Tetris Game Using an 8x8 LED Matrix

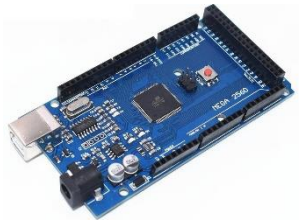
Name: Bîrlădeanu Alexandru-Gabriel

Group: 302310

Date: 22/12/2021

1. Components and Supplies

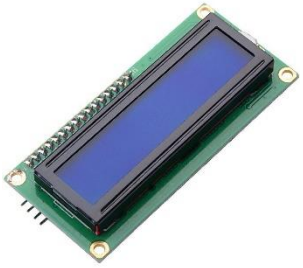
1 x Arduino Mega



1 x 8x8 Led Matrix

1 x 3x4 Keypad

1 x 16x2 Lcd Display



12 x Male/Female Jumper Wires

2. Purpose of the Project

The purpose of this project is to implement the *Tetris* game using an 8x8 led matrix to display the blocks, a 16x2 lcd display to show the current score and the buttons on a 3x4 keypad to move the blocks while playing.

3. Hardware Assembly and Setup

The led matrix, the keypad and the lcd display each require a library which must be downloaded and included in the *.ino* file.

```
#include <MaxMatrix.h>
#include <Keypad.h>
#include <LiquidCrystal.h>
```

\

The lcd pins must be declared here:

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
```

The keypad pins must be declared like this:

```
const byte ROWS = 4;
const byte COLS = 3;
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
byte rowPins[ROWS] = {49, 47, 45, 43};
byte colPins[COLS] = {41, 39, 37};
Keypad keypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
```

4. Code (data structures and functions used)

The global variables used can be seen below.

The *BlockComponent* structure represents a single dot in the 8x8 matrix, as each falling block, represented by the global variable *currentBlock*, will represent an array of *BlockComponent* structures.

We will use *startingTime* and *currentTime* for the *millis*() function, in order to move the current block one row down once a second.

The bidimensional array *occupiedSpots* represents the entire 8x8 matrix and each element has the value 0 if the led on the dot on the corresponding line and column is off, or 1 if the led is on. We will gradually assign 1 to the specific dots containing blocks which have reached the ground.

The variable *blockType* will be used to store a random value between 0 and 6 and thus decide the type of the block which will spawn next.

The variable *rotationIndex* will be incremented each time the player presses the button to rotate the falling block and, by computing the value of (*rotationIndex* % 4), the next position of the block will be determined.

The score will be incremented by 10 each time a block successfully reaches the ground and by 100 each time a full row is occupied by fallen blocks.

```
typedef struct BlockComponent{
  int x = -1;
  int y = -1;
}Point;
unsigned long startingTime, currentTime;
unsigned long secondsPassed = 1;
int currentBlockComponents = 4;
bool currentBlockReachedGround = false;
Point* currentBlock;
int occupiedSpots[8][8] = {0};
int blockType;
int rotationIndex;
int score=0;
```

```
void setup() {
 lcd.begin(16, 2);
 lcd.setCursor(0,0);
 lcd.print("YOUR SCORE: ");
 lcd.setCursor(0,1);
 lcd.print(score);
 m.init();
 m.setIntensity(8);
 keypad.addEventListener(keypadEvent);
 Serial.begin(9600);
 startingTime = millis();
 createBlock();
 setCurrentBlockCoordinates(3, -1);
}
```

In the *setup*() function we initialize the lcd display and print the score, initially 0. We also initialize the 8x8 matrix m, set the intensity of the leds, add an event listener to the keypad which will listen and call the appropriate function each time we press a button. The function for the keypad listener can be seen below. The initialization of the serial monitor is optional and is used just for debugging purposes. The function *createBlock*() initializes the array of *BlockComponents* which we named *currentBlock* and assigns a random value from 0 to 6 to *blockType*. The function *setCurrentBlockCoordinates*() sets the coordinates of each *BlockComponent* depending on the *blockType* and the *rotationIndex* and will also be called each time the player presses the rotation button. The coordinates of each block component are relative to the coordinates of the block's center, which are passed as parameters. All 7 blocks, in all 4 positions depending on *rotationIndex*, are hard coded, as seen below.

```
void keypadEvent(KeypadEvent key) {
  if(keypad.getState()==PRESSED){
    if (key == '6') {
      moveCurrentBlockRight();
    }
    if (key == '4') {
      moveCurrentBlockLeft();
    }
    if (key == '8') {
      fallOneRow();
    }
    if (key == '5') {
      rotateCurrentBlock();
    }
  } else {
    if(keypad.getState()==HOLD && key == '8') {
      while(!currentBlockReachedGround){
        fallOneRow();
      }
    }
  }
}
```

```
void setCurrentBlockCoordinates(int xCenter, int yCenter){
  switch(blockType) {
    case 0:
      if(rotationIndex % 4 == 0){
        currentBlock[0].x = xCenter;          // X
        currentBlock[0].y = yCenter;          // X
        currentBlock[1].x = xCenter;          // X
        currentBlock[1].y = yCenter-1;        // X
        currentBlock[2].x = xCenter;
        currentBlock[2].y = yCenter+1;
        currentBlock[3].x = xCenter;
        currentBlock[3].y = yCenter+2;
      }
      if(rotationIndex % 4 == 1){
        currentBlock[0].x = xCenter;          // X X X X
        currentBlock[0].y = yCenter;
        currentBlock[1].x = xCenter+1;
        currentBlock[1].y = yCenter;
        currentBlock[2].x = xCenter-1;
        currentBlock[2].y = yCenter;
        currentBlock[3].x = xCenter-2;
        currentBlock[3].y = yCenter;
      }
      if(rotationIndex % 4 == 2){
```

In the *loop*() function we call the *millis*() function in order to determine whenever a second passes. If the game is over, meaning a block topples another block when it has not fully entered the playable area, all leds turn on and there is nothing else the player can do. The function *drawCurrentBlock*() turns on the leds on which the current block is, but does not assign 1 to the corresponding position in the matrix *occupiedSpots*, as that is only for blocks that have reached ground, not for the block which is falling,

```
void loop() {
 currentTime = millis();
 keypad.getKey();
 if(gameIsOver() == true) {
  for(int columnIndex = 0; columnIndex<8; columnIndex++){
    m.setColumn(columnIndex, B11111111);
  }
  return;
 }
 drawCurrentBlock();
 if((currentTime - startingTime) / 1000 > secondsPassed){
  secondsPassed++;
  if(currentBlockReachedGround){
    score+=10;
    checkIfAnyRowIsFullyOccupied();
    createBlock();
    setCurrentBlockCoordinates(3, -1);
    currentBlockReachedGround = false;
    lcd.setCursor(0,0);
    lcd.print("YOUR SCORE: ");
    lcd.setCursor(0,1);
    lcd.print(score);
  } else {
    fallOneRow();
  }
 }
```