, 1);

            for (int i = 0; i < phrase2.length(); i++) {
                lcd.print(Display[i]);
            }
            Heart_Display();
        }

        else if (currLvl == 3) {

            //Serial.println("WE AT TWOOO");
            lcd.clear(); // Change player one to "120bMan"
            lcd.setCursor(0, 0);
            lcd.print("P1<");
            lcd.print(Letters2[Letter_Changer2]);
            lcd.print(">");
            lcd.print("      ");
            lcd.print("120B!");
            //lcd.print("P2<");
            //lcd.print(Letters2[Letter_Changer2]);
            //lcd.print(">");
            lcd.setCursor(13, 16);
            lcd.print("");
            lcd.print(lives);
            lcd.print("");
            lcd.setCursor(7, 0);
            lcd.print("L");
            lcd.print(currLvl);
            lcd.print("");
```

```
                    lcd.setCursor(0, 1);

                    for (int i = 0; i < phrase3.length(); i++) {
                        lcd.print(Display[i]);
                    }

                    Heart_Display();
                }
}

// ****************************************** Rand Function
******************************************
void Heart_Display() {
     if (lives == 6) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
          lcd.setCursor(7, 1);
          lcd.write(byte(0));
          lcd.setCursor(8, 1);
          lcd.write(byte(0));
          lcd.setCursor(9, 1);
          lcd.write(byte(0));
          lcd.setCursor(10, 1);
          lcd.write(byte(0));
          lcd.setCursor(11, 1);
          lcd.write(byte(0));
      }

    else if (lives == 5) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
          lcd.setCursor(7, 1);
          lcd.write(byte(0));
          lcd.setCursor(8, 1);
          lcd.write(byte(0));
          lcd.setCursor(9, 1);
          lcd.write(byte(0));
```

```
          lcd.setCursor(10, 1);
          lcd.write(byte(0));
     }

    else if (lives == 4) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
          lcd.setCursor(7, 1);
          lcd.write(byte(0));
          lcd.setCursor(8, 1);
          lcd.write(byte(0));
          lcd.setCursor(9, 1);
          lcd.write(byte(0));
      }

    else if (lives == 3) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
          lcd.setCursor(7, 1);
          lcd.write(byte(0));
          lcd.setCursor(8, 1);
          lcd.write(byte(0));
          lcd.setCursor(9, 1);
      }

    else if (lives == 2) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
          lcd.setCursor(7, 1);
          lcd.write(byte(0));
      }

    else if (lives == 1) {
          lcd.setCursor(6, 1);
          lcd.write(byte(0));
      }
}
```

```
// ****************************************** Reset Function
******************************************
void Reset() {
    guessNum = 0;

    for (int i = 0; i < 20; i ++)
    {
      guesses[i] = '\0';
    }

    for (int i = 0; i < 20; i++) {
      Display[i] = '_';
    }

    // lcd.clear();
}


// ****************************************** States
******************************************
enum SM1_States {SM1_INIT, Level_1, Level_2, Level_3, End_Game} state1;
  int SM1_Tick(int state1) {
    switch (state1) {

        case SM1_INIT:
            if (currLvl == 1) {
                //Screen();
                Serial.println("HMMMMM");
                phrase = Lvl_One_WordBank[set()];
                Serial.println(phrase);
                // Serial.println(set());
                state1 = Level_1;
            }

            else if (currLvl == 2) {
                Reset();
                //Screen();
                Serial.println("INITI FUNCTINONNN");
```

```
                //phrase2 = "WWWW";
                int sewrt =set();
                phrase2 = Lvl_Two_WordBank[sewrt];
                //Serial.println(Lvl_Two_WordBank[set()]);
                Serial.println(Lvl_Two_WordBank[sewrt]);
                state1 = Level_2;
            }

            else if (currLvl == 3) {
                Reset();
                Serial.println("Levelll 33333");
                phrase3 = Lvl_Three_WordBank[set()]; // // Let's try and
understand what prints from the RNG and why it does so, so we could print
                                                    // someething out
                Serial.println(phrase3);
                state1 = Level_3;
            }

            break;

        case Level_1:
            if (lives == 0) {
                state1 = End_Game;
              }

            else if (correct < 3) {
              state1 = Level_1;
            }

            else if (correct == 3) {
               Serial.println("LEVELL ONEE COMPLETE");
               currLvl += 1;
               state1 = SM1_INIT;
            }

            break;

        case Level_2:
```

```
        if (lives == 0) {
            state1 = End_Game;
        }

        else if (correct < 7) {
            //Serial.println("LETSSS GOOO");
            state1 = Level_2;
        }

        else if (correct == 7) {
           //Serial.println("LEVELL TWOOO COMPLETE");
           currLvl += 1;
           state1 = SM1_INIT;
        }

        break;

    case Level_3:
        if (lives == 0) {
            state1 = End_Game;
        }

        else if (correct < 12) {
            state1 = Level_3;
        }

        else {
           Serial.println("LEVELL THREE COMPLETE");
           state1 = End_Game;
        }

        break;

    case End_Game:
      state1 = End_Game;
      break;
}
```

```
    switch(state1) {
      case SM1_INIT:
          break;

      case Level_1:
            break;

      case Level_2:
          break;

      case Level_3:
          break;

      case End_Game:
            if (lives > 1 && correct == 12) {
                  Serial.println("YOUUU WONNNN");
                  lcd.clear();
                  lcd.setCursor(0,0);
                  lcd.print("VICTORY SCREESH!");

                  lcd.setCursor(0,1);
                  lcd.print("VICTORY SCREESH!");
                  break;
             }

            else if (lives == 0) {
                    Serial.println("YOUUU LOSTTT");
                    lcd.clear();
                    lcd.setCursor(0,0);
                    lcd.print("IT'S OVER 9000!!");

                    lcd.setCursor(0,1);
                    lcd.print("IT'S OVER 9000!!");
                    break;
            }
            break;
    }
    return state1;
```

```
  }

enum SM2_States {SM2_INIT, Game} state2;
  int SM2_Tick(int state2) {
    switch (state2) {

        case SM2_INIT:
          state2 = Game;
          break;

        case Game:
          state2 = Game;
          break;
    }

      switch(state2) {
        case SM2_INIT:
            break;

        case Game:
              while (lives >= 1 && correct < 12) {
                   Screen();

                    if (analogRead(JoyStick_X2) > 800) {
                        ++Letter_Changer2;
                        if (Letter_Changer2 == 26) {
                          Letter_Changer2 = 0;
                        }
                        Serial.println(Letter_Changer2);
                    }

                    else if (analogRead(JoyStick_X2) < 300) {
                        --Letter_Changer2;
                        if (Letter_Changer2 == -1) {
                          Letter_Changer2 = 25;
                        }
                    }
```

```
                    else if (Button_Reader2(Joystick_Btn2)) {
                          updateScreen(correctness(Letters2[Letter_Changer2]));
                          guesses[guessNum] = Letters2[Letter_Changer2];
                          Serial.println(Letters2[Letter_Changer2]);
                          Serial.println(phrase);
                          Serial.println("Correct");
                          Serial.println(correct);
                    }
               break;
        }
          return state2;
    }
}

enum SM4_States { SM4_INIT, SHIGH, SLOW}; // PWM / DUTY CYCLE
int SM4_Tick(int state4) { // 20 / (9 + 1)
switch (state4) { // State transitions
  case SM4_INIT:
    k = 0;
    state4 = HIGH;
    break;

  case SHIGH:
      if (k <= 1) { // This would have been H
        state4 = SHIGH;
      }

      else if (k > 1) { // This would have been H
        k = 0;
        state4 = SLOW;
      }
      break;

  case SLOW:
    if (k <= 9) { // This would have been L
      state4 = SLOW;
    }
```

```
      else if(k > 9) { // This would have been L
        state4 = SHIGH;
        k = 0;
      }
      break;
  }

  switch (state4) { // State Action
    case SM4_INIT:
      break;
    case SHIGH:
          if (currLvl == 2 && ServoCnt < 1) {
            for(current = 2000; current <end; current+=increment){ // This is our
pulse
                //Serial.println("WEEEE HIGHHH");
                digitalWrite(SERVO_PIN, HIGH);
                delayMicroseconds(current);
                digitalWrite(SERVO_PIN, LOW);
                delayMicroseconds(lenMicroSecondsOfPeriod - current);
            }
                ++ServoCnt;
                Serial.println(ServoCnt);
                k++;
                break;
          }


          else if (currLvl == 3 && ServoCnt2 < 1) {

            for(current = 500; current <end; current+=increment){ // This is our
pulse
                //Serial.println("WEEEE HIGHHH");

                digitalWrite(SERVO_PIN, HIGH);
                delayMicroseconds(current);
                digitalWrite(SERVO_PIN, LOW);
                delayMicroseconds(lenMicroSecondsOfPeriod - current);
            }
```

```
                ++ServoCnt2;
                Serial.println(ServoCnt2);
                k++;
                break;
          }

   case SLOW:

        if (currLvl == 2 && ServoCnt < 2) {
            for(current = end; current >first; current-=increment){
                //Serial.println("WEEEE LOWWWWW");

                digitalWrite(SERVO_PIN, HIGH);
                delayMicroseconds(current);
                digitalWrite(SERVO_PIN, LOW);
                delayMicroseconds(lenMicroSecondsOfPeriod - current);
            }
                ++ServoCnt;
                 Serial.println(ServoCnt);
                k++;
                break;
        }

         else if (currLvl == 3 && ServoCnt2 < 2) {
            for(current = end; current >first; current-=increment){
                //Serial.println("WEEEE LOWWWWW");

                digitalWrite(SERVO_PIN, HIGH);
                delayMicroseconds(current);
                digitalWrite(SERVO_PIN, LOW);
                delayMicroseconds(lenMicroSecondsOfPeriod - current);
            }
                ++ServoCnt2;
                 Serial.println(ServoCnt2);
                k++;
                break;
        }
}
```

```
  return state4;
}

  void setup() {

      lcd.createChar(0, Heart);

      seed = analogRead(A5);
      pinMode(Joystick_Btn, INPUT_PULLUP);
      pinMode(Joystick_Btn2, INPUT_PULLUP);
      lcd.begin(16, 2);
      Serial.begin(9600);

      unsigned char i = 0;
      tasks[i].state = SM1_INIT;
      tasks[i].period = 100;
      tasks[i].elapsedTime = 0;
      tasks[i].TickFct = &SM1_Tick;
      i++;
      tasks[i].state = SM2_INIT;
      tasks[i].period = 300;
      tasks[i].elapsedTime = 0;
      tasks[i].TickFct = &SM2_Tick;
      i++;
      tasks[i].state = SM4_INIT;
      tasks[i].period = 100; // 200 / (4 + 1) = 40
      tasks[i].elapsedTime = 0;
      tasks[i].TickFct = &SM4_Tick;

      delay_gcd = 5;
  }


  void loop() {
      unsigned char i;
      for (i = 0; i < tasksNum; ++i) {
        if ( (millis() - tasks[i].elapsedTime) >= tasks[i].period) {
```

```
          tasks[i].state = tasks[i].TickFct(tasks[i].state);
          tasks[i].elapsedTime = millis();
        }
      }
}
```

Part 4. Testing & Evaluation

Introduction: This section discusses the testing environment and the evaluation methodology used for the Hangman project. The Hangman game aims to provide an engaging and interactive experience by seamlessly implementing the game logic and user feedback mechanisms. The testing environment simulated various gameplay scenarios to ensure the system's reliability and responsiveness. It was necessary to fully test all levels and possible scenarios for any undesired behavior.

Parts Required for Testing

- LCD Display (Verify that it displays game parameters accordingly)
- Joystick (Verify that it reads user input correctly)
- Buzzer (Verify sound functionality)
- LED Indicators (Verify functionality in game feedback)

Testing Environment

The testing environment aimed to mimic real gameplay scenarios where players interact with the system to guess letters and receive feedback. The following components and conditions were utilized:

- LCD Display: Displaying the current game status, guessed letters, and remaining attempts.
- Joystick Calibration**:** Allowing users to input their letter guesses.
- Buzzer and LED Indicators: Providing audio and visual feedback for correct and incorrect guesses.

Calibration Methodology

To ensure the accurate and reliable operation of the Hangman game, a software calibration process was conducted. The calibration methodology involved the following steps:

- Game states: The Arduino Uno was programmed to record game state changes and user inputs, logging these events for analysis.
- Parameter Monitoring: Making sure the game’s many variables were being updated with each level, scenario, guesses, and lives remaining.
- User Interaction Simulation: Various gameplay scenarios were simulated by interacting with the joystick and observing the system's response, making sure letter toggling and “enter” inputs worked as desired.

- Display Testing: Verify that the display was showing all information for the game including lives remaining, guesses, letters, and current level.

Testing Scenarios

Scenario 1 - Testing the System's Response to Incorrect Guesses:

Objective: To evaluate the system's ability to handle multiple incorrect guesses accurately and provide appropriate feedback.

Procedure:

1. Start a new game and intentionally input incorrect letters.
2. Observe the code and where it flowed through after the incorrect guess.
3. Observe the LCD display for accurate updates of remaining attempts.
4. Listen for the buzzer to emit a distinct sound for each incorrect guess.
5. Check the red LED to ensure it lights up for each incorrect guess.
6. Continue until the game reaches the maximum number of incorrect guesses and displays the "Game Over" message.
7. Repeat steps 1-6.

Scenario 2 - Testing the System's Response to Correct Guesses:

Objective: To assess the system's ability to handle correct guesses accurately and update the game state and display accordingly.

Procedure:

1. Start a new game and input correct letters one by one.
2. Guess correctly until the word is completed.
3. Observe the LCD display for accurate updates of the guessed letters and display them in their correct positions.
4. Check the green LED to ensure it lights up for each correct guess.
5. Continue until the entire word is correctly guessed and the "Victory" message is shown.

Scenario 3 - Manual Reset and Restart Testing:

Objective: To test the manual reset and restart functionality of the game by using the joystick, once the game over screen appears, or once all guess attempts are used.

Procedure:

1. Start a new game and make a few guesses.

2. Press the reset button to restart the game.
3. Observe the LCD display for resetting to the initial blank word with underscores and no letters (clean slate).
4. Ensure the remaining attempts counter and other variables are reset, and all LEDs are turned off.
5. If all of these conditions are met successfully, then the program works as desired.
6. Achieve the same results, but once the game's levels are beaten.
7. Achieve the same results once the game over screen is shown.

Conclusion: The three test cases, including testing different game scenarios, testing the system's response to incorrect guesses, and testing the system's response to correct guesses, provide valuable insights into the performance and functionality of the Hangman project. These tests verify the system's responsiveness, accuracy, and ability to provide engaging feedback, ensuring it works as intended in the real world. The manual reset and restart testing provide verification of the seamless implementation of the game as well as providing an intuitive experience.

Part 5. Discussion

Limitations & Advances Introduction

This section will discuss the limitations encountered in our Hangman project and potential solutions to address these limitations in future iterations.

Microcontroller Choice
The Arduino Uno was chosen for this project because of its simplicity and ease of use. While there is potential to use a better microcontroller for more feature integration, the arduino uno sufficed for this project. In the future, microcontrollers with ESP32 might offer enhanced capabilities such as built-in WiFi, Bluetooth, and more robust data handling. This could provide for better interaction options, such as multiplayer capabilities and more complex game logic. This can also help in implementing a bluetooth keyboard or controller for better user input.

Data Display and User Interface:
During development, one issue encountered involved the 16x2 LCD display. While the display used in this project suffices for basic output, its size and print capabilities restrict the amount of information that can be displayed simultaneously. This display prints the values similar to an old timer or digital watch. More advanced displays, such as graphical LCDs or OLED screens, would allow for a more engaging and visually appealing interface. OLED especially would be better because of their per-pixel printing capabilities. This would enable more detailed feedback and potentially add animations or images to enhance the user experience. In the end, it was possible to display all information in this display without clarity issues.

Input Method Challenges:
The current implementation uses a joystick for letter input, which is great for this type of project. It has all the necessary inputs with the directional controls and the button click. However, these controls can be difficult to understand for some users at first. Implementing a more intuitive input method, such as a touchscreen or a keyboard interface, would streamline the user experience. This would allow faster letter selections, but would take up more space. Implementing the keyboard or bluetooth controller would also be more complicated. In the end, the joystick offered the most compact and straightforward method of user input. With a few instructions, users were able to pick up the control scheme quickly.

Audio Feedback:
The buzzer provides audio feedback for correct and incorrect guesses. However, the sound quality and variety are limited. The buzzer used is capable of playing a variety of audio chimes, but not any mp3 files or other audio file formats. Integrating a better audio component would enhance the overall experience with more complex audio capabilities including mp3 playback. For this project, I needed a basic sound that could tell the user if the guess was right or wrong, so the buzzer did its job in providing two unique sounds for correct and incorrect guesses. The buzzer was also very compact, keeping the project easy to handle and implement.

Challenges & Drawbacks

This section discusses the challenges and drawbacks encountered during the development and implementation of the Hangman project.

Realistic Challenges

Display and Data Transmission Issues:
One primary challenge was reliably transmitting and displaying game data. Initially, we struggled with updating the display consistently, often experiencing delays or incorrect characters. Troubleshooting revealed that optimizing the code and minimizing display refresh rates improved performance. Additionally, integrating all components required careful synchronization to ensure accurate and timely updates on the LCD.

Input Method and Responsiveness:
Another challenge was ensuring the responsiveness of the joystick. Calibration was essential to prevent multiple inputs from a single joystick swipe. Implementing software joystick libraries and techniques improved input accuracy, but this added complexity to the code.

Audio Feedback Limitations:
The buzzer provided basic feedback, but its sound quality was limited. Implementing varied sound effects for different game events proved challenging due to the buzzer's limited capabilities. Exploring alternative audio solutions, such as piezo speakers or small audio modules, could address this limitation.

Drawbacks

Arduino Uno Limitations:
The Arduino Uno posed several drawbacks with its lack of advanced communication features (e.g., WiFi, Bluetooth), which limited the potential for multiplayer modes or remote interactions. More versatile microcontrollers, like the Teensy 4.0, would be better suited for expanding the game's features.

Display and User Interface Constraints:
The 16x2 LCD display was restrictive, limiting the amount of information shown simultaneously. This affected the game's user interface and made it hard to fit all information in one screen. Upgrading to a more advanced display would provide a better user experience.

Improvements and the Future

Display and microcontroller:
Implementing a graphical LCD or OLED screen would improve user interaction, allowing for more detailed feedback and a visually appealing interface. Replacing the Arduino Uno with a more advanced microcontroller like the Teensy 4.0 would enable additional features, such as multiplayer modes. It would also ease the implementation of all other features in this list with more pins, power capabilities, etc.

Audio Module Improvements:
Integrating a more advanced audio module would provide richer and more varied sound effects, enhancing the overall game experience. This could include background music, different tones for correct/incorrect guesses, and celebratory sounds for winning.

User Input Methods:
Switching to a touchscreen or keyboard interface would improve the speed and accuracy of user input, making the game more enjoyable to play.

Final Product Goals

Personalized Settings:
Create tailored game settings via an intuitive interface, allowing users to adjust difficulty levels, choose different word categories, and customize audio and visual feedback.

Enhanced User Experience:
Implement a high-quality display and advanced audio module to provide an engaging and immersive game experience.

Multiplayer:
Provide remote access and multiplayer capabilities, enabling users to play with others online or adjust game settings from a distance. Implement intelligent modes that adapt the game's difficulty based on user performance.

The final product aims to meet these goals, offering a user-friendly, engaging, and versatile Hangman Game that enhances the gaming experience for users of all ages.

Part. 6 Conclusion

This project provided an opportunity for the team to deepen their understanding of microcontroller programming, circuit design, and user interface development. At first, the project appeared straightforward, but as I delved deeper, it became evident that it would require significant time and effort. Our initial plan was to use the Arduino Uno to create a functional Hangman game with an LCD display and user input buttons. However, as development of the project progressed, it was desired to add more components including the LEDs and buzzers to enhance the user experience. The implementation plan changed during the development process leading to a more refined and complete project.

Our primary goal was to build a prototype Hangman game that could provide an engaging and educational experience for users. The significance of this project lies in enhancing our programming and hardware integration skills, potentially paving the way for more complex projects in the future including more game options and multiplayer.

Our methodology began with evaluating the available components and researching current input and display technologies. The initial design included an Arduino Uno, an LCD display, and push buttons for user input. However, issues with the push buttons arose in their ease of use. Using push buttons to control a game like this did not feel right and many users had problems with the control scheme. It was then when it was decided to replace the push button with a joystick. Through extensive testing and adjustments, including software debouncing techniques and optimized code, I managed to achieve reliable user input and system performance.

The two main deliverables chosen for this project were reliable data display and user-friendly input methods. As previously mentioned, the purpose of this project was to create an engaging game that can be easily played by anyone. The Arduino Uno’s capabilities were leveraged to control the LCD display and process user inputs efficiently.

In addition to basic gameplay functionality, a buzzer was integrated to provide audio feedback for correct and incorrect guesses. This audio feedback is crucial for enhancing the user experience, making the game more interactive and enjoyable. The buzzer's sounds were programmed to indicate different game events, such as correct letters, incorrect letters, and game over scenarios. This provides the user with additional feedback as well.

Once the user input is gathered through the joystick, the Arduino processes the input to update the game state. Using an efficient sync state machine algorithm, the system checks the guessed letters against the chosen word and updates the display accordingly. The game also includes a feature to keep track of incorrect guesses, providing visual feedback through the LCD display and audio feedback through the buzzer.

During the final demo, we ensured our system's functionality was clearly demonstrated. We used the LCD to display the word progress and the number of incorrect guesses, while the joystick demonstrated smooth letter selection. After numerous calibrations and tests, we successfully demonstrated to the section TA showing our project in action.

Lessons learned from this project include the importance of pre-testing components, collaborative integration of system parts, and starting early to manage unexpected issues. Engaging with the TA proved invaluable for troubleshooting and guidance. Future research will focus on exploring alternatives to the Arduino Uno, seeking microcontrollers with built-in WiFi or Bluetooth for multiplayer capabilities, and designing a more advanced user interface with graphical displays.

In conclusion, this project was challenging and stressful, particularly during debugging. However, it was also a rewarding and educational experience. I gained practical skills in system integration and problem-solving, and we hope to revisit this project in the future, potentially transforming it from a class assignment into a more sophisticated and feature-rich game. The integration of audio feedback and an efficient user input method has added significant value to the Hangman game, enhancing its functionality and making it a more engaging experience for users.