1ST MAKER SPACE
Design.
Build.
Sustain.

Frog Microcontroller Trainer (FMT): High-Low

Length of Time: 2 days x 55 Minutes

Recommended for Advanced

Contents

Overview:

This lesson guides high school students in building a High–Low number guessing game using an Arduino-based Maker Frog board. Students write the program from scratch, integrating prior skills like reading inputs (buttons, potentiometer), using random numbers, and controlling outputs (OLED display, NeoPixels, sound).

Objectives:

Students will be able to:

Use randomSeed() and random() to generate an unpredictable secret number at the start of each game.

Use analogRead() and map() to convert a continuous potentiometer reading into a bounded integer range.

Write an if / else if / else chain that compares two numbers and branches on three possible outcomes.

Maintain a guess counter that increments once per confirmed guess and displays on the OLED.

Apply a two-state finite state machine (guessing vs. won) to control game flow.

Apply the button debounce and edge-detection pattern from prior programs, simplified to a single combined detector when both buttons perform the same action.

Write a complete, working Arduino program from a specification, using prior curriculum code as reference.

Computer Science Teacher Association Standards:

- 3A-CS-03 – Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.
- 3A-AP-13 – Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
- 3A-AP-15 – Justify the selection of specific control structures in terms of readability and performance.
- 3A-AP-16 – Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.
- 3A-AP-17 – Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
- 3A-AP-21 – Evaluate and refine computational artifacts to make them more usable and accessible.
- 3A-AP-23 – Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Materials:

- Frog Microcontroller Trainer from 1st Maker Space
- USB Power Cord
- Arduino IDE
- PC or Mac
- Chromebooks if using Arduino Create for Education App
- Copies of Student Handout

Preparation:

- Run the program yourself to see how it works.
- Read the ‘teacher troubleshooting notes.
- Print and distribute the student handouts.

Background Information:

Random Numbers on Arduino:

random(min, max) returns a pseudo-random integer from min up to, but not including, max. To get a number from 1 to 31 inclusive, the call is random(1, 32). The sequence is deterministic — the same seed always produces the same sequence — so randomSeed() is called first with an unpredictable value. Reading an unconnected (floating) analog pin with analogRead(A1) produces electrical noise that differs on every power-up, making it a simple and effective seed source.

Reading the Potentiometer with map():

analogRead() returns a 10-bit integer from 0 (pot fully left) to 1023 (pot fully right). For a guessing game the program needs a number in the range 1–31, not 0–1023. The map() function performs this scaling:
map(value, fromLow, fromHigh, toLow, toHigh)
map(analogRead(POT_PIN), 0, 1023, 1, 31) scales the 10-bit reading linearly so that 0 maps to 1 and 1023 maps to 31. map() uses integer math and does not clamp its output, so wrapping the result in constrain(result, 1, 31) protects against edge-case values outside the intended range

Three-way Comparison with if/else if/else:

Comparing a guess to the secret has exactly three outcomes: too low, too high, or correct. An if / else if / else chain handles all three in order, and exactly one branch runs:

```
if (g < secret) {
    // too low
} else if (g > secret) {
    // too high
} else {
    // correct  (g == secret is the only remaining case)
}
```

The else branch does not need to test g == secret because if neither < nor > is true, equality is guaranteed. This is an important logical pattern: if two of three cases are handled explicitly, the third is implicit.

Counting Guesses:

A guess counter is a single integer variable that starts at zero for a new game and increments by one each time the player confirms a guess. In code:

```
guessCount++;   // after every confirmed guess, before the comparison
```

Incrementing before the comparison (rather than only on incorrect guesses) means the winning guess is also counted, so the final display correctly reads "Got it in 1 guess" if the player is very lucky.

State Machine:

The game has two states. While GUESSING, button presses compare the current pot value to the secret and update the result indicator. Once the guess is correct the state changes to WON, the win screen is shown, and button presses restart the game by calling newGame() rather than comparing another guess. This two-state structure prevents the player from accidentally making a guess on the win screen while reaching for the button to restart.

Combined Button Detector:

In this game both buttons perform the same action — confirm a guess — so there is no need to track left and right separately. A single debounce loop treats the two buttons as one signal:

```
bool raw = (digitalRead(BUTTON_ONE) == PRESSED) ||
           (digitalRead(BUTTON_TWO) == PRESSED);
```

This is a simplification of the two-channel debounce used in Chess Timer and Stopwatch. The same rising-edge detection logic applies: anyPress is set true for exactly one loop iteration when the combined signal transitions from not-pressed to pressed.resetGame()

Program Architecture Reference:

This section describes the solution architecture. Use it to guide stuck students or as a pre-challenge discussion. Do not show High_Low.ino to students until after they have made a genuine attempt.

State Machine:

State	Meaning	Transition To
STATE_GUESSING	Game is active. Button presses compare the pot value to the secret.	STATE_WON when the guess equals the secret.
STATE_WON	Correct guess made. Win screen displayed with the secret and guess count.	STATE_GUESSING (via newGame()) when any button is pressed.

Key Variables:

Variable	Type	Purpose
state	GmState (enum)	Current FSM state. Controls what a button press does.
secret	uint8_t	The hidden number chosen by newGame(). Never shown while guessing.
guessCount	uint8_t	Increments on each confirmed guess. Displayed on both screens.
lastGuess	uint8_t	The number last submitted by the player. Initialized to 0 (outside 1–31). A press is ignored if the current pot reading matches lastGuess, preventing the same number from counting twice.
lastResult	Result (enum)	RESULT_NONE, RESULT_LOW, or RESULT_HIGH. Controls the feedback line and eye color.
anyStable	bool	Debounced combined button state. true = at least one button pressed.
anyPress	bool	One-shot press event. true for one loop iteration only.

Functions:

Function	Responsibility
initHardware()	Set pin modes, start OLED, initialize NeoPixels.
newGame()	Call random () to pick a new secret; reset guessCount and lastResult to starting values.
updateButtons(now)	Combined debounce for both buttons; set anyPress on rising edge.
handleButtons()	On press: compare readGuess() to secret, update lastResult / guessCount / state; on win screen: call newGame().
readGuess()	Return map(analogRead(POT_PIN), 0, 1023, 1, 31) clamped with constrain().
updateDisplay()	Draw the guessing screen (hint, large number, result line) or the won screen.
updateEyes()	Set NeoPixel color: dim white=no result, red=too low, blue=too high, green=won.

Teacher Troubleshooting Notes:

► TEACHER: Secret is always the same value: the student is calling randomSeed() with a constant, or not calling it at all. Also check they are not accidentally reassigning secret inside loop() — newGame() should only be called from setup() and on a win.
► TEACHER: Guess counter counts faster than expected: the student has guessCount++ outside the if (anyPress) guard, so it increments every loop iteration. It should only increment once per confirmed button press.
► TEACHER: map() returns unexpected values: Arduino's map() uses integer math and does not clamp. map(1023, 0, 1023, 1, 31) correctly returns 31, but floating-point rounding in the student's head may cause confusion. Demonstrate on the board: result = toLow + (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow).
► TEACHER: Win fanfare blocks the display briefly: the solution uses delay() in the win branch of handleButtons(). This is intentional and acceptable — the game is already won so no button reads are missed. If a student asks why delay() is used here but not elsewhere, explain that delay() is only safe when you are certain nothing else needs to happen during the pause.
► TEACHER: Student uses if/if/if instead of if/else if/else: all three branches will evaluate even when one has already matched. If g < secret fires and sets lastResult = RESULT_LOW, then g > secret immediately checks again. For mutually exclusive conditions, else if is both correct and more efficient. Ask: can g be both less than and greater than secret at the same time?

Lesson Elements	
Introduction 10-15 minutes	Play the finished game in front of the class. Make one obviously wrong guess (turn the knob to 1 and press). Ask: How does the Frog know what I guessed? How does it know whether to say Too Low or Too High?
Concept – 20 minutes	Draw the number line 1–31 on the board and show how binary search converges on the answer. Demonstrate map() with two specific analogRead values (0 and 512) calculated by hand. Write the three-way if/else if/else pattern on the board and ask students to trace it for three example guesses.
Challenge – 50-70 minutes	Students build the program following the step-by-step. Circulate and use the troubleshooting guide. Note: See a potential solution for the program at the end of this document.
Debrief– 10 minutes	Ask: What does map() return if analogRead() returns 0? What if it returns 1023? Why does the else branch not need to test g == secret? What would happen if guessCount were declared inside handleButtons() instead of globally?
Career Exploration: • A good resource is the IDOE Career Explorer database. • Another good resource for career information is the Bureau of Labor Statistics	
Practical Application: This program models how computers take input, compare it to a target, and give feedback—something used in everything from games to industrial control systems.	

Additional Resources:

- 1st Maker Space Frog Microcontroller Library
- 1st Maker Space Learn Arduino with the Frog Microcontroller

Performance Assessment/Check for Understanding :

- Was the student able to successfully write the program for the FMT?
- Check student handout for responses.

Additional Feedback:

Potential Solution:

```
/************************************************
 * Program: High_Low.ino
 * Copyright (C) 1st Maker Space, Inc. 2026
 *
 * Description:
 *   Number guessing game. The Frog picks a secret
 *   number between 1 and 31. Turn the knob to dial
 *   in a guess and press either button to confirm.
 *   The Frog replies "Too Low!", "Too High!", or
 *   "Got It!" and tracks the number of guesses.
 *   Press any button after a win to play again.
 *
 * Hardware:
 *   - 1ST Maker Frog board (Arduino UNO R4 Minima)
 *   - OLED display (SSD1306, 128x64) via I2C
 *   - Potentiometer (POT_PIN = A0) — dials the guess
 *   - Either button — confirms a guess / starts new game
 *   - NeoPixel eyes, piezo buzzer
 *
 * Teacher note:
 *   This is the proposed SOLUTION. Do not distribute
 *   to students before they attempt the challenge.
 ************************************************/

#include "1ST_Maker_Frog.h"

// ---------------------------------------------------------------------
//  Configuration
// ---------------------------------------------------------------------

const uint8_t  SECRET_MIN  = 1;
const uint8_t  SECRET_MAX  = 31;
const uint16_t DEBOUNCE_MS = 30;   // button debounce hold time in ms
const uint16_t DISPLAY_MS  = 80;   // OLED refresh interval in ms

// ---------------------------------------------------------------------
//  State machine
// ---------------------------------------------------------------------

enum GmState { STATE_GUESSING, STATE_WON };
GmState state = STATE_GUESSING;

// ---------------------------------------------------------------------
//  Game variables
```

```
// ---------------------------------------------------------------------------

uint8_t secret;      // the hidden number, SECRET_MIN to SECRET_MAX
uint8_t guessCount;  // confirmed guesses so far this game
uint8_t lastGuess;   // the number the player last submitted; 0 = none yet

enum Result { RESULT_NONE, RESULT_LOW, RESULT_HIGH };
Result lastResult = RESULT_NONE;

uint32_t lastDisplayMs = 0;

// ---------------------------------------------------------------------------
//  Button debounce — combined "any button" detector
// ---------------------------------------------------------------------------

bool     anyStable   = false;
uint32_t anyChangeMs = 0;
bool     anyPress    = false;   // one-shot: true for one loop when pressed

// ---------------------------------------------------------------------------
//  Function prototypes
// ---------------------------------------------------------------------------

void    initHardware();
void    newGame();
void    updateButtons(uint32_t now);
void    handleButtons();
uint8_t readGuess();
void    updateDisplay();
void    updateEyes();

// ---------------------------------------------------------------------------
//  Setup and loop
// ---------------------------------------------------------------------------

void setup() {
  initHardware();
  randomSeed(analogRead(A1));   // seed RNG from a floating (unconnected) pin
  newGame();
}

void loop() {
  uint32_t now = millis();
```

```
  updateButtons(now);
  handleButtons();

  if (now - lastDisplayMs >= DISPLAY_MS) {
    lastDisplayMs = now;
    updateDisplay();
  }

  updateEyes();
}

//
================================================================
=========
//  Functions
//
================================================================
=========

void initHardware() {
  pinMode(BUTTON_ONE, INPUT_PULLUP);
  pinMode(BUTTON_TWO, INPUT_PULLUP);
  pinMode(PIEZO, OUTPUT);
  pixel.begin();
  pixel.show();
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  display.clearDisplay();
  display.display();
}

/***********************************************
 * Function: newGame
 * Purpose:  Pick a new secret number and reset the
 *           guess counter and result indicator.
 *
 * random(min, max) returns a value from min up to
 * but NOT including max, so random(1, 32) gives
 * a number from 1 to 31 inclusive.
 ***********************************************/
void newGame() {
  secret     = random(SECRET_MIN, SECRET_MAX + 1);
  guessCount = 0;
  lastGuess  = 0;   // 0 is outside the valid range, so any first guess is accepted
```

```
  lastResult = RESULT_NONE;
  state      = STATE_GUESSING;
}

/*************************************************
 * Function: updateButtons
 * Purpose:  Debounce both buttons together as a
 *           single "any button" signal. anyPress is
 *           true for exactly one loop iteration on
 *           the rising edge.
 *
 *           Because both buttons confirm a guess,
 *           a combined detector is simpler than
 *           tracking them separately.
 *************************************************/
void updateButtons(uint32_t now) {
  anyPress = false;
  bool raw = (digitalRead(BUTTON_ONE) == PRESSED) ||
             (digitalRead(BUTTON_TWO) == PRESSED);

  if (raw != anyStable) {
    if (now - anyChangeMs >= DEBOUNCE_MS) {
      bool prev   = anyStable;
      anyStable   = raw;
      anyChangeMs = now;
      if (anyStable && !prev) anyPress = true;   // rising edge
    }
  } else {
    anyChangeMs = now;
  }
}

/*************************************************
 * Function: handleButtons
 * Purpose:  Act on a confirmed button press.
 *
 *   Guessing: read the pot, compare to the secret,
 *             update lastResult and guessCount,
 *             check for a win.
 *   Won:      start a new game.
 *
 * Notes:
 *   The win fanfare uses delay() because it is a
 *   one-time event and the brief pause is harmless.
```

```
 *   Inside a running game loop, delay() would block
 *   button reads, but a win already stops the game.
 ************************************************/
void handleButtons() {
  if (!anyPress) return;

  if (state == STATE_WON) {
    newGame();
    return;
  }

  uint8_t g = readGuess();
  if (g == lastGuess) return;   // player hasn't moved the knob; don't count
  lastGuess = g;
  guessCount++;

  if (g < secret) {
    lastResult = RESULT_LOW;
    tone(PIEZO, 300, 150);
  } else if (g > secret) {
    lastResult = RESULT_HIGH;
    tone(PIEZO, 600, 150);
  } else {
    state = STATE_WON;
    pixel.setPixelColor(0, pixel.Color(80, 0, 0));  // go green now — delay() below
    pixel.setPixelColor(1, pixel.Color(80, 0, 0));  //   blocks loop() during the fanfare
    pixel.show();
    tone(PIEZO, 1000, 80); delay(90);
    tone(PIEZO, 1300, 80); delay(90);
    tone(PIEZO, 1600, 150);
  }
}

/************************************************
 * Function: readGuess
 * Purpose:  Map the potentiometer (0–1023) to an
 *           integer in the range SECRET_MIN–SECRET_MAX.
 *
 * Returns:
 *   Current guess value (1–31).
 *
 * map(value, fromLow, fromHigh, toLow, toHigh)
 * scales the value linearly using integer math.
 * constrain() clamps the result to the valid range
```

```
 * in case analogRead() returns an edge-case value.
 *************************************************/
uint8_t readGuess() {
  int raw = analogRead(POT_PIN);
  return (uint8_t)constrain(map(raw, 0, 1023, SECRET_MIN, SECRET_MAX),
                            SECRET_MIN, SECRET_MAX);
}

/*************************************************
 * Function: updateDisplay
 * Purpose:  Draw the OLED for the current state.
 *
 * Guessing layout:
 *   Top (small):    instruction hint
 *   Middle (large): current pot value as a number
 *   Bottom (small): last result and guess count
 *
 * Won layout:
 *   "GOT IT!" (medium)
 *   Winning number (large)
 *   Guess count and restart prompt (small)
 *
 * At text size 4, each character is 24 px wide and
 * 32 px tall. A two-digit number (e.g. "18") is
 * 48 px wide; centering: x = (128 - 48) / 2 = 40.
 *************************************************/
void updateDisplay() {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  if (state == STATE_WON) {
    display.setTextSize(2);
    display.setCursor(10, 0);
    display.print("GOT IT!");

    char buf[4];
    snprintf(buf, sizeof(buf), "%u", secret);
    display.setTextSize(3);
    int16_t x = (SCREEN_WIDTH - (int16_t)(strlen(buf) * 18)) / 2;
    display.setCursor(max(x, (int16_t)0), 18);
    display.print(buf);

    display.setTextSize(1);
    display.setCursor(0, 48);
```

```
    display.print("Guesses: ");
    display.print(guessCount);
    display.setCursor(0, 57);
    display.print("Press a button");

  } else {
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("Turn knob, then press");

    char buf[4];
    snprintf(buf, sizeof(buf), "%u", readGuess());
    display.setTextSize(4);
    int16_t x = (SCREEN_WIDTH - (int16_t)(strlen(buf) * 24)) / 2;
    display.setCursor(max(x, (int16_t)0), 10);
    display.print(buf);

    display.setTextSize(1);
    display.setCursor(0, 52);
    if      (lastResult == RESULT_LOW)  display.print("Too Low!   ");
    else if (lastResult == RESULT_HIGH) display.print("Too High!  ");
    else                                display.print("           ");
    display.print("Tries: ");
    display.print(guessCount);
  }

  display.display();
}

/*************************************************
 * Function: updateEyes
 * Purpose:  Color the NeoPixel eyes to reflect the
 *           last guess result.
 *   No result: dim white (neutral / waiting)
 *   Too Low:   red  (need to go higher)
 *   Too High:  blue (need to go lower)
 *   Won:       green
 *
 *   The Frog pixel object is declared NEO_RGB but
 *   the physical LEDs are GRB, so the R and G
 *   arguments to pixel.Color() are swapped relative
 *   to their names. See inline comments below.
 *************************************************/
void updateEyes() {
```

```
  uint32_t color;
  // NOTE: the Frog pixel is declared NEO_RGB but the LEDs are physically
  // GRB, so R and G bytes are swapped on the wire. Color(r,g,b) here maps
  // r→LED green and g→LED red. Values below are written to produce the
  // intended visible color, not the intuitive argument order.
  if      (state == STATE_WON)         color = pixel.Color(80,  0,  0);  // green
  else if (lastResult == RESULT_LOW)   color = pixel.Color( 0, 80,  0);  // red
  else if (lastResult == RESULT_HIGH)  color = pixel.Color( 0,  0, 80);  // blue
  else                                 color = pixel.Color(20, 20, 20);  // dim white

  pixel.setPixelColor(0, color);
  pixel.setPixelColor(1, color);
  pixel.show();
}
```