Technical University of Cluj-Napoca
Faculty of Automation and Computer Science

Arduino-based Tetris Game

Diaconu Călin
Group: 30432

Table of Contents

Project Purpose and Objectives

This is the project marking the end of the “Design with Microprocessors” lab sessions. Thus, the purpose of the project is to apply the theoretical and practical skills obtained during the semester.

It is an implementation of the Tetris game on a 10x20 board, obtaining a device that can be used for entertainment.

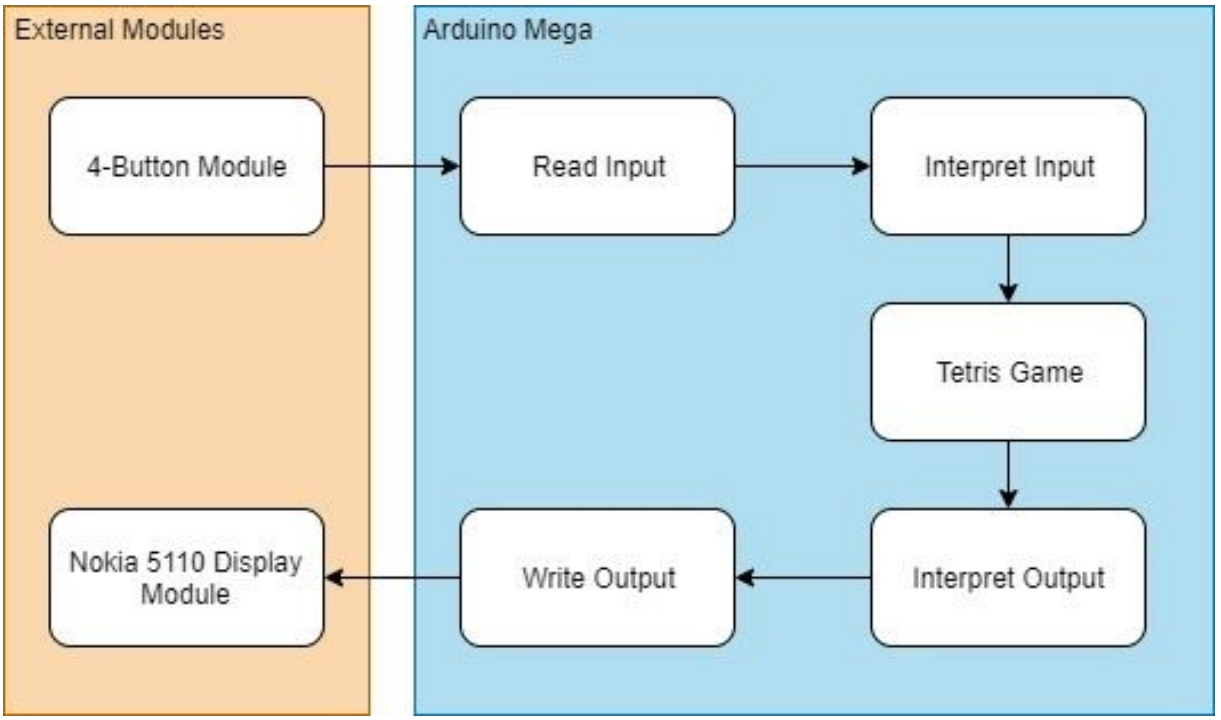
The microcontroller used is an Arduino Mega board, as an Arduino Uno’s smaller memory wouldn’t suffice because of the matrices used for the game and for displaying. The screen used is similar to the ones present on Nokia 5110 phones, with an 84x48 resolution, but mounted on a printed circuit board (PCB) and adapted to using it with an Arduino board. A module containing 4 buttons is also needed for the correct functioning of the project.

The expected result is a Tetris game playable by the standard rules:

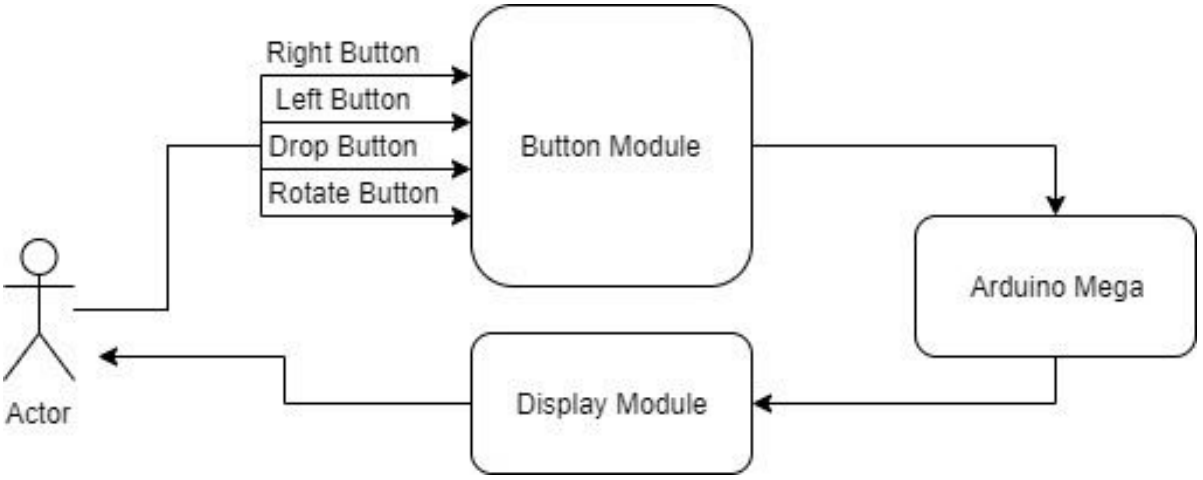
- a random piece, from the set of 7 possible pieces (officially called tetriminos), is generated in the top of the screen
- the generated tetrimino starts falling to the bottom of the screen, while the user can move it to the left or to the right, in the limits of the screen, they can accelerate its fall or rotate it
- the objective of the game is to fill as many rows as possible and avoid filling the columns in such a way that it would make impossible the creation of a new tetrimino

Analysis and Theoretical Foundation

Block diagram:



Use cases:



When the board is powered on, the game begins. Now, it accepts input from the user. They can press one of the 4 buttons:

- the Right Button, which translates the tetrimino to the right
- the Left Button, which translates the tetrimino to the left
- the Drop Button, which increases the tetrimino's falling speed
- the Rotate Button, which rotates the tetrimino to the left

The purpose of the game is to fill as many rows as possible and avoid blocking the creation of a new tetrimino by filling the board. For every row filled, the score increases with 1. If multiple rows are completed at once, the user receives a bonus (1 for 2 lines, 2 for 3 lines and 3 for 4 lines). At every 5 points, up until 45 points, the falling speed of the tetrimino increases by a factor of 1.11.

The game is over when there is not enough space left for creating a new tetrimino (the columns where the tetrimino should be created are occupied by previously placed blocks). When this happens, a "GAME OVER" message is sent through serial communication to the Serial Port, together with the score, and the game automatically restarts after 3 seconds.

Communication protocols:

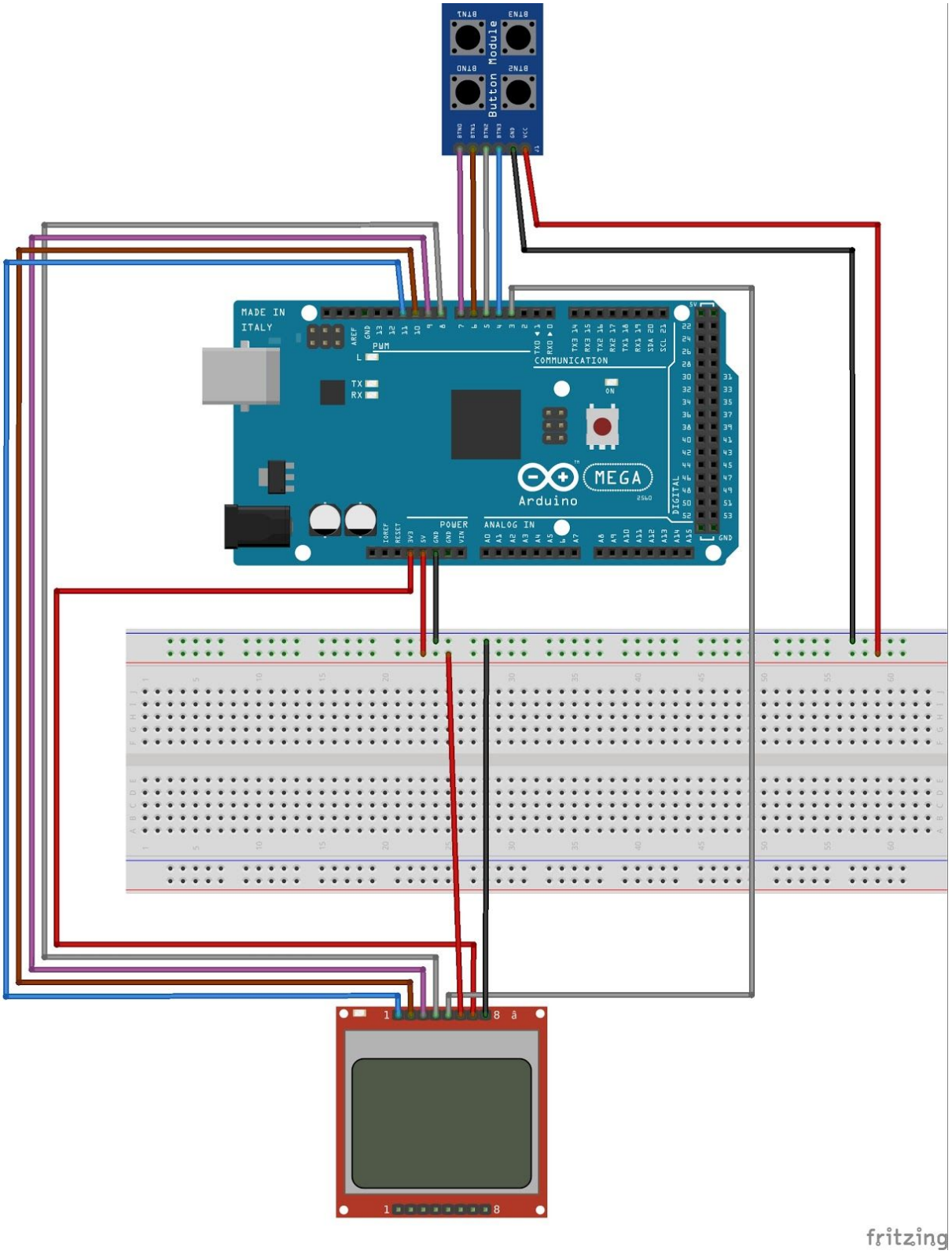
The only kind of communication used in this project is serial communication and it is used to display messages on the Serial Port. The baud rate is 9600.

Algorithms developed:

- An algorithm for interpreting the matrix of booleans representing the game board for the array used to write to the display - data written on the screen is stored into an array of bytes, where each byte represents 8 pixels on the screen (one bit for each pixel, turning it on or off)
- The actual game algorithm (adapted from an algorithm developed for a 16x8 board), which includes functions for checking if a tetrimino can move to a certain place, a function that randomly generates the new tetrimino to be creating (using a seed based on analog reads from pins 1 to 4), rotation and translation functions for the tetrimino, two functions for updating the board and the falling speed and a function that checks if the game is over

Detailed Design and Implementation

Hardware:



Software:

The software of the application is divided into three parts: the algorithms needed for the display, the algorithms needed for the 4-button module and the algorithms needed for the actual Tetris game.

1. Display module functions
    - *lcdInitialise()* - initialisation code for the display, that should be run once, in the setup()
    - *lcdLoop()* - code for the display that should run in the loop()
    - *lcdWrite(byte dc, byte data)* - function that writes one byte of code on the screen (the place on the screen doesn't need to be specified, as the data is taken in order of sending)
    - *squareToData()* - function to transform the data stored in the square matrix (a matrix of booleans containing information about the game table - whether there is a block in a certain place or not) into data for the array used to write on the display (bytes with information for the pixels, taking into account the borders and the space between the blocks)
2. Game functions
    - *arrangedBlocksToSquare()* - function to transform the data stored as integers in the arrangedBlocks matrix (where 0 represents no block in that place) into data for the square matrix (a matrix of booleans, as previosuly described)
    - *check_space()* - function to check whether a tetrimino can be moved to a certain place (checks in every direction - left, right and down, but also upward for when a tetrimino needs to be rotated)
    - *set_arrays_zero* - function used to initialise a matrix to zero
    - *newBlock()* - function that generates a new tetrimino (picked randomly from the set of 7 possible shapes)
    - *rotate()* - function for rotating the tetrimino
    - *translate_right(), translate_down(), translate_left()* - functions for translating the tetrimino into one of the three directions
    - *updating_arranged_blocks()* - function that deletes the filled rows, translates down the blocks in accordance to the number of removed rows and updates the score
    - *update_delay()* - function to update the falling speed of the tetrimino (it increases every 5 points, up to 45, by a factor of 1.1)
    - *is_the_game_over()* - function to check whether the game is over (if it is, it displays the score and resets the board for a new game)
    - *game()* - main function for the game, where a loop waits for an established amount of time (that decreases according to the update_delay() function) and then it translates down the tetrimino; it also checks whether a button is pressed or not

3. Button module functions
    - *setupButtons()* - the code for initialising the buttons, that must be run once in setup()
    - *button0(), button1(), button2(), button3()* - takes the needed action in accordance to which button is pressed
    - *checkButtons()* - function which checks whether any button is pressed and calls the appropriate function

In the *setup()* function, there are defined the 7 types of possible tetriminos and also a seed for the random generation of blocks is created. The *loop()* function only calls the game() function, described above.

Testing and validation

Developing an application consists of a lot of trial and error, but only the major results obtained from testing will be included here.

The project was first thought to be loaded onto an Arduino Uno board, but, after putting together the three parts of the code (buttons, display and the Tetris game), the memory of this board wasn't large enough, so it was replaced with an Arduino Mega board.

Because the code for the Tetris game is adapted from the code of a Tetris game built for a 16x8 board (in this application, the board has the standard dimension - 20x10), multiple problems appeared and changes were needed to be made in the check_space() function - the tetriminos wouldn't stop 2 rows before the bottom and two columns before the right margin (thus making it impossible for the user to complete a row).

However, the final validation of the application, which consisted of actually playing the game and testing all the buttons and as many of the exceptional situations (such as trying to translate a tetrimino outside of the game board) showed that all these problems were overcome.

Conclusions

Although not powerful enough for complex video output, the existence of the Nokia 5110 Display Module makes the development of graphical applications on an Arduino board possible. Although it has a small resolution (84x48 pixels), this was more than enough for a tetris game (and it may have been suitable for other games or applications as well, such as the classic Snake, that could have been found on this model of Nokia).

Also, I believe that this project could be extended into a portable mini game console by attaching multiple inputs, developing or adapting other games to this display, attaching an external power source and 3D printing a custom case for it.

Other further improvements may include permanently storing the high score and finding a way to display the current score and the tetrimino that will spawn next.

Bibliography

- https://en.wikipedia.org/wiki/Tetris
- https://create.arduino.cc/projecthub/archievkumar19/the-tetris-4de129
- https://www.instructables.com/id/Getting-Started-With-NOKIA-5110-LCD-Screen-Using-A/
- https://reference.digilentinc.com/reference/pmod/pmodbtn/start
- Radu Dănescu, Mircea Paul Mureşan, Răzvan Itu, Tiberiu Mariţa - Design with Microprocessors, Laboratory Guide

Annex

```
        // DISPLAY

// Display pins
#define LCD_CLK 3
#define LCD_DIN 8
#define LCD_DC 9
#define LCD_CE 10
#define LCD_RST 11

// Matrix to square current state of the table (inverted top-bottom)
bool squares[20][10] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};
```

```
/* data table:
    first row: EF 0F
    other row: EE 0E E0 00

   6 rows
   84 collumns

   screen row: 4m (1e 3b)x10 1e 3m
   each collumn in data, a row on screen (left -> right   bottom -> top)
*/
static byte data[] = {
  0xFF, 0xFF, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111,
0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F,
0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111,
0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111,
0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F,
0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111,
0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111,
0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F,
0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111,
0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111,
0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111, 0B11101111, 0x0F,
0B11101111, 0B11101111, 0B11101111, 0x0F, 0B11101111, 0B11101111,
0B11101111, 0x0F, 0xFF,
  0xFF, 0xFF, 0x00, 0B11100000, 0B11100000, 0B11100000, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0xFF,
  0xFF, 0xFF, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
```

0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0xFF,
0xFF, 0xFF, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0xFF,
0xFF, 0xFF, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110,
0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110, 0B11101110, 0x00,
0B11101110, 0B11101110, 0B11101110, 0x00, 0B11101110, 0B11101110,
0B11101110, 0x00, 0xFF,
0xFF, 0xFF, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110,
0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0,
0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110,
0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110,
0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0,
0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110,
0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110,
0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0,

```
0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110,
0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110,
0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110, 0B11101110, 0xE0,
0B11101110, 0B11101110, 0B11101110, 0xE0, 0B11101110, 0B11101110,
0B11101110, 0xE0, 0xFF
};

void lcdInitialise() {
  pinMode(LCD_CE, OUTPUT);
  pinMode(LCD_RST, OUTPUT);
  pinMode(LCD_DC, OUTPUT);
  pinMode(LCD_DIN, OUTPUT);
  pinMode(LCD_CLK, OUTPUT);
  digitalWrite(LCD_RST, LOW);
  digitalWrite(LCD_RST, HIGH);
  lcdWrite(LOW, 0x21 );  // LCD Extended Commands.
  lcdWrite(LOW, 0xB9 );  // Set LCD Vop (Contrast).
  lcdWrite(LOW, 0x04 );  // Set Temp coefficent.
  lcdWrite(LOW, 0x14 );  // LCD bias mode 1:48.
  lcdWrite(LOW, 0x20 );  // LCD Basic Commands
  lcdWrite(LOW, 0x0C );  // LCD in normal mode.
}

void lcdLoop() {
  digitalWrite(8, HIGH);
  for (int i = 0; i < (84 * 48) / 8; i++)
    lcdWrite(HIGH, data[i]);
}

void lcdWrite(byte dc, byte data) {
  digitalWrite(LCD_DC, dc);
  digitalWrite(LCD_CE, LOW);
  shiftOut(LCD_DIN, LCD_CLK, MSBFIRST, data);
  digitalWrite(LCD_CE, HIGH);
}
```

```
// Translate the squares matrix for the data matrix
void squareToData() {
  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++) {
      if (j % 2 == 0) {
        if (squares[i][j] == 1) {
          data[3 + 4 * i + 84 * (j / 2)] = 0B11100000 | data[3 + 4 * i + 84 * (j / 2)];
          data[4 + 4 * i + 84 * (j / 2)] = 0B11100000 | data[4 + 4 * i + 84 * (j / 2)];
          data[5 + 4 * i + 84 * (j / 2)] = 0B11100000 | data[5 + 4 * i + 84 * (j / 2)];
        } else {
          data[3 + 4 * i + 84 * (j / 2)] = 0B00011111 & data[3 + 4 * i + 84 * (j / 2)];
          data[4 + 4 * i + 84 * (j / 2)] = 0B00011111 & data[4 + 4 * i + 84 * (j / 2)];
          data[5 + 4 * i + 84 * (j / 2)] = 0B00011111 & data[5 + 4 * i + 84 * (j / 2)];
        }
      } else {
        if (squares[i][j] == 1) {
          data[3 + 4 * i + 84 * ((j + 1) / 2)] = 0B00001110 | data[3 + 4 * i + 84 * ((j + 1) /
2)];
          data[4 + 4 * i + 84 * ((j + 1) / 2)] = 0B00001110 | data[4 + 4 * i + 84 * ((j + 1) /
2)];
          data[5 + 4 * i + 84 * ((j + 1) / 2)] = 0B00001110 | data[5 + 4 * i + 84 * ((j + 1) /
2)];
        } else {
          data[3 + 4 * i + 84 * ((j + 1) / 2)] = 0B11110001 & data[3 + 4 * i + 84 * ((j + 1) /
2)];
          data[4 + 4 * i + 84 * ((j + 1) / 2)] = 0B11110001 & data[4 + 4 * i + 84 * ((j + 1) /
2)];
          data[5 + 4 * i + 84 * ((j + 1) / 2)] = 0B11110001 & data[5 + 4 * i + 84 * ((j + 1) /
2)];
        }
      }
    }
  }
}
```

```
// GAME
int block_blueprint[7][5][5] = {0};
int arranged_blocks[20][10];
short block_number;
int next = 0;
int coming_block[20][10];
int score = 0;
int delay_ = 700;
int delay_start = 700;
long int timePassed = 0;

void arrangedBlocksToSquare() {
  for (int i = 0; i < 20; i ++) {
    for (int j = 0; j < 10; j++) {
      if (arranged_blocks[i][j] > 0 || coming_block[i][j] > 0)
        squares[19 - i][j] = 1;
      else
        squares[19 - i][j] = 0;
    }
  }
}

bool check_space(int temporary_block[24][14]) {
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 14; j++) {
      if (temporary_block[i][j] != 0) {
        //Serial.println("Entered 1");
        return false;
      }
    }
  }

  for (int i = 22; i < 24; i++) {
    for (int j = 0; j < 14; j++) {
      if (temporary_block[i][j] != 0) {
        //Serial.println("Entered 2");
        return false;
      }
    }
  }
```

```
  for (int i = 0; i < 22; i++) {
    for (int j = 0; j < 2; j++) {
      if (temporary_block[i][j] != 0) {
        //Serial.println("Entered 3");
        return false;
      }
    }
  }

  for (int i = 0; i < 22; i++) {
    for (int j = 12; j < 14; j++) {
      if (temporary_block[i][j] != 0) {
        //Serial.println("Entered 4");
        return false;
      }
    }
  }

  for (int i = 2; i < 20; i++) {
    for (int j = 2; j < 12; j++) {
      if (arranged_blocks[i - 2][j - 2] != 0 && temporary_block[i][j] != 0) {
        //Serial.println("Entered 5");
        return false;
      }
    }
  }

  return true;
}

void set_arrays_zero(int mat[20][10], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      mat[i][j] = 0;
  }
}
```

```
void newBlock() {
  delay(100);
  bool finish_game = is_the_game_over();

  if (finish_game == true)
    return;

  block_number = next;
  next = random(7);
  set_arrays_zero(coming_block, 20, 10);

  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++)
      coming_block[i][j + 2] = block_blueprint[block_number][i][j];
  }

  return;
}

void rotate() {
  if (block_number == 3) // 2x2 block
    return;

  int temp[5][5] = {0};
  bool flag = 0;
  short x, y = 0; // used to find i and j where coming_block[i][j]==3 (center)

  // find coordinates of the center of the coming block
  for (int i = 0; i < 20 && !flag; i++) {
    for (int j = 0; j < 10; j++) {
      if (coming_block[i][j] == 3) {
        x = i;
        y = j;
        flag = 1;
        break;
      }
    }
  }
```

```
  // to put the matrix in a 5X5 temp matrix
  for (int i = x - 2; i < x + 3; i++) {
    for (int j = y - 2; j < y + 3; j++)
      temp[i - x + 2][j - y + 2] = coming_block[i][j];
  }

  // actual rotation
  for (int i = 0; i <= 2; i++) {
    for (int j = i; j < 4 - i; j++) {
      int mat = temp[i][j];
      temp[i][j] = temp[j][4 - i];
      temp[j][4 - i] = temp[4 - i][4 - j];
      temp[4 - i][4 - j] = temp[4 - j][i];
      temp[4 - j][i] = mat;
    }
  }

  // new thing
  int temporary_block[24][14] = {0};
  for (int i = x - 2; i < x + 3; i++) {
    for (int j = y - 2; j < y + 3; j++)
      temporary_block[i + 2][j + 2] = temp[i - x + 2][j - y + 2];
  }

  bool check = check_space(temporary_block);

  if (check == true) {
    for (int i = 0; i < 20; i++) {
      for (int j = 0; j < 10; j++)
        coming_block[i][j] = temporary_block[i + 2][j + 2];
    }
  }

  updating_arranged_blocks();
  arrangedBlocksToSquare();
}
```

```
void translate_right() {
  int temporary_block[24][14] = {0};

  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++)
      temporary_block[i + 2][j + 3] = coming_block[i][j];
  }

  bool check = check_space(temporary_block);

  if (check == true) {
    set_arrays_zero(coming_block, 20, 10);
    for (int i = 0; i < 20; i++) {
      for (int j = 0; j < 10; j++)
        coming_block[i][j] = temporary_block[i + 2][j + 2];
    }
  }

  updating_arranged_blocks();
  arrangedBlocksToSquare();
}

void translate_down() {
  int temporary_block[24][14] = {0};

  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++)
      temporary_block[i + 3][j + 2] = coming_block[i][j];
  }

  bool check = check_space(temporary_block);

  if (check == true) {
    set_arrays_zero(coming_block, 20, 10);
    for (int i = 0; i < 20; i++) {
      for (int j = 0; j < 10; j++)
        coming_block[i][j] = temporary_block[i + 2][j + 2];
    }
  } else {
    //Merging arranged blocks with coming blocks because the block can't go further
down.
```

```
    for (int i = 0; i < 20; i++) {
      for (int j = 0; j < 10; j++) {
        if (arranged_blocks[i][j] != 0 || coming_block[i][j] != 0)
          arranged_blocks[i][j] = 1;
      }
    }

    set_arrays_zero(coming_block, 20, 10);
    newBlock();
  }

  updating_arranged_blocks();
  arrangedBlocksToSquare();
  lcdLoop();
}

void translate_left() {
  int temporary_block[24][14] = {0};
  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++)
      temporary_block[i + 2][j + 1] = coming_block[i][j];
  }

  bool check = check_space(temporary_block);

  if (check == true) {
    set_arrays_zero(coming_block, 20, 10);
    for (int i = 0; i < 20; i++) {
      for (int j = 0; j < 10; j++)
        coming_block[i][j] = temporary_block[i + 2][j + 2];
    }
  }

  updating_arranged_blocks();
  arrangedBlocksToSquare();
}
```

```
void updating_arranged_blocks() {
  delay(100);
  int i, j, k;
  int number_of_lines_cleared = 0;

  for (i = 19; i >= 0; i--) {
    // check if filled row
    int count = 0;
    for (j = 0; j < 10; j++) {
      if (arranged_blocks[i][j] != 0)
        count++;
    }

    // clear row
    if (count == 10) {
      number_of_lines_cleared++;

      // move rows above
      for (k = i; k > 0; k--) {
        for (j = 0; j < 10; j++)
          arranged_blocks[k][j] = arranged_blocks[k - 1][j];
      }

      // empty first row
      for (j = 0; j < 10; j++)
        arranged_blocks[0][j] = 0;

      i++;
    }
  }

  score = score + number_of_lines_cleared;

  // bonus
  if (number_of_lines_cleared == 2)
    score = score + 1;
  else if (number_of_lines_cleared == 3)
    score = score + 2;
  else if (number_of_lines_cleared == 4)
    score = score + 3;
}
```

```
void update_delay() {
  if (score <= 5)
    delay_ = delay_start;
  else if (score > 5 && score <= 10)
    delay_ = (int) delay_start * 0.9;
  else if (score > 10 && score <= 15)
    delay_ = (int) delay_start * 0.9 * 0.9;
  else if (score > 15 && score <= 20)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9;
  else if (score > 20 && score <= 25)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9 * 0.9;
  else if (score > 25 && score <= 30)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9 * 0.9 * 0.9;
  else if (score > 30 && score <= 35)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9;
  else if (score > 35 && score <= 40)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9;
  else if (score > 40 && score <= 45)
    delay_ = (int) delay_start * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9 * 0.9;
}

bool is_the_game_over() {
  updating_arranged_blocks();
  for (int i = 0; i < 8; i++) {
    if (arranged_blocks[1][i]) {
      Serial.print("GAME OVER. SCORE: ");
      Serial.println(score);
      delay(3000);
      score = 0;
      set_arrays_zero(arranged_blocks, 20, 10);
      set_arrays_zero(coming_block, 20, 10);
    }
  }
  return false;
}
```

```
void game() {
  timePassed = millis();

  while (true) {
    if (millis() - timePassed > delay_) {
      translate_down();
      timePassed = millis();
      arrangedBlocksToSquare();
      squareToData();
      lcdLoop();
    }

    checkButtons();
  }
}

// BUTTONS
/*
   b0 - move left
   b1 - rotate
   b2 - move right
   b3 - play / drop
*/

// Button pins
#define b0 7
#define b1 6
#define b2 5
#define b3 4

void setupButtons() {
  pinMode(b0, INPUT_PULLUP);
  pinMode(b1, INPUT_PULLUP);
  pinMode(b2, INPUT_PULLUP);
  pinMode(b3, INPUT_PULLUP);
}
```

```
// Move left
void button0() {
  //Serial.println("Left");
  translate_left();
  squareToData();
  lcdLoop();
}

// Rotate
void button1() {
  //Serial.println("Rotate");
  rotate();
  squareToData();
  lcdLoop();
}

// Move right
void button2() {
  //Serial.println("Right");
  translate_right();
  squareToData();
  lcdLoop();
}

// Drop
void button3() {
  //Serial.println("Down");
  translate_down();
  squareToData();
  lcdLoop();
}

void checkButtons() {
  if (digitalRead(b0))
    button0();
  if (digitalRead(b1))
    button1();
  if (digitalRead(b2))
    button2();
  if (digitalRead(b3))
    button3();
```

```
}

void setup() {
  Serial.begin(9600);
  // 3 = center of block (used at rotation)
  // Block blueprint 0
  //..2..
  //..2..
  //..3..
  //..2..
  //.....
  block_blueprint[0][0][2] = 2;
  block_blueprint[0][1][2] = 2;
  block_blueprint[0][2][2] = 3;
  block_blueprint[0][3][2] = 2;

  // Block blueprint 1
  //.....
  //..2..
  //..32.
  //..2..
  //.....
  block_blueprint[1][1][2] = 2;
  block_blueprint[1][2][2] = 3;
  block_blueprint[1][2][3] = 2;
  block_blueprint[1][3][2] = 2;

  // Block blueprint 2
  //.....
  //.2...
  //.23..
  //..2..
  //.....
  block_blueprint[2][1][1] = 2;
  block_blueprint[2][2][1] = 2;
  block_blueprint[2][2][2] = 3;
  block_blueprint[2][3][2] = 2;
```

```
// Block blueprint 3
//.....
//.22..
//.23..
//.....
//.....
block_blueprint[3][1][1] = 2;
block_blueprint[3][1][2] = 2;
block_blueprint[3][2][1] = 2;
block_blueprint[3][2][2] = 3;

// Block blueprint 4
//.....
//..2..
//.23..
//.2...
//.....
block_blueprint[4][1][2] = 2;
block_blueprint[4][2][1] = 2;
block_blueprint[4][2][2] = 3;
block_blueprint[4][3][1] = 2;

// Block blueprint 5
//.....
//.22..
//..3..
//..2..
//.....
block_blueprint[5][1][1] = 2;
block_blueprint[5][1][2] = 2;
block_blueprint[5][2][2] = 3;
block_blueprint[5][3][2] = 2;
// Block blueprint 6
//.....
//..22.
//..3..
//..2..
//.....
block_blueprint[6][1][2] = 2;
block_blueprint[6][1][3] = 2;
block_blueprint[6][2][2] = 3;
```

```
  block_blueprint[6][3][2] = 2;
  // Generate random seed for generating blocks
  int seed = (analogRead(0) + 1) * (analogRead(1) + 1) * (analogRead(2) + 1) *
(analogRead(3) + 1);
  randomSeed(seed);
  random(10, 9610806);
  seed = seed * random(3336, 15679912) + analogRead(random(4)) ;
  randomSeed(seed);
  random(10, 98046);

  setupButtons();
  lcdInitialise();

  next = random(7);
  newBlock();
}

void loop() {
  game();
}
```