Numele proiectului: Jocul Tetris

Autor: Raluca-Simona Iordache

Scop: Dezvoltarea unui joc interactiv bazat pe regulile clasice ale Tetrisului, unde jucătorul interacționează cu o matrice și piese pentru a construi linii complete și a evita umplerea ecranului de joc. Scopul principal este de a oferi o experiență distractivă și captivantă pentru utilizator, păstrând esența și familiaritatea cu jocul Tetris.

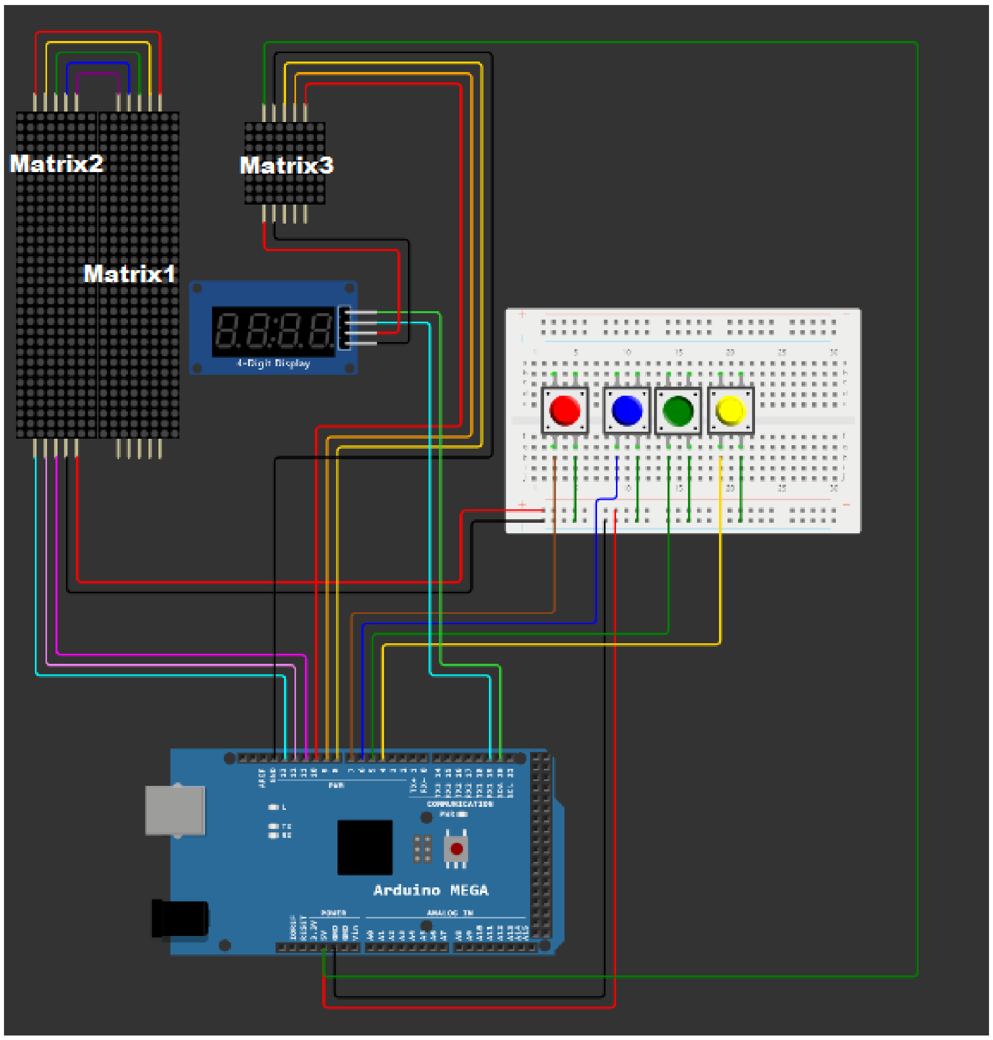
Piese folosite:

- 1x Arduino Mega
- 2x LED Matrix: 32x8
- 1x LED Matrix: 8x8
- 1x Half-size Breadboard
- 4x Pushbutton
- 1x Seven Segment Display (TM1637)

Descrierea proeictului: În Tetris, jucătorii își completează liniile prin mișcarea pieselor cu forme variate care coboară în spațiul de joc. Liniile completate dispar, oferind jucătorului puncte (10 puncte pentru fiecare linie eliminată), iar acesta poate progresa prin ocuparea spațiilor libere. Jocul se încheie atunci când ecranul este complet ocupat. Cu cât jucătorul reușește să amâne umplerea ecranului, cu atât scorul său va fi mai mare. Scorul este afișat în timp real pe un display alăturat, încurajând astfel jucătorul să continue să manevreze piesele pentru a obține un scor cât mai mare. Un ecran mai mic furnizează informații utile, afișând următoarea piesă care urmează să "cadă".

În cele ce urmează am atașat o imagine, plus câteva explicații pentru a vă ghida în ceea ce privește conectarea pieselor hardware:

- Matrix3:V+ -> mega:5V
- Matrix3:CLK -> mega:10
- Matrix2:CS -> mega:9
- Matrix2:DIN -> mega:8
- Matrix3:V+.2 -> sevseg1:VCC
- Matrix3:GND.2 -> sevseg1:GND
- Sevseg1:CLK -> mega:20
- Sevseg1:DIO -> mega:19



Codul:

```
#include <LedControl.h>
#include <TM1637Display.h>

#define CLK 13
#define CS  12
#define DIN 11

#define PIN_CLK 10
#define PIN_CS  9
#define PIN_DIN 8

#define DISPLAY_PIN_CLK 20
#define DISPLAY_PIN_DIO 19

#define BUTTON_ROTATE 7
#define BUTTON_LEFT 6
```

```
#define BUTTON_DOWN 5
#define BUTTON_RIGHT 4

LedControl lc=LedControl(DIN, CLK, CS, 8);
LedControl lc1=LedControl(PIN_DIN, PIN_CLK, PIN_CS, 2);

TM1637Display display = TM1637Display(DISPLAY_PIN_CLK, DISPLAY_PIN_DIO);
int score = 0;

const int rows = 32;
const int cols = 16;
int board[rows][cols];

const int PIECE_SIZE = 4; // Dimensiunea matricei pentru o piesa
int currentPiece[PIECE_SIZE][PIECE_SIZE];

// Pozitionare initiala a piesei in partea de sus pe mijloc
int initialY = 0;  // Partea de sus a tablei de joc
int initialX = cols / 2 - 1;  // Centrul ecranului

// Definire coordonate curente ale piesei
int currentPieceY = initialY;
int currentPieceX = initialX;

int nextPiece[PIECE_SIZE][PIECE_SIZE];

int randomIndex;

int maxHeight;
int maxWidth;

int Y;
int X;

// Forme pentru piese Tetris
const int tetrisShapes[][PIECE_SIZE][PIECE_SIZE] = {
  {
    { 1, 1, 1, 1 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 1, 1, 0, 0 },
    { 1, 1, 0, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 0, 1, 0, 0 },
    { 1, 1, 1, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 1, 0, 0, 0 },
    { 1, 1, 1, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 0, 0, 1, 0 },
    { 1, 1, 1, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 1, 1, 0, 0 },
    { 0, 1, 1, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
  {
    { 0, 1, 1, 0 },
    { 1, 1, 0, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }
  },
};

void setup() {
  lc.shutdown(0, false);
  lc.setIntensity(0, 8);
  lc.clearDisplay(0);

  display.clear();
  display.setBrightness(10);

  //Serial.begin(9600);

  pinMode(BUTTON_ROTATE, INPUT_PULLUP);
  pinMode(BUTTON_LEFT, INPUT_PULLUP);
  pinMode(BUTTON_DOWN, INPUT_PULLUP);
  pinMode(BUTTON_RIGHT, INPUT_PULLUP);

  for (int i = 0; i < rows; i++){
    for (int j = 0; j < cols; j++){
      board[i][j] = 0;
    }
  }

  choosePiece();
  chooseNextPiece();
}
```

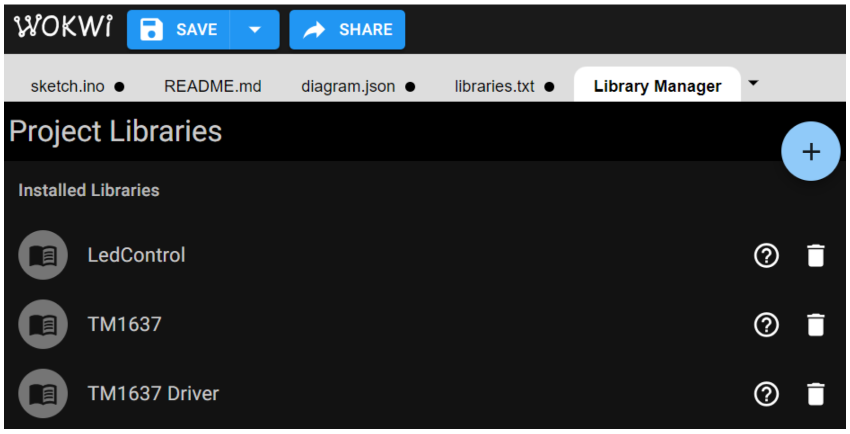
```
// Functia show transforma matricea board de 16x32 in 8 matrici de 8x8
// ca sa pot sa le afisez pe "ecranul" alaturat
void show(){
  int k;
  int x;
  int y;

  for(int i = 0; i < rows; i++){
    for(int j = 0; j < cols; j++){
      if(j <= 7){
        if(i <= 7)
          k = 3;
        else if(i <= 15)
          k = 2;
        else if(i <= 23)
          k = 1;
        else if(i <= 31)
          k = 0;
        x = 7 - j;
        y = 7 - i % 8;
      }
      else if(j <= 15){
        if(i <= 7)
          k = 4;
        else if(i <= 15)
          k = 5;
        else if(i <= 23)
          k = 6;
        else if(i <= 31)
          k = 7;
        x = j - 8;
        y = i % 8;
      }
      lc.setLed(k, x, y, board[i][j]);
    }
  }
}

void choosePiece(){
  // Generare piesei
  randomIndex = random(0, 7);
  memcpy(currentPiece, tetrisShapes[randomIndex], sizeof(currentPiece));

  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if(currentPiece[i][j] != 0)
        board[initialY + i][initialX + j] = currentPiece[i][j];
    }
  }
  calculatePieceHeight(currentPiece);
  calculatePieceWidth(currentPiece);
```

```
}

void chooseNextPiece(){
  randomIndex = random(0, 7);

  memcpy(nextPiece, tetrisShapes[randomIndex], sizeof(nextPiece));

  calculatePieceHeight(nextPiece);
  calculatePieceWidth(nextPiece);


  Y = (8 - maxHeight) / 2;
  X = (8 - maxWidth) / 2;

  // Plasare piesei în matricea mica
  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if(nextPiece[i][j] == 1)
        lc1.setLed(0, X + j, Y + i, 1);
    }
  }
}

void takeTheNextPiece(){
  memcpy(currentPiece, nextPiece, sizeof(currentPiece));

  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if (nextPiece[i][j] == 1) {
        lc1.setLed(0, X + j, Y + i, 0);
      }
    }
  }
  chooseNextPiece();
}

int calculatePieceHeight(int piece[PIECE_SIZE][PIECE_SIZE]) {
  maxHeight = 0;
  for (int j = 0; j < PIECE_SIZE; j++) {
    int currentHeight = 0;

    for (int i = 0; i < PIECE_SIZE; i++) {
      if (piece[i][j] == 1) {
        currentHeight++;
      }
    }

    if (currentHeight > maxHeight) {
      maxHeight = currentHeight;
    }
  }
  //Serial.print("H: ");
  //Serial.println(maxHeight);
  return maxHeight;
}

int calculatePieceWidth(int piece[PIECE_SIZE][PIECE_SIZE]) {
  maxWidth = 0;
  for (int i = 0; i < PIECE_SIZE; i++) {
    int currentWidth = 0;

    for (int j = 0; j < PIECE_SIZE; j++) {
      if (piece[i][j] == 1) {
        currentWidth++;
      }
    }

    if (currentWidth > maxWidth) {
      maxWidth = currentWidth;
    }
  }
  //Serial.print("W: ");
  //Serial.println(maxWidth);
  return maxWidth;
}

void erasePiece() {
  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if (currentPiece[i][j] == 1) {
        board[currentPieceY + i][currentPieceX + j] = 0;
      }
    }
  }
}

void drawPiece() {
  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if (currentPiece[i][j] == 1) {
        board[currentPieceY + i][currentPieceX + j] = 1;
      }
    }
  }
}

// Functie pentru a verifica coliziunea piesei cu partea de jos sau cu alte piese
bool hasCollision(int x, int y) {
  for (int i = 0; i < PIECE_SIZE; i++) {
    for (int j = 0; j < PIECE_SIZE; j++) {
      if (currentPiece[i][j] == 1 && (y + i >= rows || board[y + i][x + j] == 1)) {
        //lc.setLed(3, 7, 7, 1);
        return true;
      }
    }
  }
  return false;
}

// Functie pentru a roti o matrice 4x4 în sensul acelor de ceasornic
void rotateMatrix(int mat[PIECE_SIZE][PIECE_SIZE]) {
  for (int i = 0; i < PIECE_SIZE / 2; i++) {
    for (int j = i; j < PIECE_SIZE - i - 1; j++) {
      int temp = mat[i][j];
      mat[i][j] = mat[PIECE_SIZE - 1 - j][i];
      mat[PIECE_SIZE - 1 - j][i] = mat[PIECE_SIZE - 1 - i][PIECE_SIZE - 1 - j];
      mat[PIECE_SIZE - 1 - i][PIECE_SIZE - 1 - j] = mat[j][PIECE_SIZE - 1 - i];
      mat[j][PIECE_SIZE - 1 - i] = temp;
    }
  }

  int zeroOnLine = 4;
  while(zeroOnLine == 4){
    zeroOnLine = 0;
    for (int j = 0; j < PIECE_SIZE; j++) {
      if(mat[0][j] == 0){
        zeroOnLine++;
      }
    }
    if(zeroOnLine == 4){

      for (int i = 0; i < PIECE_SIZE - 1; i++) {
        for (int j = 0; j < PIECE_SIZE; j++) {
          mat[i][j] = mat[i+1][j];
        }
      }
      for (int j = 0; j < PIECE_SIZE; j++) {
        mat[3][j] = 0;
      }
    }
  }

  int zeroOnColumn = 4;
  while(zeroOnColumn == 4){
    zeroOnColumn = 0;
    for (int i = 0; i < PIECE_SIZE; i++) {
      if(mat[i][0] == 0){
        zeroOnColumn++;
      }
    }
    if(zeroOnColumn == 4){
      for (int j = 0; j < PIECE_SIZE - 1; j++) {
        for (int i = 0; i < PIECE_SIZE; i++) {
          mat[i][j] = mat[i][j+1];
        }
      }
      for (int i = 0; i < PIECE_SIZE; i++) {
        mat[i][3] = 0;
      }
    }
  }
}

void rotatePiece() {
  erasePiece();

  rotateMatrix(currentPiece);

  if (!hasCollision(currentPieceX, currentPieceY)) {
    drawPiece();
  } else {
    rotateMatrix(currentPiece);
    rotateMatrix(currentPiece);
    rotateMatrix(currentPiece);
    drawPiece();
  }

  calculatePieceHeight(currentPiece);
  calculatePieceWidth(currentPiece);
}

void movePieceLeft() {
  erasePiece();
  if (!hasCollision(currentPieceX - 1, currentPieceY) && currentPieceX != 0) {
    currentPieceX--;
  }
  drawPiece();
}

void movePieceRight() {
  erasePiece();
  if (!hasCollision(currentPieceX + 1, currentPieceY) && currentPieceX != cols - 1 - maxWidth + 1) {
    currentPieceX++;
  }
  drawPiece();
}

void movePieceDown() {
    erasePiece();
  if (!hasCollision(currentPieceX, currentPieceY + 4)) {
    currentPieceY = currentPieceY + 4;
  }
    drawPiece();
}

void displayAndMovePiece() {
  erasePiece();
  if (hasCollision(currentPieceX, currentPieceY + 1)) {
    drawPiece();
    //show();
    isGameOver();
    currentPieceY = initialY;
    currentPieceX = initialX;
    takeTheNextPiece();
  } else {
    currentPieceY++;
    drawPiece();
  }
}

void deleteFullLine(){
  for(int i = rows - 1; i >= 0; i--){
    int numberOfZeros = 0;
    for(int j = cols - 1; j >= 0; j--){
      if(board[i][j] == 0){
        numberOfZeros++;
      }
    }
    if(numberOfZeros == 0){
      for(int j = cols - 1; j >= 0; j--){
        board[i][j] = 0;
      }

      score = score + 10;
      display.showNumberDec(score);

      delay(1000);
      show();

      currentPieceY = initialY;
      currentPieceX = initialX;

      for(int k = i; k > 0; k--){
        for(int j = cols - 1; j >= 0; j--){
          board[k][j] = board[k-1][j];
        }
      }
      takeTheNextPiece();
      show();
    }
  }
}

void youLose(){
  board[3][1] = 1;
  board[4][1] = 1;
  board[5][2] = 1;
  board[6][2] = 1;
  board[7][3] = 1;
  board[8][3] = 1;
```

```
board[9][3] = 1;
board[10][3] = 1;
board[5][4] = 1;
board[6][4] = 1;
board[3][5] = 1;
board[4][5] = 1;

board[3][8] = 1;
board[10][8] = 1;
board[3][11] = 1;
board[3][13] = 1;
board[10][12] = 1;

board[25][0] = 1;
board[25][1] = 1;
board[25][2] = 1;
board[25][5] = 1;
board[17][0] = 1;
board[17][5] = 1;
board[17][12] = 1;
board[17][13] = 1;
board[17][14] = 1;
board[21][13] = 1;
board[25][12] = 1;
board[25][13] = 1;
board[25][14] = 1;

//S
board[17][9] = 1;
board[21][9] = 1;
board[25][9] = 1;
board[18][10] = 1;
board[18][8] = 1;
board[19][8] = 1;
board[20][8] = 1;
board[22][10] = 1;
board[23][10] = 1;
board[24][10] = 1;
board[24][8] = 1;

for(int i = 4; i <= 9; i++){
  board[i][7] = 1;
  board[i][9] = 1;
  board[i][11] = 1;
  board[i][13] = 1;
}

for(int i = 18; i <= 24; i++){
  board[i][0] = 1;
  board[i][4] = 1;
  board[i][6] = 1;
  board[i][12] = 1;
  }

  show();
}

void isGameOver(){
  if(hasCollision(initialX, initialY)){
    for(int i = 0; i < rows; i++){
      for(int j = 0; j < cols; j++){
        board[i][j] = 0;
      }
    }
    show();

    youLose();
    delay(120000);
  }
}

void loop() {
  show();

  if (digitalRead(BUTTON_ROTATE) == LOW) {
    rotatePiece();
  }
  if (digitalRead(BUTTON_LEFT) == LOW) {
    movePieceLeft();
  }
  if (digitalRead(BUTTON_DOWN) == LOW) {
    movePieceDown();
  }
  if (digitalRead(BUTTON_RIGHT) == LOW) {
    movePieceRight();
  }

  displayAndMovePiece();
  show();
  deleteFullLine();
}
```

Diagram.json: În cazul în care doriți să testați jocul am atașat si fișierul json și o imagine cu bibilotecile necesare, având în vedere că proiectul a fost realizat în wokwi.

```
{
  "version": 1,
  "author": "Raluca",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-breadboard-half", "id": "bb1", "top": -281.4, "left": 377.2, "attrs": {} },
    { "type": "wokwi-arduino-mega", "id": "mega", "top": 125.4, "left": 44.4, "attrs": {} },
{
  "type": "wokwi-max7219-matrix",
  "id": "matrix1",
  "top": -349.84,
  "left": -131.4,
  "rotate": 270,
  "attrs": { "chain": "4" }
},
{
  "type": "wokwi-max7219-matrix",
  "id": "matrix2",
  "top": -350.16,
  "left": -208.6,
  "rotate": 90,
  "attrs": { "chain": "4" }
},
{
  "type": "wokwi-pushbutton",
  "id": "btn1",
  "top": -211.3,
  "left": 550.7,
  "rotate": 90,
  "attrs": { "color": "yellow" }
},
{
  "type": "wokwi-pushbutton",
  "id": "btn2",
  "top": -211.1,
  "left": 505.5,
  "rotate": 270,
  "attrs": { "color": "green" }
},
{
  "type": "wokwi-pushbutton",
  "id": "btn3",
  "top": -211.3,
  "left": 454.7,
  "rotate": 90,
  "attrs": { "color": "blue" }
},
{
  "type": "wokwi-pushbutton",
  "id": "btn4",
  "top": -211.3,
  "left": 397.1,
  "rotate": 90,
  "attrs": { "color": "red" }
},
{
  "type": "wokwi-max7219-matrix",
  "id": "matrix3",
  "top": -453.51,
      "left": 116.5,
      "rotate": 270,
      "attrs": { "chain": "1" }
    },
    {
      "type": "wokwi-tm1637-7segment",
      "id": "sevseg1",
      "top": -307.24,
      "left": 83.83,
      "attrs": { "color": "red" }
    }
  ],
  "connections": [
    [ "matrix2:CLK", "mega:13", "cyan", [ "v201.6", "h183" ] ],
    [ "matrix2:CS", "mega:12", "violet", [ "v192", "h182.4" ] ],
    [ "matrix2:DIN", "mega:11", "magenta", [ "v182.4", "h182.8" ] ],
    [ "matrix2:V+.2", "matrix1:V+", "purple", [ "v-19.04", "h28.8" ] ],
    [ "matrix2:GND.2", "matrix1:GND", "blue", [ "v-28.64", "h38.4" ] ],
    [ "matrix2:DOUT", "matrix1:DIN", "green", [ "v-38.24", "h38.4" ] ],
    [ "matrix2:CS.2", "matrix1:CS", "gold", [ "v-47.84", "h57.6" ] ],
    [ "matrix2:CLK.2", "matrix1:CLK", "red", [ "v-57.44", "h105.6" ] ],
    [ "bb1:5b.h", "bb1:bn.4", "green", [ "v0" ] ],
    [ "bb1:11b.h", "bb1:bn.9", "green", [ "v0" ] ],
    [ "bb1:16b.h", "bb1:bn.13", "green", [ "v0" ] ],
    [ "bb1:21b.h", "bb1:bn.17", "green", [ "v0" ] ],
    [ "bb1:bp.7", "mega:5V", "red", [ "v459.9", "h-269.5" ] ],
    [ "bb1:bn.6", "mega:GND.2", "black", [ "v440.3", "h-250.15" ] ],
    [ "bb1:3b.h", "mega:7", "#8f4814", [ "v144", "h-192" ] ],
    [ "bb1:9b.h", "mega:6", "blue", [ "v38.4", "h-19.2", "v115.2", "h-192" ] ],
    [ "bb1:14b.h", "mega:5", "green", [ "v163.2", "h-182.4" ] ],
    [ "bb1:19b.h", "mega:4", "gold", [ "v172.8", "h-220.8" ] ],
    [ "matrix2:GND", "bb1:bn.1", "black", [ "v124.8", "h374.4", "v-65.9" ] ],
    [ "matrix2:V+", "bb1:bp.1", "red", [ "v115.2", "h355.2", "v-66.3" ] ],
    [ "matrix3:CLK", "mega:10", "red", [ "v-19.2", "h144", "v316.8", "h-134.1" ] ],
    [ "matrix3:CS", "mega:9", "orange", [ "v-28.8", "h163.2", "v336", "h-133.7" ] ],
    [ "matrix3:DIN", "mega:8", "gold", [ "v-38.4", "h182.4", "v355.2", "h-133.8" ] ],
    [ "matrix3:GND", "mega:GND.1", "black", [ "v-48", "h201.6", "v374.4", "h-201.4" ] ],
    [ "btn1:1.l", "bb1:21t.b", "", [ "$bb" ] ],
    [ "btn1:2.l", "bb1:19t.b", "", [ "$bb" ] ],
    [ "btn1:1.r", "bb1:21b.g", "", [ "$bb" ] ],
    [ "btn1:2.r", "bb1:19b.g", "", [ "$bb" ] ],
    [ "btn2:1.l", "bb1:14b.g", "", [ "$bb" ] ],
    [ "btn2:2.l", "bb1:16b.g", "", [ "$bb" ] ],
    [ "btn2:1.r", "bb1:14t.b", "", [ "$bb" ] ],
    [ "btn2:2.r", "bb1:16t.b", "", [ "$bb" ] ],
    [ "btn3:1.l", "bb1:11t.b", "", [ "$bb" ] ],
    [ "btn3:2.l", "bb1:9t.b", "", [ "$bb" ] ],
    [ "btn3:1.r", "bb1:11b.g", "", [ "$bb" ] ],
    [ "btn3:2.r", "bb1:9b.g", "", [ "$bb" ] ],
    [ "btn4:1.l", "bb1:5t.b", "", [ "$bb" ] ],
    [ "btn4:2.l", "bb1:3t.b", "", [ "$bb" ] ],
    [ "btn4:1.r", "bb1:5b.g", "", [ "$bb" ] ],
    [ "btn4:2.r", "bb1:3b.g", "", [ "$bb" ] ],
    [ "sevseg1:CLK", "mega:20", "limegreen", [ "h57.6", "v326.4", "h48" ] ],
    [ "sevseg1:DIO", "mega:19", "cyan", [ "h48", "v326.4", "h48", "v0", "h7.7" ] ],
    [ "matrix3:V+.2", "sevseg1:VCC", "red", [ "v25.01", "h124.8", "v76.8" ] ],
    [ "matrix3:GND.2", "sevseg1:GND", "black", [ "v15.41", "h124.8", "v96" ] ],
    [ "matrix3:V+", "mega:5V", "green", [ "v-57.6", "h604.8", "v336", "h0", "v528", "h-528" ] ]
  ],
  "dependencies": {}
}
```



Project Libraries

Installed Libraries

- LedControl
- TM1637
- TM1637 Driver