**LCD Game Using Arduino**

**Components Needed for LCD Approach:**

1. Arduino Uno
2. 16x2 LCD Display
3. Resistor – 220Ω
4. Potentiometer – 10kΩ
5. Tactile Switch/Button
6. Jumper Wires

**Components Needed for OLED Approach:**

1. Arduino Uno
2. OLED Display
3. Three Resistors – 10kΩ
4. Potentiometer – 10kΩ
5. Three Tactile Switch/Button
6. Jumper Wires

**Software Used for both approaches:**

* Arduino software

**Library Used for LCD Approach:**

* LiquidCrystal Library for the LCD.
* Bounce2 Library for digital debouncing.

**Library Used for OLED Approach:**

* Bounce2.h Library for the digital debouncing.
* SPI.h Library for Serial Peripheral Interfacing
* Wire.h Library for I2C communication
* Adafruit\_GFX.h Library for OLED Display
* Adafruit\_SSD1306.h Library for OLED Display

**Workshop Skill Break-up:** 45% hardware, 55% software

**Workshop Overview**

The objective of this workshop is to build a game on different types of displays for the Arduino using both regular wiring and I2C communication approaches.

The Arduino is a low cost, open-source microcontroller board using the ATMega328 with 32KB of Flash memory storage.

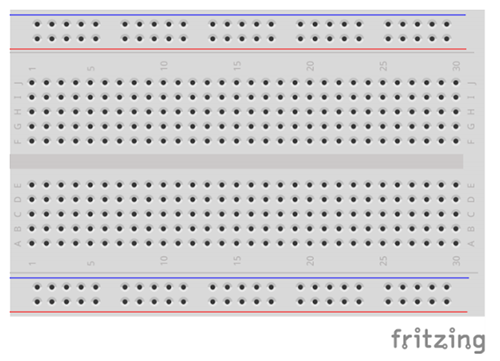
By the end of this workshop, we should be able to set-up a game with Artificial Intelligence to play against with different levels of difficulty.

**Workshop Steps for the LCD Approach**

1. Circuit Connections:

Before we get started, we need to be familiar with the wiring connections of the breadboard and the type of rails we have.

The Blue and Red Rails are connected vertically and are usually used for power. Red is the Voltage Supply (Vcc) and the Blue is usually used for Ground (GND). In the diagram you can see how each rail is connected in green.



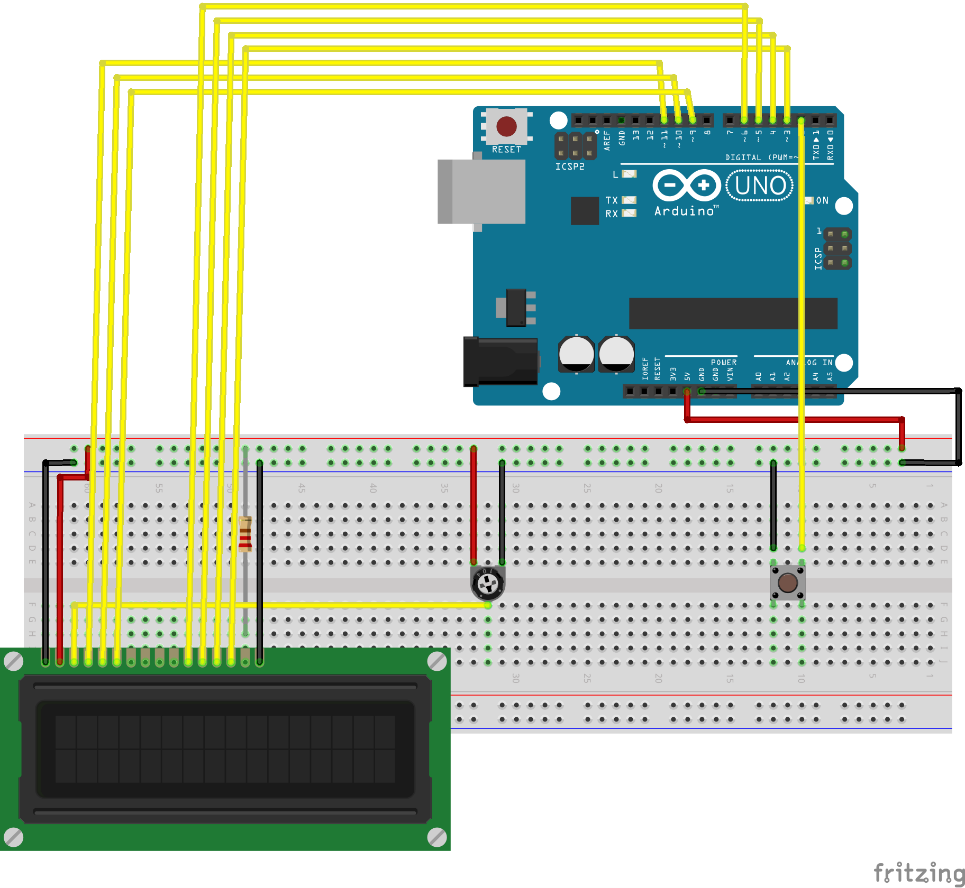
1. LCD Wiring:

The LCD has 16 Pins that are used for connections. The Pins in order from 1 to 16 are:

1. VSS/GND – Ground Pin
2. VDD – 5V Pin
3. Vo – Display Contrast Pin
4. RS – Register Select
5. R/W – Read/Write
6. E – Enable
7. D0 – Data Pin
8. D1 – Data Pin
9. D2 – Data Pin
10. D3 – Data Pin
11. D4 – Data Pin
12. D5 – Data Pin
13. D6 – Data Pin
14. D7 – Data Pin
15. A – Anode/Positive Power
16. K – Cathode/GND

Follow the steps below and refer to the diagram to make the connections. The numbers start from the left and go to the right.

* Connect VSS to GND rail.
* Connect VDD to 5V rail.
* Connect Vo to the middle pin of the potentiometer.
* Connect the other ends of the potentiometer to GND and VCC rails as in the diagram.
* Connect RS Pin to Pin 11 on the Arduino.
* Connect R/W Pin to Pin 10 on the Arduino.
* Connect E Pin to Pin 9 on the Arduino.
* Data Pins D0-D3 are unused.
* Connect D4 Pin to Pin 6 on the Arduino.
* Connect D5 Pin to Pin 5 on the Arduino.
* Connect D6 Pin to Pin 4 on the Arduino.
* Connect D7 Pin to Pin 3 on the Arduino.
* Connect the Anode to 5V rail through a 220Ω resistor.
* Connect the Cathode to the GND rail.
* Place the button in the middle and connect the top right leg to pin 2 on the Arduino.
* Connect the top left leg of the button to GND.
* Connect 5V from the Arduino to the 5V rail.
* Connect the GND Pin from the Arduino to the GND rail on the breadboard.



1. Coding the Jump Game:

Begin by including the necessary library for the LCD:

* #include <LiquidCrystal.h>

Next, we will include the pins for the LCD R/W and button:

* #define PIN\_BUTTON 2
* #define PIN\_AUTOPLAY 1
* #define PIN\_READWRITE 10

Now we define the Sprite character:

* #define SPRITE\_RUN1 1
* #define SPRITE\_RUN2 2
* #define SPRITE\_JUMP 3
* #define SPRITE\_JUMP\_UPPER '.'         // Use the '.' character for the head
* #define SPRITE\_JUMP\_LOWER 4
* #define SPRITE\_TERRAIN\_EMPTY ' '      // User the ' ' character
* #define SPRITE\_TERRAIN\_SOLID 5
* #define SPRITE\_TERRAIN\_SOLID\_RIGHT 6
* #define SPRITE\_TERRAIN\_SOLID\_LEFT 7

Next, we define horizontal position of the character and the terrain properties:

* #define HERO\_HORIZONTAL\_POSITION 1    // Horizontal position of hero on screen
* #define TERRAIN\_WIDTH 16
* #define TERRAIN\_EMPTY 0
* #define TERRAIN\_LOWER\_BLOCK 1
* #define TERRAIN\_UPPER\_BLOCK 2

Now we define the character positions:

* #define HERO\_POSITION\_OFF 0          // Hero is invisible
* #define HERO\_POSITION\_RUN\_LOWER\_1 1  // Hero is running on lower row (pose 1)
* #define HERO\_POSITION\_RUN\_LOWER\_2 2  //                              (pose 2)
* #define HERO\_POSITION\_JUMP\_1 3       // Starting a jump
* #define HERO\_POSITION\_JUMP\_2 4       // Half-way up
* #define HERO\_POSITION\_JUMP\_3 5       // Jump is on upper row
* #define HERO\_POSITION\_JUMP\_4 6       // Jump is on upper row
* #define HERO\_POSITION\_JUMP\_5 7       // Jump is on upper row
* #define HERO\_POSITION\_JUMP\_6 8       // Jump is on upper row
* #define HERO\_POSITION\_JUMP\_7 9       // Half-way down
* #define HERO\_POSITION\_JUMP\_8 10      // About to land
* #define HERO\_POSITION\_RUN\_UPPER\_1 11 // Hero is running on upper row (pose 1)
* #define HERO\_POSITION\_RUN\_UPPER\_2 12 //                              (pose 2)

Then we define the pins for the LCD connections:

* LiquidCrystal lcd(11, 9, 6, 5, 4, 3);

Then we define the upper terrain and lower terrain and define the button as a 0 in the beginning, so the game doesn’t start:

* **static** **char** terrainUpper[TERRAIN\_WIDTH + 1];
* **static** **char** terrainLower[TERRAIN\_WIDTH + 1];
* **static** **bool** buttonPushed = **false**;

In the initializeGraphics() function, we will initialize the graphics that will be used for the game:

* **void** initializeGraphics(){
* **static** byte graphics[] = {
* // Run position 1
* B01100,
* B01100,
* B00000,
* B01110,
* B11100,
* B01100,
* B11010,
* B10011,
* // Run position 2
* B01100,
* B01100,
* B00000,
* B01100,
* B01100,
* B01100,
* B01100,
* B01110,
* // Jump
* B01100,
* B01100,
* B00000,
* B11110,
* B01101,
* B11111,
* B10000,
* B00000,
* // Jump lower
* B11110,
* B01101,
* B11111,
* B10000,
* B00000,
* B00000,
* B00000,
* B00000,
* // Ground
* B11111,
* B11111,
* B11111,
* B11111,
* B11111,
* B11111,
* B11111,
* B11111,
* // Ground right
* B00011,
* B00011,
* B00011,
* B00011,
* B00011,
* B00011,
* B00011,
* B00011,
* // Ground left
* B11000,
* B11000,
* B11000,
* B11000,
* B11000,
* B11000,
* B11000,
* B11000,
* };
* **int** i;
* // Skip using character 0, this allows lcd.print() to be used to
* // quickly draw multiple characters
* **for** (i = 0; i < 7; ++i) {
* lcd.createChar(i + 1, &graphics[i \* 8]);
* }
* **for** (i = 0; i < TERRAIN\_WIDTH; ++i) {
* terrainUpper[i] = SPRITE\_TERRAIN\_EMPTY;
* terrainLower[i] = SPRITE\_TERRAIN\_EMPTY;
* }
* }

In the advanceTerrain() function, we slide the terrain to the left in half-character increments to match the pace of the character:

* **void** advanceTerrain(**char**\* terrain, byte newTerrain){
* **for** (**int** i = 0; i < TERRAIN\_WIDTH; ++i) {
* **char** current = terrain[i];
* **char** next = (i == TERRAIN\_WIDTH-1) ? newTerrain : terrain[i+1];
* **switch** (current){
* **case** SPRITE\_TERRAIN\_EMPTY:
* terrain[i] = (next == SPRITE\_TERRAIN\_SOLID) ? SPRITE\_TERRAIN\_SOLID\_RIGHT : SPRITE\_TERRAIN\_EMPTY;
* **break**;
* **case** SPRITE\_TERRAIN\_SOLID:
* terrain[i] = (next == SPRITE\_TERRAIN\_EMPTY) ? SPRITE\_TERRAIN\_SOLID\_LEFT : SPRITE\_TERRAIN\_SOLID;
* **break**;
* **case** SPRITE\_TERRAIN\_SOLID\_RIGHT:
* terrain[i] = SPRITE\_TERRAIN\_SOLID;
* **break**;
* **case** SPRITE\_TERRAIN\_SOLID\_LEFT:
* terrain[i] = SPRITE\_TERRAIN\_EMPTY;
* **break**;
* }
* }
* }

In the function drawHero(), we do as the name suggests and draw the character by using the inputs from the terrains:

* **bool** drawHero(byte position, **char**\* terrainUpper, **char**\* terrainLower, unsigned **int** score) {
* **bool** collide = **false**;
* **char** upperSave = terrainUpper[HERO\_HORIZONTAL\_POSITION];
* **char** lowerSave = terrainLower[HERO\_HORIZONTAL\_POSITION];
* byte upper, lower;
* **switch** (position) {
* **case** HERO\_POSITION\_OFF:
* upper = lower = SPRITE\_TERRAIN\_EMPTY;
* **break**;
* **case** HERO\_POSITION\_RUN\_LOWER\_1:
* upper = SPRITE\_TERRAIN\_EMPTY;
* lower = SPRITE\_RUN1;
* **break**;
* **case** HERO\_POSITION\_RUN\_LOWER\_2:
* upper = SPRITE\_TERRAIN\_EMPTY;
* lower = SPRITE\_RUN2;
* **break**;
* **case** HERO\_POSITION\_JUMP\_1:
* **case** HERO\_POSITION\_JUMP\_8:
* upper = SPRITE\_TERRAIN\_EMPTY;
* lower = SPRITE\_JUMP;
* **break**;
* **case** HERO\_POSITION\_JUMP\_2:
* **case** HERO\_POSITION\_JUMP\_7:
* upper = SPRITE\_JUMP\_UPPER;
* lower = SPRITE\_JUMP\_LOWER;
* **break**;
* **case** HERO\_POSITION\_JUMP\_3:
* **case** HERO\_POSITION\_JUMP\_4:
* **case** HERO\_POSITION\_JUMP\_5:
* **case** HERO\_POSITION\_JUMP\_6:
* upper = SPRITE\_JUMP;
* lower = SPRITE\_TERRAIN\_EMPTY;
* **break**;
* **case** HERO\_POSITION\_RUN\_UPPER\_1:
* upper = SPRITE\_RUN1;
* lower = SPRITE\_TERRAIN\_EMPTY;
* **break**;
* **case** HERO\_POSITION\_RUN\_UPPER\_2:
* upper = SPRITE\_RUN2;
* lower = SPRITE\_TERRAIN\_EMPTY;
* **break**;
* }
* **if** (upper != ' ') {
* terrainUpper[HERO\_HORIZONTAL\_POSITION] = upper;
* collide = (upperSave == SPRITE\_TERRAIN\_EMPTY) ? **false** : **true**;
* }
* **if** (lower != ' ') {
* terrainLower[HERO\_HORIZONTAL\_POSITION] = lower;
* collide |= (lowerSave == SPRITE\_TERRAIN\_EMPTY) ? **false** : **true**;
* }
* byte digits = (score > 9999) ? 5 : (score > 999) ? 4 : (score > 99) ? 3 : (score > 9) ? 2 : 1;
* // Draw the scene
* terrainUpper[TERRAIN\_WIDTH] = '\0';
* terrainLower[TERRAIN\_WIDTH] = '\0';
* **char** temp = terrainUpper[16-digits];
* terrainUpper[16-digits] = '\0';
* lcd.setCursor(0,0);
* lcd.print(terrainUpper);
* terrainUpper[16-digits] = temp;
* lcd.setCursor(0,1);
* lcd.print(terrainLower);
* lcd.setCursor(16 - digits,0);
* lcd.print(score);
* terrainUpper[HERO\_HORIZONTAL\_POSITION] = upperSave;
* terrainLower[HERO\_HORIZONTAL\_POSITION] = lowerSave;
* **return** collide;
* }

Next, we handle the button inputs as interrupts which removes false triggering:

* **void** buttonPush() {
* buttonPushed = **true**;
* }

In the setup function, we define the pins are outputs or inputs based on what they are and begin the LCD. Also, the button interrupt is defined to interrupt when the button is falling from its cycle:

* **void** setup(){
* pinMode(PIN\_READWRITE, OUTPUT);
* digitalWrite(PIN\_READWRITE, LOW);
* pinMode(PIN\_CONTRAST, OUTPUT);
* digitalWrite(PIN\_CONTRAST, LOW);
* pinMode(PIN\_BUTTON, INPUT);
* digitalWrite(PIN\_BUTTON, HIGH);
* pinMode(PIN\_AUTOPLAY, OUTPUT);
* digitalWrite(PIN\_AUTOPLAY, HIGH);
* // Digital pin 2 maps to interrupt 0
* attachInterrupt(0/\*PIN\_BUTTON\*/, buttonPush, FALLING);
* initializeGraphics();
* lcd.begin(16, 2);
* }

In the loop function, we begin by defining the hero positions and new terrain types:

* **static** byte heroPos = HERO\_POSITION\_RUN\_LOWER\_1;
* **static** byte newTerrainType = TERRAIN\_EMPTY;
* **static** byte newTerrainDuration = 1;
* **static** **bool** playing = **false**;
* **static** **bool** blink = **false**;
* **static** unsigned **int** distance = 0;

If the game is not being played, the hero will blink in its position until the button starts the game:

* **if** (!playing) {
* drawHero((blink) ? HERO\_POSITION\_OFF : heroPos, terrainUpper, terrainLower, distance >> 3);
* **if** (blink) {
* lcd.setCursor(0,0);
* lcd.print("Press Start");
* }
* delay(250);
* blink = !blink;
* **if** (buttonPushed) {
* initializeGraphics();
* heroPos = HERO\_POSITION\_RUN\_LOWER\_1;
* playing = **true**;
* buttonPushed = **false**;
* distance = 0;
* }
* **return**;
* }

Now we shift the terrain to the left or make a new terrain to the right:

* // Shift the terrain to the left
* advanceTerrain(terrainLower, newTerrainType == TERRAIN\_LOWER\_BLOCK ? SPRITE\_TERRAIN\_SOLID : SPRITE\_TERRAIN\_EMPTY);
* advanceTerrain(terrainUpper, newTerrainType == TERRAIN\_UPPER\_BLOCK ? SPRITE\_TERRAIN\_SOLID : SPRITE\_TERRAIN\_EMPTY);
* // Make new terrain to enter on the right
* **if** (--newTerrainDuration == 0) {
* **if** (newTerrainType == TERRAIN\_EMPTY) {
* newTerrainType = (random(3) == 0) ? TERRAIN\_UPPER\_BLOCK : TERRAIN\_LOWER\_BLOCK;
* newTerrainDuration = 2 + random(10);
* } **else** {
* newTerrainType = TERRAIN\_EMPTY;
* newTerrainDuration = 10 + random(10);
* }
* }

If the button is pushed:

* **if** (buttonPushed) {
* **if** (heroPos <= HERO\_POSITION\_RUN\_LOWER\_2) heroPos = HERO\_POSITION\_JUMP\_1;
* buttonPushed = **false**;
* }

If the character collides with an object:

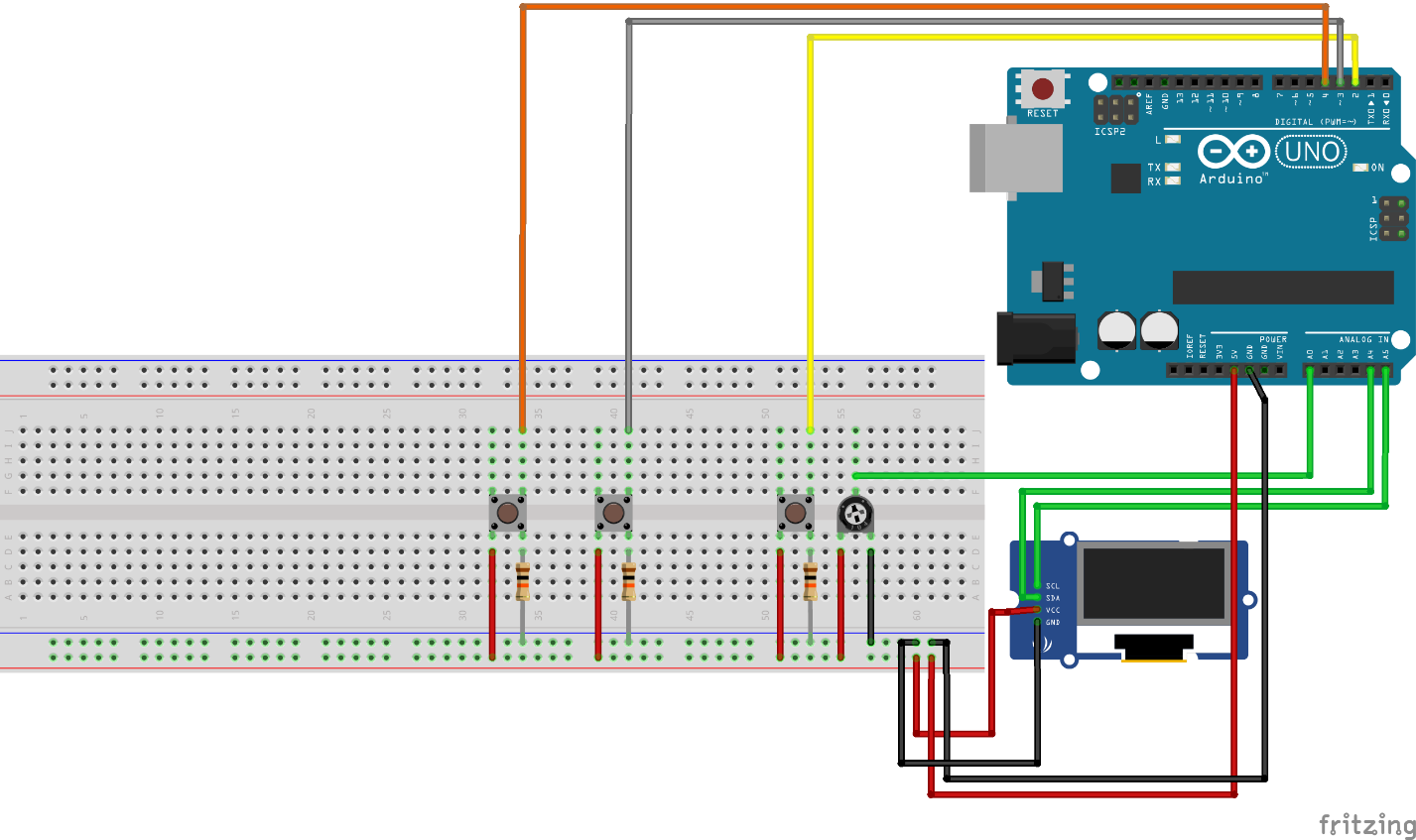
* **if** (drawHero(heroPos, terrainUpper, terrainLower, distance >> 3)) {
* playing = **false**; // The hero collided with something. Too bad.
* } **else** {
* **if** (heroPos == HERO\_POSITION\_RUN\_LOWER\_2 || heroPos == HERO\_POSITION\_JUMP\_8) {
* heroPos = HERO\_POSITION\_RUN\_LOWER\_1;
* } **else** **if** ((heroPos >= HERO\_POSITION\_JUMP\_3 && heroPos <= HERO\_POSITION\_JUMP\_5) && terrainLower[HERO\_HORIZONTAL\_POSITION] != SPRITE\_TERRAIN\_EMPTY) {
* heroPos = HERO\_POSITION\_RUN\_UPPER\_1;
* } **else** **if** (heroPos >= HERO\_POSITION\_RUN\_UPPER\_1 && terrainLower[HERO\_HORIZONTAL\_POSITION] == SPRITE\_TERRAIN\_EMPTY) {
* heroPos = HERO\_POSITION\_JUMP\_5;
* } **else** **if** (heroPos == HERO\_POSITION\_RUN\_UPPER\_2) {
* heroPos = HERO\_POSITION\_RUN\_UPPER\_1;
* } **else** {
* ++heroPos;
* }
* ++distance;
* digitalWrite(PIN\_AUTOPLAY, terrainLower[HERO\_HORIZONTAL\_POSITION + 2] == SPRITE\_TERRAIN\_EMPTY ? HIGH : LOW);
* }

**Workshop Steps for the OLED Approach**

1. Circuit Wiring:

The OLED has 4 Pins, two pins are for VCC and GND to power the display and the other two are SCL and SDA. The SCL is for clocking and SDA is for data line used in I2C communications. The pins that SCL and SDA are connected to are A5 and A4 respectively but for the Arduino Uno and Nano but can vary for other models of the Arduino. Follow the steps below and refer to the diagram for the connections:

* Place the OLED Display on the breadboard horizontally.
* Connect the Vcc pin to the 5V rail.
* Connect the GND pin to the GND rail.
* Connect the SCL pin to A5 Pin on the Arduino Uno.
* Connect the SDA pin to A4 Pin on the Arduino Uno.
* Place the potentiometer and connect the middle pin to A0 Pin on the Arduino.
* Connect the other two legs of the potentiometer to GND and VCC respectively.
* Place three button on the breadboard.
* Connect the big button’s top right pin to D2 Pin on the Arduino.
* Connect the bottom right pin to GND rail through a 10k resistor
* Connect the bottom left pin to 5V rail.
* Connect the smaller buttons top right pins to D3 and D4 respectively.
* Connect the smaller buttons bottom pins the same way as the big buttons.
* Connect the 5V and GND pins from the Arduino to the 5V and GND rails of the breadboard

****

1. Coding the OLED game (Pong)

Begin by including the necessary libraries:

* #include <SPI.h>  // Including Necessary libraries for the OLED Display
* #include <Wire.h>
* #include <Adafruit\_GFX.h>
* #include <Adafruit\_SSD1306.h>
* #include <Bounce2.h>

Next we initialize the display using this line:

* Adafruit\_SSD1306 display(128, 64, &Wire, 4); //Initializing the Display

Now we include the setup variables to run the game:

* + **int** resolution[2] = {128, 64}, ball[2] = {20, (resolution[1] / 2)}; // Screen Resolution
  + **const** **int** PIXEL\_SIZE = 8, WALL\_WIDTH = 4, PADDLE\_WIDTH = 4, BALL\_SIZE = 4;
  + **int** SPEED = 3; // These are constants that help the compiler run faster
  + **int** playerScore = 0, aiScore = 0, playerPos = 0, aiPos = 0; // This stores the positions of the player and AI
  + **char** ballDirectionHori = 'R', ballDirectionVerti = 'S'; // This controls the ball direction
  + boolean inProgress = **true**; // To begin the game

To make the game harder, the SPEED variable can be altered to make the ball move faster. If this is done, the AI would need to be altered to be able to cope with the new speed.

Next we define the button we want to debounce and their variables:

* Bounce debouncer = Bounce(); // Digitally debouncing buttons to remove false triggering
* Bounce easy = Bounce();
* Bounce hard = Bounce();
* **const** **int** buttonPin = 2; // Defining the button input pins
* **const** **int** easyPin = 4;
* **const** **int** hardPin = 3;
* **int** buttonCount = 0; // Variables used for button counting
* **int** even = 0;
* **int** easymode = 0;
* **int** hardmode = 0;

In the setup, we begin the display and define the buttons as inputs:

* **void** setup()   {
* display.begin(SSD1306\_SWITCHCAPVCC, 0x3C); // Begin the display
* display.display(); // Display what's on the buffer
* display.clearDisplay(); // Clear the buffer to begin the game
* pinMode(buttonPin, INPUT); // Define the button pins as inputs to the system
* pinMode(easyPin, INPUT);
* pinMode(hardPin, INPUT);
* debouncer.attach(buttonPin, INPUT\_PULLUP); // Debouncing function by the library
* easy.attach(easyPin, INPUT\_PULLUP);
* hard.attach(hardPin, INPUT\_PULLUP);
* debouncer.interval(25); // Stable interval of 25ms
* easy.interval(5);
* hard.interval(5);
* Serial.begin(9600); // Start the Serial Monitor
* }

In the loop, the main tasks occur. We begin by initializing the button library to keep checking for inputs:

* debouncer.update(); // Begin checking for the button inputs
* easy.update();
* hard.update();

Then if anything is found from the button called the debouncer (the big button on the breadboard), it should increment the counter by one each time:

* **if** (debouncer.fell()) { // If the button is falling (Transitionign from High to Low after the click)
* buttonCount++; // Increase the counter
* }

Then we use modulus to check if the number of button clicks reached is an even number. The function modulus returns a 1 if there is a remainder and a 0 if it’s an even number:

* even = buttonCount % 2; // Check if the button input is an even number

Now we will code the Welcome screen for the OLED:

* // Welcome Screen
* **if** (buttonCount == 0) { // If there is no button inputs yet
* display.display(); // Initialize Display
* display.clearDisplay(); // Clear the Display buffer
* display.setCursor(2, 0); // Position where the first character pixel will appear
* display.setTextSize(3); // Set the text size
* display.setTextColor(WHITE); // Set the text colour
* display.println("WELCOME"); // Print a message on the display
* display.setCursor(38, 25); // Move the cursor down and to the center
* display.setTextSize(2); // Different size
* display.setTextColor(WHITE); // Set colour
* display.println("CLICK     START"); // Print message
* }

The screen will stay on the welcome screen until a button is pressed, if it is pressed once, it will enter the difficulty menu page:

* **else** **if** (buttonCount == 1) { // If the button has been clicked once - Difficulty Menu
* display.display();
* display.clearDisplay();
* display.setCursor(30, 0);
* display.setTextSize(2);
* display.setTextColor(WHITE);
* display.println("CHOOSE");
* display.setCursor(6, 20);
* display.println("DIFFICULTY");
* display.setCursor(6, 40);
* display.println("EASY  HARD");
* **if** (easy.fell()) {
* easymode++; // Increment easy mode button to trigger the game to start
* buttonCount++; // Increment counter to go back to an even number
* }
* **if** (hard.fell()) {
* hardmode++; // Increment easy mode button to trigger the game to start
* buttonCount++; // Increment counter to go back to an even number
* }
* }

Now, if the left button is clicked, the game will enter the easy mode, inside easy mode is where the game functions. The code below shows how the game enters easy mode and the initial state when the number is even.

* **if** (easymode >= 1) { // If the easy mode button is clicked from the difficulty menu, this will start easy mode
* **if** (even == 0) {
* display.clearDisplay();
* **if** (aiScore > 4 || playerScore > 4) { // Sets the maximum score to be reached (in this case it's first to 5)
* // check game state
* inProgress = **false**; // Ends the game
* }

Next the main game occurs in the function called inProgress which controls the ball. The code is commented through explaining each line of code and how the ball moves and bounces:

* **if** (inProgress) {
* eraseScore(); // Clear the previous score
* eraseBall(ball[0], ball[1]); // Initialize the balls first position
* **if** (ballDirectionVerti == 'U') {
* // Move ball up diagonally
* ball[1] = ball[1] - SPEED;
* }
* **if** (ballDirectionVerti == 'D') {
* // Move ball down diagonally
* ball[1] = ball[1] + SPEED;
* }
* **if** (ball[1] <= 0) {
* // Bounce the ball off the top
* ballDirectionVerti = 'D';
* }
* **if** (ball[1] >= resolution[1]) {
* // Bounce the ball off the bottom
* ballDirectionVerti = 'U';
* }
* **if** (ballDirectionHori == 'R') {
* ball[0] = ball[0] + SPEED; // Move ball
* **if** (ball[0] >= (resolution[0] - 6)) {
* // Ball is at the AI edge of the screen
* **if** ((aiPos + 12) >= ball[1] && (aiPos - 12) <= ball[1]) { // Ball hits AI paddle
* **if** (ball[1] > (aiPos + 4)) {
* // Deflect ball down
* ballDirectionVerti = 'D';
* }
* **else** **if** (ball[1] < (aiPos - 4)) {
* // Deflect ball up
* ballDirectionVerti = 'U';
* }
* **else** {
* // Deflect ball straight
* ballDirectionVerti = 'S';
* }
* // Change ball direction
* ballDirectionHori = 'L';
* }
* **else** {
* // GOAL!
* ball[0] = 6; // Move ball to other side of screen
* ballDirectionVerti = 'S'; // Reset ball to straight travel
* ball[1] = resolution[1] / 2; // Move ball to middle of screen
* ++playerScore; // Increase player score
* }
* }
* }
* **if** (ballDirectionHori == 'L') {
* ball[0] = ball[0] - SPEED; // Move ball
* **if** (ball[0] <= 6) {
* // Ball is at the player edge of the screen
* **if** ((playerPos + 12) >= ball[1] && (playerPos - 12) <= ball[1]) {
* // Ball hits player paddle
* **if** (ball[1] > (playerPos + 4)) {
* // Deflect ball down
* ballDirectionVerti = 'D';
* }
* **else** **if** (ball[1] < (playerPos - 4)) {
* // Deflect ball up
* ballDirectionVerti = 'U';
* }
* **else** {
* // Deflect ball straight
* ballDirectionVerti = 'S';
* }
* ballDirectionHori = 'R';
* }
* **else** {
* ball[0] = resolution[0] - 6; // Move ball to other side of screen
* ballDirectionVerti = 'S'; // Reset ball to straight travel
* ball[1] = resolution[1] / 2; // Move ball to middle of screen
* ++aiScore; // Increase AI score
* }
* }
* }
* drawBall(ball[0], ball[1]);
* erasePlayerPaddle(playerPos);
* playerPos = analogRead(A0); // Read player potentiometer (Player's control stick)
* playerPos = map(playerPos, 0, 1023, 8, 54); // Convert value from 0 - 1023 to 8 - 54
* drawPlayerPaddle(playerPos); // Move the paddle to the player's chosen position
* moveAi(); // Run the AI function to play
* drawNet(); // Draw the middle line
* drawScore(); // Update the Score
* }

The variable playerPos is the input of the player. The function moveAi() is what controls the AI bot.

The only difference between the easy and hard mode falls in the moveAi() function which will be explained in a bit.

If the game is not running in the function inProgress, it means that someone has either won or lost:

* **else** {
* // Somebody has won
* display.clearDisplay();
* display.setTextSize(3);
* display.setTextColor(WHITE);
* display.setCursor(0, 0);
* **if** (aiScore > playerScore) {    // Figure out who
* display.println("YOU    LOSE!");
* display.setTextSize(1);
* display.setTextColor(WHITE);
* display.setCursor(0, 50);
* display.println("Reset to replay");
* }
* **else** **if** (playerScore > aiScore) {
* display.println("YOU    WIN!");
* display.setTextSize(1);
* display.setTextColor(WHITE);
* display.setCursor(0, 50);
* display.println("Press Reset to replay");
* }
* }
* }
* **else** {
* display.clearDisplay();
* display.setCursor(12, 20);
* display.setTextSize(3);
* display.setTextColor(WHITE);
* display.println("PAUSED");
* }
* }

The last else statement is the one that check if the button input is an odd number and if it is, the game pauses until the button is clicked again and the game resumes.

The moveAi() function controls the AI bot, the way it does this is by moving the paddle position either up or down by 1 above the ball.

* **void** moveAi() {
* // Move the AI paddle
* eraseAiPaddle(aiPos);
* **if** (ball[1] > aiPos) {
* ++aiPos;
* //aiPos += 2;
* }
* **else** **if** (ball[1] < aiPos) {
* --aiPos;
* //aiPos -= 2;
* }
* drawAiPaddle(aiPos);
* }

The commented aiPos is for the hard mode. The difference between easy and hard is how fast the ai reacts to the ball movement and usually hits more angles since it moves up and down by 2 blocks instead of 1.

The functions drawScore() and eraseScore() write and update the game score respectively:

* **void** drawScore() {
* // Draw AI and player scores
* display.setTextSize(2);
* display.setTextColor(WHITE);
* display.setCursor(45, 0);
* display.println(playerScore);
* display.setCursor(75, 0);
* display.println(aiScore);
* }
* **void** eraseScore() {
* // Erase AI and player scores
* display.setTextSize(2);
* display.setTextColor(BLACK);
* display.setCursor(45, 0);
* display.println(playerScore);
* display.setCursor(75, 0);
* display.println(aiScore);
* }

The rest of the functions are just related to drawing and updating the paddles and the net:

* **void** drawNet() {
* **for** (**int** i = 0; i < (resolution[1] / WALL\_WIDTH); ++i) {
* drawPixel(((resolution[0] / 2) - 1), i \* (WALL\_WIDTH) + (WALL\_WIDTH \* i), WALL\_WIDTH);
* }
* }
* **void** drawPixel(**int** posX, **int** posY, **int** dimensions) {
* // draw group of pixels
* **for** (**int** x = 0; x < dimensions; ++x) {
* **for** (**int** y = 0; y < dimensions; ++y) {
* display.drawPixel((posX + x), (posY + y), WHITE);
* }
* }
* }
* **void** erasePixel(**int** posX, **int** posY, **int** dimensions) {
* // erase group of pixels
* **for** (**int** x = 0; x < dimensions; ++x) {
* **for** (**int** y = 0; y < dimensions; ++y) {
* display.drawPixel((posX + x), (posY + y), BLACK);
* }
* }
* }
* **void** erasePlayerPaddle(**int** row) {
* erasePixel(0, row - (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* erasePixel(0, row - PADDLE\_WIDTH, PADDLE\_WIDTH);
* erasePixel(0, row, PADDLE\_WIDTH);
* erasePixel(0, row + PADDLE\_WIDTH, PADDLE\_WIDTH);
* erasePixel(0, row + (PADDLE\_WIDTH + 2), PADDLE\_WIDTH);
* }
* **void** drawPlayerPaddle(**int** row) {
* drawPixel(0, row - (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* drawPixel(0, row - PADDLE\_WIDTH, PADDLE\_WIDTH);
* drawPixel(0, row, PADDLE\_WIDTH);
* drawPixel(0, row + PADDLE\_WIDTH, PADDLE\_WIDTH);
* drawPixel(0, row + (PADDLE\_WIDTH + 2), PADDLE\_WIDTH);
* }
* **void** drawAiPaddle(**int** row) {
* **int** column = resolution[0] - PADDLE\_WIDTH;
* drawPixel(column, row - (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* drawPixel(column, row - PADDLE\_WIDTH, PADDLE\_WIDTH);
* drawPixel(column, row, PADDLE\_WIDTH);
  + drawPixel(column, row + PADDLE\_WIDTH, PADDLE\_WIDTH);
* drawPixel(column, row + (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* }
* **void** eraseAiPaddle(**int** row) {
* **int** column = resolution[0] - PADDLE\_WIDTH;
* erasePixel(column, row - (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* erasePixel(column, row - PADDLE\_WIDTH, PADDLE\_WIDTH);
* erasePixel(column, row, PADDLE\_WIDTH);
* erasePixel(column, row + PADDLE\_WIDTH, PADDLE\_WIDTH);
* erasePixel(column, row + (PADDLE\_WIDTH \* 2), PADDLE\_WIDTH);
* }
* **void** drawBall(**int** x, **int** y) {
* display.drawCircle(x, y, BALL\_SIZE, WHITE);
* }
* **void** eraseBall(**int** x, **int** y) {
* display.drawCircle(x, y, BALL\_SIZE, BLACK);
* }

**References and Tutorials**

LCD Jump Game:

* <https://www.instructables.com/id/Arduino-LCD-Game/>

OLED Pong Game:

* <https://www.makeuseof.com/tag/arduino-retro-gaming-oled-display/>