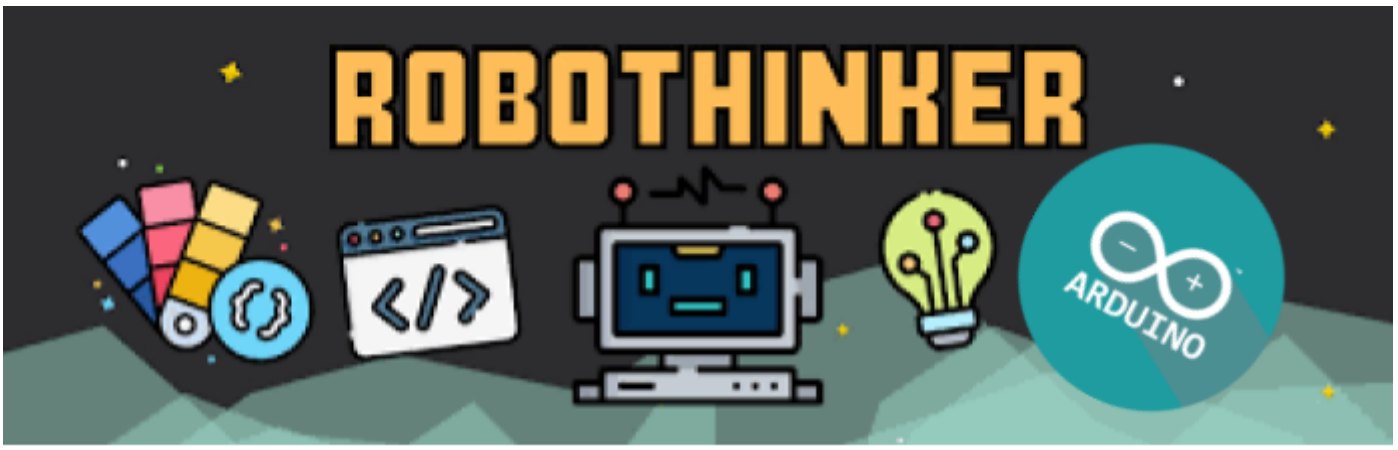
TEAM ROBOT LEAGUE STARTS HERE



LED BUTTON GAME

IMPLEMENTING ARRAYS AND CONTROL STATEMENTS IN JAVA

INTRODUCTION

In this project, you will use arrays and control statements to create a game that interacts with 6 LEDs and 6 buttons. Your goal is to write a program that generates a sequence of LED flashes and requires the player to repeat the sequence by pressing the corresponding buttons.

Requirements

REQUIREMENTS

- 6 LEDs
- 6 buttons
- A microcontroller (such as Arduino)
- Jumper wires
- A breadboard
- A computer with the Java development environment set up

HINTS

- Use a loop (e.g., for, while) to generate the LED sequence
- Use the random class to generate random numbers for the LED sequence
- Use control statements (if, while) to control the game logic
- Use the digitalRead() and digitalWrite() functions to control the buttons and LEDs

APPROPRIATE ALGORITHM STEPS

1. Declare and initialise an array to store the sequence of LED flashes
2. Write a Java program that generates a new sequence of LED flashes for each round of the game
3. Use control statements (if, while) and the LED array to control the game logic
4. Use Java code to turn on and off the LED lights in the sequence
5. Monitor the player's button presses and compare them to the current sequence
6. Test your code on the microcontroller to make sure it works as expected
7. Refine and improve your code to make the game more engaging and challenging

APPROPRIATE SETUP STEPS

STEP 1: SET UP THE HARDWARE

- Connect 6 LEDs and 6 buttons to the microcontroller using jumper wires and a breadboard
- Connect the microcontroller to your computer

STEP 2: CREATE AN ARRAY FOR THE LED SEQUENCE

- Declare and initialise an array to store the sequence of LED flashes
- The array should have a length of 6 to match the number of LEDs
- Example: int[] sequence = {1, 3, 5, 4, 2, 6};

STEP 3: WRITE THE GAME LOGIC

- Write a Java program that generates a new sequence of LED flashes for each round of the game
- The program should monitor the player's button presses and compare them to the current sequence
- The program should update the game status (e.g., game over, next round, etc.) based on the player's button presses
- Hint: Use control statements (e.g., if, while) and the LED array to control the game logic

STEP 4: CONTROL THE LED LIGHTS AND BUTTONS

- Use Java code to turn on and off the LED lights in the sequence
- Monitor the player's button presses and compare them to the current sequence
- Hint: Use the digitalRead() and digitalWrite() functions to control the buttons and LEDs

STEP 5: TEST AND DEBUG

- Test your code on the microcontroller to make sure it works as expected
- Debug any issues that you encounter

STEP 6: REFINE AND IMPROVE

- Refine your code to make the game more engaging and challenging
- Add additional features or improvements, such as different levels of difficulty, different game modes, etc.

CODING HELP

HINT: USE OF ARRAYS

An array is a data structure that stores a collection of values. In Java, an array is declared with the syntax: type[] name = {element1, element2, element3, ...};

For example, if you have an array of 6 integers to store the sequence of LED flashes, you can declare it as follows: int[] sequence = {1, 3, 5, 4, 2, 6};

Techniques for working with an array

The sequence array contains 6 integers, each representing the sequence of the 6 buttons. Your goal is to press the buttons in the correct order according to the sequence.

```
int[] sequence = {1, 3, 5, 4, 2, 6};
```

Iterating through the Array

You can use a for loop to go through each element in the sequence array. For example:

```
main()
{
   int[] sequence = {1, 3, 5, 4, 2, 6};
   playSequence(sequence);
}

void playSequence( int[] arr)
{
  for (int i = 0; i < arr.length; i++)
  {
     currentFlash = arr[i];
     digitalWrite(currentFlash, HIGH);
     delay(1000);
     digitalWrite(currentFlash, LOW);
   }
}
```

HINT: TO KEEP TRACK OF PROGRESS

You can use a variable to keep track of your progress in the sequence.

For example:

```
int progress = 0;
int currentVal = sequence[progress];

if ( digitalRead(currentVal) == LOW)
{
        // correct button pressed
        progress ++;
        // turn on LED
        // ...
}
else
{
        // incorrect button pressed
        // ...
        // triple beep and quit?
}
```

Check if the Sequence is Completed: You can use an if statement to check if the sequence is completed.

For example:

```
if (progress == 6) {
  // sequence is completed
  // ...
}
```

HINT: USE OF CONTROL STATEMENTS

Control statements are used to control the flow of execution in a program. In this project, you will use the following control statements:

- if: Used to make a decision based on a condition
- while: Used to repeat a block of code while a condition is true

Here are some hints on how to control the button so that it only records the first button pressed:

1. **Debouncing**:
   Buttons can bounce when they are pressed, which can cause multiple readings to be recorded even if the button is only pressed once. You can use the delay() function to add a small delay between readings to reduce bouncing. For example:

```
int buttonState = digitalRead(buttonPin);
if (buttonState == LOW)
{
    delay(10); // add debouncing delay
    if (digitalRead(buttonPin) == LOW)
    {
    // button is pressed
    // ...
  }
}
```

2. **Keep Track of Button State**
   You can use a variable to keep track of the button state. For example:

```
int buttonState = digitalRead(buttonPin);
if (buttonState == LOW && !buttonPressed) {
  // button is pressed for the first time
  buttonPressed = true;
  // ...
} else if (buttonState == HIGH && buttonPressed) {
  // button is released
  buttonPressed = false;
  // ...
}
```

3. **Only Record the First Button Pressed:**
   You can use an if statement to only record the first button pressed. For example:

```
if (buttonState == LOW && !buttonPressed)
{
    // button is pressed for the first time
    buttonPressed = true;
    if (progress == i)
    {
       // correct button pressed
       progress++;
       // turn on LED
       // ...
```

```
      }
      else
      {
      // incorrect button pressed
      // ...
     }
}

else if (buttonState == HIGH && buttonPressed)
{
  // button is released
  buttonPressed = false;
}
```

HINT: BUZZER CODE EXAMPLE

```
int buzzer = 13; // pin 13 connected to buzzer
int correctTone = 1000; // frequency in Hz for correct tone
int incorrectTone = 500; // frequency in Hz for incorrect tone

void setup()
{
  pinMode(buzzer, OUTPUT); // initialize buzzer as output
}

void playCorrectTone()
{
  tone(buzzer, correctTone); // play correct tone
  delay(500); // delay for half a second
  noTone(buzzer); // stop playing tone
}

void playIncorrectTone()
{
  tone(buzzer, incorrectTone); // play incorrect tone
  delay(500); // delay for half a second
  noTone(buzzer); // stop playing tone
}
```