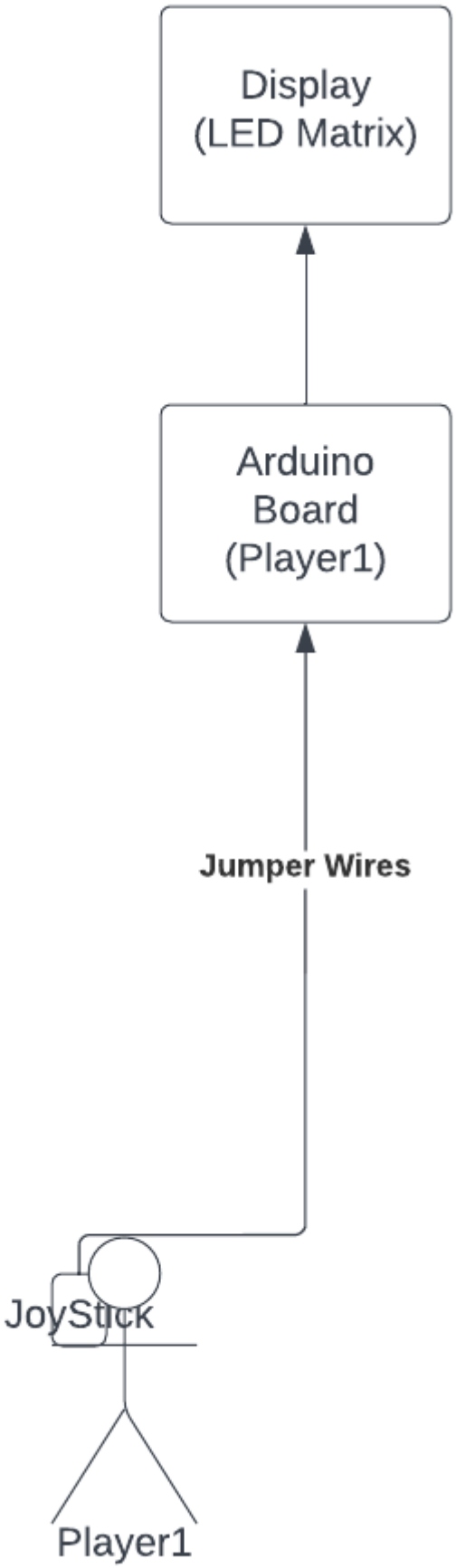
Tetris on a Matrix of LEDs

Introduction

Caruntu Vlad-Andrei 1222A This project is meant to recreate the all-famous game known as Tetris. It does this by displaying the game in a distinct and charming way, through LED lightups which the blocks and their falling across a 32×8 Matrix of LEDs. In case you don't know what the game consists of, here is a short description of it: Tetris is a puzzle game in which geometric shapes called “tetrominoes” fall down onto a playing field, and the player has to arrange them to form gapless lines.

General Description

The following is a diagram representing the overall idea for the project:


Display
(LED Matrix)
Arduino
Board
(Player1)
Jumper Wires
JoyStick
Player1

Hardware Design

Parts List: -8×8 LED Matrices -Joystick -Arduino Uno Board -several Jumper Wires connecting the other components together

Software Design

TETRIS by GEEKEE CEEBEE #include < LedControl.h > We always have to include the library #include “LedControl.h”

***Global Variables and Definitions** LedControl lc = LedControl(12, 11, 10, 4); const int SW_pin = 2;* digital pin connected to SW const int X_pin = A0; *analog pin connected to VRx const int Y_pin = A1; float r = 7.0;* row input int c = 3; *column input int add = 0;* address input int add_prev = 1000; int r_prev = 1000; int c_prev = 1000; int ro = 0; *to set block orientation int ro_prev; int c2; float r2; int check1, check2, check3, check4; int left_check1, left_check2, left_check3, left_check4; int right_check1, right_check2, right_check3, right_check4; int block = 0;* ***Right Translate Functions** int Right_translate(int c1, boolean state1) {

```
int X = analogRead(X_pin);
if (state1 == true && X < 200) {
  return c--;
}
else if (state1 == false && X < 200) {
  return c;
}
```

}

***Left Translate Functions** int Left_translate(int c1, boolean state1) { int X = analogRead(X_pin); if (state1 == true) { if (X > 800) { return c++; } } else if (state1 == false) { if (X > 800) { return c; } } }* ***Down Translate Functions** float Down_translate(float r1, boolean state1) {

```
int Y = analogRead(Y_pin);
//Serial.print(" Y_pin: ");
//        Serial.print(Y);
if (state1 == true ) {
  if (Y > 800) {
    return r--;
  }
}
else if (state1 == false) {
  if (Y > 800) {
    return   r;
  }
}
```

}

I BLOCK FUNCTION ***void I_block(int add, float r, int c, int ro) { switch (ro) { case 0: Horizontal Orientation if (ro_prev == 1) { lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev - 1, c_prev, false); lc.setLed(add_prev, r_prev - 2, c_prev, false); lc.setLed(add_prev, r_prev - 3, c_prev, false); } lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev, c_prev - 2, false); lc.setLed(add_prev, r_prev, c_prev - 3, false); lc.setLed(add, r, c, true); lc.setLed(add, r, c - 1, true); lc.setLed(add, r, c - 2, true); lc.setLed(add, r, c - 3, true); check1 = lc.getstatus(add, r - 1, c); check2 = lc.getstatus(add, r - 1, c - 1); check3 = lc.getstatus(add, r - 1, c - 2); check4 = lc.getstatus(add, r - 1, c - 3); left_check1 = lc.getstatus(add, r , c + 1); right_check1 = lc.getstatus(add, r , c - 4); add_prev = add; r_prev = r; c_prev = c; ro_prev = ro; break; case 1: Vertical Orientation if (ro_prev == 0) { lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev, c_prev - 2, false); lc.setLed(add_prev, r_prev, c_prev - 3, false); } lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev - 1, c_prev, false); lc.setLed(add_prev, r_prev - 2, c_prev, false); lc.setLed(add_prev, r_prev - 3, c_prev, false); lc.setLed(add, r, c, true); lc.setLed(add, r - 1, c, true); lc.setLed(add, r - 2, c, true); lc.setLed(add, r - 3, c, true); check1 = lc.getstatus(add, r - 4, c); left_check1 = lc.getstatus(add, r , c + 1); right_check1 = lc.getstatus(add, r , c - 1); add_prev = add; r_prev = r; c_prev = c; ro_prev = ro; break; } } S BLOCK FUNCTION void S_block(int add, float r, int c, int ro) { switch (ro) { case 0: Horizontal Orientation if (ro_prev == 1) { lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev - 1, c_prev - 1, false); lc.setLed(add_prev, r_prev + 1, c_prev, false); } lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev - 1, c_prev, false); lc.setLed(add_prev, r_prev - 1, c_prev + 1, false); lc.setLed(add, r, c, true); lc.setLed(add, r, c - 1, true); lc.setLed(add, r - 1, c, true); lc.setLed(add, r - 1, c + 1, true); check1 = lc.getstatus(add, r - 2, c); check2 = lc.getstatus(add, r - 2, c + 1); check3 = lc.getstatus(add, r - 1, c - 1); left_check1 = lc.getstatus(add, r , c + 2); right_check1 = lc.getstatus(add, r , c - 2); add_prev = add; r_prev = r; c_prev = c; ro_prev = ro; break; case 1: Vertical Orientation if (ro_prev == 0) { lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev - 1, c_prev, false); lc.setLed(add_prev, r_prev - 1, c_prev + 1, false); } lc.setLed(add_prev, r_prev, c_prev, false); lc.setLed(add_prev, r_prev, c_prev - 1, false); lc.setLed(add_prev, r_prev - 1, c_prev - 1, false); lc.setLed(add_prev, r_prev + 1, c_prev, false); lc.setLed(add, r, c, true); lc.setLed(add, r, c - 1, true); lc.setLed(add, r - 1, c - 1, true); lc.setLed(add, r + 1, c, true); check1 = lc.getstatus(add, r - 1, c); check2 = lc.getstatus(add, r - 2, c - 1); left_check1 = lc.getstatus(add, r , c + 1); right_check1 = lc.getstatus(add, r , c - 2); add_prev = add; r_prev = r; c_prev = c; ro_prev = ro; break; } } MAIN SETUP void setup() { we have already set the number of devices when we created the LedControl int devices = lc.getDeviceCount(); lc.line_break(); int che = lc.getstatus(add, r, c); pinMode(SW_pin, INPUT); digitalWrite(SW_pin, HIGH); randomSeed(analogRead(0)); Serial.begin(9600); we have to init all devices in a loop for (int address = 0; address < devices; address++) { lc.shutdown(address, false); lc.setIntensity(address, 8); lc.clearDisplay(address); } } * MAIN LOOP FUNCTION void loop() { block = random(-1, 2); Block selection by randomly picking value between 0 and 1 r = 8; Starting row for new block c = 3; starting column for new block add = 0; address is 0 if using only one 8×8 LED module switch (block) { * I_block Horizontal case 1: do { int X = analogRead(X_pin); int Y = analogRead(Y_pin); int Rot = digitalRead(SW_pin); if (Rot == LOW) { ro++; if (ro == 2) { ro = 0; } } if (c >= -1 && c ⇐ 8) { switch (ro) { case 0: if (c == 2 || c == 1 || c == 0) { c = 3; } if (c == 8) { c = 7; } if (left_check1 == 1 &&***

right_check1 == 0) { c = Right_translate(c, true); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, true); Down_translate(r, true); } else if (left_check1 == 1 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 0) { c2 = Right_translate(c, true); c2 = Left_translate(c, true); r2 = Down_translate(r, true); } break; * I block_Vertical case 1: if (c == -1) { c = 0; } if (c == 8) { c = 7; } if (left_check1 == 1 && right_check1 == 0) { c = Right_translate(c, true); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, true); Down_translate(r, true); } else if (left_check1 == 1 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 0) { c2 = Right_translate(c, true); c2 = Left_translate(c, true); r2 = Down_translate(r, true); } break; } } r = r - 0.125; I_block(add, r, c, ro); if (check1 == 1 || check2 == 1 || check3 == 1) { r = 0.8; } delay(225); } while (r > 0.9); add_prev = 100; r_prev = 100; c_prev = 100; break; *S_Block case* case 0: do { int Rot = digitalRead(SW_pin); if (Rot == LOW) { ro++; if (ro == 2) { ro = 0; } } if (c >= 0 && c ⇐ 7) { switch (ro) { case 0: if (c == 0) { c = 1; } if (c == 7) { c = 6; } if (left_check1 == 1 && right_check1 == 0) { c = Right_translate(c, true); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, true); Down_translate(r, true); } else if (left_check1 == 1 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 0) { c2 = Right_translate(c, true); c2 = Left_translate(c, true); r2 = Down_translate(r, true); } break; *S block case 1: if (c == -1) { c = 0; } if (c == 8) { c = 7; } if (left_check1 == 1 && right_check1 == 0) { c = Right_translate(c, true); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, true); Down_translate(r, true); } else if (left_check1 == 1 && right_check1 == 1) { c = Right_translate(c, false); Left_translate(c, false); Down_translate(r, true); } else if (left_check1 == 0 && right_check1 == 0) { c2 = Right_translate(c, true); c2 = Left_translate(c, true); r2 = Down_translate(r, true); } break; } } r = r - 0.125; S_block(add, r, c, ro); if (check1 == 1 || check2 == 1 || check3 == 1) { r = 1.8; } delay(225); } while (r > 1.9); add_prev = 100; r_prev = 100; c_prev = 100; break; } lc.line_break(); Line Break function called from LED control libraries lc.gameover(); Game Over function called from LED control libraries }