CS 362: Milestone 9

Due on April 24, 2024 at 11:59pm

Professor Troy 11:00am

John Ezra See
Ryan Magdaleno
jsee4@uic.edu
rmagd2@uic.edu

Group Information

Group 62: VSRG-UNO-R3
Names:

1. Ryan Magdaleno: rmagd2@uic.edu
   netID: rmagd2, UIN: 668523658
2. John Ezra See: jsee4@uic.edu
   netID: jsee4, UIN: 668942698

Abstract

This project introduces an original Arduino/C++ endeavor aimed at enhancing the gaming experience through innovative hardware and software integration. Utilizing Arduino microcontrollers and a photoresistor array, it offers physical interaction with rhythm games, synchronizing gameplay with real-time data processing. The system manages communication between components and supports customization preferences. Through this project, users can enjoy an immersive and customizable VSRG experience, demonstrating creativity and innovation in Arduino/C++ technology.

---

1 Project Idea

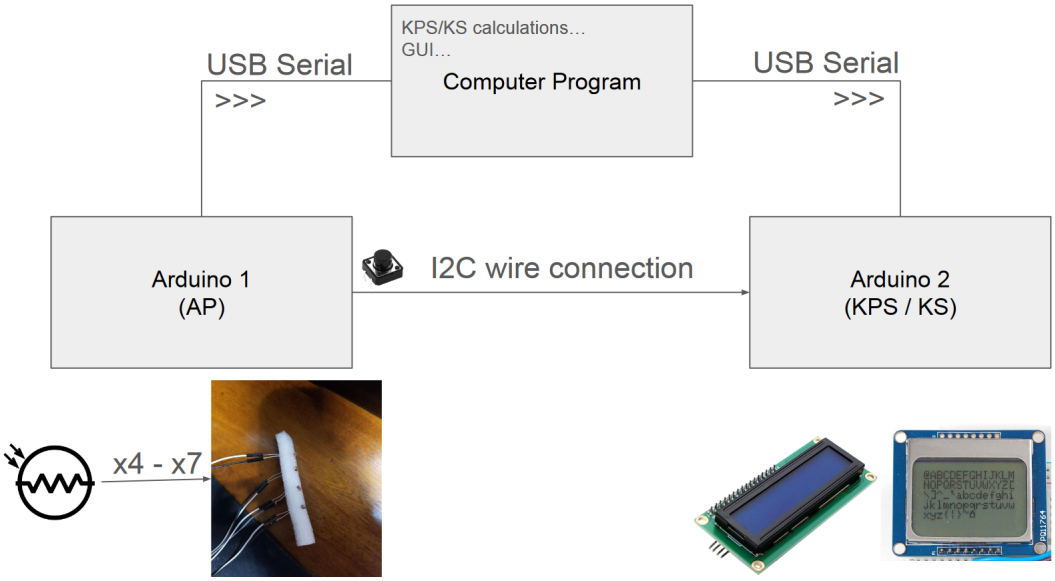
Our project idea centers around enhancing the experience of playing Vertical Scrolling Rhythm Games (VSRGs), particularly focusing on key presses. We plan to utilize two Arduino microcontrollers to manage two displays and a photoresistor array. One Arduino will control the displays, showcasing both the keys pressed per second and the key press strokes. This data will be calculated and sent via serial communication to a computer-side program. We intend to update the display with the keys per second value every 20 milliseconds on a 16x2 LCD screen. Additionally, we will transmit a binary value representing the keys pressed, also updating every 20 milliseconds.

For gameplay, we've chosen Osu Mania due to its easily parseable .osu file format, which contains hit timings for beatmaps. Although Osu Mania supports maps of various key counts, we will focus on maps ranging from 4 to 7 keys. We've changed our original idea of using a solenoid array to a photoresistor array because there were some complexities in the solenoid circuit. Now one Arduino will read the incoming notes and check if it's in front of the photoresistor. The photoresistor array Arduino will send the binary encoding of the current keys pressed to a computer side program. The computer side program will then press those keys. The computer program will also continuously monitor these key presses, calculating both keystrokes and keys per second, and sending the accurate values to the display Arduino. For the Arduino managing keystrokes, we will employ a Nokia 5110 LCD to display the pressed keys, animating them upwards for visual feedback. The computer side program will also allow you to run without the auto player and only the key strokes, or vice versa.

---

2 Project Design

One Arduino will read the screen with photoresistors and send the current notes to press, then it will be sent to a computer side program. The other Arduino will receive keyboard press information, including keys per second and recent key presses. The computer-side program will start both Arduinos upon pressing key H and stop them upon pressing key J. An I2C wire connection signal will be transmitted from the photoresistor array Arduino to the display Arduino via a pushbutton. This I2C signal will turn the photoresistor array and displays off, pressing them again will turn both Arduinos on again.



---

3 Arduino Communication

For communication, we'll utilize a computer-side program to transmit the necessary data as required. The photoresistor Arduino will send hit data, while the display Arduino will receive keys per second (KPS) and keystroke data.

For the photoresistor array Arduino, we will have 4 photoresistors. Each photoresistor will be reading the falling notes and transmitting the note data every 20ms. The note data is like so: column 0 in the game represents the first bit in our sent number for example. Take the number 0101, this means column 0 has a note, C1 has no note, C2 has a note, and C3 has no note. The computer side program will unpack this data and press the corresponding keys.

The display Arduino follows a similar data pack scheme. The computer side program will be monitoring the currently pressed keys and create a binary mapping similar to the photoresistor array, with the added KPS, the first 10 bits are reserved for the key columns pressed while the latter 22 bits are used for the KPS value.

Initiating and stopping both Arduinos' Serial connection will be controlled by the computer-side program. To begin operation, the key H must be pressed. When this key is pressed, the Serial USB connection starts receiving data from the photoresistor array Arduion to the computer side program, and starts sending currently pressed keys on the computer to the display Arduino. The Serial connection will terminate when the key J is pressed, the GUI terminate button is pressed, or the beatmap ends/player exits in-game.

---

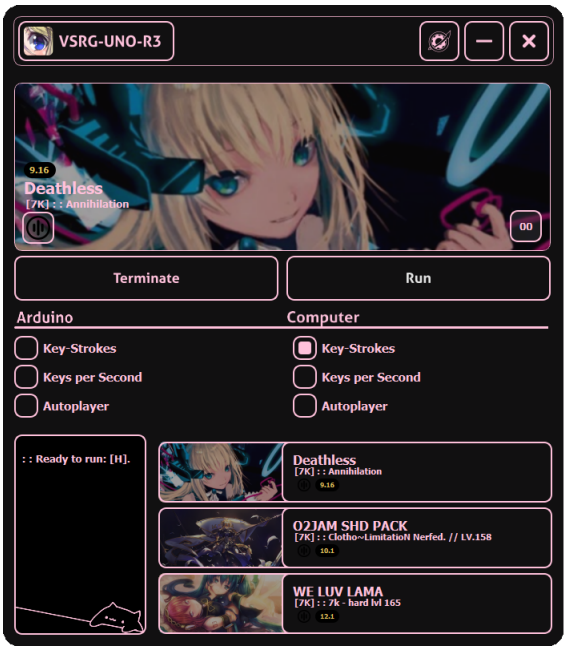
4 Expected Inputs/Outputs

Expected Inputs

1. **Keyboard Input:** The user would input commands through the keyboard. For example, pressing the key H to start the system and the key J to terminate it. The KPS and keystrokes are generated based on keyboard input.
2. **Beatmap Hit Data (.osu file):** This data would be inputted into the computer-side program, to provide the timing and sequence of hits in the osu beatmap.
3. **Note Light (photoresistors):** The photoresistors will be reading the falling notes, it requires the note to be in front of the photoresistor.
4. **I2C Input (pushbutton):** A single pushbutton will send a signal via a I2C connection to turn both Arduinos on and off.

Expected Outputs

1. **Display Data(16x2 LCD + Nokia 5110 LCD):** The display Arduino would output data to the connected display, showing information such as keys per second and recent key presses. This provides visual feedback to the user. KPS on a 16x2 LCD, key strokes on a Nokia 5110 LCD.
2. **Current keys pressed(LEDs)** There will be an array of LEDs that will light up if the corresponding photoresistor detects a note in front of it.
3. **Overall GUI program:** The computer-side program has a GUI, it will display many options to the user about what is going on, what is being transmitted, etc.



Here's a youtube video showcasing the project fully in action:
https://www.youtube.com/watch?v=WKqHV_tVNM8

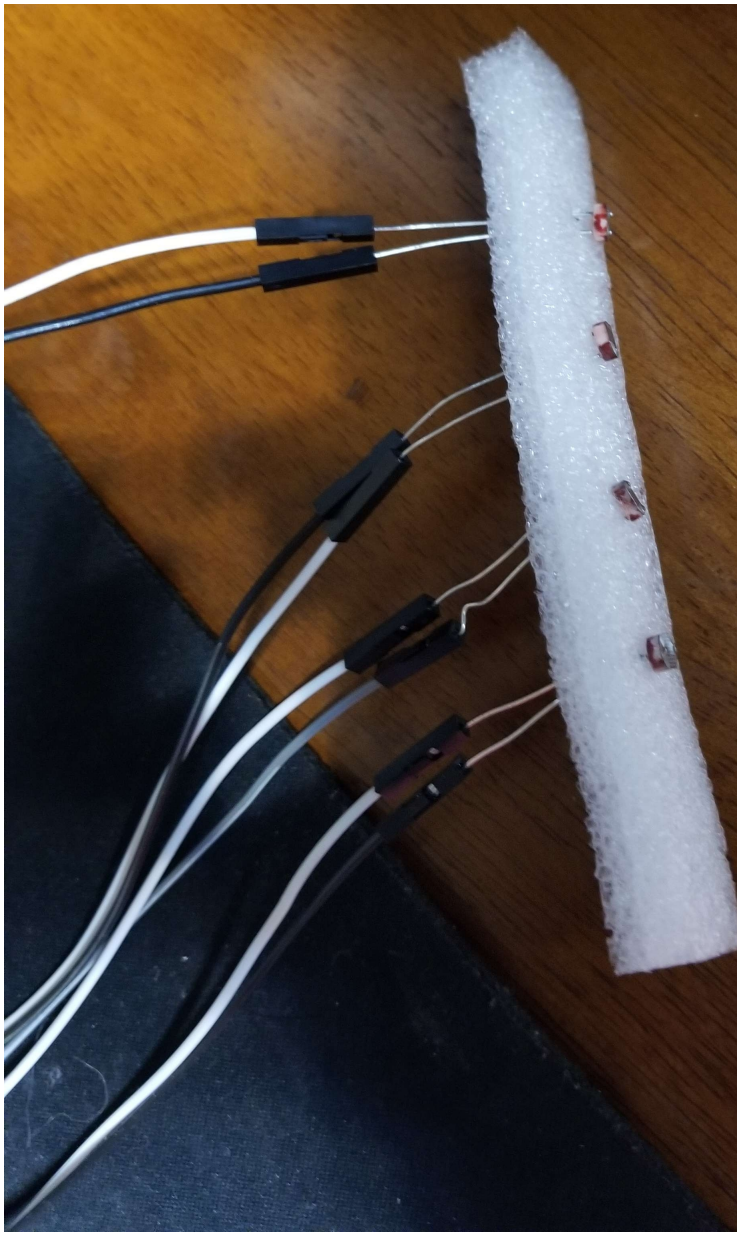
5 Original Work Justification

This project presents an original Arduino/C++ endeavor through its fusion of hardware and software elements to enhance the Osu Mania gaming experience. By integrating custom hardware components like Arduino microcontrollers and a photoresistor array, the system goes beyond typical software-based modifications, allowing for physical interaction with the game. It processes real-time data, including beatmap hit timings and key presses, synchronizes physical actions with gameplay, abd showcases advanced programming techniques in C++ (multi-threading, chrono timing, windows serial handles). Managing communication and synchronization between multiple components, including the computer-side program, Arduinos, photoresistor array, and displays, demonstrates a sophisticated understanding of system architecture and programming principles. This project stands out as an original and innovative Arduino/C++ project, offering a creative VSRG gaming experience. A GUI is used to facilitate this.

---

6 How to Build the Project

1. Head over to https://github.com/typeRYOON/VSRG-UNO-R3
2. You can build the GUI program from source following the instructions I set in the description or just get an already built binary from the Release section (Windows only).
3. For Arduino hardware please look at the diagrams below for what wiring you need. Because the computer side program allows using 0, 1, or both Arduino configurations, build whichever ones you'd like.
4. For the photoresistor Arduino, put your photoresistors through something like Foam and tape it to the osu mania game screen, it needs to read incoming notes. Please download the osu skin from the res GitHub folder, it's optimal for this program.



7 How to Use the Project

1. Connect both Arduinos to the computer you're going to use. Note the port name.
2. Download the git repository onto your system.
3. From the project's git repository, get the Arduino ino files from the arduino folder.
4. Upload the respective ino program to each Arduino.
5. You must have Osu installed and running before you open the GUI, it needs to latch to the Osu process to read memory signatures (for GUI purposes).
6. Make sure to install the Noto Sans JP font from the res/osu! folder onto your system.
7. Before launching the GUI, enter Osu and set your osu mania 4K, 5K, 6K, 7K keys, the program needs a valid set of keys to work with, please set up an osu! account before doing this (GUI needs a config file).
8. If you built your binary, please move the res/res folder to the executable's folder.
9. Launch osu! then the GUI and go to the settings page (gear icon at top), set the Arduino settings with the port selector, put the ports corresponding to the Arduinos. (Note: if you're running off a USB hub, enter the ports exactly like COM5 or com5, otherwise entering 5 should auto detect and mark COM5).
10. To run a configuration, select the beatmap in game and click on the GUI's run button along with the features you want to run.
11. Assuming you're using Arduino features, press H to start the two Arduinos.
12. If you're using the computer autoplayer feature, press H on the first note.
13. To terminate early, press J. You can also wait until the map finishes and the Arduinos will terminate for you.
14. An extra note, if you're using the Arduino autoplayer feature, please use the osu skin in the res/osu! folder.

---

8 Timeline of Development

Completed Items:

1. **Week 9 (03/11/24):**
   Ryan finished up the display Arduino code, and made a CLI for interacting communicating with the Arduinos.
   John's circuit catches on fire after connecting a 950 mA solenoid with a 600 mA transistor.
2. **Week 10 (03/18/24):**
   Ryan and John finished the photoresistor circuit/general code.
3. **Week 11 (03/25/24):**
   Ryan learned Qt6 over spring break and makes a working GUI.
4. **Week 12 (04/01/24):**
   Ryan worked on the GUI.
5. **Week 13 (04/08/24):**
   Ryan finished the GUI
   John finished the photoresistor array.
   Made final stand for both Arduinos to sit on.
6. **Week 14 (04/15/24):**
   Created presentation slides and presented on 04/15/24 in lab.
7. **Week 15 (04/22/24):**
   Did finishing tweaks and did project demonstration on 04/22/24 in lab.

---

9 List of Materials Expected

Display Arduino

1. 1x Arduino UNO-R3
2. 1x Breadboard
3. 1x 16x2 LCD
4. 1x Nokia 5110 LCD
5. 1x 2KΩ resistor
6. 1x 220Ω resistor
7. 1x 330Ω resistor
8. 4x 10KΩ resistors
9. 1x 1KΩ resistor

Photoresistor Array Arduino

1. 1x Arduino UNO-R3
2. 1x Breadboard
3. 5x 10KΩ resistors
4. 4x photoresistors
5. 1x Pushbutton
6. 4x LEDs
7. 4x 220Ω resistors
8. 2x 5KΩ resistors
9. Many wires
10. Tape and cardboard.

Miscellaneous

1. C++23 Compiler
2. Windows Computer

---

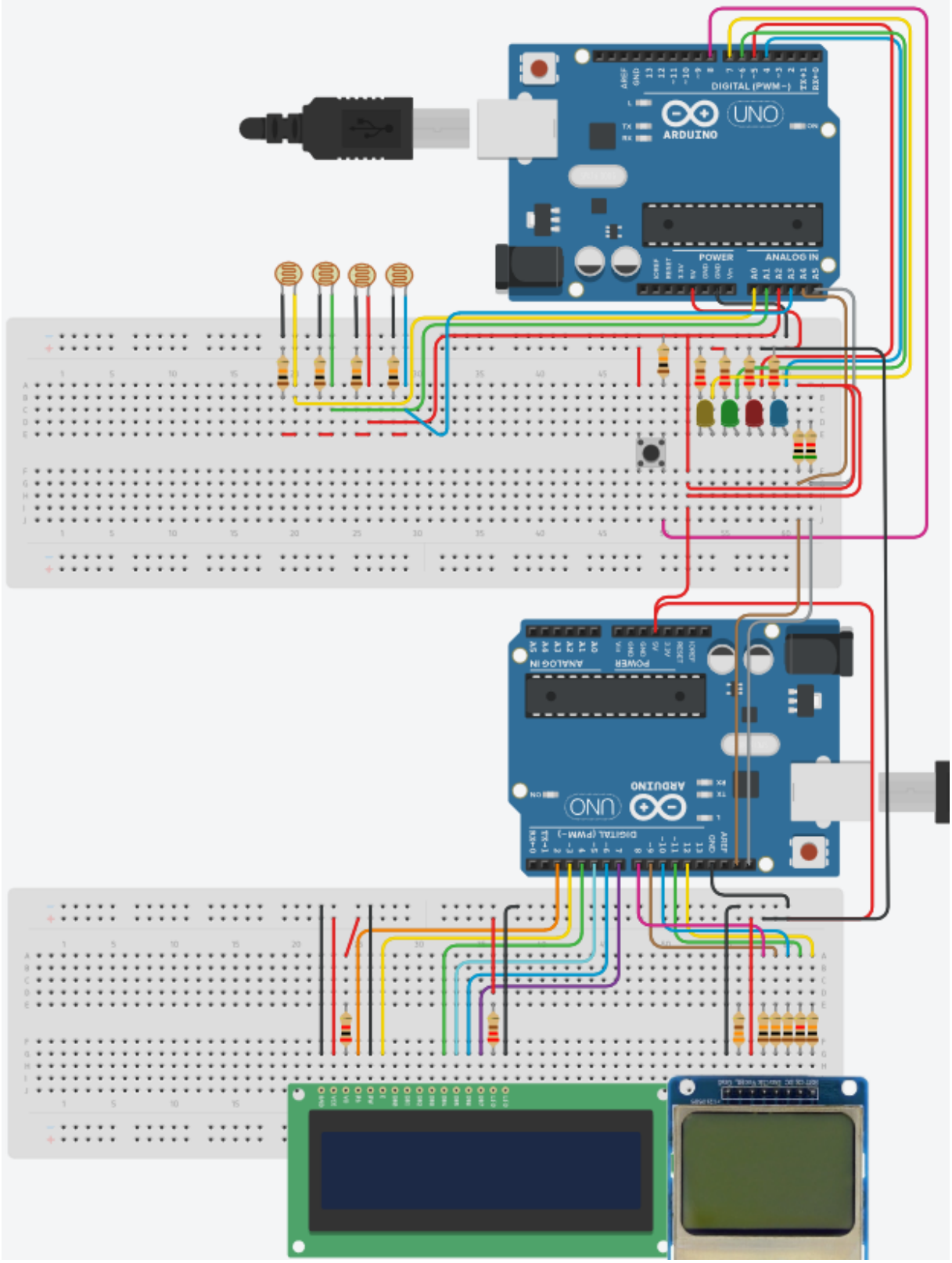
10 List of References

1. https://en.cppreference.com/w/cpp/header/chrono
2. https://en.cppreference.com/w/cpp/header/thread
3. https://learn.microsoft.com/en-us/windows/win32/api/winbase/ns-winbase-dcb
4. https://www.theengineeringprojects.com/2019/06/introduction-to-pn2222.html
5. https://www.youtube.com/watch?v=haUWO7BGYTE
6. https://www.build-electronic-circuits.com/rectifier-diode/
7. https://users.ece.utexas.edu/ valvano/Datasheets/PN2222-D.pdf
8. https://www.vishay.com/docs/88503/1n4001.pdf
9. https://www.onsemi.com/pdf/datasheet/tip120-d.pdf

---

11 Diagram + Code

TinkerCad Link



Photoresistor Array Arduino:

```
/* A1-AP.ino */
/* ------------------------------------------------------------------------------------------
 >> Assignment details and provided code are created and owned by Patrick Troy.
 >> University of Illinois at Chicago - CS 362, Spring 2024
 >> ------------------------------------------------------------------------------------------
 >> File    :: A1-AP.ino :: VSRG-UNO-R3: Arduino 1, Photoresistor Master I2C
 >> Course  :: CS 362 (47019), SP24
 >> Author  :: John Ezra See
 >> UIN/nID :: 668942698 (jsee4)
 >> System  :: Windows 10 w/ Mingw-w64
 - -                                 - -
 >> File overview ::
 >> This file implements Arduino 1. This program controls a photoresistor array and sends
 >> the osu mania key binary mapping to the GUI. This program is meant to be run with the
 >> VSRG-UNO-R3 GUI. Link below
 - -              >> https://github.com/typeRYOON/VSRG-UNO-R3 <<                        - -
 >> Please read the instructions on the github page to understand how to run this
 >> Arduino. Hardware required is listed in the repository README/documentation files.
------------------------------------------------------------------------------------------ */
// Preprocessor directives ::
#include <Wire.h>

// Global variables :: - -                                                              - -
int32_t light, buttonState = 1, buttonPin = 8, buttonStateOutput = 9;
const uint8_t  pinValues[] = {A0, A1, A2, A3, 7, 6, 5, 4};
const uint16_t luxValues[] = {550, 550, 550, 550}; // Calibrate as needed
uint8_t columnHits, curButton = LOW, prevButton = LOW;
uint32_t prevTime = 0, curTime;

// Set up all the pins :: - -                                                           - -
void setup()
{
    Serial.begin(9600);
    for (uint8_t i = 4; i < 8; ++i) {
        pinMode(pinValues[i], OUTPUT);
    }
    pinMode(buttonPin, INPUT);
    pinMode(buttonStateOutput, OUTPUT);
    Wire.begin();
}
```

```
// Main Arduino program loop :: - -                                    - -
void loop()
{
    curTime = millis();
    curButton = digitalRead(buttonPin);

    if (curTime - prevTime < 20) { return; }

    // Debounce
    if (prevButton == LOW && curButton == HIGH) {
      buttonState = !buttonState;

      // Send to Arduino 2 only when button is pressed:
      // buttonState = 0 -> OFF
      // buttonState = 1 -> ON
      Wire.beginTransmission(5);
      Wire.write(buttonState);
      Wire.endTransmission();
    }

    prevTime = curTime;
    prevButton = curButton;

    columnHits = 0;
    // Read all photoresistors and write to LEDs :: - -                 - -
    for (uint8_t i = 0; i < 4; ++i)
    {
        light = analogRead(pinValues[i]);
        if (buttonState == 0) light = 0; // If button state is OFF, photoresistor is OFF.
        if (light >= luxValues[i])  j
        {
            columnHits |= 1 << (i);
            digitalWrite(pinValues[i + 4], HIGH);
        }
        else {
            digitalWrite(pinValues[i + 4], LOW);
        }
    }
    Serial.print(columnHits ? columnHits : 0); // Send column mapping to GUI.
}
```

KPS/Key-strokes Arduino:

```
/* A2-KPS-KS.ino */
/* ------------------------------------------------------------------------------------------
 >> Assignment details and provided code are created and owned by Patrick Troy.
 >> University of Illinois at Chicago - CS 362, Spring 2024
 >> ------------------------------------------------------------------------------------------
 >> File    :: A2-KPS-KS.ino :: VSRG-UNO-R3: Arduino 2, Nokia 5110 LCD + 16x2 LCD
 >> Course  :: CS 362 (47019), SP24
 >> Author  :: Ryan Magdaleno
 >> UIN/nID :: 668523658 (rmagd2)
 >> System  :: Windows 10 w/ Mingw-w64
 - -                                 - -
 >> File overview ::
 >> This file is the implementation for the VSRG-UNO-R3 Arduino 2 keystrokes + KPS stat
 >> displays. This program can receive a signal from Arduino 1 that turns the displays
 >> off. This program is meant to be run with the VSRG-UNO-R3 GUI. Link below
 - -              >> https://github.com/typeRYOON/VSRG-UNO-R3 <<                        - -
 >> Please read the instructions on the github page to understand how to run this
 >> Arduino. Hardware required is listed in the repository README/documentation files.
------------------------------------------------------------------------------------------ */
// Preprocessor directives ::
#include <Adafruit_PCD8544.h>
#include <LiquidCrystal.h>
#include <Adafruit_GFX.h>
#include <Wire.h>
#include <SPI.h>

// Global variables :: - -                                                              - -
const uint8_t VSRG[] PROGMEM = { // 'VSRG' bitmap :: 10x48px
  0x0c, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x0c, 0x00,
  0x0c, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x77, 0x00, 0x61, 0x80, 0x61, 0xc0,
  0x61, 0xc0, 0x01, 0xc0, 0x01, 0x80, 0x27, 0x00, 0x3e, 0x00, 0x00, 0x00, 0x71, 0x80,
  0x31, 0x80, 0x19, 0x80, 0x1d, 0x80, 0x39, 0x80, 0x31, 0x80, 0x31, 0x80, 0x3f, 0x80,
  0x07, 0x00, 0x0f, 0x00, 0x3f, 0x80, 0x30, 0x00, 0x30, 0x00, 0x3c, 0x00, 0x0f, 0x00,
  0x03, 0x80, 0x03, 0x80, 0x3f, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x1e, 0x00, 0x1e, 0x00,
  0x33, 0x00, 0x33, 0x00, 0x33, 0x80, 0x61, 0x80, 0x61, 0x80, 0xc0, 0xc0
};

const uint8_t Kurisu[] PROGMEM = { // 'Kurisu' bitmap :: 48x70px
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x47,
  0x80, 0x00, 0x00, 0x00, 0x02, 0xe7, 0x80, 0x09, 0xe0, 0x00, 0x03, 0xf7, 0x80, 0x0d,
  0x08, 0x00, 0x07, 0x7d, 0xc0, 0x0b, 0xc4, 0x00, 0x03, 0x7e, 0xc0, 0x0a, 0x10, 0x00,
  0x03, 0x3e, 0xc0, 0x00, 0x89, 0x08, 0x03, 0x1f, 0xc0, 0x00, 0xc0, 0x08, 0x03, 0x2f,
  0xe0, 0x01, 0x04, 0x18, 0x01, 0x37, 0xf0, 0x03, 0x10, 0x88, 0x01, 0x77, 0xf8, 0x02,
  0x02, 0x00, 0x00, 0xc7, 0xfc, 0x30, 0x0a, 0x20, 0x00, 0x23, 0xfc, 0x80, 0x00, 0xc0,
  0x00, 0x23, 0xef, 0x00, 0x04, 0x40, 0x00, 0x23, 0xfe, 0x72, 0x05, 0x80, 0x00, 0x03,
```

```
  0xd7, 0xe8, 0x04, 0x80, 0x00, 0x13, 0xdb, 0x30, 0x05, 0x00, 0x00, 0x0f, 0xdb, 0x21,
  0x05, 0x00, 0x00, 0x0f, 0xef, 0x00, 0x07, 0x00, 0x00, 0x0f, 0xff, 0x00, 0x07, 0x00,
  0x00, 0x0f, 0xff, 0x01, 0x0f, 0x00, 0x00, 0x0f, 0xfe, 0x05, 0x1f, 0x00, 0x00, 0x0f,
  0xfe, 0x1a, 0x3f, 0x00, 0x00, 0x0f, 0xdc, 0x7c, 0x7f, 0x00, 0x00, 0x0f, 0x82, 0xb8,
  0xff, 0x00, 0x00, 0x1f, 0x00, 0x39, 0xff, 0x00, 0x00, 0x1e, 0x00, 0x11, 0xff, 0x00,
  0x00, 0x3c, 0x40, 0x13, 0xff, 0x00, 0x00, 0x38, 0x00, 0x17, 0x7f, 0x00, 0x00, 0x38,
  0x00, 0x1f, 0xbf, 0x80, 0x00, 0x70, 0x00, 0x0f, 0xbf, 0x80, 0x00, 0x70, 0x00, 0x0f,
  0xff, 0x80, 0x00, 0x70, 0x00, 0x5f, 0xff, 0x80, 0x00, 0x71, 0x01, 0xbf, 0xff, 0x80,
  0x00, 0x71, 0x71, 0xff, 0xff, 0x80, 0x00, 0x77, 0x37, 0xff, 0xff, 0xc0, 0x00, 0x77,
  0xfd, 0xef, 0xff, 0xc0, 0x00, 0x7e, 0xfe, 0x2f, 0xff, 0xc0, 0x00, 0x7d, 0xff, 0x6f,
  0xff, 0xc0, 0x00, 0x79, 0xff, 0x7f, 0xff, 0xc0, 0x00, 0x79, 0xff, 0xff, 0xff, 0xc0,
  0x00, 0x7f, 0xff, 0xff, 0xff, 0x80, 0x00, 0x7b, 0xff, 0xff, 0xdf, 0x80, 0x00, 0x7f,
  0xc7, 0xe0, 0x4f, 0x80, 0x00, 0x1e, 0x07, 0xfc, 0x4f, 0x80, 0x00, 0x32, 0x5f, 0xff,
  0xcf, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff, 0x00, 0x00, 0x13, 0xff, 0xff, 0xff, 0x00,
  0x00, 0x07, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x07,
  0xff, 0xff, 0xf8, 0x00, 0x00, 0x03, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x01, 0xff, 0xff,
  0xe0, 0x00, 0x00, 0x00, 0x7f, 0xff, 0x80, 0x00, 0x00, 0x00, 0x0f, 0xfc, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

// 16x2 display variables :: - -                                                - -
LiquidCrystal LCD(2, 3, 4, 5, 6, 7);
char s_KPS[9], s_KS[16]; // Top/Bottom row buffers.

// Nokia 5110 display variables :: - -                                          - -
Adafruit_PCD8544 Nokia = Adafruit_PCD8544(8, 9, 10, 11, 12);
const uint8_t WIDTH = 14, HEIGHT = 48;
uint8_t KS_Bits[WIDTH][HEIGHT] = {0}, drawIterations = 0;
int32_t rawKPS_KS, KS;

// Program flow variables :: - -                                                - -
uint32_t now, prev = millis();
bool draw = true, off = false;

// Set up external devices + communication pipelines :: - -                     - -
void setup()
{ // Set up the 16x2 LCD :: - -                                                 - -
  LCD.begin(16, 2);
  LCD.println("KPS :: 0        ");
  LCD.print  ("                ");

  // Set up the Nokia 5110 LCD :: - -                                           - -
  Nokia.begin();
  Nokia.setContrast(50);
  Nokia.clearDisplay();
  Nokia.drawBitmap(10, -15, Kurisu, 48, 70, BLACK);
  Nokia.drawBitmap(9 ,  0 , VSRG  , 10, 48, BLACK);
```

```
  Nokia.display();

  // Set up the I2C wire connection :: - -                                    - -
  Wire.begin(5);
  Wire.onReceive(buttonPressed);

  // Set up the 9600 baud rate serial USB connection :: - -                   - -
  Serial.begin(9600);
}

// Given a keystroke binary mapping, update the KS bitmap :: - -              - -
void updateBitmap(const int32_t KS)
{ // Shift values up :: - -                                                  - -
  for (uint8_t x = 0; x < WIDTH; ++x) {
    for (int8_t y = HEIGHT - 1; y >= 0; y -= 2)
    {
      KS_Bits[x][y-1] = KS_Bits[x][y-2];
      KS_Bits[x][y]   = KS_Bits[x][y-1];
    }
  }
  // Add new 14x2px row to bottom of KS bitmap :: - -                         - -
  for (uint16_t i = 2, x = 0; i <= 128; i <<= 1)
  {
    KS_Bits[x][0]   = KS & i; // Note ::
    KS_Bits[x++][1] = KS & i; //  Each column is 2 pixels wide.
    KS_Bits[x][0]   = KS & i; //  So we must increment x forward
    KS_Bits[x++][1] = KS & i; //  2 pixels for each loop iteration.
  }
}

// Draw current KS_Bits bitmap onto Nokia display :: - -                      - -
void drawBitmap(const bool print) // 'print' turns drawing/clearing on/off.
{
  for (uint8_t x = 0; x < WIDTH; ++x) {
    for (uint8_t y = 0; y < HEIGHT; ++y)
// Given a raw binary mapping, retrieve the KPS and update the 16x2 display :: - -   - -
void print16_2(const int32_t KPS)
{ // First 10 bits are for KS, so get latter 22 bits:
  sprintf(s_KPS, "%d       ", (KPS & 0xFFFFC00) >> 10);

  // Get KS mapping, bits 2-8:
  for (int i = 2, j = 0; i <= 128; i <<= 1) {
    s_KS[j++] = i & KPS ? '\xDB' : ' ';
  }
  // Update 16x2 display:
  LCD.setCursor(7, 0);
  LCD.print(s_KPS);
  LCD.setCursor(0, 1);
  LCD.print(s_KS);
```

```
}

// Arduino 1 will send a number via I2C, changes display states :: - -                - -
void buttonPressed(int32_t _)
{
  while (Wire.available()) {
    int32_t i = Wire.read();
    if (i)
    { // Displays are now set to ON state:
      LCD.setCursor(0, 0);
      LCD.println("KPS :: 0        ");
      LCD.print  ("                ");
      off = false;
    } else
    { // Displays are now set to OFF state:
      drawBitmap(false);
      LCD.clear();
      off = true;
    }
  }
}

// Main Arduino loop, every 20ms there should be a serial value :: - -                - -
void loop()
{
  if (Serial.available())
  { // Example :: "#12345678#"
    String rawSerial = Serial.readStringUntil('#');
    if (rawSerial.length() > 0) {
      rawSerial.trim();
      rawKPS_KS = rawSerial.toInt();

      // 0xFE = 0b11111110, 7 relevant key columns:
      updateBitmap(rawKPS_KS & 0xFE);

      // Displays are in off state, do not draw to them:
      if (off) { return; }

      // Only update Nokia LCD every 60ms, performance fix:
      if (drawIterations++ == 3) {
        drawIterations = 0;
        drawBitmap(true);
      }
      print16_2(rawKPS_KS);
      prev = millis();
      draw = true;
    }
  }
  // Clear displays if USB serial becomes inactive :: - -                            - -
```

```
  now = millis();
  if (draw && now - prev >= 500)
  {
    LCD.clear();
    LCD.println("KPS :: 0        ");
    LCD.print  ("                ");
    drawBitmap(false);
    draw = false;
    prev = now;
  }
} // - -                                                                    - -
```