Man Zou and Reihaneh Tamizkar
Professor Elio Bidinost
CART 360
30-09-2022

Étude 2

Part 2 A: Code Observation

1- We start with defining a variable associated with all the corresponding pin's number (ex. #define PIN_LED_BUTTON_1 3), then define the rest of the variables/array that we need to use in the code.

2- We define a few functions the game needs to have. For example, we need one function for lighting up the LED and producing a sound associated to that LED from the piezo. In the code, this function is called displayLightAndSound(). We also need a function that starts the light and sound sequence at the beginning, hinting a new start for the game, which is called startUpLightsAndSound() (this function is called only in void setup and when sequence number is 0). Another function that we need is one that defines the gameplay routine associated with each sequence, which the code decided to put in void loop. Basically, those lines of codes in void loop create a random number for the LED to light up. If the player don't push the button under 3 seconds, then the sequence number becomes 0 and the piezo will make a "sad" sound, meaning the game is lost, and the sequence becomes 0 and gets back to 1 again. The same goes if the player pushes the wrong button. If the player pushes the right button, the sequence will increment and the next round begins in 1 second (delay(1000)).

Part 2 B: How Does The Game Works

The computational structure the game employs to establish and maintain state is what's in the void loop.

```
void loop() {
  static int sequenceLength = 1; //*********THIS IS WHAT MAKES THE GAME PROCEED TO THE NEXT LEVEL
  int i;

  // Display current sequence
  //*********Randomly lght up an LED
  randomSeed(gameSeed);
  for (i = sequenceLength; i > 0; --i) {
    displayLightAndSound(buttonLookup[random(LED_BUTTON_COUNT)], 250); //Illuminate the button that has been pushed
    delay(250);
  }

  //********Make a random value
  randomSeed(gameSeed);


  for (int matches = 0; matches < sequenceLength; ++matches) {
    int button = -1;
    int timeout;
    // Wait for the player to press a button (with 3 second timeout)
    for (button = -1, timeout = 30; timeout && button < 0; --timeout) {
      button = getButtonPush();
      delay(100);
    }

    // Check if the correct button was pushed
    if (button != random(LED_BUTTON_COUNT)) {
      // Play sad sound and start a new game
      displayLightAndSound(-1, 1000);
      gameSeed = micros();
      delay(1000);
      sequenceLength = 0;
      startUpLightsAndSound();
    }
  }
  ++sequenceLength;
  delay(1000);
}
```

ii) What is special about this computational structure?

What is special about this computational structure is that it uses a variable (sequenceLength) to define the state of the game. Everytime the sequence adds 1, one more level is added to the game, which means one more random LED will light up. Inside the computational structure of the void loop, there is one for loop that checks if the button has been pressed under 3 seconds and if the wrong button number has been pressed. If no right button is pushed under 3 seconds, sequence number goes back to 0 and the sequence starts again from 1. If the right button has been pressed, the following code will hot execute and sequenceLenght will add 1, thus generating a new gameSeed (random LED lights up) on top of the previous values.

```
// Check if the correct button was pushed
if (button != random(LED_BUTTON_COUNT)) {
  // Play sad sound and start a new game
  displayLightAndSound(-1, 1000);
  gameSeed = micros();
  delay(1000);
  sequenceLength = 0;
  startUpLightsAndSound();
}
```