

Технологично училище „Електронни системи“ към
Технически Университет – София

Курсова работа

Tetris Display

Изготвил

Божидар Павлов

Преподавател

Енчо Шахънов

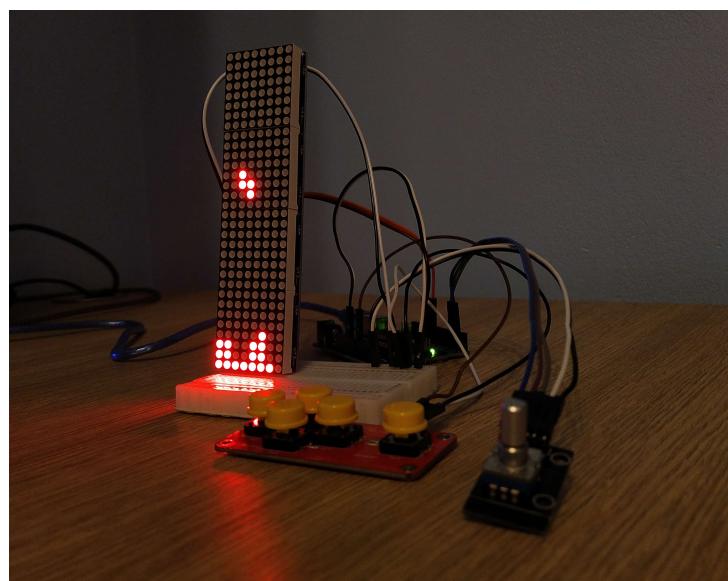
2023 г.

Увод

Tetris Display е конзола, на която потребителят може да играе *Tetris* с приятел. Това е идеалният начин да се забавляват заедно, като се наслаждават на една от най-популярните игри на всички времена.

Оригиналната игра Тетрис е създадена през 1984 г. в Москва от двама компютърни инженери от *Компютърния център към Руската академия на науките*. В оригиналната игра, единственият играч има за цел да подреди падащи фигури с различна форма, наречени Тетромино, като изцяло запълнените от фигури редове изчезват.

Този проект представлява модифициран вариант на Тетрис, който се играе от двама играчи, които играят един срещу друг. Играч 1 подрежда падащите Тетроминота, а Играч 2 управлява кое Тетромино ще падне на следващия ход. Ако Играч 2 пусне едно и също парче два пъти последователно, той губи, а пък ако неизчистените редове на Играч 1 достигнат горната част на игралното поле, Играч 2 печели.



Фигура 1: Първи прототип на Tetris Display

Глава 1

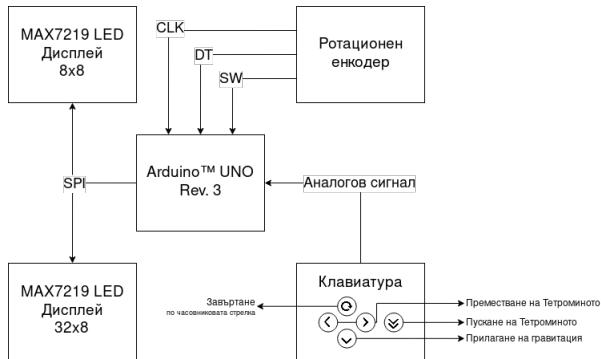
Теория

1.1. Принцип на работа на проекта

За реализацията на Tetris Display са необходими микроконтролер, клавиатура с минимум 5 бутона за контролиране на играта от Играч 1, ротационен енкодер за избор на следващо Тетромино от Играч 2, еcran, изобразяващ текущото състояние на играта и втори еcran, изобразяващ следващото Тетромино.

От момента на пускането си, микроконтролерът започва продължително да чете текущото състояние на клавиатурата и на ротационния сензор. В случай, че някой от бутоните за завъртане на падащото Тетромино бъде натиснат от Играч 1, или ротационният енкодер се завърти от Играч 2, промяната е отразена във вътрешното състояние на играта.

При натискане на бутона на оста на енкодера от Играч 2, текущата игра бива прекратена и започва нова игра отначало. Това е реализирано със софтуерно нулиране на UNO платката.



Фигура 1.1: Функционална схема на Tetris Display

1.2. Избор на хардуерни компоненти за реализацијата на проекта

1.2.1. Избор на микроконтролер

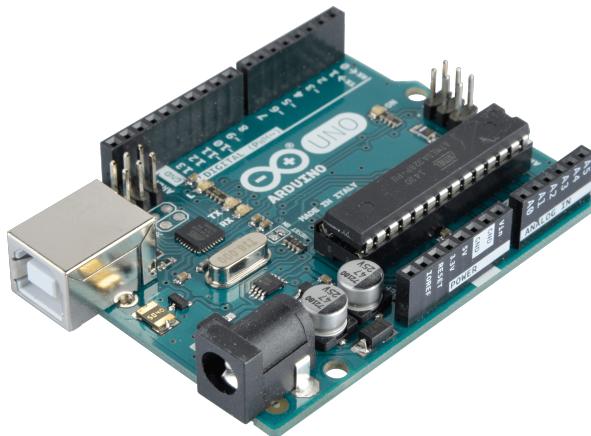
Arduino™ UNO ревизия 3

Arduino™ UNO [8] е най-достъпната, удобна и лесна за програмиране и употреба платка, както за начинаещи, така и за напреднали, правейки я най-разпространената *Arduino™* платка. Поради тази причина има изключително много информация за това как работи и как трябва да се програмира в интернет. Освен това заедно с платката е предоставена и лесна и интерактивна среда за програмиране – *Arduino™ IDE*, осигуряваща всичко необходимо откъм допълнителни библиотеки и софтуер свързани с програмирането на платката. Тези фактори я правят добър избор за проекта.

UNO разполага със 14 цифрови входно-изходни порта, 6 от които могат да са ШИМ¹ изходи, 6 аналогови входа и кварцов резонатор на 16 MHz.

В заданието на този курсов проект се изисква задължителната употреба на *Arduino™ UNO* платка.

¹Широчинно-импулсна модулация



Фигура 1.2: Развойна платка Arduino™ UNO

1.2.2. Избор на клавиатура

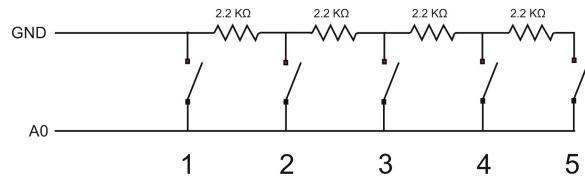
Проектът използва клавиатура с 5 бутона [1], която има аналогов вход за прочитане на петте състояния на всеки бутон. По този начин се спестява ресурса на процесора на управляващата платка. Платката на клавиатурата разполага с повече свободни входно/изходни пинове, които са неизползвани в реализацията на този проект. Платката разполага и с изходен конектор с три извода. Два от тях са за захранване, третият е аналогов изход. Клавиатурата е подходяща за връзка както с развойните платки UNO, NANO, така и PIC, MSP и други.



Фигура 1.3: Аналогова клавиатура

Схемата на свързване на тази клавиатура, изобразена на фигура 1.4 се състои от 5 ключа, които ако са затворени, дават успоредно свързаните резистори на късо, променяйки изходното напрежение. То може да бъде четено

тривиално от аналогов пин на развойната платка UNO.



Фигура 1.4: Принципна схема на аналогова клавиатура

1.2.3. Избор на сензор за завъртане

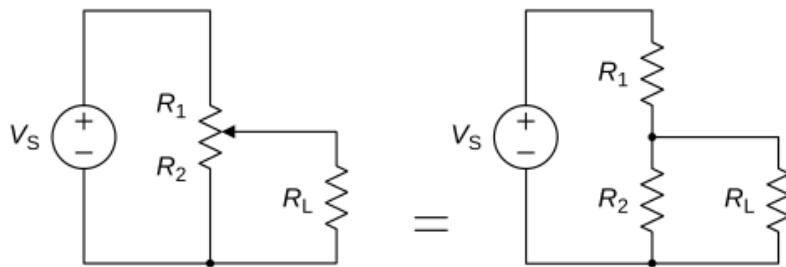
Потенциометър

За входно устройство възможен избор би бил потенциометър [6] – резистор с 3 извода, с който е възможно при промяната на съпротивлението му чрез плъзгащ контакт в електрическата верига, да се променя изходното напрежение.



Фигура 1.5: Кръгъл потенциометър

На фигура 1.6 е изобразена еквивалентната електрическа схема на един потенциометър. Ако свържем високия потенциал от входа към захранващия пин на нашия микроконтролер, а високия – към някой от заземените (GND) пинове, то тогава потенциалната разлика между средната точка и нулата може да бъде управлявана от ориентацията на плъзгача.



Фигура 1.6: Потенциометър в електрическа верига и неговата еквивалентна схема

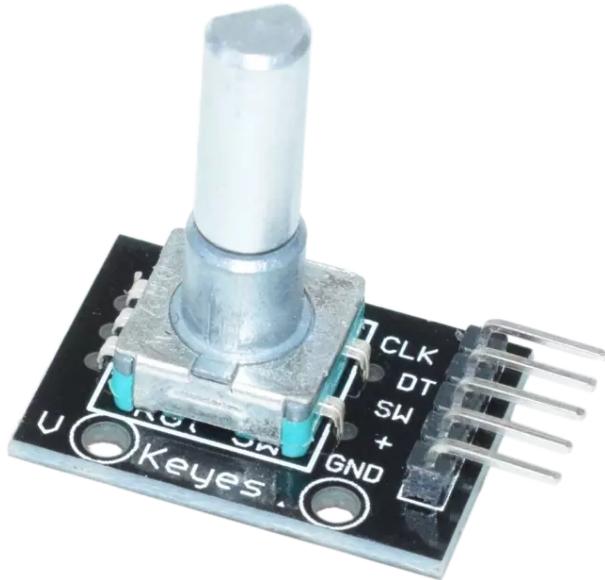
Ако свържем третия извод на потенциометъра към някой от аналоговите входове на UNO микроконтролера, ние ще можем софтуерно да четем уникална чисрова стойност от 0 до 1023, описваща настоящата позиция на плъзгача. Тази стойност може да управлява изборът на тетромино в нашата игра.

Главният недостатък на потенциометърът за проекта е, че плъзгачът може да бъде завъртан единствено в интервал от $\frac{3}{4}$ от целия кръг, но за реализацията на този проект е необходимо той да може свободно да бъде завъртан двупосочко в целия интервал от 360° . Потенциометърът е полезен в случаи, в които искаме да научим точната настояща позиция на плъзгача, но за управлението тя не е важна, тъй като промяната на изборът на Тетромино се управлява от промяната на ориентацията и посоката на въртене.

Ротационен енкодер

Следващият избор за сензор за въртене е ротационен енкодер [2]. Подобно на потенциометърът, ротационният енкодер представлява електромеханично устройство, което преобразува ъгловото положение или движението на вал или ос в аналогови или цифрови изходни сигнали, но за разлика от него, той може да бъде завъртан в интервал от 360° – както в посока на

часовниковата стрелка, така и в обратна на нея.



Фигура 1.7: Модул ротационен енкодер

Съществуват два основни типа ротационни енкодери: абсолютни и инкрементални. Изходът на абсолютния енкодер показва текущото положение на вала, което го прави преобразовател на ъгъл. Изходът на инкременталния енкодер предоставя информация за движението на вала, която обикновено се обработва на друго място в информация като позиция, скорост и разстояние.

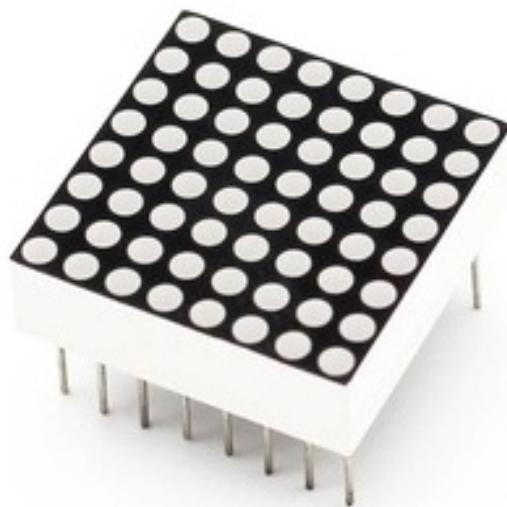
Краен избор

Тъй като на нас ни е необходимо да знаем по какъв начин е предвижен валът, за реализацията на този проект е използван инкрементален цифров ротационен енкодер.

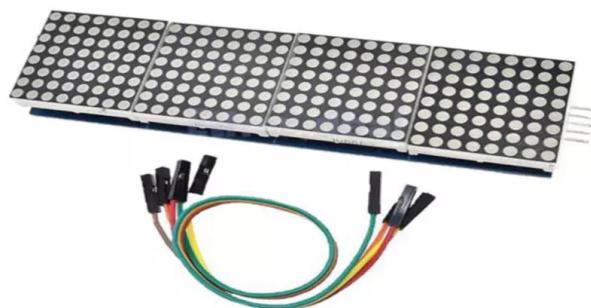
1.2.4. Избор на дисплеи

Проектът използва два матрични 1.8 LED дисплея – един с резолюция 32x8 пиксела и един с резолюция 8x8 пиксела. Дисплей 1 се управлява от четири MAX7219 [3] драйвера за дисплеи, всеки от които управлява зона от 8x8

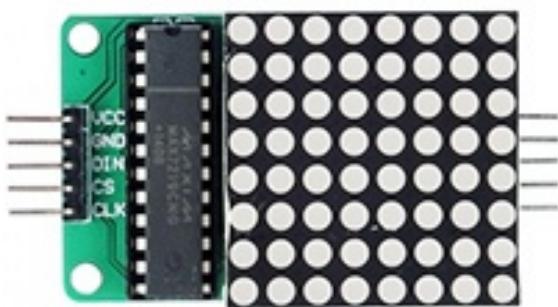
пиксела. Драйверите са свързани последователно, като част от FC-16 модула. Микроконтролерът управлява състоянието на дисплея чрез серийна комуникация, използваща SPI протокола. Дисплей 2 е управляван от единичен MAX7219 модул (фиг. 1.10).



Фигура 1.8: LED матрица с размер 8x8 без модул за управление



Фигура 1.9: MAX7219 FC-16 LED матрица с размер 32x8



Фигура 1.10: MAX7219 LED матрица с размер 8x8

Глава 2

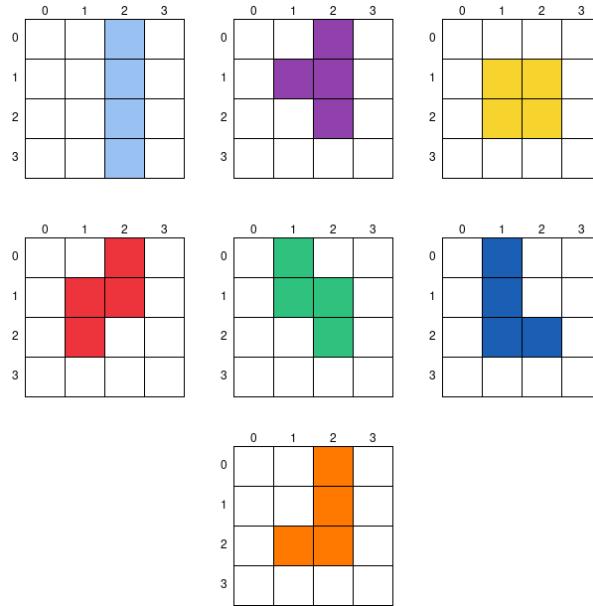
Практика

2.1. Изходен код

Тъй като реализацията на играта Тетрис изисква нетривиално количество изходен код, в тази документация са прикачени единствено ключовите части. Целият код може да бъде намерен в GitHub хранилището на проекта [7].

2.1.1. Кодиране на падащото Тетромино

В Тетрис съществуват седем различни Тетроминота, като всяко едно от тях може да бъде обърнато по 4 начина и всяко от тях може да бъде нанесено върху мрежа с размер 4x4 по начинът, изображен на фигура 2.1.



Фигура 2.1: *Тетроминота*, нанесени върху 4x4 мрежа с координатна система

Съответно в код се представят празните и запълнените позиции на фигурите като масив от символни низове.

```
constexpr char *tetrominoes[] =
{
    "...X..",
    "...X..",
    "...X..",
    "...X..",
    "...X..,"

    "...X..",
    ".XX..",
    "...X..",
    "... . . . ,

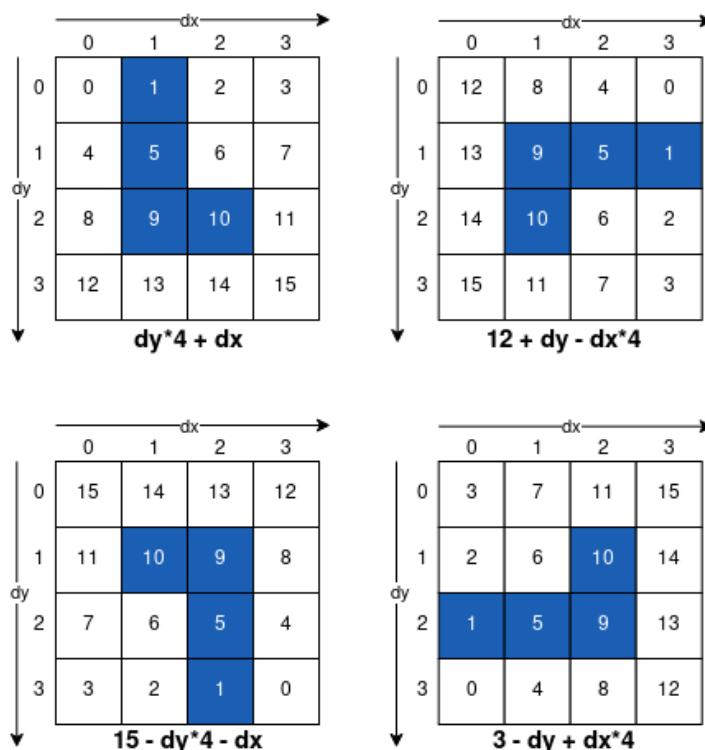
    // ...

    "...X..",
    "...X..",
    ".XX..",
    "... . . . ,
};
```

```
constexpr size_t num_tetrominoes =
    sizeof tetrominoes / sizeof *tetrominoes;
```

Вида Тетромино се представя с индекс от този масив – число от 0 до `num_tetrominoes`.

Различните ориентации на всяко едно Тетромино се получават като индексираме символните низове в различен ред. Всеки квадрат от 4×4 мрежата представлява точка от координатна система, описана на фигура 2.1. Нека точките в тази система имат координати dx и dy . Всичките възможни видове индексиране са описани по следния начин:



Фигура 2.2: Формули за индексиране на символните низове на Тетроминотата

```
const char (*rotations[])(int dx, int dy, int tetromino_idx) =
{
    [tetrominoes](int dx, int dy, int tetromino_idx)
        { return tetrominoes[tetromino_idx][dy * 4 + dx]; },
    [tetrominoes](int dx, int dy, int tetromino_idx)
        { return tetrominoes[tetromino_idx][12 + dy - (dx * 4)]; },
    [tetrominoes](int dx, int dy, int tetromino_idx)
        { return tetrominoes[tetromino_idx][15 - dy * 4 - dx]; },
    [tetrominoes](int dx, int dy, int tetromino_idx)
        { return tetrominoes[tetromino_idx][3 - dy + dx * 4]; }
};
```

```
[tetrominoes](int dx, int dy, int tetromino_idx)
{ return tetrominoes[tetromino_idx][15 - (dy * 4) - dx]; },
[tetrominoes](int dx, int dy, int tetromino_idx)
{ return tetrominoes[tetromino_idx][3 - dy + (dx * 4)]; },
};

constexpr size_t num_rotations = sizeof rotations / sizeof *rotations;
```

За оптимизация на бързодействието на алгоритъма, всяка ориентация представлява C++ ламбда функция, взимаща координатите dx и dy и индекса на вида Тетромино и връща ' .' , ако тази позиция е празна и 'X' , ако тя е запълнена.

Указатели към функциите се пазят в масива rotations, който е с дължина num_rotations. Следователно ориентацията на едно Тетромино се представя като индекс в този масив.

Освен вида и ориентацията на едно Тетромино, в цифровото представяне на едно Тетромино се съхраняват и неговите координати x и y, спрямо горния ляв пиксел на дисплея.

Така изглежда структурата за едно Тетромино:

```
struct Tetromino
{
    int x, y, type, rotation;
} tetromino;
```

tetromino е глобална променлива за падащата фигура в играта.

За тази структура съществуват следните методи:

- Tetromino :: move_left() и Tetromino :: move_right – премества позицията с 1 наляво/надясно
- Tetromino :: rotate() – променя ориентацията веднъж по часовниковата стрелка

- Tetromino :: apply_gravity() – премества позицията с 1 надолу
- Tetromino :: place() – утвърждава окончателно настоящата позиция, като я прибавя към вече поставените парчета (т. 2.1.2) и нулира позицията на Тетроминото.
- Tetromino :: drop() – премества позицията надолу, докато не стигне до най-ниското, достижимо ниво

2.1.2. Кодиране на поставените парчета

Поставените парчета се съхраняват в двумерен масив от тип `bool` с размери 8x32:

```
bool stack[DPY_WIDTH][DPY_HEIGHT]{false};
```

Ако `stack[x][y] = true`, позицията на дисплея с координати x, y е запълнена.

2.1.3. Четене на натиснат клавиши от клавиатурата

С цел по-четим код, се дефинира изброим тип, изреждащ всичките клавиши.

```
enum struct Key : byte
{
    None, Up, Down, Left, Right, Drop
};
```

Функцията `get_pressed_key()` връща стойност от Key изброимия тип, описваща натиснатия клавиши в момента на нейното извикване.

Стойностите на напрежението на аналоговия сигнал са нестабилни и поради тази причина е необходимо да се прави проверка в интервал около желаната стойност.

```
#define APPROX_EQ(val, ref) ((ref)-50 <= (val) && (val) <= (ref)+50)

Key key = Key::None;

Key get_pressed_key(uint8_t pin)
{
    int val = analogRead(pin);
    Key old_key = key;
    if (APPROX_EQ(val, 144))
        key = Key::Up;
    else if (APPROX_EQ(val, 328))
        key = Key::Down;
    else if (APPROX_EQ(val, 0))
        key = Key::Left;
    else if (APPROX_EQ(val, 504))
        key = Key::Right;
    else if (APPROX_EQ(val, 738))
        key = Key::Drop;
    else
        key = Key::None;
    return key != old_key ? key : Key::None;
}
```

2.1.4. Прочитане на състоянието на ротационния енкодер

За по-четим код, се дефинира структура, съдържаща необходимите методи.

```
struct Encoder
{
    uint8_t clk_pin, dt_pin, sw_pin;
    int rotation=0, last_clk=0;
    bool is_clockwise=false, button=false, has_rotated=false,
         was_button_pressed=false;

    Encoder(uint8_t clk_pin, uint8_t dt_pin, uint8_t sw_pin)
        : clk_pin(clk_pin), dt_pin(dt_pin), sw_pin(sw_pin)
    {
```

```

    }

void begin();
void read();
};

```

Като полета се запазват номерата на цифровите пинове, към които са свързани изходите на енкодера:

- CLK
- DT
- SW

Тяхната функция е документирана на по-късен етап.

Структурата също така съхранява полета за настоящото състояние на енкодера.

Инициализация на енкодера

`Encoder::begin()` функцията инициализира необходимите цифрови пинове като входни за микроконтролера и запазва настоящото състояние на CLK пина.

```

void Encoder::begin()
{
    pinMode(clk_pin, INPUT);
    pinMode(dt_pin, INPUT);
    pinMode(sw_pin, INPUT_PULLUP);
    last_clk = digitalRead(clk_pin);
}

constexpr int ENC_CLK_PIN=2, ENC_DT_PIN=3, ENC_SW_PIN=4;

Encoder enc(ENC_CLK_PIN, ENC_DT_PIN, ENC_SW_PIN);

```

```
void setup()
{
    enc.begin();
    // ...
}
```

Прочитане на състоянието на енкодера

`Encoder :: read()` функцията прочита състоянието от цифровите пинове, използвани от енкодера, в полетата на структурата.

Ако стойността на CLK е променена спрямо предходното му състояние, енкодерът е бил завъртян, като можем да научим посоката от DT пина.

От SW изхода на енкодера разбираме дали бутонът е натиснат.

```
void Encoder :: read()
{
    int current_clk = digitalRead(clk_pin);
    if (!last_clk && current_clk)
    {
        if (digitalRead(dt_pin))
            ++rotation, is_clockwise=true;
        else
            --rotation, is_clockwise=false;
        has_rotated = true;
    }
    else
        has_rotated = false;
    last_clk = current_clk;

    bool new_button = !digitalRead(sw_pin);
    was_button_pressed = !button && new_button;
    button = new_button;
}
```

2.1.5. SPI комуникация с матричния дисплей

За SPI комуникация с дисплеите е използвана `MD_MAX72XX` библиотеката от *magicDesigns* [4].

По следният начин се дефинира и инициализира дисплеите:

```
constexpr int DPY_WIDTH=8, DPY_HEIGHT=32;
constexpr int DPY_CS_PIN=10;

MD_MAX72XX dpy(MD_MAX72XX::FC16_HW, DPY_CS_PIN, DPY_HEIGHT/8);

void setup()
{
    // ...
    dpy.begin();
}
```

Обновяване на изображението на 32x8 дисплея

След всяка съществена промяна на състоянието на играта е необходимо изображението на дисплея да бъде обновено. Това става с `redraw()` функцията, която е извиквана при необходимост.

```
void redraw()
{
    dpy.clear();           // Целият дисплей се нулира.
    draw_tetromino(tetromino); // Падащото Тетромино се изобразява.
    draw_stack();          // Поставените Тетроминота се изобразяват.
}
```

Рисуването на Тетромино се изпълнява по следния начин:

```
void draw_tetromino(Tetromino t)
{
    for (int dx=0; dx<4; ++dx)
        for (int dy=0; dy<4; ++dy)
            dpy.setPoint(DPY_WIDTH-t.x-1-dx, DPY_HEIGHT-t.y-1-dy,
```

```
    rotations[t.rotation](dx, dy, t.type) = 'X');
```

Тъй като координатната система, използвана от кода, и координатната система на 32x8 MAX7219 дисплея не съвпадат, е необходимо да се извърши трансформация на координатите. По подобен начин се рисуват и вече поставените парчета:

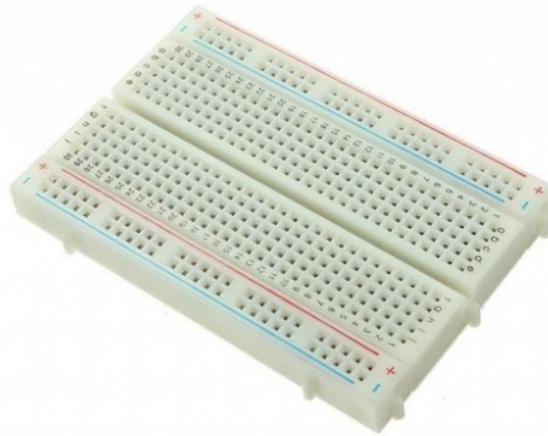
```
void draw_stack()
{
    for (int x=0; x<DPY_WIDTH; ++x)
        for (int y=0; y<DPY_HEIGHT; ++y)
            if (stack[x][y])
                dpy.setPoint(DPY_WIDTH-x-1, DPY_HEIGHT-y-1, true);
}
```

2.2. Свързване на електронните компоненти

При свързването на електронните компоненти, захранването (+5V) и земята (GND) е необходимо да бъдат общи за всичките компоненти. Допълнително, информационните и синхронизиращите входове на SPI комуникацията с двата дисплея се изисква да са свързани с обща шина. За реализирането на това, този проект си служи с breadboard. [5]

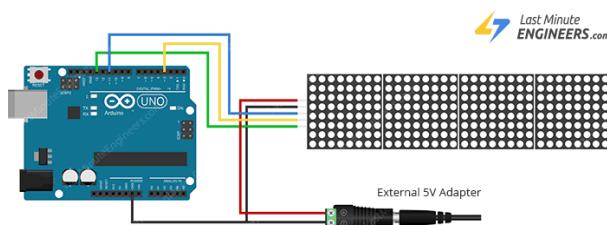
2.2.1. Свързване на MAX7219 дисплейте

Пиновете на 32x8 дисплея се свързват със следните пинове на ArduinoTM UNO микроконтролера:



Фигура 2.3: Breadboard

Пин от MAX7219	Пин от Arduino™ UNO
VCC	+5V
GND	GND
DIN	Цифров 11
CS	Цифров 10
CLK	Цифров 13



Фигура 2.4: Свързване на MAX7219 FC-16 модул

Пиновете на 8x8 дисплея се свързват със следните пинове на Arduino™ UNO микроконтролера:

Пин от MAX7219	Пин от Arduino™ UNO
VCC	+5V
GND	GND
DIN	Цифров 11
CS	Цифров 8
CLK	Цифров 13

2.2.2. Свързване на аналоговата клавиатура

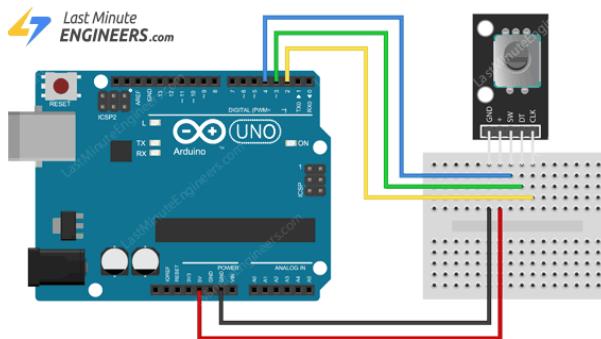
Пиновете на аналоговата клавиатура се свързват със следните пинове на Arduino™ UNO микроконтролера:

Пин от клавиатура	Пин от Arduino™ UNO
OUT	Аналогов 0
VCC	+5V
GND	GND

2.2.3. Свързване на ротационния енкодер

Пиновете на ротационния енкодер се свързват със следните пинове на Arduino™ UNO микроконтролера:

Пин от енкодера	Пин от Arduino™ UNO
CLK	Цифров 2
DT	Цифров 3
SW	Цифров 4
+	+5V
GND	GND



Фигура 2.5: Свързване на ротационен енкодер

2.3. Програмиране на ArduinoTM UNO

Процесът на програмиране на микроконтролера включва следните стъпки:

1. Свързване на електронните компоненти (т. 2.2)
2. Свързване на ArduinoTM UNO микроконтролера към компютър с USB-В кабел.
3. Изтегляне на инсталационен скрипт за Tetris Display от GitHub [7] на компютъра.
4. Изпълняване на инсталационния скрипт от компютъра.

Изтегляне на инсталационен скрипт за Tetris Display от GitHub на компютъра.

Всичките издания на Tetris Display са видими в Releases секцията от GitHub ханилището на проекта [7]. За различните операционни системи е необходимо да се използва различен инсталационен скрипт:

Изпълняване на инсталационния скрипт от компютъра.

Операционна система	Име на файл
GNU/Linux	tetris_display_install_gnu.sh
Windows	tetris_display_install_win.ps1

Необходимо е скриптът за операционната система на компютъра да бъде изтеглен от GitHub хранилището, да бъде изпълнен и да бъдат следвани указанията, написани в конзолния прозорец.

След успешно изпълнение, скриптът ще завърши със следното съобщение:

...
Tetris Display successfully flashed!

В този случай Arduino™ UNO микроконтролерът е успешно програмиран и играта ще бъде пусната на свързания дисплей.

Заключение

В заключение, проектът Tetris Display представя нова вариация на старата и харесвана игра Тетрис, която е играна от двама играчи.

В бъдеще Tetris Display може да бъде развит със въвеждането на цветен дисплей, фонова музика и звукови ефекти по време на играта, помагащи на играчите да се потопят в нея.

Tetris Display е идеална конзола за партита, рождения дни или просто за една релаксираща игра с приятели.

Използвана литература

- [1] Martyn Currey. *Keypads and Button Switches on the Arduino*.
<https://www.martyncurrey.com/keypads/>. Дек. 2013.
- [2] Last Minute Engineers. *How Rotary Encoder Works and Interface It with Arduino*.
<https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/>.
- [3] Last Minute Engineers. *Interfacing MAX7219 LED Dot Matrix Display with Arduino*.
<https://lastminuteengineers.com/max7219-dot-matrix-arduino-tutorial/>.
- [4] majicDesigns. *MAX72xx LED Matrix Display Library*.
https://github.com/MajicDesigns/MD_MAX72XX.
- [5] Wikipedia. *Breadboard*.
<https://en.wikipedia.org/wiki/Breadboard>.
- [6] Wikipedia. *Потенциометър*.
<https://bg.wikipedia.org/wiki/Потенциометър>.
- [7] Божидар Павлов. *GitHub Хранилище на Tetris Display*.
<https://github.com/bvpav/tetris-display>.
- [8] Уебсайт на *Arduino™*.
<https://www.arduino.cc/>.

Съдържание

Увод	2
1 Теория	3
1.1 Принцип на работа на проекта	3
1.2 Избор на хардуерни компоненти за реализацията на проекта	4
2 Практика	11
2.1 Изходен код	11
2.2 Свързване на електронните компоненти	20
2.3 Програмиране на Arduino™ UNO	23
Заключение	25
Използвана литература	26
Съдържание	27