Project report on

Arduino-Powered 128x64 OLED Display for Playing Classic Games Including Flappy Bird and Pong.

Submitted by

Student Name: BADE NAVEEN KUMAR (2241011121)

Student Name: SAI SHANKET (2241019295)

B. Tech. (CSE(IOT) 5th Semester (Section–2241009)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Bhubaneswar, Odisha, India.
(Oct, 2024)

Abstract

This project showcases the design and development of an Arduino-powered gaming system using a 128x64 OLED display. The system allows users to play classic games such as Flappy Bird, Pong, and Ball Bounce, implemented using the Arduino IDE, two push-button controls, and a buzzer for sound effects. The project highlights the flexibility and affordability of Arduino hardware in recreating nostalgic gaming experiences. This system is a perfect demonstration of the power of microcontrollers in small-scale gaming, combining simplicity and innovation to create a low-cost, fun, and educational project.

Contents

Chapter 01: Introduction

1.1. Introduction

This project involves the design of a **game-playing system** using an **Arduino Uno** and a **128x64 OLED display**. The primary goal is to run simple yet enjoyable games like **Flappy Bird**, **Pong**, and **Ball Bounce**. The games are coded in **Embedded C** and use **two push buttons** for player interaction, simulating the feel of old arcade games.

1.2. Background

As retro gaming continues to gain popularity, there is an increased interest in developing **cost-effective gaming systems** using **microcontrollers**. The project explores how an **Arduino Uno** can be used for gaming purposes, making use of basic components like an OLED display, buttons, and a buzzer.

1.3. Project Objectives

- **Create an Arduino-based gaming system** using a 128x64 OLED display.
- Implement three classic games: **Flappy Bird**, **Pong**, and **Ball Bounce**.
- Develop a **low-cost** and **engaging** platform that demonstrates the versatility of Arduino for small-scale game development.

1.4. Scope

This project focuses on demonstrating **Arduino's capabilities** in handling gaming logic and display output with limited resources. It serves as a foundation for **expanding** into more complex game designs and **further integration** with sensors or sound modules.

1.5. Project Management

The project is organized into distinct phases: **planning**, **hardware setup**, **software development**, **testing**, and **final presentation**. Each phase is tracked to ensure timely completion with minimal resource expenditure.

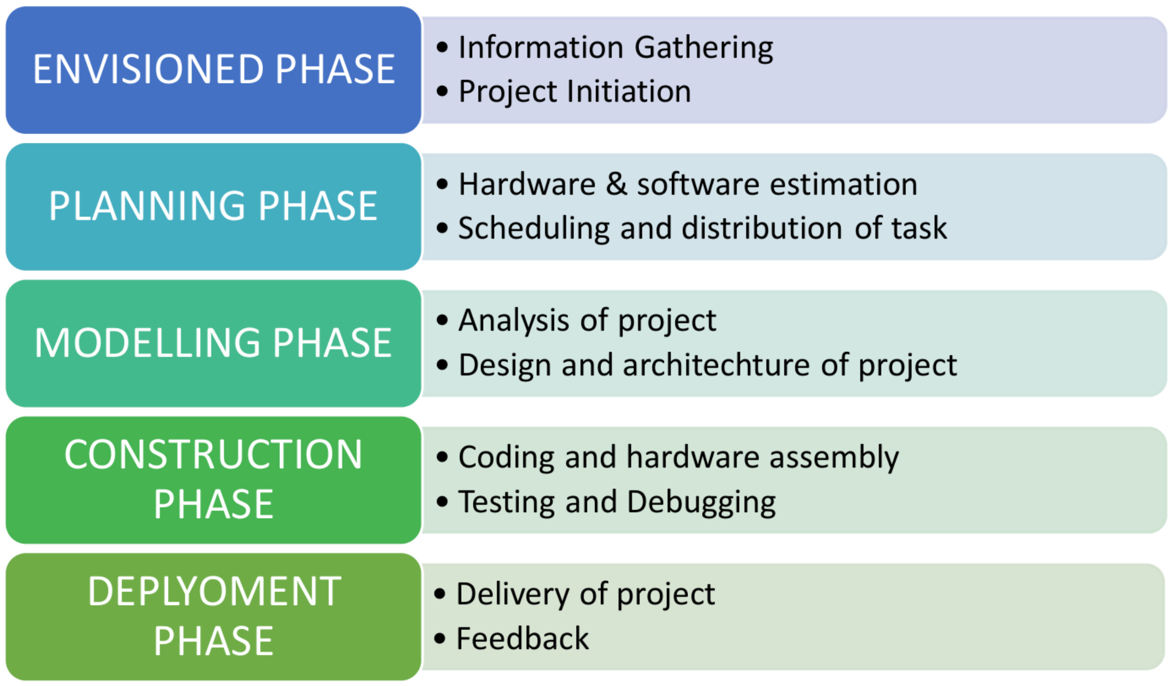


Figure 1. Model of phases in project management.

1.6. Overview and Benefits

This project highlights the **cost-effectiveness** of using **Arduino** to create a working gaming system. It provides an **educational platform** for students to learn **hardware programming** while enjoying **classic games** in a **compact and efficient design**.

1.7. Organization of the Report

The report is organised into the following chapters. Each chapter is unique on its own and is described with necessary theory to comprehend it.

Chapter 2 deals with background survey and review, Chapter 3 has the description of the theoretical aspects that has been acquired to commence the project work.

Chapter 02: Background Review & Survey

2.1. Related Works

Arduino projects are widely used for developing **retro games** due to their affordability and simplicity. Several projects have explored classic games like **Pong** and **Flappy Bird** using **OLED displays** and **push buttons**.

- **Pong on Arduino**: This project demonstrates Pong on an OLED display, similar to our implementation. The game uses simple logic for paddle and ball interaction.
- **Flappy Bird on Arduino**: In this Flappy Bird project, an OLED display and a button control the bird's movement. Our project follows a similar approach.
- **Arduino Games with Buzzer**: Adding sound for a more interactive experience, this project shows how a **buzzer** can enhance the gameplay with audio feedback.

These projects provided valuable insights into **game development** on Arduino, inspiring us to integrate **multiple games** and **buzzer-based sound effects**. For further reference, you can check out:

- Arduino Gaming with OLED
- Classic Pong on Arduino
- Flappy Bird on Arduino

Chapter 03: Theoretical Aspects

3.1. Internet of Things (IoT)

The gaming system consists of an **Arduino Uno**, a **128x64 OLED display**, and **two push buttons**. The user interacts with the games through the buttons, and the OLED display shows the game visuals. The programming is done in **Embedded C** using the **Arduino IDE**, and the games are designed to run efficiently within the hardware limitations.

3.2. Features of IoT

a) Intelligence
b) Connectivity
c) Dynamic Nature
d) Enormous Scale
e) Sensing
f) Heterogeneity
g) Security

3.3. Advantages of IoT

a) Communication
b) Automation and Control
c) Information
d) Monitoring
e) Efficiency

3.4. Disadvantages of IoT

a) Compatibility
b) Complexity
c) Privacy/Security
d) Safety

3.5. Application areas of IoT

- **Smart Homes**: Automated lighting, security, and appliances.
- **Healthcare**: Wearable health monitors, remote patient monitoring.
- **Industrial IoT (IIOT)**: Smart factories, predictive maintenance.
- **Agriculture**: Smart irrigation, crop monitoring, and livestock management.
- **Transportation**: Connected vehicles, fleet management, and smart traffic systems.
- **Retail**: Inventory tracking, smart shelves, and personalized shopping.

• **Smart Cities**: Smart grids, waste management, and public safety.
• **Environmental Monitoring**: Air and water quality monitoring.

3.6. IOT Technologies and Protocols

a) Bluetooth

b) Zigbee

c) Z-Wave

d) Wi-Fi

e) Cellular

f) NFC

g) LoRaWAN

3.7. Project Layout

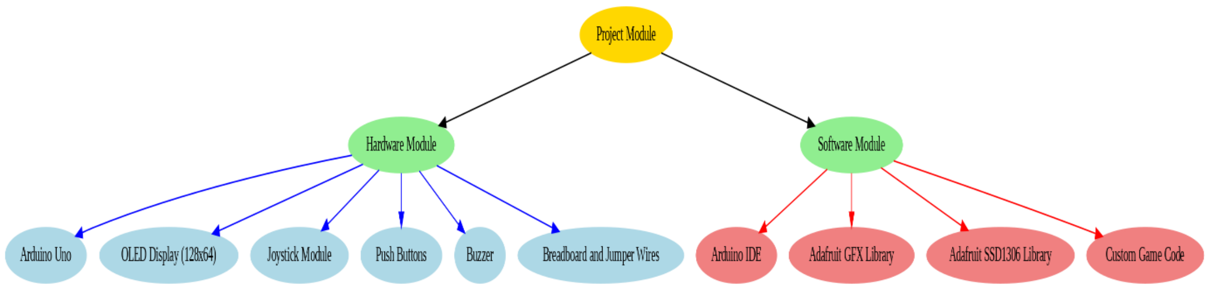


Figure 2. Layout of project module

3.7.1. Brief Description

1. Hardware Module (Green Oval)

This module includes the physical components required to build the project. These components are connected to the Hardware Module, and each serves a specific purpose:

- **Arduino Uno**: The microcontroller that acts as the brain of the project, controlling other components.
- **OLED Display (128x64)**: Used to display information or graphics, likely the game interface.
- **Joystick Module**: Provides directional input from the user.
- **Push Buttons**: Allow additional user input, such as selection or navigation.

- **Buzzer**: Possibly used for audio feedback, such as sounds or alerts.
- **Breadboard and Jumper Wires**: Essential for wiring and prototyping connections between components.

2. Software Module (Green Oval)

The software side of the project consists of tools and libraries required to program and control the hardware:

- **Arduino IDE**: The programming environment used to write, compile, and upload code to the Arduino.
- **Adafruit GFX Library**: A library used to handle graphics on the OLED display.
- **Adafruit SSD1306 Library**: A library for interfacing the Arduino with the OLED display.
- **Custom Game Code**: Likely contains the logic and functionality of a custom game or application that will run on the hardware.

3.Overall Project

The **Project Module** integrates both hardware and software elements to create a functional system. Based on the layout, it seems like the project revolves around building a small game or interactive system using an Arduino and an OLED display, with inputs from a joystick and buttons, and feedback from a buzzer.

Chapter 04: Hardware Requirements

4.1. Arduino Uno

The **Arduino Uno** serves as the heart of the project, controlling the game logic and handling user input.

4.1.1. Features

- **Microcontroller**: ATmega328P
- **Operating Voltage**: 5V
- **Digital I/O Pins**: 14 (6 PWM outputs)
- **Clock Speed**: 16 MHz

4.1.2. Pin Configuration

 OLED Display:

- **VCC** → Connect to the **5V** pin on the Arduino.
- **GND** → Connect to a **GND** pin on the Arduino.
- **SCL** → Connect to the **A5** pin on the Arduino.
- **SDA** → Connect to the **A4** pin on the Arduino.

 Buzzer:

- **Positive terminal** → Connect to digital pin **8** on the Arduino.
- **Negative terminal** → Connect to **GND**.

 Push Buttons:

- **Button 1**:
  - One terminal → Connect to digital pin **2**.
  - Other terminal → Connect to **GND**.
- **Button 2**:
  - One terminal → Connect to digital pin **3**.
  - Other terminal → Connect to **GND**.

 Joystick Module:

- **VCC** → Connect to the **5V** pin on the Arduino.
- **GND** → Connect to a **GND** pin on the Arduino.
- **VRX** → Connect to analog pin **A0**.
- **VRY** → Connect to analog pin **A1**.
- **SW (optional button)** → Connect to digital pin **4**.

4.2. ESP–01

The **ESP-01** module is a compact Wi-Fi module based on the ESP8266 chip. It can be used to enable wireless communication between the Arduino and other devices or applications. In this project, it can be included for future extensions like adding IoT functionality or remote game monitoring.

4.2.1. Features

- **Microcontroller**: ESP8266
- **Operating Voltage**: 3.3V
- **Wi-Fi**: 802.11 b/g/n
- **Flash Memory**: 1 MB (upgradable)
- **Communication**: UART, SPI

4.3. Sensors

Though sensors are not directly used in the game-playing system, they can be included for added functionality like controlling gameplay based on real-world inputs.

4.3.1. Pressure Sensor

The **Pressure Sensor** can be used to add a new feature where the user can control aspects of the game (such as character speed) by applying different pressure levels. For example, varying pressure might control the bird's movement in the Flappy Bird game.

The **Pressure Sensor** can be used to add a new feature where the user can control aspects of the game (such as character speed) by applying different pressure levels. For example, varying pressure might control the bird's movement in the Flappy Bird game.

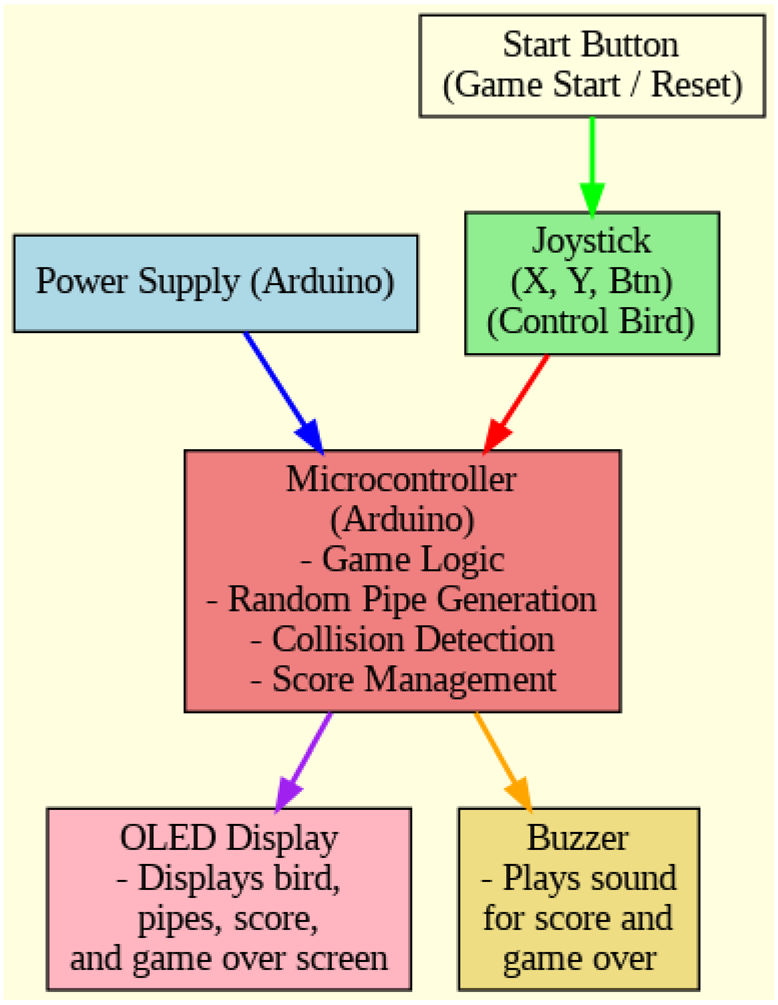
4.3.2. Temperature Sensor

A **Temperature Sensor** like the **DHT11** can be used to monitor the environment. It could be added to showcase dynamic backgrounds or in-game elements that respond to the current temperature, adding an extra layer of interactivity.

4.4. Additional components

- **Resistors**: To limit current flow to protect the Arduino and components.
- **LED**: To indicate power or in-game status changes.

4.5. Block diagram of the proposed system

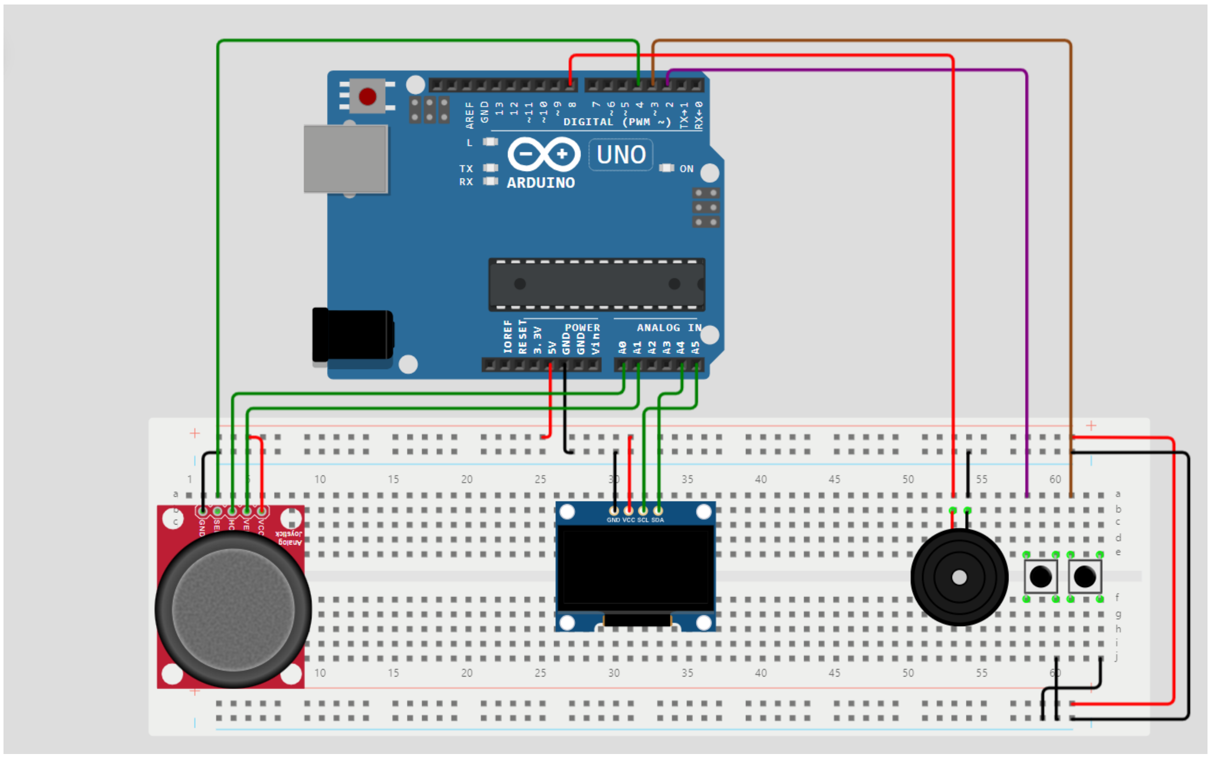


4.5.1. Working of the system

• **Game Initialization**: When the system is powered on, the Arduino loads the game (e.g., Flappy Bird) onto the OLED display using the Adafruit libraries.
• **User Input**: The player uses the joystick to control the game character (e.g., the bird in Flappy Bird). The joystick's VRX and VRY pins provide the movement data, while buttons can be used to select options or trigger actions.
• **Game Logic**: The Arduino processes user inputs and updates the game state accordingly. Collision detection, scoring, and other mechanics are handled by the microcontroller.
• **Display Output**: The updated game state is rendered on the OLED display.

• **Sound Feedback**: The buzzer provides sound effects for in-game actions like collisions or scoring.
• **Sensors (optional)**: Pressure or temperature sensors may influence in-game elements if implemented.
• **Wi-Fi Module (optional)**: The ESP-01 could be used to transmit game data to a cloud server or communicate with a mobile app (for example, using Blynk or Google Assistant).

4.5.2. Circuit Diagram



4.5.3. Components Required

Table 1. Component listing.

Sl. No.	Component and Specification	Quantity
1.	Arduino Uno	1
2.	OLED Display (128x64)	1
3.	Push Buttons	2
4.	Buzzer	1
5.	Breadboard	1
6.	Jumper Wires	Various

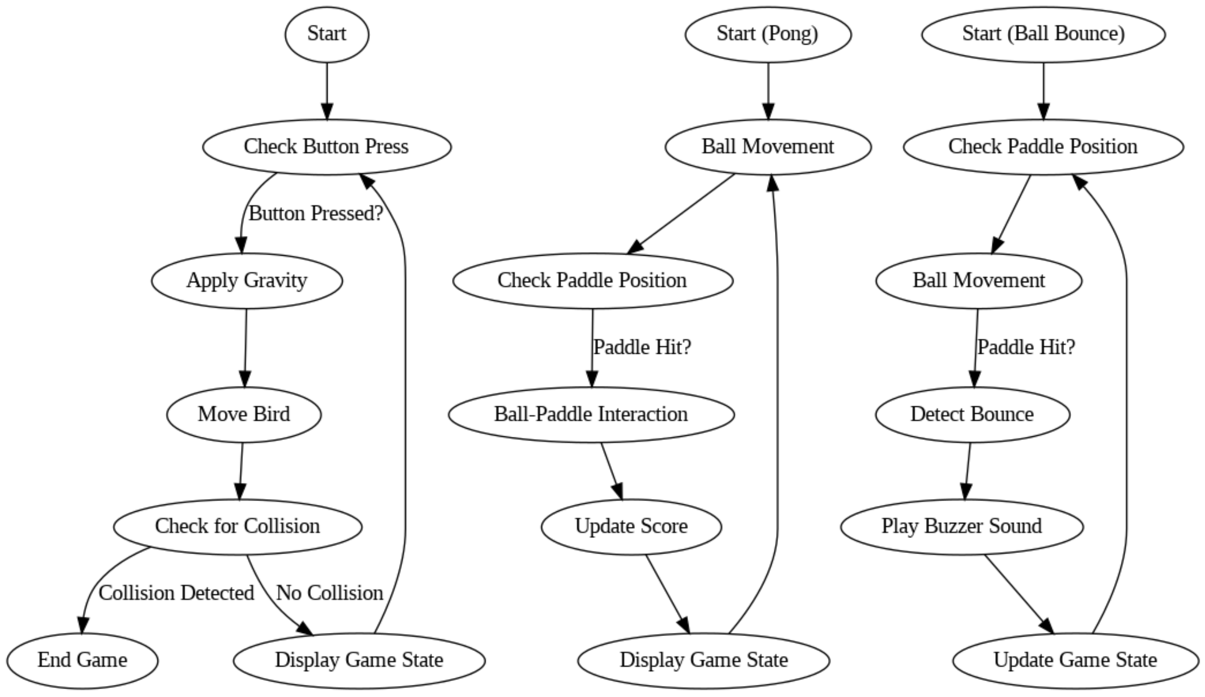
Chapter 05: Software Requirements

5.1. Arduino IDE (Embedded C / C++)

The software development for the games is done in **Embedded C** using the **Arduino IDE**. The games' mechanics, such as ball movement in Pong or obstacle generation in Flappy Bird, are implemented with efficient logic to ensure **smooth gameplay**. The buzzer provides **audio feedback** for events like scoring or game over.

5.2. Logic and Flowchart

Each game follows a defined flowchart. For example, **Flappy Bird** has logic for **gravity, bird movement**, and **obstacle collision**. Similarly, **Pong** uses logic for **ball movement, paddle control**, and **score tracking**, and **Ball Bounce** involves paddle-ball interaction with buzzer sounds on successful bounces.



Chapter 06: Project development & Testing Aspects

6.1. The project was developed using an iterative process of coding, testing, and debugging. Each game was individually tested to ensure smooth user interaction, accurate game mechanics, and reliable input recognition. The OLED display was calibrated to ensure proper resolution and smooth frame transitions, while the buzzer was tested for responsive sound effects.

Development Process

1. **Coding**:
    - Each game was programmed individually using Arduino IDE. The code was structured to facilitate easy debugging and modification, with comments added for clarity.
    - Libraries such as **Adafruit_GFX** and **Adafruit_SSD1306** were utilized to manage graphics on the OLED display, enhancing rendering quality.
2. **Testing**:
    - **Individual Game Testing**: Each game underwent rigorous testing to evaluate user interaction and game mechanics. This included:
        - **Flappy Bird**: Focused on jump responsiveness, gravity implementation, and collision detection with obstacles.
        - **Pong**: Examined ball movement, paddle control, and score updating mechanisms.
        - **Ball Bounce**: Tested paddle-ball interaction and sound effects triggered by successful bounces.
    - **User Interaction**: User input recognition was thoroughly evaluated, ensuring that the buttons and joystick responded promptly to user actions.
3. **Debugging**:
    - Debugging sessions were conducted to identify and fix any issues that arose during testing. This involved:
        - Reviewing console output for error messages.
        - Adjusting parameters such as button debounce time to enhance input accuracy.
        - Utilizing serial monitoring to track variable states during gameplay.

Calibration and Optimization

- **OLED Display Calibration**:

    - The OLED display was calibrated to ensure optimal resolution and clarity. This involved:
        - Adjusting contrast and brightness settings for better visibility.
        - Testing different frame rates to achieve smooth animations and transitions, preventing flickering or lag during gameplay.
- **Buzzer Sound Effects**:
    - The buzzer was tested to ensure that sound effects were responsive and synchronized with game events. This included:
        - Tuning sound frequencies for different game actions (e.g., jump, score, bounce).
        - Evaluating sound duration and volume to create an engaging audio experience without being overwhelming.

Final Testing

- After individual testing and debugging, a comprehensive playtest session was conducted where all three games were tested together. This allowed for:
    - **Performance Evaluation**: Assessing how the system handled multiple inputs and outputs simultaneously.
    - **User Experience Feedback**: Gathering feedback from users to identify any areas for improvement in gameplay mechanics or user interface.

Chapter 07: Conclusion & Future Scope

7.1. Result

The **Arduino-powered OLED gaming system** was successfully created and can run three games: **Flappy Bird**, **Pong**, and **Ball Bounce**. The system responds efficiently to button inputs, with **audio feedback** from the buzzer enhancing the user experience. The games are simple, yet engaging, providing a **fun and interactive experience**.

7.2. Conclusion

This project proves that **Arduino**, paired with a simple OLED display, two buttons, a buzzer, and a breadboard, can serve as a **versatile platform** for developing and running classic games. It is a perfect example of how **basic hardware** can deliver a high-quality experience when paired with **effective programming**.

7.3. Limitations

The system's main limitations are due to the **Arduino's limited processing power** and the small size of the **OLED display**, which restricts the complexity of the games that can be developed.

The system's main limitations are due to the Arduino's limited processing power and the small size of the OLED display, which restricts the complexity of the games that can be developed.

7.4. Further Enhancement and Future Scope

The system could be enhanced by adding features such as **more complex sound effects**, **additional games**, and **sensor-based inputs**. Additionally, more games could be developed, and multiplayer functionality could be introduced by integrating wireless communication modules.

References

- https://www.youtube.com/watch?v=sfm1XXSoj0U
- https://github.com/makeabilitylab/arduino/blob/master/OLED/FlappyBird/FlappyBird.ino

Appendix 01

A01. Main Code

The following is the main code for three classic games developed for the **Arduino Uno** using a **128x64 OLED Display**, a joystick for control, and buttons for game interaction.

Flappy Bird Code

cpp

Copy code

```cpp
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Pin Definitions
#define OLED_RESET -1
Adafruit_SSD1306 display(OLED_RESET);
#define BUZZER_PIN 8
#define BUTTON_PIN 2
#define JOYSTICK_X A0
#define JOYSTICK_Y A1
#define JOYSTICK_BTN 4

// Game Variables
int birdY = 30;
int birdX = 10;
int pipeX = 128;
int pipeGap = 20;
int pipeHeight = random(10, 40);

int score = 0;

int gravity = 1;

int birdVelocity = 0;

bool gameOver = false;


void setup() {

  // Initialize OLED display

  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  display.clearDisplay();

  display.display();


  // Initialize buzzer and buttons

  pinMode(BUZZER_PIN, OUTPUT);

  pinMode(BUTTON_PIN, INPUT_PULLUP); // Use internal pull-up resistor for button

  pinMode(JOYSTICK_BTN, INPUT_PULLUP);


  // Seed the random number generator for random pipe heights

  randomSeed(analogRead(0));


  // Welcome Screen

  display.setTextSize(1);

  display.setTextColor(WHITE);

  display.setCursor(0, 10);

  display.println("Flappy Bird!");
  display.println("Press to start...");
  display.display();

  // Wait for button press to start the game
  while (digitalRead(BUTTON_PIN) == HIGH);

  display.clearDisplay();
  display.display();
}

void loop() {
  if (!gameOver) {
    updateBird();
    updatePipes();
    checkCollision();
    drawGame();
    delay(50); // Controls the speed of the game
  } else {
    gameOverScreen();
    if (digitalRead(BUTTON_PIN) == LOW) {
      resetGame();
    }
  }
}
```

```
void updateBird() {
 // Read joystick to move bird
 int joystickY = analogRead(JOYSTICK_Y);
 if (joystickY < 400) {
   birdVelocity = -3; // Move bird up
 } else {
   birdVelocity += gravity;
 }

 birdY += birdVelocity;
 if (birdY < 0) birdY = 0; // Prevent bird from going out of bounds
 if (birdY > 63) birdY = 63; // Prevent bird from falling off the screen
}

void updatePipes() {
 pipeX -= 2; // Move pipes left

 if (pipeX < -10) { // Pipe goes off screen, reset it
   pipeX = 128;
   pipeHeight = random(10, 40);
   score++;
   tone(BUZZER_PIN, 1000, 100); // Play beep sound when score increases
 }
}
```

```
void checkCollision() {
  // Check if bird hits the top or bottom of the pipe
  if ((birdY < pipeHeight || birdY > pipeHeight + pipeGap) && pipeX < birdX + 6 && pipeX
> birdX - 6) {
    gameOver = true;
    tone(BUZZER_PIN, 500, 1000); // Play game over sound
  }
}

void drawGame() {
  display.clearDisplay();

  // Draw bird
  display.fillRect(birdX, birdY, 6, 6, WHITE);

  // Draw pipe
  display.fillRect(pipeX, 0, 10, pipeHeight, WHITE);
  display.fillRect(pipeX, pipeHeight + pipeGap, 10, 64 - pipeHeight - pipeGap, WHITE);

  // Draw score
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("Score: ");
  display.print(score);
  display.display();
}

void gameOverScreen() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 20);
  display.println("Game Over!");
  display.setCursor(0, 30);
  display.print("Score: ");
  display.println(score);
  display.setCursor(0, 40);
  display.println("Press button to restart.");
  display.display();
}

void resetGame() {
  birdY = 30;
  birdVelocity = 0;
  pipeX = 128;
  score = 0;
  gameOver = false;
}
```

Ball Bounce Code

cpp

Copy code

```
#include <Adafruit_SSD1306.h>
#include <Wire.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Pin assignments
#define JOYSTICK_X_PIN A0
#define JOYSTICK_Y_PIN A1
#define BUTTON1_PIN 2  // Button to start/restart the game
#define BUTTON2_PIN 3  // Optional button (not used in this game)
#define BUZZER_PIN 8

// Ball properties
int ballX, ballY;
int ballSpeedX = 2;
int ballSpeedY = 2;
```

```
const int ballRadius = 3;


// Paddle properties

int paddleX;

const int paddleY = SCREEN_HEIGHT - 10;

const int paddleWidth = 20;

const int paddleHeight = 3;


// Game state variables

int score = 0;

const int maxScore = 15; // Winning score

bool gameStarted = false;


void setup() {

  pinMode(BUTTON1_PIN, INPUT_PULLUP);

  pinMode(BUTTON2_PIN, INPUT_PULLUP);  // Not used in this game

  pinMode(BUZZER_PIN, OUTPUT);


  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {

    while (true); // Stay here if the display doesn't initialize

  }


  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(SSD1306_WHITE);
  // Display welcome screen
  welcomeScreen();
}

void loop() {
  if (!gameStarted) {
    // Check if Button 1 is pressed to start the game
    if (digitalRead(BUTTON1_PIN) == LOW) {
      gameStarted = true;
      resetGame();
    }
    return; // Exit loop if game hasn't started yet
  }

  // Move the paddle with the joystick
  int joystickX = analogRead(JOYSTICK_X_PIN);
  paddleX = map(joystickX, 0, 1023, 0, SCREEN_WIDTH - paddleWidth);

  // Update ball position
  updateBallPosition();

  // Check for ball collision with paddle
  checkPaddleCollision();
  // Draw everything on the display
  drawGame();
}

// Display the welcome screen
void welcomeScreen() {
  display.setCursor(0, 0);
  display.println("Ball Bounce Game");
  display.println("Press Button 1 to Start");
  display.display();
}

// Reset the game state
void resetGame() {
  ballX = SCREEN_WIDTH / 2; // Center the ball horizontally
  ballY = 10; // Set ball just below the top of the screen
  score = 0; // Reset score
  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.println("Set Max Score");
  display.display();
}
```

```
// Update ball position
void updateBallPosition() {
 ballX += ballSpeedX;
 ballY += ballSpeedY;

 // Ball collision with walls
 if (ballX - ballRadius <= 0 || ballX + ballRadius >= SCREEN_WIDTH) {
  ballSpeedX = -ballSpeedX; // Reverse direction on horizontal wall collision
 }
 if (ballY - ballRadius <= 0) {
  ballSpeedY = -ballSpeedY; // Reverse direction on upper wall collision
 }

 // Check if the ball falls to the bottom (game over)
 if (ballY + ballRadius >= SCREEN_HEIGHT) {
  playBuzzer(500, 500); // Play game over sound
  displayWinner(false);
  resetGame(); // Reset game
 }
}

// Check for ball collision with paddle
void checkPaddleCollision() {
 if (ballY + ballRadius >= paddleY && ballX >= paddleX && ballX <= paddleX +
paddleWidth) {
    ballSpeedY = -ballSpeedY; // Reverse direction on paddle collision
    score++; // Increase score
    playBuzzer(1200, 100); // Play sound when scoring

    // Check if the score reaches the winning score
    if (score >= maxScore) {
      displayWinner(true);
      resetGame(); // Reset game after displaying winner
    }
  }
}

// Draw game elements
void drawGame() {
  display.clearDisplay();
  drawPaddle();
  drawBall();
  displayScore();
  display.display();
}

// Draw paddle
void drawPaddle() {
  display.fillRect(paddleX, paddleY, paddleWidth, paddleHeight, SSD1306_WHITE);
```

Pong Game

```
#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128 // OLED display width, in pixels

#define SCREEN_HEIGHT 64 // OLED display height, in pixels


// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)

#define OLED_RESET -1 // Set to -1 as we are not using the reset pin

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


// Pin Definitions

const int JOYSTICK_X_PIN = A0; // Joystick X-axis (VRX)

const int JOYSTICK_Y_PIN = A1; // Joystick Y-axis (VRY)

const int BUZZER_PIN = 8;      // Buzzer pin

const int BUTTON1_PIN = 2;     // Start/Confirm button

const int BUTTON2_PIN = 3;     // Move Left paddle button


// Game Variables

int paddleHeight = 20;

int paddleWidth = 3;
```

```
// Right Paddle (controlled by joystick)

int rightPaddleX = SCREEN_WIDTH - paddleWidth - 1;

int rightPaddleY = (SCREEN_HEIGHT / 2) - (paddleHeight / 2);


// Left Paddle (controlled by button)

int leftPaddleX = 0;

int leftPaddleY = (SCREEN_HEIGHT / 2) - (paddleHeight / 2);


// Ball

int ballX = SCREEN_WIDTH / 2;

int ballY = SCREEN_HEIGHT / 2;

int ballRadius = 2;

int ballSpeedX = 2;

int ballSpeedY = 2;


int rightPlayerScore = 0;

int leftPlayerScore = 0;


int maxScore = 15;  // Default maximum score

bool gameStarted = false;    // Variable to track if the game has started

bool scoreAdjusted = false;  // To track if the score adjustment screen has been passed


// Function to draw the paddles

void drawPaddles() {

  // Draw right paddle
  display.fillRect(rightPaddleX, rightPaddleY, paddleWidth, paddleHeight,
SSD1306_WHITE);

  // Draw left paddle
  display.fillRect(leftPaddleX, leftPaddleY, paddleWidth, paddleHeight, SSD1306_WHITE);
}

// Function to draw the ball
void drawBall() {
  display.fillCircle(ballX, ballY, ballRadius, SSD1306_WHITE);
}

// Function to play sound on the buzzer
void playBuzzer(int frequency, int duration) {
  tone(BUZZER_PIN, frequency, duration);
}

// Setup function
void setup() {
  // Initialize the display
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }
  display.clearDisplay();
  // Pin Modes
  pinMode(JOYSTICK_X_PIN, INPUT);
  pinMode(JOYSTICK_Y_PIN, INPUT);
  pinMode(BUTTON1_PIN, INPUT_PULLUP); // Start/Confirm button
  pinMode(BUTTON2_PIN, INPUT_PULLUP); // Left paddle button
  pinMode(BUZZER_PIN, OUTPUT);

  // Initialize variables
  ballX = SCREEN_WIDTH / 2;
  ballY = SCREEN_HEIGHT / 2;

  // Display the initial screen
  showStartScreen();
}

// Function to show the initial welcome screen
void showStartScreen() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(20, 10);
  display.print("WELCOME TO PONG!");
  display.setCursor(20, 30);
  display.print("Press Button 1");
  display.setCursor(20, 40);
  display.print("to Start");
  display.display();

  // Wait for the player to press Button 1 to start the game
  while (digitalRead(BUTTON1_PIN) == HIGH) {
    // Wait for button press (loop until the button is pressed)
  }
  playBuzzer(1000, 200); // Play a sound to indicate game start
  adjustMaxScoreScreen();
}

// Function to adjust the maximum score using VRX joystick
void adjustMaxScoreScreen() {
  while (!scoreAdjusted) {
    // Read the joystick X-axis to adjust the max score
    int joystickXVal = analogRead(JOYSTICK_X_PIN);
    int adjustedScore = map(joystickXVal, 0, 1023, 5, 50);  // Map VRX to range between 5
and 50
    adjustedScore = (adjustedScore / 5) * 5;  // Round to nearest multiple of 5

    // Display the max score selection screen
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(20, 10);
    display.print("Set Max Score");
    display.setCursor(20, 30);
    display.print("Max Score: ");
    display.print(adjustedScore);
    display.setCursor(20, 50);
    display.print("Press Button 1");
    display.display();

    // Update the max score with the adjusted value
    maxScore = adjustedScore;

    // Wait for the player to press Button 1 to confirm
    if (digitalRead(BUTTON1_PIN) == LOW) {
      playBuzzer(1000, 200); // Play a sound to confirm
      scoreAdjusted = true;
      gameStarted = true;
    }
  }
}

// Function to reset the game and show the start screen
void resetGame() {
  if (digitalRead(BUTTON1_PIN) == LOW) {
    // Reset ball, paddles, and scores
    ballX = SCREEN_WIDTH / 2;
    ballY = SCREEN_HEIGHT / 2;
    ballSpeedX = 2;
    ballSpeedY = 2;
    rightPlayerScore = 0;
    leftPlayerScore = 0;

    // Play a reset sound
    playBuzzer(2000, 500);

    // Return to the welcome screen and wait for a new game to start
    gameStarted = false;
    scoreAdjusted = false;
    showStartScreen();
  }
}

// Function to declare a winner and reset the game
void declareWinner(const char* winner) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(20, 20);
  display.print(winner);
  display.setCursor(20, 40);
  display.print("Wins!");
  display.display();

  // Play a winning sound
 // playBuzzer(1500, 1000);

  // Wait for 3 seconds to display the winner
  delay(3000);

  // Reset the game and return to start screen
  showStartScreen();
}

// Function to move the right paddle using the joystick
void moveRightPaddle() {
  int joystickYVal = analogRead(JOYSTICK_Y_PIN);
  int mappedY = map(joystickYVal, 0, 1023, 0, SCREEN_HEIGHT - paddleHeight);
  rightPaddleY = mappedY;
}

// Function to move the left paddle using button control
void moveLeftPaddle() {
  if (digitalRead(BUTTON2_PIN) == LOW) {
    leftPaddleY -= 2; // Move up when button pressed
    if (leftPaddleY < 0) leftPaddleY = 0; // Keep within bounds
  } else {
    leftPaddleY += 2; // Move down otherwise
    if (leftPaddleY + paddleHeight > SCREEN_HEIGHT) leftPaddleY = SCREEN_HEIGHT - paddleHeight;
  }
}

// Game loop
void loop() {
  if (!gameStarted) {
    return;  // Don't run the game loop until the game has started
  }

  display.clearDisplay();

  // Move paddles
  moveRightPaddle();
  moveLeftPaddle();

  // Reset game if the reset button is pressed
  resetGame();

  // Move ball
  ballX += ballSpeedX;
  ballY += ballSpeedY;
  // Ball collision with top or bottom walls
  if (ballY - ballRadius <= 0 || ballY + ballRadius >= SCREEN_HEIGHT) {
    ballSpeedY = -ballSpeedY;
  }

  // Ball collision with right paddle
  if (ballX + ballRadius >= rightPaddleX && ballY >= rightPaddleY && ballY <=
rightPaddleY + paddleHeight) {
    ballSpeedX = -ballSpeedX;
    rightPlayerScore++;
    playBuzzer(1200, 100); // Play sound on paddle hit
  }

  // Ball collision with left paddle
  if (ballX - ballRadius <= leftPaddleX + paddleWidth && ballY >= leftPaddleY && ballY <=
leftPaddleY + paddleHeight) {
    ballSpeedX = -ballSpeedX;
    leftPlayerScore++;
    playBuzzer(1200, 100); // Play sound on paddle hit
  }

  // Ball out of bounds (miss) on right side
  if (ballX + ballRadius >= SCREEN_WIDTH) {
    ballX = SCREEN_WIDTH / 2;
    ballY = SCREEN_HEIGHT / 2;
    ballSpeedX = -2;
    ballSpeedY = 2;
    leftPlayerScore++;
    playBuzzer(500, 500); // Play sound on miss
  }

  // Ball out of bounds (miss) on left side
  if (ballX - ballRadius <= 0) {
    ballX = SCREEN_WIDTH / 2;
    ballY = SCREEN_HEIGHT / 2;
    ballSpeedX = 2;
    ballSpeedY = 2;
    rightPlayerScore++;
    playBuzzer(500, 500); // Play sound on miss
  }

  // Draw paddles and ball
  drawPaddles();
  drawBall();

  // Display the scores
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
```

```
  display.print("L: ");
  display.print(leftPlayerScore);
  display.setCursor(SCREEN_WIDTH - 30, 0);
  display.print("R: ");
  display.print(rightPlayerScore);

  // Check if either player reached the maximum score
  if (leftPlayerScore >= maxScore) {
    declareWinner("Left Player");
  } else if (rightPlayerScore >= maxScore) {
    declareWinner("Right Player");
  }

  display.display();
}
```

A01.3. Libraries

The following libraries are required to run the code on the **Arduino Uno** with a **128x64 OLED display**:

1. **Adafruit GFX Library**:
   - This library is essential for handling graphical operations such as drawing shapes, text, and other elements on the display.
   - **Installation**: You can install this via the Arduino IDE Library Manager by searching for **Adafruit GFX**.
2. **Adafruit SSD1306 Library**:
   - This library is used to interface with the SSD1306-based OLED display over I2C.
   - **Installation**: You can install this via the Arduino IDE Library Manager by searching for **Adafruit SSD1306**.

These libraries handle the display rendering and communication between the Arduino and the OLED screen. They are vital for drawing graphics and text for the Flappy Bird, Pong, and Ball Bounce games.

Appendix 02

A02.1. Project Proposal Form

The project proposal form was prepared and duly signed from our Faculty-in-Charge Dr. Biswaranjan Swain. The same is attached at the last of this report.

A02.2. Project Management

#	Component	Individual Contributions in %		Total
		Name 01	**Name 02**	
1.	Planning	50%	50%	100%
2.	Background Research and Analysis	90%	10%	100%
3.	Hardware design	60%	40%	100%
4.	Software design	80%	20%	100%
5.	Testing	30%	70%	100%
6.	Final Assembling	50%	50%	100%
7.	Project report writing	50%	50%	100%
8.	Presentation	80%	20%	100%
9.	Logistics	10%	90%	100%

A02.3. Bill of Material

Table 1. Component listing.

#	Component	Specification	Unit Cost	Quantity	Total
1.	Arduino Uno	ATmega328P	500	1	500
2.	OLED Display (128x64)	I2C interface	150	1	150
3.	Push Buttons	2-pin	5	2	10
4.	Jumper Wires	Standard	5	20	50
5.	USB Cable for Arduino Uno	Type A to Type B	60	1	60
6.	Buzzer	Active/passive	20	1	20
7.	Joystick Module	2-axis (X, Y) + Button	100	1	100
Grand Total					890

Appendix 03

A03.1. Data Sheets

For the successful completion of this project, understanding the specifications and operational details of each hardware component is crucial. Below is a list of components used, along with references to their data sheets for detailed information.

1. Arduino Uno (ATmega328P Microcontroller)

- **Description**: The Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins, 6 analog inputs, USB connection, power jack, and a reset button.
- **Key Specifications**:
  - **Microcontroller**: ATmega328P
  - **Operating Voltage**: 5V
  - **Input Voltage (recommended)**: 7-12V
  - **Digital I/O Pins**: 14 (6 PWM outputs)
  - **Analog Input Pins**: 6
  - **Flash Memory**: 32 KB
  - **Clock Speed**: 16 MHz
- **Data Sheet Reference**:
  - ATmega328P Data Sheet: Microchip Technology - ATmega328P
  - Arduino Uno Technical Specs: Arduino Official Website

2. 128x64 OLED Display (SSD1306 Driver)

- **Description**: A monochrome 0.96-inch OLED display module based on the SSD1306 driver, communicating via I2C interface.
- **Key Specifications**:
  - **Resolution**: 128x64 pixels
  - **Interface**: I2C

- **Operating Voltage**: 3.3V to 5V
- **Driver IC**: SSD1306

- **Data Sheet Reference**:
  - SSD1306 Driver Data Sheet: SSD1306 Data Sheet
  - OLED Module Specifications: Adafruit OLED Display

3. Joystick Module

- **Description**: An analog joystick module similar to those found in game controllers, providing two axes (X and Y) and a button when pressed down.
- **Key Specifications**:
  - **Operating Voltage**: 5V
  - **Outputs**: Two analog outputs for X and Y axes, one digital output for the button
- **Data Sheet Reference**:
  - Joystick Module Details: Generic Joystick Module Data Sheet

4. Push Buttons

- **Description**: Standard momentary push buttons used for starting, restarting, or interacting with the games.
- **Key Specifications**:
  - **Type**: Normally Open (NO)
  - **Operating Voltage**: Compatible with 5V logic
- **Data Sheet Reference**:
  - Push Button Specifications: Refer to the specific model used, commonly available from electronics suppliers.

5. Buzzer

- **Description**: An audio signaling device that can produce sounds of various frequencies, used here for game sound effects.
- **Key Specifications**:
  - **Type**: Active or Passive (depending on model)
  - **Operating Voltage**: Typically 5V
- **Data Sheet Reference**:
  - Buzzer Module Details: Passive and Active Buzzer Modules

6. USB Cable for Arduino Uno

- **Description**: A USB Type A to Type B cable used for powering the Arduino Uno and uploading code.
- **Key Specifications**:
  - **Connector Types**: USB Type A (computer side) to USB Type B (Arduino side)
- **Data Sheet Reference**:
  - USB Cable Specifications: Standard USB 2.0 Cable Details

7. Breadboard and Jumper Wires

- **Description**: Prototyping tools used to build and test circuits without soldering.
- **Key Specifications**:
  - **Breadboard**: Standard solderless breadboard
  - **Jumper Wires**: Male-to-male, male-to-female, or female-to-female connectors
- **Data Sheet Reference**:
  - Breadboard Specifications: Standard Breadboard Info
  - Jumper Wires: Standard wire specifications

Note: The data sheets provide essential information such as pin configurations, electrical characteristics, timing diagrams, and application notes. It is recommended to consult the data sheets to ensure correct and safe usage of each component.

Accessing Data Sheets:

- **Arduino Uno**:
  - Official Arduino Documentation: Arduino Uno Rev3
  - ATmega328P Microcontroller Data Sheet: Microchip ATmega328P
- **OLED Display (SSD1306)**:
  - Manufacturer's Data Sheet: SSD1306 OLED Driver
  - Module Details: Adafruit 128x64 OLED
- **Joystick Module**:
  - General Module Information: Joystick Module
- **Push Buttons and Buzzer**:
  - Refer to the specific models used in the project. Data sheets are typically available from the component manufacturers or suppliers.

Important: Always verify the electrical ratings and pinouts from the data sheets before connecting components to prevent damage to the components or the Arduino board.

Result Diagram