Max Ficco & Turner Piercy 28 April 2025

EE 10200, Spring 2025

# Final Project Report

## Executive Summary:

Our project recreates a simplified "Bop It" game using an Arduino Uno, an LCD screen, a button, a potentiometer, a tilt sensor, and a piezo buzzer. Players must quickly respond to random commands—"Bop it" (press the button), "Twist it" (turn the knob), or "Tilt it" (tilt the board)—before time runs out. Correct actions score points and reset the timer, while mistakes or delays end the game. Each action has a unique success sound, and a distinct game-over sound plays when the player fails. The game combines quick reflexes and sound feedback for a fun and interactive experience.

## Background:

Motivation: We designed this project for the sake of entertainment, distilling parts from the classic *Bop It* game into a single breadboard of Arduino components. While limited in size, power, and actuator quality, the game is still entertaining and fast-paced, challenging players' reaction time and coordination. It could be used as a fun party game, a reaction-time tester, or even a basic educational tool to introduce students to electronics. Indeed, the open-faced components of the game provide a sense of transparency and faithfulness to the rapid-fire signals that players wish to send.

Attribution: Our project is inspired by the original *Bop It* toy, a popular children's toy first released in 1996. The original game (soon followed by many varying successors) contained three main commands: "Bop it!", "Twist it!", and "Pull it!", as described on Wikipedia and the official Hasbro website. Constrained by the limited range of sounds which the Piezo Buzzer can produce, instructions are displayed visually on an LCD screen. The game timing, scoring system, and specific physical inputs (button press, potentiometer twist, and tilt sensor movement) are also uniquely tailored for this Arduino implementation. We did not directly base our project on any existing Arduino project, however, there are many projects similar in their inspiration to the original *Bop It* game. For instance, one project from a user on Autodesk Instructables outlines a "Task Giving Arduino Machine (aka: making a 'Bop-It')" in which multiple breadboards, LEDs, force sensors, and an MP3 player, as well as an SD card and many other components are used to give instructions to a player who must then complete the corresponding action. Compared to such a project, as well as the original *Bop It* game, our project is still unique and innovative in its simplicity as well as accessibility. Our game provides an easily hand-held game capable of being played by anyone who can read, regardless of hearing impairments. The time bar displayed during each command provides an intuitive solution that still gives the player a fun sense of urgency.

<u>Project Evolution:</u> Our completed project closely matches the goals outlined in our original proposal. One minor change is that we decided not to implement music, instead focusing on unique sounds for each of the physical inputs. This is primarily due to our belief that the music did not add substantially to the quality of our final design, especially given the quality of the piezo buzzer as a speaker. Allowing the tones of correct moves to be heard clearly has a positive effect on the satisfaction while playing the game, and ensures players are aware of their status to keep up with the fast pace of the game.

<u>Environmental Scan:</u> Commercially, the *Bop It* toy is widely available and sold by Hasbro, Inc. There are also various Arduino recreations online, as mentioned earlier. Our project remains distinct in using a combination of a button, potentiometer, and tilt sensor, each with unique success sounds and an LCD-based instruction system, rather than more complex sensors with voice prompts.

## Functional Description and Product Operation:

Our simplified "Bop-It" game tests a player's reaction time and agility by issuing random commands that must be completed before the 5-second countdown timer expires. Each successful action is rewarded with a unique sound and decreases the starting countdown timer by 250 milliseconds until it ultimately reaches a minimum of 2 seconds, thereby increasing the difficulty of the game as it progresses.

<u>Instructions To Play:</u>

1. When the device is powered on, the LCD displays a welcome message prompting the player to "Press button to start Bop It!"
2. After the player presses the button to begin the game, a random command ("Bop it," "Twist it," or "Tilt it") is displayed on the LCD screen.
3. The player must quickly perform the corresponding action:
    a. Bop It: Press the button on the right side of the breadboard.

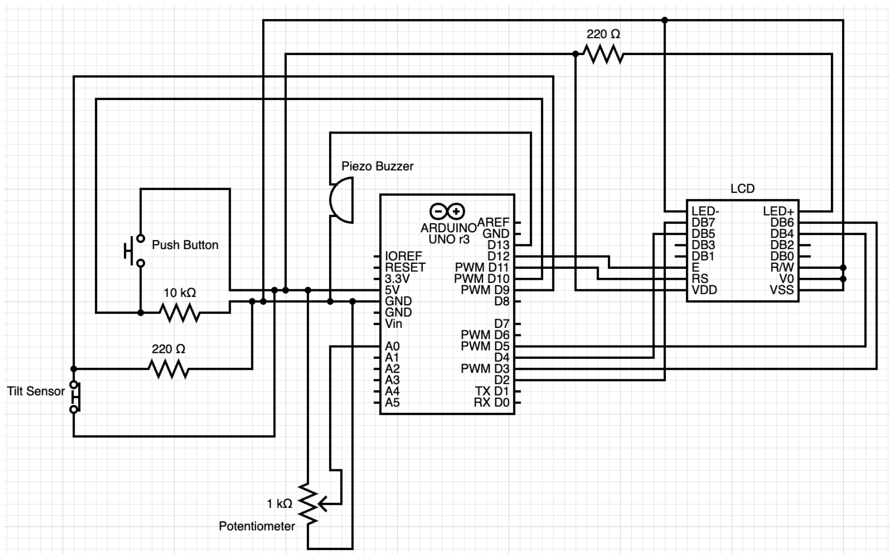
   b. Twist It: Twist the potentiometer on the left side of the breadboard.
   c. Tilt It: Rotate the board to trigger a tilt reading.
4. A countdown timer runs for each command. If the player performs the correct action in time, a unique success sound plays, and a new random command is displayed.
5. If the player fails to respond in time, a "Game Over!" screen appears with their score. Every 1.5 seconds, the display cycles between this screen, the highest score since power-on/reset, and a "Press button to restart!" prompt.
6. The player may then press the button at any time, beginning a new game.

**Hardware:**

Sensors and Actuators Used: **Push Button (input):** Signals 1 when pushed, 0 when off ("Bop it!"). **Potentiometer (input):** Changes voltage from 0-5V, detecting changes >2V ("Twist it!"). **Tilt Sensor (input):** Signals 1 when not tilted, 0 when tilted ("Tilt it!"). **Piezo Buzzer (output):** Plays sequences of tones for each input, as well as for game over.

**Liquid-Crystal Display (output):** Displays startup instructions, game commands, visual timer, score, and high score

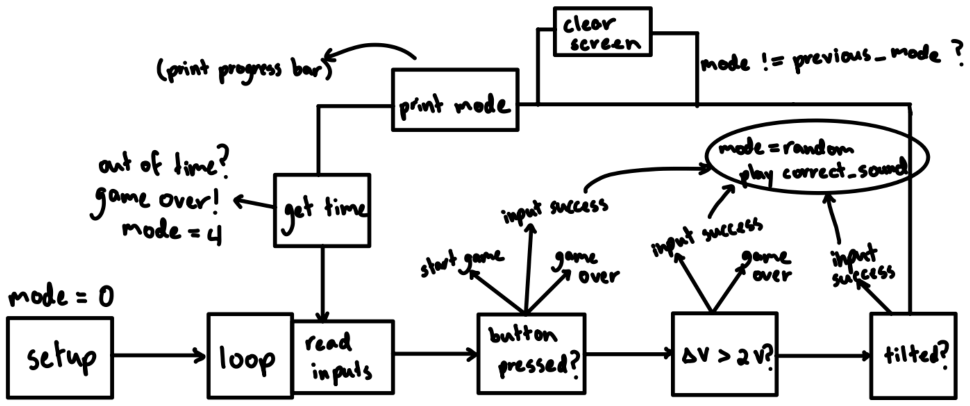
Circuit Diagram and Description:



Here, as listed above, the Arduino microcontroller, three sensors, as well as the Piezo Buzzer and LCD are visualized. The potentiometer is connected to the analog input pin 0, while the push button and tilt sensors are connected to digital input pins 10 and 9, respectively. Each also has a pull down resistor, labeled above. The piezo buzzer is connected directly from digital pin 13 as output to ground. All other sensors are connected to 5V supplied by the Arduino microcontroller, with the LCD being connected from its VDD and LED+ pins, the latter being connected with a 220Ω resistor. Pins LED-, VSS, V0, and R/W are all connected to ground. Notably, V0 is connected to ground to ensure maximum contrast on the display, and R/W is connected to ground to set the LCD to write mode. Pins DB4-DB7 are assigned to digital pins 2-5 in reverse order for convenience, and E and RS pins are connected to digital pins 12 and 11, respectively.

**<u>Software:</u>**

Our code works by first initializing a wide set of global variables to keep track of the gamemode, time, input states, and score. Most notable is the variable "mode": 0 represents the startup sequence, 1 is "Bop it", 2 is "Twist it", 3 is "Tilt it", and 4 is the game over sequence. An abstracted flow chart is provided, with functions explained further below.



**`void setup();`** sets up the Piezo Buzzer and LCD as outputs, seeds the random function. **`void loop();`** calculates the delta_time of each passing loop, and decreases the game time counter by its value. The three inputs are read using analogRead and digitalRead, with logic to keep track of changes in the potentiometer across loops. Three if statements follow, as shown above. If an input is made during its corresponding mode, input_success is called. Otherwise, game_over is called. Pressing the button starts/restarts the game. Finally, if the mode is changed, the screen is cleared. **`void correct_sound(int success_mode);`** plays a distinct success tone on the piezo buzzer corresponding to the input, success_mode, used. **`void print_progress_bar(int time, int t0);`** divides time by the total time t0 and multiplies by 16 to convert to a number of '#' characters printed to the LCD. **`void print_mode(int mode, int time, int t0, int delta_time);`** prints the mode to the screen and calls print_progress bar. When mode==4, displays score and high_score.

**`void game_over_sound();`** plays a tone alerting the user that the game is over**. `void game_over();`** sets the mode to 4, resets time, and saves to high_score if broken. Finally, **`void input_success(int success_mode);`** resets the timer and sets a new random mode.

**Project Post-Mortem:**

Changes Made: As stated above in our project evolution, our completed project closely matched the goals outlined in our original proposal. The most notable change was the decision to not implement music. By focusing on unique sounds for each of the physical inputs, we both lowered the complexity and bloat of our code and streamlined the user experience. We wanted the tones of correct moves to be heard clearly by players, and believe this had a positive effect on the satisfaction while playing the game. An additional minor change was the decision to use consistent polling over interrupts. Given the nature of the game, polling inputs every loop proved to be the best solution and allowed us to check for incorrect inputs (e.g. player pressing “Bop it” when instructed “Twist it”).

Novelty: In our opinion, the most novel or interesting part of our project is likely our use of the LCD screen. Most versions of *Bop It!*, both official and inspired, are entirely based on audio commands. The changing screen, with the timer bar displayed, allows for an intuitive game, complete with starting instructions and the ability to store and display a high score.

Challenges: The most difficult part of our project was implementing the correct logic for handling multiple user inputs simultaneously. At every moment, the program must monitor all three inputs and determine whether any action is correct or incorrect based on the current mode, and then proceed to execute the correct response. Fine-tuning the sensitivity of the twist and tilt sensors to make the game both responsive and fair was also a difficult but important part of polishing the user experience.

Future Changes: If we were to do this project again, two points of focus would be on the tilt sensor and the overall housing of the game. The tilt sensor fits loosely into the breadboard, and is incredibly sensitive—so much so that we needed to modify our game logic to ensure that players would not inadvertently lose the game at a small movement during the wrong mode. Having a more precise or tunable tilt sensor could allow the game to be expanded. Adding an external housing could also improve the aesthetic quality of the game. It would also contribute positively to the stability of the screen, which can flicker or temporarily lose connection if the player is not careful when tilting or handling the game.

**Appendix:**

Below is a complete program listing of our code, which can also be found on GitHub.

```
#include <LiquidCrystal.h>

// Pin definitions
#define BUTTON_PIN 10
#define TWIST_PIN A0
#define TILT_PIN 9
#define PIEZO_PIN 13

// Initialize lcd pins: RS, EN, D4, D5, D6, D7
LiquidCrystal lcd(11,12,5,4,3,2);

// Game variables
int mode; // 0=startup, 1=bop it, 2=twist it, 3=tilt it, 4=game
over
int previous_mode = -1; // Tracks previous mode to detect changes

// Timer variables
unsigned long prev_time = 0;
unsigned long current_time;
unsigned long delta_time;
int t0 = 5000; // Initial timer value (5 seconds)
int time = t0;

// Input states
int button = 0;
float twist = 0, twist_start = -1.0;
int tilt = 0;

// Score tracking
int score = 0;
int high_score = 0;
// Pause timer for game over animations
int pause_ms = 0;

void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(A1)); // Seed random (A1 pin is floating)
  lcd.begin(16, 2);
  lcd.clear();
  pinMode(PIEZO_PIN, OUTPUT); // Piezo buzzer as output
  mode = 0; // Start at startup mode
}


void loop() {
  // Track elapsed time (milliseconds)
  current_time = millis();
  delta_time = current_time - prev_time;
  prev_time = current_time;

  // Decrease timer in active mode
  if (mode != 0 && mode != 4) time -= delta_time;

  // If time runs out, go to game over
  if (time < 0) {
    game_over();
  }

  // Read inputs
  button = digitalRead(BUTTON_PIN); // button: 1 on, 0 off
  twist = analogRead(TWIST_PIN)*5.0/1023.0; // potentiometer:
0.0-5.0V
  if (twist_start == -1.0) {
    twist_start = twist; // Save starting twist position
  }
  tilt = digitalRead(TILT_PIN); // tilt sensor: 0 tilted, 1 flat

  // Handle button press
  if (button) {
    if (mode == 0 || mode == 4) { // starting new game
      mode = 1; // always start with "bop it"
      time = t0;
      button = 0;
      lcd.clear();
      score = 0;
    }
    if (time < t0-250) { // make sure button isn't triggered twice
within 250ms ("debouncing")
      if (mode == 1) { // successful "bop it"
        input_success(mode);
      } else { // button pressed during wrong mode
        game_over();
      }
    }
    button = 0; // clear button state
  }

  if (abs(twist - twist_start) > 2) { // Detect significant change
(potentiometer changes by >2V, or more than 2/5th of turn)
    if (mode == 0 || mode == 4) { // starting new game
      correct_sound(2); // play happy sound just for fun/testing,
will still wait for button press
    } else if (mode == 2) { // successful "twist it"
      input_success(mode);
    } else { // twisted during wrong mode
      game_over();
    }
    twist_start = -1.0; // reset twister
  }

  if (tilt==0) {
    if (mode == 0 || mode == 4) { // starting new game
      Serial.println("tilted!");
    }
    if (time < t0-500) { // make sure tilt isn't triggered twice
within 0.5 second (give player time to reset, avoid "bouncing")
      if (mode == 3) { // successful "tilt it"
        input_success(mode);
      }
      // No penalty if tilted during wrong mode (tilt sensor is too
sensitive)
    }
    tilt = 1; // Clear tilt state
  }
  // clear screen when mode changes
  if (mode != previous_mode) {
    lcd.clear();
    previous_mode = mode;
  }

  // display the game on LCD
  print_mode(mode, time, t0, delta_time);

}

// Prints the progress bar for time remaining
void print_progress_bar(int time, int t0) {
    lcd.setCursor(0,1);
    int bars = time / (float)t0 * 16;
    for (int i=0; i<bars; i++) {
      lcd.print('#');
    }
    for (int i=bars; i<16; i++) {
      lcd.print(' ');
    }
}

// Displays the current mode (gameplay and game over screens)
void print_mode(int mode, int time, int t0, int delta_time) {
  if (mode == 0) {
    lcd.setCursor(0, 0);
    lcd.print("Press button    ");
    lcd.setCursor(0, 1); // column 0, line 1
    lcd.print("to start Bop It!");
  } else if (mode == 1) {
    lcd.setCursor(0, 0);
    lcd.print("-> Bop it!      ");
    print_progress_bar(time, t0);
  } else if (mode == 2) {
    lcd.setCursor(0, 0);
    lcd.print("-> Twist it!    ");
    print_progress_bar(time, t0);
  } else if (mode == 3) {
    lcd.setCursor(0, 0);
    lcd.print("-> Tilt it!");
    print_progress_bar(time, t0);
  } else if (mode == 4) {
    if (pause_ms < 1500) {
      lcd.print("Game Over!      ");
      lcd.setCursor(0, 1); // column 0, line 1
      lcd.print("Score: ");
      lcd.print(score);
      int num_digits = (score == 0) ? 1 : (int)log10(score) + 1;
      for (int i = 0; i < 16 - (7 + num_digits); i++) {
        lcd.print(' '); // extra spaces to overwrite old content
      }
    } else if (pause_ms < 3000) {
      lcd.setCursor(0, 0);
      lcd.print("High Score:     ");
      lcd.setCursor(0, 1); // column 0, line 1
      lcd.print("       ");
      lcd.print(high_score);
      int num_digits = (score == 0) ? 1 : (int)log10(score) + 1;
      for (int i = 0; i < 16 - (7 + num_digits); i++) {
        lcd.print(' '); // extra spaces to overwrite old content
      }
    } else if (pause_ms < 4500) {
      lcd.setCursor(0, 0);
      lcd.print("Press button    ");
      lcd.setCursor(0, 1); // column 0, line 1
      lcd.print("to restart!     ");
      lcd.setCursor(0, 0);
    } else {
      pause_ms = 0; // Cycle back to beginning
    }
    pause_ms += delta_time;
  }
}

// Plays a sound for correct input
void correct_sound(int success_mode) {
  if (success_mode == 1) {
    tone(PIEZO_PIN, 880, 80);    // A5
    delay(90);
    noTone(PIEZO_PIN);
    delay(30);
    tone(PIEZO_PIN, 880, 80);    // A5
    delay(60);
    noTone(PIEZO_PIN);
  }
  else if (success_mode == 2) {
    tone(PIEZO_PIN, 1000, 50);
    delay(60);
    tone(PIEZO_PIN, 800, 50);
    delay(60);
    tone(PIEZO_PIN, 1000, 50);
    delay(60);
    noTone(PIEZO_PIN);
  }
  else if (success_mode == 3) {
    for (int freq = 400; freq <= 900; freq += 10) {
      tone(PIEZO_PIN, freq);
      delay(5);
    }
    noTone(PIEZO_PIN);
  }
}

// Plays sad sound at game over
void game_over_sound()
{
  tone(PIEZO_PIN, 220, 400); // A3, low and slow
  delay(400);
  tone(PIEZO_PIN, 196, 400); // G3, a little lower
  delay(400);
  tone(PIEZO_PIN, 174, 600); // F3, even lower and longer
  delay(600);
  noTone(PIEZO_PIN);
}
```

```
// Handles successful input
void game_over() {
    pause_ms = 0;
    mode = 4;
    game_over_sound(); // << play sad sound once
    pause_ms -= 1400; // account for delay by audio influencing
delta_time
    t0 = 5000; // starting speed back to 5 sec
    time = t0;
    high_score = (score > high_score) ? score : high_score;
}

// Handles successful input
void input_success(int success_mode) {
    correct_sound(success_mode);
    mode = random(3) + 1; // Pick random mode 1-3
    if (t0 > 2000) t0 -= 250; // time per action decreases from 5
to 2 seconds
    time = t0; // Reset timer
    score++;
}
```