**Simplified version of Arkanoid**

***Authors:***

**Bucatos Carla-Daiana**

**Bulzan Andrei-Florin**

***Requirements for making a simplified version of Arkanoid***

* You will have to define 3 kinds of blocks. You will have to keep tabs on durability.
* You will have to use a LED (light-emitting diode) matrix (dimensions: 8\*8 or bigger)/screen of type OLED, TFT etc, to show the game’s pieces.
* The player will be represented as a block which is placed at screen’s base.
* The left-right circulation will be effectuated using a suitable button for each direction. One press means one move.
* There can be implemented bonuses: extra lifes,a more powerful ball etc.

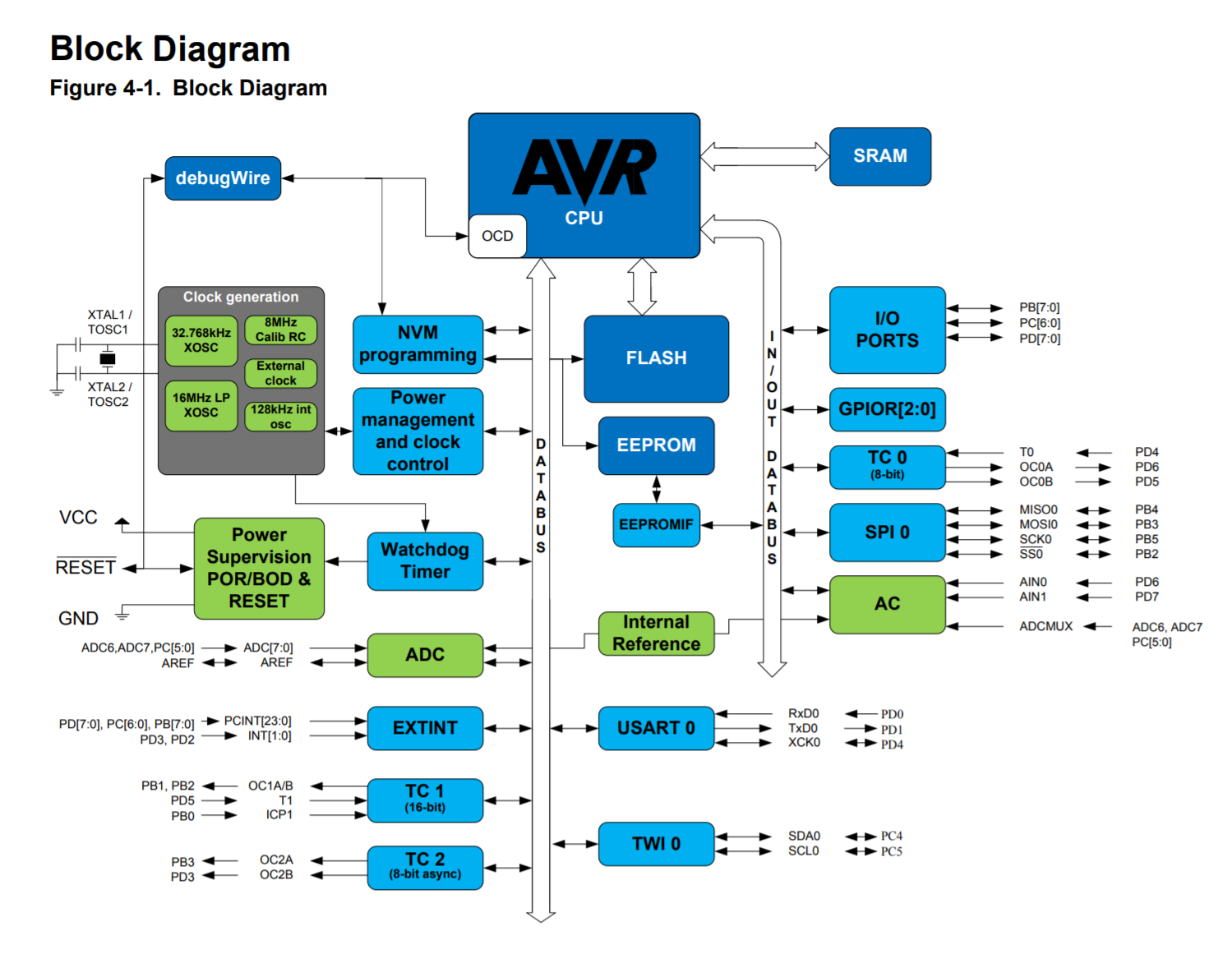
The development board we use is ***UNO R3 ATmega328p*** and it has the following characteristics:

• Microcontroller: ATmega328p  
• USB Chip: CH340G  
• Operating Voltage: 5V  
• Input Voltage (recommended): 7-12V  
• Input Voltage (limits): 6-20V  
• Digital I/O Pins: 14 (of which 6 provide PWM output)  
• Analog Input Pins: 6  
• DC Current per I/O Pin: 40 mA  
• DC Current for 3.3V Pin: 50 mA  
• Flash Memory: 32 KB (ATmega328) of which 0.5 KB used by bootloader  
• SRAM: 2 KB (ATmega328)  
• EEPROM: 1 KB (ATmega328)  
• Clock Speed: 16 MHz

The microcontroller used (***ATmega328p***) has the following features:

(High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family)

* Advanced RISC Architecture
  + 131 Powerful Instructions
  + Most Single Clock Cycle Execution
  + 32 x 8 General Purpose Working Registers
  + Fully Static Operation
  + Up to 20 MIPS Throughput at 20MHz
  + On-chip 2-cycle Multiplier
* High Endurance Non-volatile Memory Segments
  + 32KBytes of In-System Self-Programmable Flash program Memory
  + 1KBytes EEPROM
  + 2KBytes Internal SRAM
  + Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  + Data Retention: 20 years at 85°C/100 years at 25°C(1)
  + Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  + Programming Lock for Software Security
* Atmel® QTouch® Library Support
  + Capacitive Touch Buttons, Sliders and Wheels
  + QTouch and QMatrix® Acquisition
  + Up to 64 sense channels
* Peripheral Features
  + Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  + One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  + Real Time Counter with Separate Oscillator
  + Six PWM Channels
  + 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  + 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  + Two Master/Slave SPI Serial Interface
  + One Programmable Serial USART
  + One Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
  + Programmable Watchdog Timer with Separate On-chip Oscillator
  + One On-chip Analog Comparator
  + Interrupt and Wake-up on Pin Change
* Special Microcontroller Features
  + Power-on Reset and Programmable Brown-out Detection
  + Internal Calibrated Oscillator
  + External and Internal Interrupt Sources
  + Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
* I/O and Packages
  + 23 Programmable I/O Lines
  + 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
* Operating Voltage:
  + 1.8 - 5.5V
* Temperature Range:
  + -40°C to 105°C
* Speed Grade:
  + 0 - 4MHz @ 1.8 - 5.5V
  + 0 - 10MHz @ 2.7 - 5.5V
  + 0 - 20MHz @ 4.5 - 5.5V
* Power Consumption at 1MHz, 1.8V, 25°C
  + Active Mode: 0.2mA
  + Power-down Mode: 0.1μA
  + Power-save Mode: 0.75μA (Including 32kHz RTC)



In the project, the **SPI (Serial Peripheral Interface)** is used in order to transfer data between the development board and the MAX7219 circuit(led matrix).

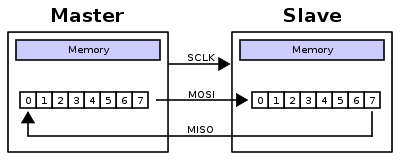
The Serial Peripheral Interface is used to transfer data between integrated circuits using a reduced number of data lines. This article provides the background information needed for novices to understand the interface.

Capabilities and Characteristics

The Serial Peripheral Interface Bus provides full-duplex synchronous communication between a master device and a slave using four data lines.

Basic Master-Slave Configuration

The Serial Peripheral Interface allows bits of data to be shifted out of a master device into a slave, and at the same time, bits can be shifted out of the slave into the master.



Typically there are three lines common to all the devices:

* MISO (Master In Slave Out) - The Slave line for sending data to the master(***In this project MISO is not used***),
* MOSI (Master Out Slave In) - The Master line for sending data to the peripherals,
* SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master

and one line specific for every device:

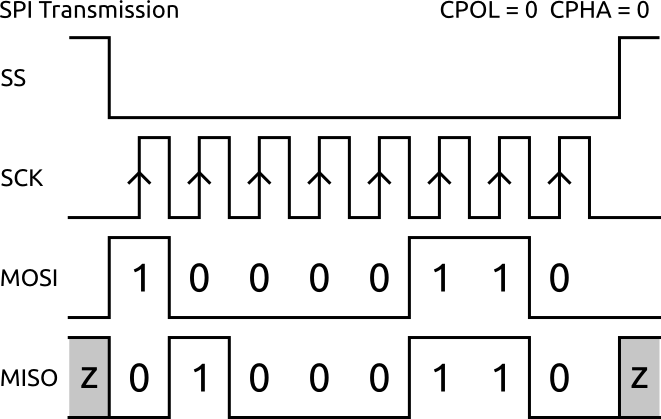
* SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

To write code for a new SPI device you need to note a few things:

* What is the maximum SPI speed your device can use? This is controlled by the first parameter in SPISettings. If you are using a chip rated at 15 MHz, use 15000000. Arduino will automatically use the best speed that is equal to or less than the number you use with SPISettings.
* Is data shifted in Most Significant Bit (MSB) or Least Significant Bit (LSB) first? This is controlled by second SPISettings parameter, either MSBFIRST or LSBFIRST. Most SPI chips use MSB first data order.
* Is the data clock idle when high or low? Are samples on the rising or falling edge of clock pulses? These modes are controlled by the third parameter in SPISettings.

The SPI standard is loose and each device implements it a little differently. This means you have to pay special attention to the device's datasheet when writing your code.



Clock Polarity and Phase

Clock transitions govern the shifting and sampling of data. SPI has four modes (0,1,2,3) that correspond to the four possible clocking configurations.

Each transaction begins when the slave-select line is driven to logic low (slave select is typically an active-low signal). The exact relationship between the slave-select, data, and clock lines depends on how the clock polarity (CPOL) and clock phase (CPHA) are configured.

With non-inverted clock polarity (i.e., the clock is at logic low when slave select transitions to logic low):

* Mode 0: Clock phase is configured such that data is sampled on the rising edge of the clock pulse and shifted out on the falling edge of the clock pulse. This corresponds to the first blue clock trace in the above diagram. Note that data must be available before the first rising edge of the clock.
* Mode 1: Clock phase is configured such that data is sampled on the falling edge of the clock pulse and shifted out on the rising edge of the clock pulse. This corresponds to the second blue clock trace in the above diagram.

With inverted clock polarity (i.e., the clock is at logic high when slave select transitions to logic low):

* Mode 2: Clock phase is configured such that data is sampled on the falling edge of the clock pulse and shifted out on the rising edge of the clock pulse. This corresponds to the first orange clock trace in the above diagram. Note that data must be available before the first falling edge of the clock.
* Mode 3: Clock phase is configured such that data is sampled on the rising edge of the clock pulse and shifted out on the falling edge of the clock pulse. This corresponds to the second orange clock trace in the above diagram.

Advantages:

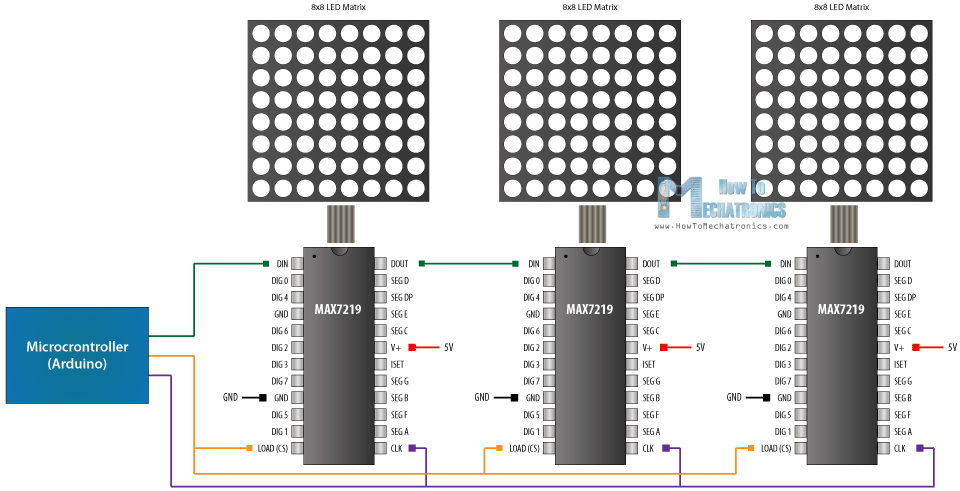
* Full duplex communication in the default version of this protocol
* Push-pull drivers (as opposed to open drain) provide good signal integrity and high speed
* Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speed
* Complete protocol flexibility for the bits transferred
  + Not limited to 8-bit words
  + Arbitrary choice of message size, content, and purpose
* Extremely simple hardware interfacing
  + Typically lower power requirements than I²C or SMBus due to less circuitry (including pull up resistors)
  + No arbitration or associated failure modes
  + Slaves use the master's clock and do not need precision oscillators
  + Slaves do not need a unique address – unlike I²C or GPIB or SCSI
  + Transceivers are not needed
* Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
* At most one unique bus signal per device (chip select); all others are shared
* Signals are unidirectional allowing for easy galvanic isolation
* Simple software implementation

Disadvantages:

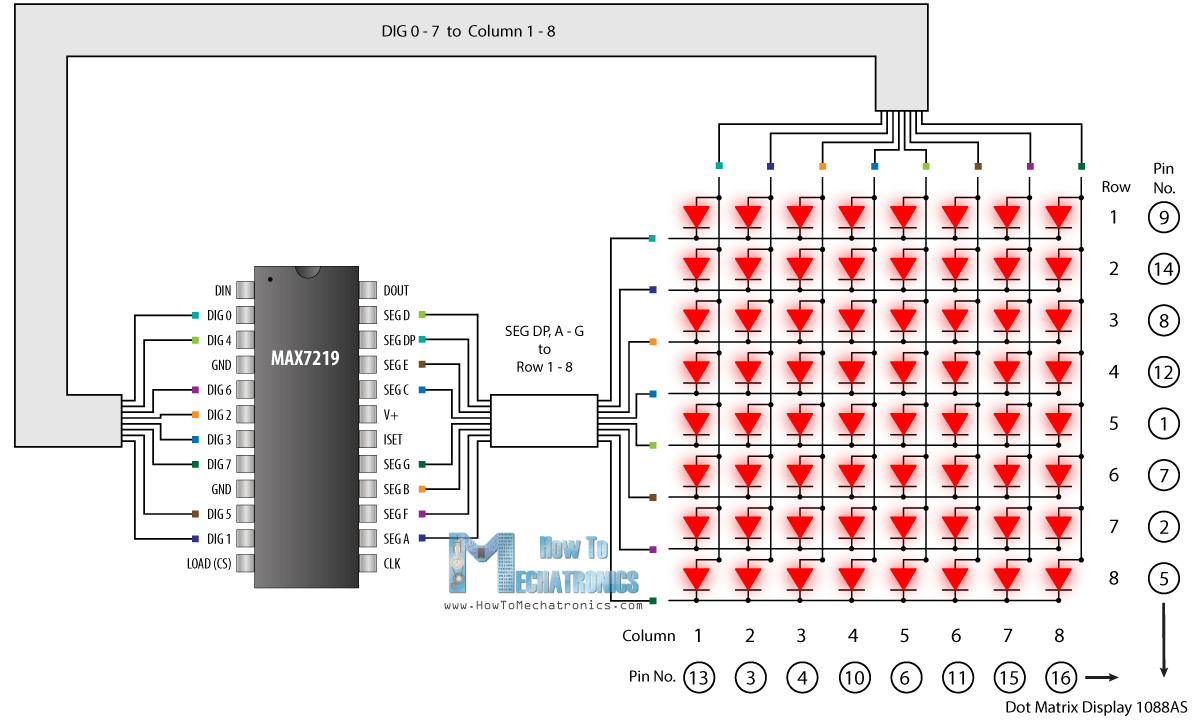
* Requires more pins on IC packages than I²C, even in the *three-wire* variant
* No in-band addressing; out-of-band chip select signals are required on shared buses
* No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
* No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
* Typically supports only one master device (depends on device's hardware implementation)
* No error-checking protocol is defined
* Without a formal standard, validating conformance is not possible
* Only handles short distances compared to RS-232, RS-485, or CAN-bus. (its distance can be extended with use of transceivers like RS-422)
* Many existing variations, making it difficult to find development tools like host adapters that support those variations
* SPI does not support hot swapping (dynamically adding nodes).
* Interrupts must either be implemented with out-of-band signals or be faked by using periodic polling similarly to USB 1.1 and 2.0.
* Some variants like dual SPI, quad SPI, and three-wire serial buses defined below are half-duplex.

The development board does not have an USB port, it only has CHIP CH340g which helps connecting it to the USB on a computer. Even though the USB is not integrated in the board, connecting the board to the USB is crucial in this project, because that is the way in which the code is transmitted to the board. So basically, the USB plays an important role in the project.

The **display module** used in this project is an **8×8 LED Matrix** using the **MAX7219 driver**. Let’s take a closer look at the MAX7219 driver. The IC is capable of driving 64 individual LEDs while using only 3 wires for communication with the Arduino. For example, what’s more is that we can daisy chain multiple drivers and matrixes and still use the same 3 wires. (picture below – of course, we have only one matrix)

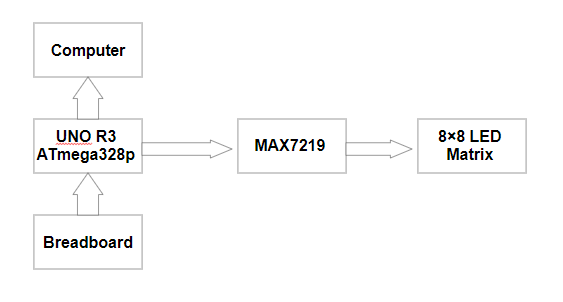


The 64 LEDs are driven by 16 output pins of the IC. The question now is how is that possible. Well the maximum number of LEDs light up at the same time is actually eight. The LEDs are arranged as 8×8 set of rows and columns. So the MAX7219 activates each column for a very short period of time and at the same time it also drives each row. So by rapidly switching through the columns and rows the human eye will only notice a continuous light. Basically, below we can see how the multiplexing method is taking place.



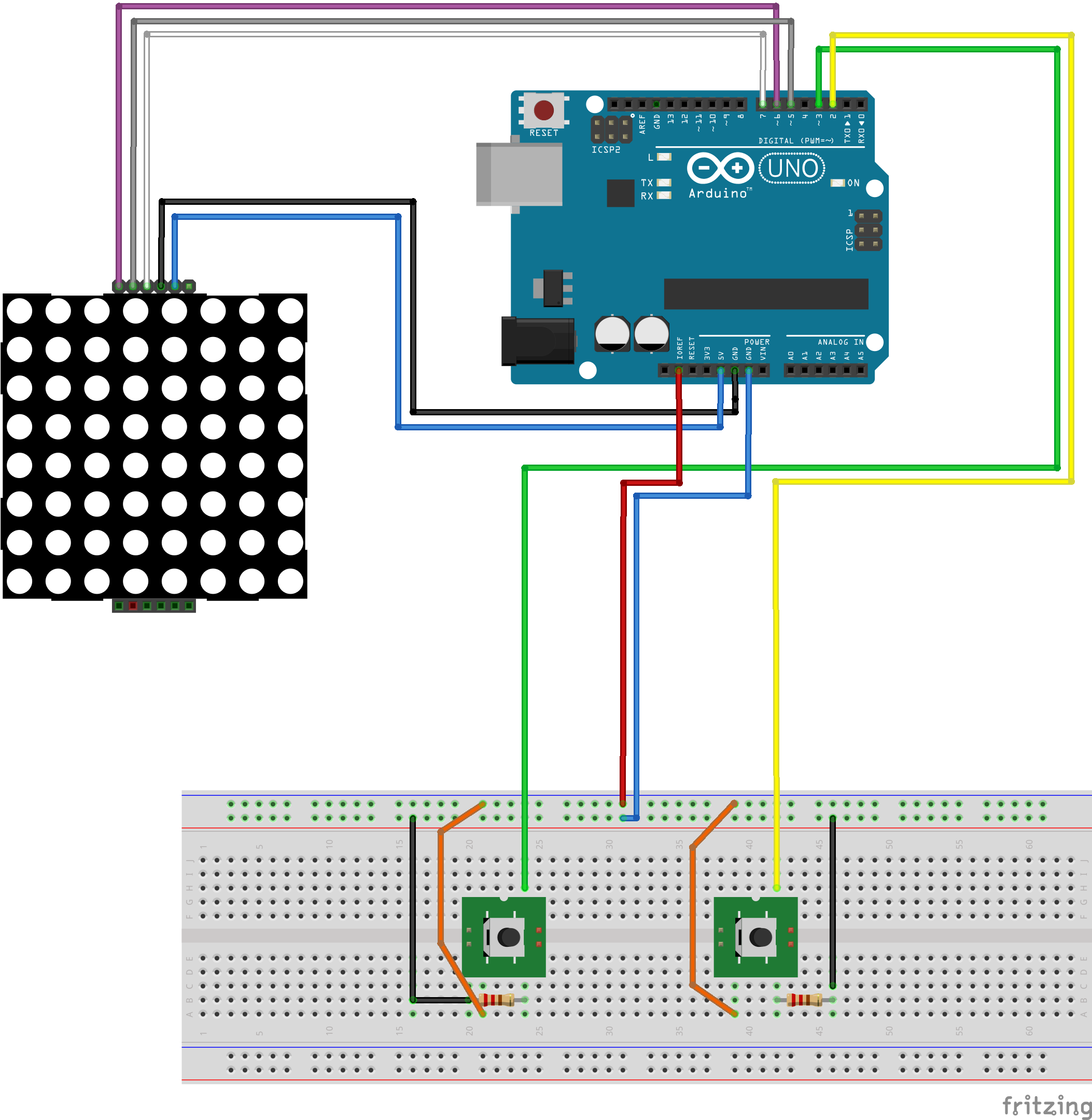
The pins of a common 8×8 LED Matrix are internally arranged, so if you are building a matrix on your own you should consider it. A common breakout board for the MAX7219 comes with a resistor between the 5V and the IC pin number 18. The resistor is used for setting the brightness or the current flow to the LEDs.

**Block Diagram**



The development board is connected to the MAX7219, which is connected to the LED matrix. This circuit, like it was said before, makes the connection board-matrix easier, using the multiplexing method. The breadboard is connected to the development board in order to help us connect the push buttons. The UNO R3 board is also connected to the computer, since that is the place where the code is extracted from.

Below is the projects’ scheme made in Fritzing. The led matrix, which is attached to the MAX7219 circuit, is connected to the UNO R3 board. With the help of the breadboard we connected 2 pushbuttons along with a couple of 10 kOhm resistors(their purpose is to prevent the leds from burning).



**The program:**

#include <Arduino.h>

#include <MaxMatrix.h>

#include <avr/pgmspace.h>

#define NO\_OF\_LEVELS 3

const int buttonPin1 = 2; // the number of the pushbutton pin

const int buttonPin2 = 3;

int buttonState1 = 0; // variable for reading the pushbutton status

int buttonState2 = 0;

char victorySign[] = {8, 8,

B00000001,

B00000001,

B00000001,

B00000001,

B00000001,

B00000001,

B00000001,

B00000001

};

char smile[] = {8, 8,

B00111100,

B01000010,

B10010101,

B10100001,

B10100001,

B10010101,

B01000010,

B00111100

};

PROGMEM const unsigned char LEVELS[] = {

//level 1

B00000000,

B01111110,

B01111110,

B00000000,

B00000000,

B00000000,

B00000000,

B00000000,

//level 2

B10101010,

B01010101,

B10101010,

B00000000,

B00000000,

B00000000,

B00000000,

B00000000,

//level 3

B10001111,

B11000111,

B11100011,

B11110001,

B00000000,

B00000000,

B00000000,

B00000000

};

PROGMEM const unsigned char CH[] = {

3, 8, B00000000, B00000000, B00000000, B00000000, B00000000, // space

1, 8, B01011111, B00000000, B00000000, B00000000, B00000000, // !

3, 8, B00000011, B00000000, B00000011, B00000000, B00000000, // "

5, 8, B00010100, B00111110, B00010100, B00111110, B00010100, // #

4, 8, B00100100, B01101010, B00101011, B00010010, B00000000, // $

5, 8, B01100011, B00010011, B00001000, B01100100, B01100011, // %

5, 8, B00110110, B01001001, B01010110, B00100000, B01010000, // &

1, 8, B00000011, B00000000, B00000000, B00000000, B00000000, // '

3, 8, B00011100, B00100010, B01000001, B00000000, B00000000, // (

3, 8, B01000001, B00100010, B00011100, B00000000, B00000000, // )

5, 8, B00101000, B00011000, B00001110, B00011000, B00101000, // \*

5, 8, B00001000, B00001000, B00111110, B00001000, B00001000, // +

2, 8, B10110000, B01110000, B00000000, B00000000, B00000000, // ,

4, 8, B00001000, B00001000, B00001000, B00001000, B00000000, // -

2, 8, B01100000, B01100000, B00000000, B00000000, B00000000, // .

4, 8, B01100000, B00011000, B00000110, B00000001, B00000000, // /

4, 8, B00111110, B01000001, B01000001, B00111110, B00000000, // 0

3, 8, B01000010, B01111111, B01000000, B00000000, B00000000, // 1

4, 8, B01100010, B01010001, B01001001, B01000110, B00000000, // 2

4, 8, B00100010, B01000001, B01001001, B00110110, B00000000, // 3

4, 8, B00011000, B00010100, B00010010, B01111111, B00000000, // 4

4, 8, B00100111, B01000101, B01000101, B00111001, B00000000, // 5

4, 8, B00111110, B01001001, B01001001, B00110000, B00000000, // 6

4, 8, B01100001, B00010001, B00001001, B00000111, B00000000, // 7

4, 8, B00110110, B01001001, B01001001, B00110110, B00000000, // 8

4, 8, B00000110, B01001001, B01001001, B00111110, B00000000, // 9

2, 8, B01010000, B00000000, B00000000, B00000000, B00000000, // :

2, 8, B10000000, B01010000, B00000000, B00000000, B00000000, // ;

3, 8, B00010000, B00101000, B01000100, B00000000, B00000000, // <

3, 8, B00010100, B00010100, B00010100, B00000000, B00000000, // =

3, 8, B01000100, B00101000, B00010000, B00000000, B00000000, // >

4, 8, B00000010, B01011001, B00001001, B00000110, B00000000, // ?

5, 8, B00111110, B01001001, B01010101, B01011101, B00001110, // @

4, 8, B01111110, B00010001, B00010001, B01111110, B00000000, // A

4, 8, B01111111, B01001001, B01001001, B00110110, B00000000, // B

4, 8, B00111110, B01000001, B01000001, B00100010, B00000000, // C

4, 8, B01111111, B01000001, B01000001, B00111110, B00000000, // D

4, 8, B01111111, B01001001, B01001001, B01000001, B00000000, // E

4, 8, B01111111, B00001001, B00001001, B00000001, B00000000, // F

4, 8, B00111110, B01000001, B01001001, B01111010, B00000000, // G

4, 8, B01111111, B00001000, B00001000, B01111111, B00000000, // H

3, 8, B01000001, B01111111, B01000001, B00000000, B00000000, // I

4, 8, B00110000, B01000000, B01000001, B00111111, B00000000, // J

4, 8, B01111111, B00001000, B00010100, B01100011, B00000000, // K

4, 8, B01111111, B01000000, B01000000, B01000000, B00000000, // L

5, 8, B01111111, B00000010, B00001100, B00000010, B01111111, // M

5, 8, B01111111, B00000100, B00001000, B00010000, B01111111, // N

4, 8, B00111110, B01000001, B01000001, B00111110, B00000000, // O

4, 8, B01111111, B00001001, B00001001, B00000110, B00000000, // P

4, 8, B00111110, B01000001, B01000001, B10111110, B00000000, // Q

4, 8, B01111111, B00001001, B00001001, B01110110, B00000000, // R

4, 8, B01000110, B01001001, B01001001, B00110010, B00000000, // S

5, 8, B00000001, B00000001, B01111111, B00000001, B00000001, // T

4, 8, B00111111, B01000000, B01000000, B00111111, B00000000, // U

5, 8, B00001111, B00110000, B01000000, B00110000, B00001111, // V

5, 8, B00111111, B01000000, B00111000, B01000000, B00111111, // W

5, 8, B01100011, B00010100, B00001000, B00010100, B01100011, // X

5, 8, B00000111, B00001000, B01110000, B00001000, B00000111, // Y

4, 8, B01100001, B01010001, B01001001, B01000111, B00000000, // Z

2, 8, B01111111, B01000001, B00000000, B00000000, B00000000, // [

4, 8, B00000001, B00000110, B00011000, B01100000, B00000000, // \ backslash

2, 8, B01000001, B01111111, B00000000, B00000000, B00000000, // ]

3, 8, B00000010, B00000001, B00000010, B00000000, B00000000, // hat

4, 8, B01000000, B01000000, B01000000, B01000000, B00000000, // \_

2, 8, B00000001, B00000010, B00000000, B00000000, B00000000, // `

4, 8, B00100000, B01010100, B01010100, B01111000, B00000000, // a

4, 8, B01111111, B01000100, B01000100, B00111000, B00000000, // b

4, 8, B00111000, B01000100, B01000100, B00101000, B00000000, // c

4, 8, B00111000, B01000100, B01000100, B01111111, B00000000, // d

4, 8, B00111000, B01010100, B01010100, B00011000, B00000000, // e

3, 8, B00000100, B01111110, B00000101, B00000000, B00000000, // f

4, 8, B10011000, B10100100, B10100100, B01111000, B00000000, // g

4, 8, B01111111, B00000100, B00000100, B01111000, B00000000, // h

3, 8, B01000100, B01111101, B01000000, B00000000, B00000000, // i

4, 8, B01000000, B10000000, B10000100, B01111101, B00000000, // j

4, 8, B01111111, B00010000, B00101000, B01000100, B00000000, // k

3, 8, B01000001, B01111111, B01000000, B00000000, B00000000, // l

5, 8, B01111100, B00000100, B01111100, B00000100, B01111000, // m

4, 8, B01111100, B00000100, B00000100, B01111000, B00000000, // n

4, 8, B00111000, B01000100, B01000100, B00111000, B00000000, // o

4, 8, B11111100, B00100100, B00100100, B00011000, B00000000, // p

4, 8, B00011000, B00100100, B00100100, B11111100, B00000000, // q

4, 8, B01111100, B00001000, B00000100, B00000100, B00000000, // r

4, 8, B01001000, B01010100, B01010100, B00100100, B00000000, // s

3, 8, B00000100, B00111111, B01000100, B00000000, B00000000, // t

4, 8, B00111100, B01000000, B01000000, B01111100, B00000000, // u

5, 8, B00011100, B00100000, B01000000, B00100000, B00011100, // v

5, 8, B00111100, B01000000, B00111100, B01000000, B00111100, // w

5, 8, B01000100, B00101000, B00010000, B00101000, B01000100, // x

4, 8, B10011100, B10100000, B10100000, B01111100, B00000000, // y

3, 8, B01100100, B01010100, B01001100, B00000000, B00000000, // z

3, 8, B00001000, B00110110, B01000001, B00000000, B00000000, // {

1, 8, B01111111, B00000000, B00000000, B00000000, B00000000, // |

3, 8, B01000001, B00110110, B00001000, B00000000, B00000000, // }

4, 8, B00001000, B00000100, B00001000, B00000100, B00000000, // ~

};

int DIN = 7; // DIN pin of MAX7219 module

int CLK = 6; // CLK pin of MAX7219 module

int CS = 5; // CS pin of MAX7219 module

int maxInUse = 1;

MaxMatrix m(DIN, CS, CLK, maxInUse);

byte buffer[10];

unsigned long previousMillis = 0;

const long interval = 200;

void(\*resetFunc)(void) = 0;

typedef struct {

int x, y;

} point\_t;

point\_t ball;

point\_t pad[2];

char bricks[8][8];

enum Direction {U, U\_R, R\_D, D, D\_L, L\_U};

Direction currentDirection;

int level;

int activeBricks;

void activateBrick(int x, int y);

void deactivateBrick(int x, int y);

void bricksSetup(int level);

void setBall(int x, int y);

void setPad(int x, int y);

void movePadRight();

void movePadLeft();

void moveBall(Direction direction);

void gameOver();

point\_t getNextPosition();

Direction getNextDirection(point\_t nextPosition);

void victory();

void nextLevel();

void setup();

void loop();

void serialEvent();

void activateBrick(int x, int y) {

bricks[x][y] = '1';

m.setDot(y, x, true);

++activeBricks;

}

void deactivateBrick(int x, int y) {

bricks[x][y] = '0';

m.setDot(y, x, false);

--activeBricks;

}

void bricksSetup(int level) {

int i, j;

memcpy\_P(buffer, LEVELS + 8\*(level - 1), 8);

m.clear();

activeBricks = 0;

for(i = 0; i < 8; ++i) {

for(j = 0; j < 8; ++j)

if((buffer[i] & (1 << j)) != 0)

activateBrick(i, 7 - j);

else

bricks[i][7 - j] = '0';

}

}

void setBall(int x, int y) {

if(x >= 0 && x <= 7 && y >= 0 && y <= 7) {

//erase current ball

m.setDot(ball.y,ball.x,false);

//set new coordinates

ball.x = x;

ball.y = y;

m.setDot(ball.y,ball.x,true);

}

}

void setPad(int x, int y) {

if(x >= 0 && x <= 7 && y >= 0 && y < 7) {

//erase current pad

m.setDot(pad[0].y,pad[0].x,false);

m.setDot(pad[1].y,pad[1].x,false);

//set new coordinates

pad[0].x = x; pad[1].x = x; //pad[2].x = x;

pad[0].y = y; pad[1].y = y + 1; //pad[2].y = y + 2;

m.setDot(pad[0].y,pad[0].x,true);

m.setDot(pad[1].y,pad[1].x,true);

}

}

void movePadRight() {

setPad(pad[0].x, pad[0].y + 1);

}

void movePadLeft() {

setPad(pad[0].x, pad[0].y - 1);

}

//move ball one step in a certain direction

void moveBall(Direction direction) {

switch(direction) {

case U: setBall(ball.x - 1, ball.y); break;

case U\_R: setBall(ball.x - 1, ball.y + 1); break;

case R\_D: setBall(ball.x + 1, ball.y + 1); break;

case D: setBall(ball.x + 1, ball.y); break;

case D\_L: setBall(ball.x + 1, ball.y - 1); break;

case L\_U: setBall(ball.x - 1, ball.y - 1); break;

}

}

void gameOver() {

int i;

m.writeSprite(0, 0, smile);

delay(2000);

resetFunc();

}

point\_t getNextPosition() {

point\_t nextPosition;

nextPosition.x = ball.x;

nextPosition.y = ball.y;

switch(currentDirection) {

case U: --nextPosition.x; break;

case U\_R: --nextPosition.x; ++nextPosition.y; break;

case R\_D: ++nextPosition.x; ++nextPosition.y; break;

case D: ++nextPosition.x; break;

case D\_L: ++nextPosition.x; --nextPosition.y; break;

case L\_U: --nextPosition.x; --nextPosition.y; break;

}

return nextPosition;

}

Direction getNextDirection(point\_t nextPosition) {

if(nextPosition.x > 7)

gameOver();

if(nextPosition.y < 0) {

switch(currentDirection) {

case D\_L: return R\_D;

case L\_U: return U\_R;

}

}

else

if(nextPosition.y > 7) {

switch(currentDirection) {

case R\_D: return D\_L;

case U\_R: return L\_U;

}

}

else

if(nextPosition.x < 0) {

switch(currentDirection) {

case U: return D;

case U\_R: return R\_D;

case L\_U: return D\_L;

}

}

else

if(nextPosition.x == pad[0].x && nextPosition.y == pad[0].y) {

switch(currentDirection) {

case R\_D: return L\_U;

case D: return L\_U;

case D\_L: return L\_U;

}

}

else

if(nextPosition.x == pad[0].x && nextPosition.y == pad[0].y - 1) {

switch(currentDirection) {

case R\_D: gameOver();

case D: gameOver();

case D\_L: return L\_U;

}

}

else

if(nextPosition.x == pad[1].x && nextPosition.y == pad[1].y) {

switch(currentDirection) {

case R\_D: return U\_R;

case D: return U\_R;

case D\_L: return U\_R;

}

}

if(nextPosition.x == pad[1].x && nextPosition.y == pad[1].y + 1) {

switch(currentDirection) {

case R\_D: return U\_R;

case D: gameOver();

case D\_L: gameOver();

}

}

else {

switch(currentDirection) {

case U:

if(bricks[nextPosition.x][nextPosition.y] == '0')

return currentDirection;

deactivateBrick(nextPosition.x, nextPosition.y);

return D;

case U\_R:

if(bricks[nextPosition.x][nextPosition.y] == '0' && bricks[nextPosition.x][nextPosition.y - 1] == '0' && bricks[nextPosition.x + 1][nextPosition.y] == '0')

return currentDirection;

if(bricks[nextPosition.x + 1][nextPosition.y] == '1') {

if(bricks[nextPosition.x][nextPosition.y - 1] == '1') {

deactivateBrick(nextPosition.x + 1, nextPosition.y);

deactivateBrick(nextPosition.x, nextPosition.y - 1);

return D\_L;

}

else {

deactivateBrick(nextPosition.x + 1, nextPosition.y);

return L\_U;

}

}

else {

if(bricks[nextPosition.x][nextPosition.y - 1] == '1') {

deactivateBrick(nextPosition.x, nextPosition.y - 1);

return R\_D;

}

else {

deactivateBrick(nextPosition.x, nextPosition.y);

return D\_L;

}

}

case R\_D:

if(bricks[nextPosition.x][nextPosition.y] == '0' && bricks[nextPosition.x][nextPosition.y - 1] == '0' && bricks[nextPosition.x - 1][nextPosition.y] == '0')

return currentDirection;

if(bricks[nextPosition.x - 1][nextPosition.y] == '1') {

if(bricks[nextPosition.x][nextPosition.y - 1] == '1') {

deactivateBrick(nextPosition.x - 1, nextPosition.y);

deactivateBrick(nextPosition.x, nextPosition.y - 1);

return L\_U;

}

else {

deactivateBrick(nextPosition.x - 1, nextPosition.y);

return D\_L;

}

}

else {

if(bricks[nextPosition.x][nextPosition.y - 1] == '1') {

deactivateBrick(nextPosition.x, nextPosition.y - 1);

return U\_R;

}

else {

deactivateBrick(nextPosition.x, nextPosition.y);

return L\_U;

}

}

case D:

if(bricks[nextPosition.x][nextPosition.y] == '0')

return currentDirection;

deactivateBrick(nextPosition.x, nextPosition.y);

return U; break;

case D\_L:

if(bricks[nextPosition.x][nextPosition.y] == '0' && bricks[nextPosition.x][nextPosition.y + 1] == '0' && bricks[nextPosition.x - 1][nextPosition.y] == '0')

return currentDirection;

if(bricks[nextPosition.x - 1][nextPosition.y] == '1') {

if(bricks[nextPosition.x][nextPosition.y + 1] == '1') {

deactivateBrick(nextPosition.x - 1, nextPosition.y);

deactivateBrick(nextPosition.x, nextPosition.y + 1);

return U\_R;

}

else {

deactivateBrick(nextPosition.x - 1, nextPosition.y);

return R\_D;

}

}

else {

if(bricks[nextPosition.x][nextPosition.y + 1] == '1') {

deactivateBrick(nextPosition.x, nextPosition.y + 1);

return L\_U;

}

else {

deactivateBrick(nextPosition.x, nextPosition.y);

return U\_R;

}

}

case L\_U:

if(bricks[nextPosition.x][nextPosition.y] == '0' && bricks[nextPosition.x][nextPosition.y + 1] == '0' && bricks[nextPosition.x + 1][nextPosition.y] == '0')

return currentDirection;

if(bricks[nextPosition.x + 1][nextPosition.y] == '1') {

if(bricks[nextPosition.x][nextPosition.y + 1] == '1') {

deactivateBrick(nextPosition.x + 1, nextPosition.y);

deactivateBrick(nextPosition.x, nextPosition.y + 1);

return R\_D;

}

else {

deactivateBrick(nextPosition.x + 1, nextPosition.y);

return U\_R;

}

}

else

if(bricks[nextPosition.x][nextPosition.y + 1] == '1') {

deactivateBrick(nextPosition.x, nextPosition.y + 1);

return D\_L;

}

else {

deactivateBrick(nextPosition.x, nextPosition.y);

return R\_D;

}

}

}

}

void victory() {

int i;

char text[]= "Congratulations! :) ";

printStringWithShift(text, 100);

delay(1000);

//resetFunc();

}

void nextLevel() {

++level;

if(level == NO\_OF\_LEVELS + 1)

victory();

m.clear();

memcpy\_P(buffer, CH + 7 \* (16 + level), 7);

m.writeSprite(2, 1, buffer); // (text, scrolling speed)

delay(1000);

bricksSetup(level);

setPad(7, 1);

setBall(pad[0].x - 2, pad[0].y);

currentDirection = D;

delay(1000);

}

void setup() {

m.init(); // MAX7219 initialization

m.setIntensity(4); // initial led matrix intensity, 0-15

pinMode(buttonPin1, INPUT); // the pushbutton pin is initialized as an input

pinMode(buttonPin2, INPUT);

level = 0;

activeBricks = 0;

}

void loop() {

unsigned long currentMillis = millis();

//move ball after each interval

if(activeBricks == 0)

nextLevel();

if(currentMillis - previousMillis >= interval) {

//change ball direction based on its next position

point\_t nextPosition;

Direction nextDirection;

while(1) {

nextPosition = getNextPosition();

nextDirection = getNextDirection(nextPosition);

if(nextDirection == currentDirection)

break;

currentDirection = nextDirection;

}

moveBall(currentDirection);

previousMillis = currentMillis;

}

// read the state of the pushbutton value

buttonState1 = digitalRead(buttonPin1);

buttonState2 = digitalRead(buttonPin2);

// check if the pushbutton is pressed.

// if it is, the buttonState is HIGH

if (buttonState1 == HIGH) {

delay(200);

movePadRight();

}

if (buttonState2 == HIGH) {

delay(200);

movePadLeft();

}

}

void printCharWithShift(char c, int shift\_speed) {

if (c < 32) return;

c -= 32;

memcpy\_P(buffer, CH + 7 \* c, 7);

m.writeSprite(32, 0, buffer);

m.setColumn(32 + buffer[0], 0);

for (int i = 0; i < buffer[0] + 1; i++)

{

delay(shift\_speed);

m.shiftLeft(false, false);

}

}

// Extract the characters from the text string

void printStringWithShift(char\* s, int shift\_speed) {

while (\*s != 0) {

printCharWithShift(\*s, shift\_speed);

s++;

}

}

The game has 3 levels. In case you “die”, a smile face appears on the screen and then it takes you all over from the beginning of the game. If you successfully finish all 3 levels, a “Congratulations!” message will appear.

**Bibliography:**

* http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P\_Datasheet.pdf
* https://ardushop.ro/ro/electronice/286-placa-de-dezvoltare-uno-compatibila-arduino-dip-cablu.html
* https://howtomechatronics.com/tutorials/arduino/8x8-led-matrix-max7219-tutorial-scrolling-text-android-control-via-bluetooth/
* https://embeddedmicro.com/blogs/tutorials/serial-peripheral-interface-spi
* https://en.wikipedia.org/wiki/Serial\_Peripheral\_Interface\_Bus
* https://www.arduino.cc/en/Reference/SPI
* https://www.allaboutcircuits.com/technical-articles/spi-serial-peripheral-interface/