

Проектирование устройств

Курсовой проект

Евгений Зеленин

1 июня 2025 г.

1. Постановка задачи

Условия задачи. Реализация проекта игровой консоли на базе *Arduino UNO* с SPI TFT экраном 320x240. Цель - сделать портативную консоль с двумя встроенными играми, реализовать сохранение параметров в EEPROM, обработку внешних событий через прерывания, отслеживание заряда батареи, звуковую индикацию. Управление осуществляется энкодером. Предусмотреть возможность спящего режима для снижения энергопотребления во время простоя. Разработать корпус устройства.

Введение

Существует достаточно большое количество проектов игровых консолей на ардуино. В текущем проекте сделан упор на следующие ключевые моменты, добавляющие элементы новизны в устройство:

- Сохранение настроек консоли и достижений игрока в энергонезависимой памяти
- Спящий режим для обеспечения длительной автономности устройства
- Реализация цепи защиты питания от переполюсовки
- Доработка платы Arduino с целью снижения паразитного энергопотребления
- Повышение эффективности прошивки с помощью использования структур с битовыми полями, битовых массивов, рекурсивных алгоритмов
- Реализация системы меню
- Воплощение устройства в корпусе с возможностью полноценного использования

2. Схема устройства

Для реализации устройства выбраны типовые компоненты: SPI дисплей 320x240, пьезоизлучатель, модуль энкодера для Arduino, резисторы, р-канальный MOSFET для защиты от переполюсовки. Контроллер SPI-дисплея рассчитан на логические уровни в 3.3в, в нашем случае, для достижения таких уровней будет достаточно использовать ограничительные резисторы номиналом 1.5к. Мониторинг напряжения питания осуществляется с помощью АЦП и резистивного делителя 10к:100к. В качестве опорного напряжения АЦП используется встроенный в МК делитель на 1.1в. Устройство собрано на монтажной плате, схема показана на рисунке 1.

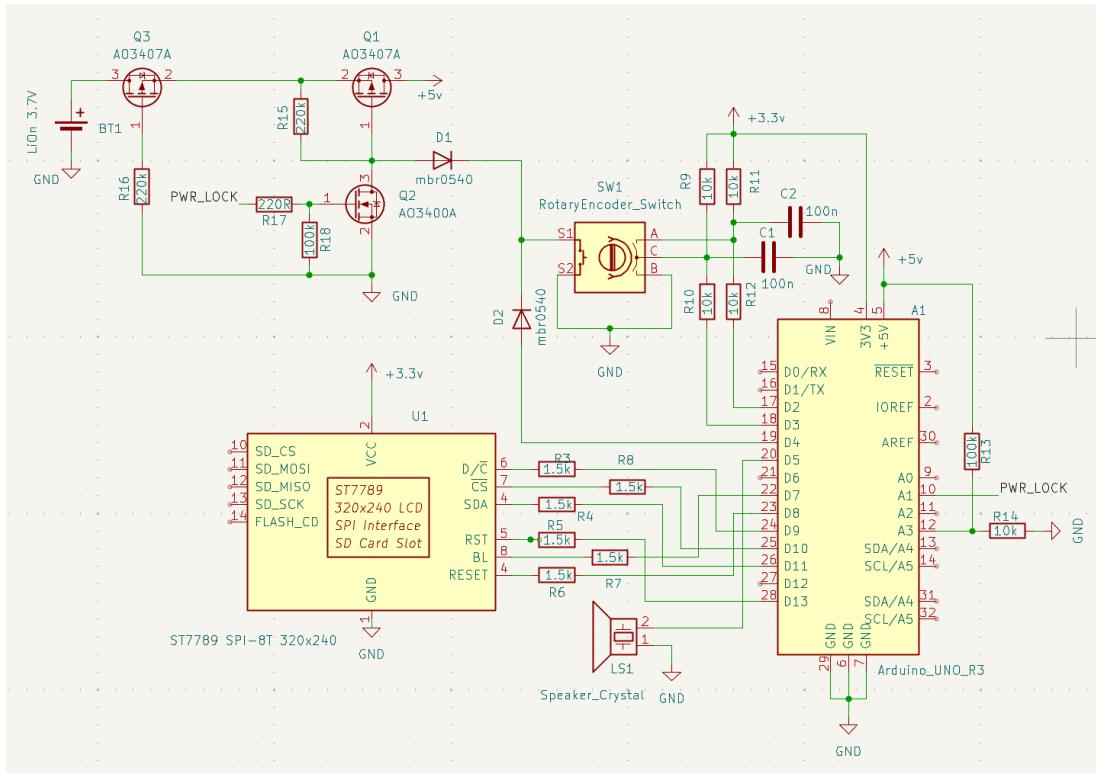


Рис. 1: Схема устройства

3. Проектирование корпуса

Корпус был спроектирован в САПР Компас-3Д. Размеры снимались штангенциркулем с собранного на монтажной плате устройства. В корпусе предусмотрен отсек для батареи со встроенной пластиковой пружиной и съемной крышкой.

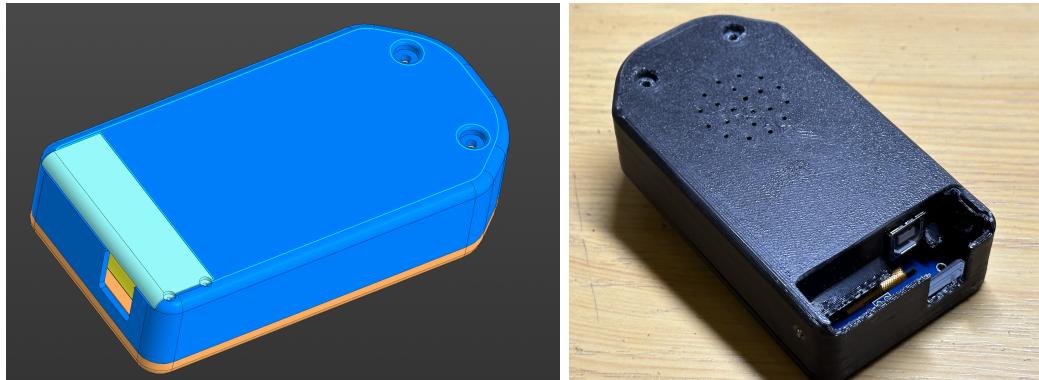


Рис. 2: Корпус устройства

Также, предусмотрена возможность подключения USB-кабеля, когда батарея извлечена из корпуса. Верхняя крышка выполнена съемной, с одной стороны крепится на защелку, с другой на два самореза. В нижней крышке предусмотрен крепеж для динамика и упоры под энкодер (чтобы избежать продавливания). Дисплей и обе платы также надежно фиксируются пластиковыми упорами встроеннымными в элементы корпуса. На рисунках 2, 3 показаны скриношоты проекта и его реализация с помощью 3D-печати.

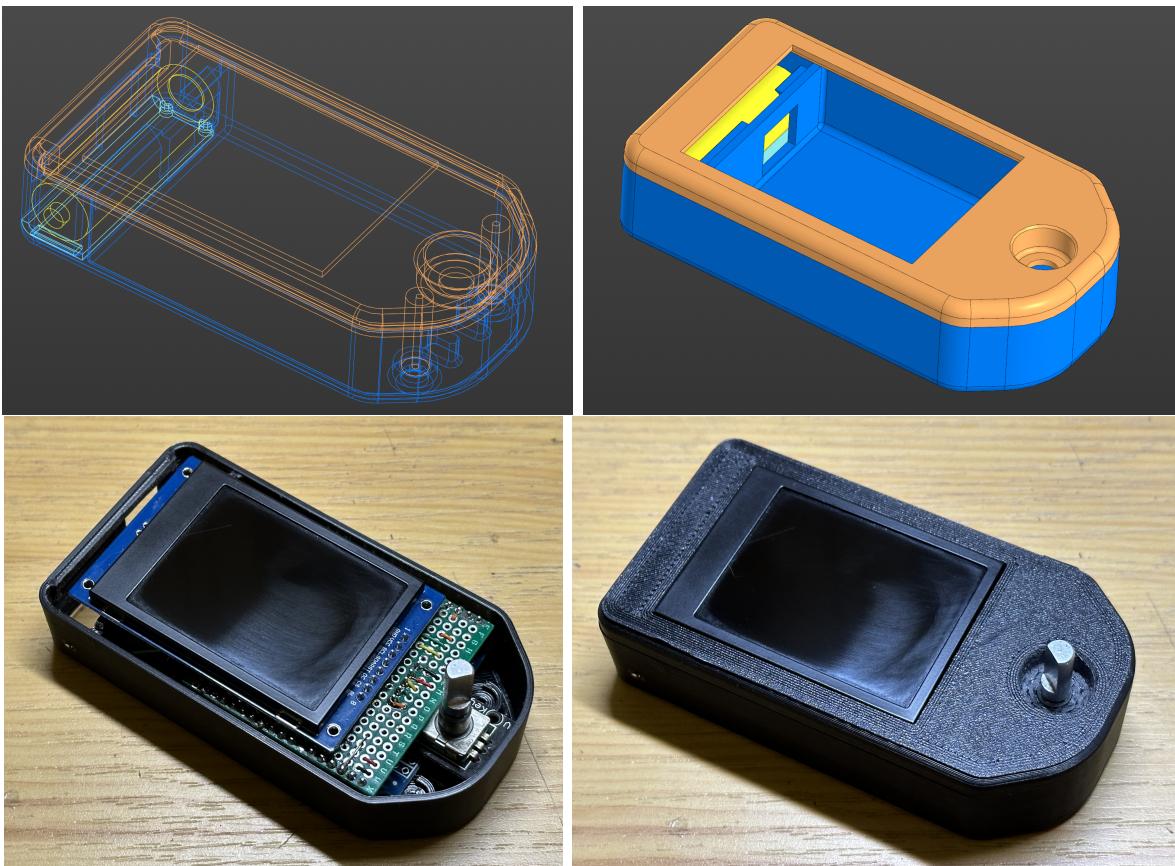


Рис. 3: Корпус устройства

Чтобы все компоненты встали на свои места с посадкой в 0.2мм, итеративно было разработано несколько прототипов корпуса. На рисунке 4 показан процесс сборки устройства, и прототипы на различных этапах проектирования.



Рис. 4: Сборка устройства и прототипы

4. Интерфейс пользователя Взаимодействие с устройством осуществляется с помощью энкодера, снабженного кнопкой. Для вывода консоли из энергосберегающего режима необходимо нажать на энкодер. По умолчанию, запускается игра выбранная в настройках (тетрис или колонки), а для запуска меню нужно зажать энкодер в момент простоя. Меню содержит следующие пункты:

- SOUND (звук)
- TETRIS (режим тетриса)
- RESET EEPROM (сброс настроек)
- COLOR (цвет фигур тетриса)
- SCREEN TIME (время заставки)
- EXIT MENU (выйти из меню)

Навигация по меню: выбор пункта осуществляется вращением энкодера, а изменение значения - кратковременным нажатием. На рисунке 5 показан интерфейс пользователя в различных ситуациях: меню, игра Columns, игра Tetris.



Рис. 5: Меню, Columns, Tetris

Цель игры колонки - собрать 4 и более квадратов одного цвета. Чем больше различных фигур соберется за раз - тем больше очков получит пользователь.

Цель игры тетрис - заполнять линии. Максимальное количество очковдается за 4 заполненные линии. Изменение сочетания цветов и поворота фигуры осуществляется кратковременным нажатием на энкодер, а падение фигуру осуществляется длительным нажатием. Повороты энкодера влево и вправо приводят к перемещению фигур по горизонтали.

5. Спящий режим

В активном режиме устройство имеет ток потребления в 85mA. После определенного промежутка времени (задается в меню), консоль переходит в ждущий режим. Ток спящего режима составляет 2.5mA (в сон переводится как микроконтроллер дисплея и сам дисплей, так и основной контроллер). Для достижения этих показателей с платы Arduino UNO пришлось демонтировать светодиоды, а питание USB-TTL преобразователя на микросхеме CH341 отключить от цепи батареи и перенести непосредственно на USB-порт (Рисунок 6). Дисплей в режиме сна обладает потреблением около 1.5mA, а плата Arduino UNO - около 1mA. Питание устройства осуществляется от LiOn аккумулятора формата АА емкостью 900mAч, время разряда аккумулятора в простое составляет около двух недель.

После сдачи курсового было добавлено функционал по отключению питания с помощью двух полевых транзисторов: Q1 и Q2. Устройство включается нажатием на кнопку энкодера (открывается транзистор Q1 через диод D3), затем, после включения MK, подается управляющий сигнал на Q2 с пина A1 и таким образом устройство удерживается во включенном состоянии необходимое время. Для включения устройства требуется удерживать кнопку включения около 0.1с. Диод D4 необходим чтобы избежать самопроизвольного (частичного) открывания Q1 из-за протекания тока через подтягивающий резистор порта D4.

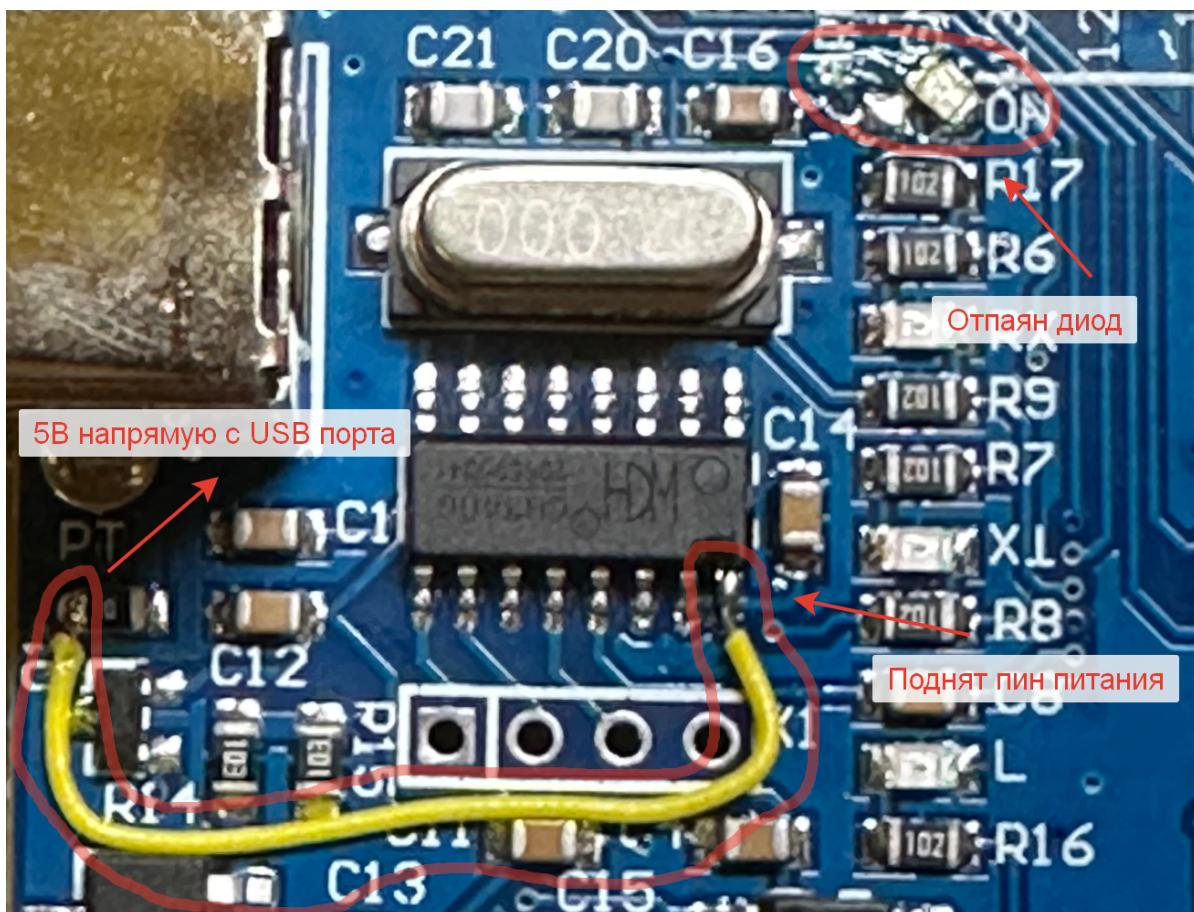


Рис. 6: Доработка Arduino UNO

6. Исходный код проекта

```
/* @Molnija3D
// I was inspired with https://mysku.club/blog/diy/65413.html when working on this project
// It could work with 3.7v LiOn AA battery.
// If you dismantle all LEDS and disconnect power pin from CH341 it will be capable to run
// You can solder CH341 power directly from USB socket.
// To launch menu hold encoder button during screensaver
// Battery monitoring was made with 10:100k resistors divider at ADC port (PIN_BATT A3)
// To erase EEPROM, set "ERASE EEPROM" to 1 and remove a power from the device.
// Here is used 320x240 SPI display.
// Идея игры "Columns" и принцип управления энкодером из этого очень интересного проекта
// Если отпаять светодиоды с платы и перенести питание CH341 на прямую от USB порта,
// то в режиме сна потребление около 2.5mA. Для выхода из спящего режима - нажать на кнопку
// В игре реализована система меню. Для входа в которое нужно зажать энкодер во время проектирования
// Из меню можно включить или отключить звук, выбрать режим тетриса, цвет кубиков, длительность
// для сброса EEPROM нужно включить в 1 соответствующий параметр меню и перезагрузить устройство
// Индикатор заряда сделан с помощью делителя 10:100кОм и АЦП.
// Используется SPI дисплей 320x240
*/
#include <Adafruit_GFX.h>
#include <Adafruit_ST7789.h>
#include <SPI.h>
#include <TimerOne.h>
#include <Encod_erer.h>
#include <avr/eeprom.h>
#include <avr/sleep.h>

#define DEBUG 0
// DEBUG 0 - нет отладки через Serial
// DEBUG 0-9 - уронви отладки
// пин LED+ на TFT_BL через резистор 1к
#define TFT_CS 10 // пин 10 подключаем к CS дисплея
#define TFT_RST 8 // пин 8 подключаем к RST дисплея
#define TFT_DC 9 // пин 9 подключаем к DS дисплея
#define TFT_BL 7 // пин 7 подсветка
#define PIN_DT 4 // пин 4 подключаем к DT энкодера
#define PIN_CLK 2 // пин 2 подключаем к CLK энкодера
#define PIN_SW 3 // пин 3 подключаем к SW энкодера
#define PIN_SPK 5 // пин 5 подключаем к + пищалки (- пищалки на землю)
#define PIN_BATT A3 // аналоговый пин 3 - делитель батареи
#define PIN_LOCK A1 // защелка питания

#define no_push 0
#define short_push 1
#define long_push 2
#define LONG_PUSH_MS 350 // длительность длинного нажатия
```

```

#define DEBOUNCE 50          // устранение дребезка кнопки

#define MAX_X 13
#define MAX_Y 22

#define SHFT_X 3
#define SHFT_Y 5

#define L_INFO 198 //левая координата информационного окна
#define B_FIELD 270 //нижняя координата игрового поля
#define Y_INFO 80 // верхняя координата информационного поля (надпись LEVEL)

#define MENU_X 24 //размеры меню
#define MENU_Y 34
#define MENU_W 142
#define MENU_H 160
#define MENU_T 70 //положение текста меню

#define TIMER_MS 250
#define PUZZLE_SZ 15 // размер квадрата
#define TOTAL_COLORS 6 // количество цветов квадратиков
#define MAX_LEVEL 9 // макс. кол-во уровней
#define NEXT_LEVEL 80 // через сколько столбиков повышать уровень

#define EEPROM_KEY 14
#define KEY_ADDR 101
#define PARAM_ADDR 50
#define GDATA_ADDR 1

uint8_t a_field[MAX_X][MAX_Y], a_prev[MAX_X][MAX_Y];

Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);
Encod_er encoder(PIN_CLK, PIN_DT, PIN_SW);

uint8_t fig_count, full_fig_count, cur_level = 1, prev_level, speed;
const uint16_t colors[] = { ST77XX_WHITE, ST77XX_BLACK, ST77XX_RED, ST77XX_BLUE, ST77XX_
uint32_t last_activity = 0, battery_time = 0;
uint16_t TScore = 0, top_score = 0, score = 0;
int prev_batt = 2000;
int8_t enc_rot, menu_item = 0, push_res, x, y;
const uint8_t a_speeds[MAX_LEVEL] = { 50, 45, 40, 35, 30, 25, 20, 10, 5 }; // задержки

typedef struct {
    uint16_t columns_record;
    uint16_t columns_games;
    uint8_t columns_level;
    uint8_t tetris_level;

```

```

        uint16_t tetris_record;
        uint16_t tetris_games;
    } eeprom_data;
eeprom_data game_data;

typedef struct { //упаковка в структуру
    uint8_t demo : 1;
    uint8_t in_game : 1;
    uint8_t new_game : 1;
    uint8_t allow_mv : 1;
    uint8_t tetris : 1;
    uint8_t sound : 1;
    uint8_t menu : 1;
    uint8_t menu_update : 1;
    uint8_t t_color : 3;
    uint8_t reserved : 1; // зарезервировано на будущее
    uint8_t screen_time : 4;
} b_params;
b_params params{ 0 };

typedef struct block { //определение списка с координатами одинаковых блоков
    uint8_t x;
    uint8_t y;
    block *next;
} block;

typedef struct { //отвечает за разрешение обновления информации в полях
    uint8_t score : 1;
    uint8_t top_score : 1;
    uint8_t level : 1;
    uint8_t max_level : 1;
    uint8_t games : 1;
} my_bools;
my_bools changed = { 1, 1, 1, 1, 1 };

typedef struct {
    uint8_t shape : 3;
    int8_t x : 5;
    uint8_t color : 3;
    uint8_t y : 5;
    uint8_t rot : 2;
    uint8_t do_rotate : 1;
} tetris_figure;
tetris_figure tt;

//=====
//      array 0, 2:          array 1,3:           y - rotation axis

```

```

//      x3 x2 x1 x0      *  * y3      * | *
//      * * * * y1      *  * y2      * * | * *
//      * * * * y0      *  * y1      ----- x - rotation axis
//      x3 x2 x1 x0 .   *  * y0      *  * | * *
//                           x1 x0 .   * | *
//
const uint8_t tetris_shapes[7][4] = { { 0b00001111, 0b00001111, 0b11110000, 0b11110000 },
                                     { 0b00100111, 0b11100100, 0b01110010, 0b01001110 },
                                     { 0b01100110, 0b01100110, 0b01100110, 0b01100110 },
                                     { 0b00110110, 0b11000110, 0b00110110, 0b11000110 },
                                     { 0b01100011, 0b01101100, 0b01100011, 0b01101100 },
                                     { 0b10001110, 0b11101000, 0b01110001, 0b00010111 },
                                     { 0b00010111, 0b01110001, 0b11101000, 0b10001110 } }
//                                hor      vert      hor      vert
//                                rot = 0    rot = 1    rot = 2    rot = 3
//=====================================================================
//=====================================================================

// Прототипы функций
void navigate_menu();
void upd_info();
void menu();
void draw_menu_asterisk(uint8_t yy, bool mode);
void process_pushes();
void get_movements();
void my_delay(int delay);
void navigate_tetris();
void navigate_columns();
void upd_field();
void update_tetris();
void draw_columns();
bool if_move(int8_t dx, int8_t dy);
bool move_tetris(int8_t dx, int8_t dy);
void full_tetris();
bool check_tetris(int8_t t_x, int8_t t_y, uint8_t shape, uint8_t rot);
void draw_tetris(int8_t t_x, int8_t t_y, uint8_t shape, uint8_t rot, uint8_t color);
bool read_shape(uint8_t tx, uint8_t ty, uint8_t shape, uint8_t rot);
bool move_columns(int8_t dx, int8_t dy);
void draw_square(uint8_t xx, uint8_t yy, uint8_t mycolor);
void draw_next();
uint8_t my_push();
bool add_node(uint8_t xx, uint8_t yy, block *full_body);
void find_full_block(uint8_t xx, uint8_t yy, block *full_body, uint8_t dir);
void free_block(block *full_body);
void paint_black(block *full_body);
bool delete_black();
bool search();
bool analyze();

```

```

void update();
void new_game();
void begin_tetris();
void begin_columns();
void gameover();
void my_tone(uint8_t pwm, uint16_t delay);
void screensaver();
void go_to_sleep();
void init_tft();
void my_print_num(uint32_t num, uint8_t digits);
void show_battery_level();
void draw_battery_bars(uint8_t num, uint16_t color);
void eeprom_enable();
void wakeup();
void timerInterrupt();
//=====
//=====

void setup() {
    pinMode(PIN_LOCK, OUTPUT);
    digitalWrite(PIN_LOCK, HIGH);

#if (DEBUG)
    Serial.begin(9600);
#endif

    eeprom_enable();
    randomSeed(analogRead(0));
    analogReference(INTERNAL);

    init_tft();
    upd_info();
    draw_next();

    Timer1.initialize(TIMER_MS);           // инициализация таймера 1, период
    Timer1.attachInterrupt(timerInterrupt, TIMER_MS); // задаем обработчик прерываний
    pinMode(PIN_DT, INPUT);
    pinMode(PIN_CLK, INPUT);
    pinMode(PIN_BATT, INPUT);
    pinMode(PIN_SW, INPUT_PULLUP);
    pinMode(TFT_BL, OUTPUT);

    digitalWrite(PIN_SW, HIGH); // подтягиваем к 5V
    delay(100);
    encoder.timeLeft = 0;
    encoder.timeRight = 0;

    memset(a_field, 0, sizeof(a_field));
}

```

```

last_activity = millis();
}

//=====
void loop() {
    if (params.in_game) {
        my_delay(10 * speed);
        if_move(0, 1); //падение вниз на 1 клетку
    } else if (params.demo) {
        screensaver();
    } else if (params.new_game) {
        new_game();
    } else if (params.menu) {
        menu();
        my_delay(200);
    }

    if (millis() - last_activity > ((uint32_t)params.screen_time + 1) * 5000) {
        go_to_sleep();
    }
}

//=====
void navigate_menu() { // навигация по меню
    if (enc_rot != 0) {
        menu_item += enc_rot; //если было вращение энкодера, переместиться на другой пункт
        menu_item < 0 ? menu_item = 5 : menu_item > 5 ? menu_item = 0 : menu_item;
        last_activity = millis();
    }
    switch (menu_item) {
        case 0:
            draw_menu_asterisk(MENU_T + 15 * 5, 0); //затереть звездочку в старом положении
            draw_menu_asterisk(MENU_T, 1);
            draw_menu_asterisk(MENU_T + 15, 0); //поставить новую звездочку
            if (push_res == short_push) {
                params.sound = !params.sound;
                params.menu_update = true;
            }
            break;
        case 1:
            draw_menu_asterisk(MENU_T + 15 * (menu_item - 1), 0);
            draw_menu_asterisk(MENU_T + 15 * menu_item, 1);
            draw_menu_asterisk(MENU_T + 15 * (menu_item + 1), 0);
            if (push_res == short_push) {
                params.tetris = !params.tetris;
                params.menu_update = true;
            }
    }
}

```

```

        memset(a_field, 0, sizeof(a_field));
        top_score = 0;
    }
    break;
case 2:
    draw_menu_asterisk(MENU_T + 15 * (menu_item - 1), 0);
    draw_menu_asterisk(MENU_T + 15 * menu_item, 1);
    draw_menu_asterisk(MENU_T + 15 * (menu_item + 1), 0);
    if (push_res == short_push) {
        eeprom_write_byte(KEY_ADDR, eeprom_read_byte(KEY_ADDR) != 0 ? 0 : EEPROM_KEY);
        params.menu_update = true;
    }
    break;
case 3:
//{ ST77XX_WHITE, ST77XX_BLACK, ST77XX_RED, ST77XX_BLUE, ST77XX_GREEN, ST77XX_YELLOW
    draw_menu_asterisk(MENU_T + 15 * (menu_item - 1), 0);
    draw_menu_asterisk(MENU_T + 15 * menu_item, 1);
    draw_menu_asterisk(MENU_T + 15 * (menu_item + 1), 0);
    if (push_res == short_push) {
        tt.color < 7 ? tt.color++ : tt.color = 2;
        params.t_color = tt.color;
        params.menu_update = true;
    }
    break;
case 4:
    draw_menu_asterisk(MENU_T + 15 * (menu_item - 1), 0);
    draw_menu_asterisk(MENU_T + 15 * menu_item, 1);
    draw_menu_asterisk(MENU_T + 15 * (menu_item + 1), 0);
    if (push_res == short_push) {
        params.screen_time < 15 ? ++params.screen_time : params.screen_time = 0;
        params.menu_update = true;
    }
    break;
case 5:
    draw_menu_asterisk(MENU_T + 15 * (menu_item - 1), 0);
    draw_menu_asterisk(MENU_T + 15 * menu_item, 1);
    draw_menu_asterisk(MENU_T, 0);

    if (push_res == short_push) {
        params.menu = false;
        params.demo = true;
        menu_item = 0;
        eeprom_write_block((void *)&params, PARAM_ADDR, sizeof(params));
        changed = (my_bools){ 1, 1, 1, 1, 1 };
    }

    init_tft();
    upd_info();

```

```

        draw_next();

        prev_batt = 2000;
        show_battery_level();
    }
    break;
}
}

//=====
void upd_info() {
    uint32_t cur_time = millis();
    if (cur_time - battery_time > 1000) {
        show_battery_level();
        battery_time = cur_time;
    }

    tft.setTextColor(ST77XX_BLUE);
    tft.setTextSize(1);
    /**
     if (changed.score) {                                //если есть изменения, пере
        tft.fillRect(L_INFO, Y_INFO + 45, 30, 8, ST77XX_WHITE); // 113 - 60
        tft.setCursor(L_INFO, Y_INFO + 45);                  // 113 - 60
        my_print_num(score, 5);
        changed.score = false;
    }
    /**
     if (changed.top_score) {
        tft.setCursor(L_INFO, Y_INFO + 90);
        tft.fillRect(L_INFO, Y_INFO + 90, 30, 8, ST77XX_WHITE);
        my_print_num(params.tetris ? game_data.tetris_record : game_data.columns_record, 5);
        changed.top_score = false;
    }
    /**
     if (changed.level) {
        tft.fillRect(L_INFO + 8, Y_INFO + 15, 30, 20, ST77XX_WHITE);
        tft.setCursor(L_INFO + 8, Y_INFO + 15);
        tft.setTextSize(2);
        tft.print(cur_level);
        changed.level = false;
    }
    /**
     if (changed.max_level) {
        tft.fillRect(L_INFO + 8, Y_INFO + 125, 30, 10, ST77XX_WHITE);
        tft.setCursor(L_INFO + 8, Y_INFO + 125);
        tft.setTextSize(1);
        tft.print(params.tetris ? game_data.tetris_level : game_data.columns_level);
    }
}

```

```

        changed.max_level = false;
    }
    /**
     if (changed.games) {
        tft.fillRect(L_INFO, Y_INFO + 165, 30, 8, ST77XX_WHITE);
        tft.setCursor(L_INFO, Y_INFO + 165);
        tft.setTextSize(1);
        params.tetris ? my_print_num(game_data.tetris_games, 5) : my_print_num(game_data.col
        changed.games = false;
    }
}

//=====
void menu() { // отрисовка меню
    if (params.menu_update) {
        tft.drawRect(MENU_X, MENU_Y, MENU_W, MENU_H, ST77XX_BLACK);
        tft.fillRect(MENU_X + 1, MENU_Y + 1, MENU_W - 2, MENU_H - 2, ST77XX_WHITE);
        tft.setCursor(MENU_X + 1, MENU_T - 25);
        tft.setTextSize(2);
        tft.setTextColor(ST77XX_BLUE);
        tft.print(" MENU: ");

        tft.setCursor(MENU_X + 1, MENU_T);
        tft.setTextSize(1);
        tft.setTextColor(ST77XX_BLACK);
        params.sound ? tft.print(" SOUND = 1") : tft.print(" SOUND = 0");

        tft.setCursor(MENU_X + 1, MENU_T + 15);
        tft.setTextSize(1);
        tft.setTextColor(ST77XX_BLACK);
        params.tetris ? tft.print(" TETRIS = 1") : tft.print(" TETRIS = 0");

        tft.setCursor(MENU_X + 1, MENU_T + 30);
        tft.setTextSize(1);
        tft.setTextColor(ST77XX_BLACK);
        eeprom_read_byte(KEY_ADDR) == EEPROM_KEY ? tft.print(" RESET EEPROM = 0") : tft.pr

        //{{ ST77XX_WHITE, ST77XX_BLACK, ST77XX_RED, ST77XX_BLUE, ST77XX_GREEN, ST77XX_YELLOW
        // отображение выбранного цвета
        tft.setCursor(MENU_X + 1, MENU_T + 45);
        tft.setTextSize(1);
        tft.setTextColor(ST77XX_BLACK);
        tft.print(" COLOR = ");
        tft.fillRect(MENU_X + 60, MENU_T + 42, 26, 13, colors[tt.color]);

        tft.setCursor(MENU_X + 1, MENU_T + 60);

```

```

tft.setTextSize(1);
tft.setTextColor(ST77XX_BLACK);
tft.print("  SCREEN TIME = ");
tft.print(5 * (params.screen_time + 1));

tft.setCursor(MENU_X + 1, MENU_T + 75);
tft.setTextSize(1);
tft.setTextColor(ST77XX_BLACK);
tft.print("  EXIT MENU");

memset(a_prev, 255, sizeof(a_prev)); //чтобы обновить экран при выходе из меню
params.menu_update = false;
}

//=====
void draw_menu_asterisk(uint8_t yy, bool mode) { //поставить звездочку
    tft.setCursor(MENU_X + 1, yy);
    tft.setTextSize(1);
    tft.setTextColor(ST77XX_BLACK);
    if (mode) {
        tft.print(" *");
    } else {
        tft.fillRect(MENU_X + 1, yy, 12, 7, ST77XX_WHITE);
    }
}

//=====
void process_pushes() { //обработка нажатий в паузе
    if (enc_rot != 0 || push_res == short_push) {
        params.new_game = true;
        params.demo = false;
    } else if (push_res == long_push) {
        params.menu = true;
        params.menu_update = true;
        params.demo = false;
    }
}

//=====
void get_movements() { //обработка движений в паузе
    enc_rot = 0;
    push_res = my_push();
    if (encoder.timeRight) // обрабатываем повороты энкодера
    {
        enc_rot = 1;
        encoder.timeRight = 0;
    } else if (encoder.timeLeft) {

```

```

    enc_rot = -1;
    encoder.timeLeft = 0;
}

if (!digitalRead(PIN_SW)) {
    enc_rot = 0;
    // last_activity = millis();
}
}

//=====
void my_delay(int delay) { // работаем, пока пауза
    unsigned long starttime = millis();
    while (millis() - starttime < delay) {
        get_movements();
        if (params.in_game && params.allow_mv) {
            if (params.tetris) {
                navigate_tetris();
            } else {
                navigate_columns();
            }
        } else if (params.menu && !params.menu_update) {
            navigate_menu();
        } else if (!params.in_game) {
            process_pushes();
        }
    }
}

//=====
void navigate_tetris() { //навигация в тетрисе
    if (enc_rot) {
        if_move(enc_rot, 0); //движение в бок
    } else if (push_res == short_push) { // вращение
        tt.do_rotate = true;
        if_move(0, 0); //вращение
    } else if (push_res == long_push) {
        speed = 5; // падение
    }
}

//=====
void navigate_columns() { //навигация в игре "columns"
    if (enc_rot != 0 && if_move(enc_rot, 0)) {
        for (uint8_t yy = 0; yy < 3; yy++) {
            a_field[x + enc_rot][y + yy] = a_field[x][y + yy];
            a_field[x][y + yy] = 0;
    }
}

```

```

    }
    upd_field();
    if (a_field[x][y + 3] == 0)
        tft.drawFastHLine(SHFT_X + x * (PUZZLE_SZ - 1) + 1, SHFT_Y + y * (PUZZLE_SZ - 1) -
x += enc_rot;
}
if (push_res == short_push) // перемена цветов фигуры
{
    uint8_t aa = a_field[x][y];
    a_field[x][y] = a_field[x][y + 2];
    a_field[x][y + 2] = a_field[x][y + 1];
    a_field[x][y + 1] = aa;
    upd_field();
}
if (push_res == long_push) {
    speed = 5; // падение
}
}
//=====
void upd_field() { //перерисовать поле
    if (params.tetris) {
        update_tetris();
    } else {
        draw_columns();
    }
}
//=====
void update_tetris() { //обновить поле тетриса
    for (uint8_t yy = 3; yy < MAX_Y; yy++)
        for (uint8_t xx = 0; xx < MAX_X; xx++) {
            draw_square(xx, yy, a_field[xx][yy]);
        }
}
//=====
void draw_columns() { //обновить поле columns
    for (uint8_t yy = 3; yy < MAX_Y; yy++)
        for (uint8_t xx = 0; xx < MAX_X; xx++) {
            if (a_field[xx][yy] != a_prev[xx][yy]) {
                draw_square(xx, yy, a_field[xx][yy]);
            }
        }
    memcpy(a_prev, a_field, sizeof(a_field));
}
//=====
bool if_move(int8_t dx, int8_t dy) { //проверка возможности движения и движение, если о

```

```

bool res = true;
if (params.tetris) {
    res = move_tetris(dx, dy);
} else {
    res = move_columns(dx, dy);
}
return res;
}
//=====

bool move_tetris(int8_t dx, int8_t dy) { //движение тетриса

    if (dx && check_tetris(tt.x + dx, tt.y, tt.shape, tt.rot)) { //проверка движения в бок
        draw_tetris(tt.x, tt.y, tt.shape, tt.rot, 0);
        tt.x += dx;
        draw_tetris(tt.x, tt.y, tt.shape, tt.rot, tt.color);
        tt.do_rotate = false; //если была попытка поворота в момент движения в бок, обнулит
    } else if (dy) { //движение вниз
        if (check_tetris(tt.x, tt.y + dy, tt.shape, tt.rot)) {
            draw_tetris(tt.x, tt.y, tt.shape, tt.rot, 0);
            tt.y += dy;
            draw_tetris(tt.x, tt.y, tt.shape, tt.rot, tt.color);
        } else { // если упали на фигуру или дно
            for (uint8_t xx = 0; xx < 4; xx++)
                for (uint8_t yy = 0; yy < 4; yy++) {
                    if (read_shape(xx, yy, tt.shape, tt.rot)) {
                        a_field[1 & tt.rot ? xx + tt.x + 1 : xx + tt.x][1 & tt.rot ? yy + tt.y : yy]
                    }
                }
            if (tt.y < 3) {
                gameover();
            } else { // если фигура остановилась, проверяем не нужно ли начислить очки
                TScore = 0;
                full_tetris(); // поиск полных линий
                tetris_score(); // подсчет очков
            }
            draw_next();
        }
    } else if (tt.do_rotate && check_tetris(tt.x, tt.y, tt.shape, tt.rot < 3 ? tt.rot + 1
        draw_tetris(tt.x, tt.y, tt.shape, tt.rot, 0);
        tt.do_rotate = 0;
        tt.rot < 3 ? ++tt.rot : tt.rot = 0;
        draw_tetris(tt.x, tt.y, tt.shape, tt.rot, tt.color);
    }
    last_activity = millis();
    return false;
}

```

```

//=====
void tetris_score() {
    switch (TScore) {
        case 0:
            break;
        case 1:
            score += 10;
            break;
        case 2:
            score += 30;
            break;
        case 3:
            score += 70;
            break;
        case 4:
            score += 150;
        default:
            score += 10;
    }
    if (score) {
        changed.score = true;
    }
    if (top_score < score) {
        top_score = score;
        changed.top_score = true;
    }
}
//=====

void full_tetris() {           // поиск и удаление полных строк
    params.allow_mv = false;   // запрет движения в паузе
    for (uint8_t yy = MAX_Y - 1; yy > 2; yy--) {
        uint8_t cnt = 0;
        for (uint8_t xx = 0; xx < MAX_X - 1; xx++) {
            if (a_field[xx][yy] && a_field[xx + 1][yy]) {
                cnt++; // считаем количество квадратов в одной линии
            }
        }
        if (cnt == MAX_X - 1) { // если количество равно длине линии
            my_delay(50);
            for (uint8_t xx = 0; xx < MAX_X; xx++) {
                a_field[xx][yy] = 1;
                draw_square(xx, yy, 1); // окрашиваем линию в черный цвет
            }
            TScore++;
        }
    }
    for (uint8_t xx = 0; xx < MAX_X; xx++) {

```

```

for (uint8_t yy = MAX_Y - 1; yy > 2; yy--) {
    if (a_field[xx][yy] == 1) {
        for (uint8_t my = yy; my > 2; my--) {
            a_field[xx][my] = a_field[xx][my - 1]; //удаляем все черные квадраты и сдвигаем
        }
        yy++; // если удалили линию и переместили содержимое, то возвращаемся на предыдущую
    }
}
my_delay(200);
upd_field();
if (TScore) {
    my_tone(60, 100);
}
params.allow_mv = true; //опять разрешаем движение
}

//=====
bool check_tetris(int8_t t_x, int8_t t_y, uint8_t shape, uint8_t rot) { //проверка доступности
    int8_t real_x, real_y;
    bool res = true;
    for (uint8_t xx = 0; xx < 4; xx++)
        for (uint8_t yy = 0; yy < 4; yy++) {
            if (read_shape(xx, yy, shape, rot)) { //если есть пиксель по упакованной
                real_x = 1 & rot ? xx + t_x + 1 : xx + t_x; //1&rot - вертикаль или горизонталь
                real_y = 1 & rot ? yy + t_y : yy + t_y + 1;
                if (real_x < 0 || real_x >= MAX_X || real_y >= MAX_Y || a_field[real_x][real_y])
                    res = false;
            }
        }
    return res;
}

//=====
void draw_tetris(int8_t t_x, int8_t t_y, uint8_t shape, uint8_t rot, uint8_t color) { //рисование
    int8_t real_x, real_y;
    for (uint8_t xx = 0; xx < 4; xx++)
        for (uint8_t yy = 0; yy < 4; yy++) {
            if (read_shape(xx, yy, shape, rot)) { //если есть пиксель по координате
                real_x = 1 & rot ? xx + t_x + 1 : xx + t_x; //получить реальную координату и нарисовать
                real_y = 1 & rot ? yy + t_y : yy + t_y + 1;
                draw_square(real_x, real_y, color);
                if (real_y + 2 > MAX_Y) {
                }
            }
        }
}

```

```

}

//=====
//          h      v      h      v
// uint8_t tetris_shapes[7][4] = {{0b00001111, 0b00001111, 0b11110000, 0b11110000}, //
bool read_shape(uint8_t tx, uint8_t ty, uint8_t shape, uint8_t rot) { //преобразование
    bool val;
    if (rot == 0 || rot == 2) { // horizontal orientation
        val = 1 & ((tetris_shapes[shape][rot]) >> (tx + 4 * ty));
    } else { // vertical orientation
        val = 1 & ((tetris_shapes[shape][rot]) >> (ty + 4 * tx));
    }
    return val;
}

//=====

bool move_columns(int8_t dx, int8_t dy) {
    bool allow = false, res = true;
    if (x + dx < 0 || x + dx > MAX_X - 1) {
        res = false; // попытка выйти за стенки стакана
    } else {
        if (dx != 0) { //движение в бок
            if (a_field[x + dx][y + dy + 2] == 0) { //если нижний кубик не задевает
                res = true;
            } else {
                res = false;
            }
        } else if (dy > 0) { // падение вниз
            if (y + dy + 2 > MAX_Y - 1 || a_field[x + dx][y + dy + 2] > 0) { // либо на дне,
                //если стакан за
                if (y < 3) {
                    gameover();
                } else {
                    allow = true;
                }
            } else { //перемещение вниз на 1 кубик
                for (uint8_t yy = 0; yy < 3; yy++) {
                    a_field[x][y + 2 - yy + dy] = a_field[x][y + 2 - yy];
                }
                a_field[x][y] = 0;
                y += dy;
            }
        }
        if (allow) {
            update(); //пересчитать поле и очки
            draw_next(); //нарисовать следующий набор кубиков
        }
        upd_field();
        last_activity = millis();
    }
}

```

```

        }
    }
    return res;
}

//=====
void draw_square(uint8_t xx, uint8_t yy, uint8_t mycolor) // отрисовка квадрата
{
    if (yy > 2) {
        uint8_t y = SHFT_Y + (yy - 3) * PUZZLE_SZ - yy;
        uint8_t x = SHFT_X + xx * (PUZZLE_SZ - 1);
        if (mycolor != 0) {
            if (!params.tetris) {
                tft.drawRect(x, y, PUZZLE_SZ, PUZZLE_SZ, ST77XX_BLACK);
            }
            tft.fillRect(x + 1, y + 1, PUZZLE_SZ - 2, PUZZLE_SZ - 2, colors[mycolor]);
        } else
            tft.fillRect(x + 1, y - 1, PUZZLE_SZ - 2, PUZZLE_SZ, ST77XX_WHITE);
    }
}

//=====
void draw_next() {
    uint8_t dx = 14; // смещение на 14 кубиков вправо
    uint8_t dy = 4; // смещение на 4 кубика вниз
    static uint8_t a_next_color[3];
    static struct {
        uint8_t shape : 3;
        uint8_t rot : 2;
    } prev_tt{ 0, 1 };
    params.tetris ? x = 5 : x = 6; //позиция, от куда будут падать кубики
    y = 0;
    if (params.tetris) {
        draw_tetris(dx - 1, dy, prev_tt.shape, 1, 0);
        tt.shape = prev_tt.shape;
        tt.rot = prev_tt.rot;
        tt.x = x;
        tt.y = y;
        prev_tt.shape = random(7);
        prev_tt.rot = random(4);
        draw_tetris(dx - 1, dy, prev_tt.shape, 1, tt.color);
    } else {
        for (uint8_t yy = 0; yy < 3; yy++) {
            if (!params.demo) {
                a_field[x][yy] = a_next_color[yy];
            }
        }
        a_next_color[yy] = random(TOTAL_COLORS) + 2;
    }
}

```

```

        draw_square(dx, dy + yy, a_next_color[yy]);
    }
}

if (!params.demo) {
    fig_count++;
    if (fig_count == NEXT_LEVEL) {
        fig_count = 0;
        cur_level < MAX_LEVEL ? cur_level++ : cur_level = MAX_LEVEL;
        changed.level = true;
    }
}
if (params.tetris) {
    if (cur_level > game_data.tetris_level) {
        game_data.tetris_level = cur_level;
        changed.max_level = true;
    }
} else {
    if (cur_level > game_data.columns_level) {
        game_data.tetris_level = cur_level;
        changed.max_level = true;
    }
}
}

speed = a_speeds[cur_level - 1];

upd_info();
}

//=====
uint8_t my_push() // возвращает длинное-2, короткое-1 или отсутствие нажатия-0
{
    uint32_t cur_time = millis();
    static uint32_t push_time = 0, depress_time = 0;
    uint8_t res = no_push;
    static bool b_long = false, b_short = false;
    //если кнопка нажата
    if (!digitalRead(PIN_SW)) {
        if (!push_time) {
            push_time = cur_time;
        }
        if (cur_time - push_time > LONG_PUSH_MS) { //длинное нажатие
            b_short = false;
            b_long = true;
            res = long_push;
            push_time = 0;
        } else if (cur_time - push_time > DEBOUNCE && !b_long) { //устранение дребезга
            b_short = true;
        }
    }
}
```

```

    }
} else {
    //если кнопка отпущена
    if (!depress_time) {
        depress_time = cur_time;
    }
    if (cur_time - depress_time > DEBOUNCE) { //устранение дребезга
        if (!b_long && b_short) {
            res = short_push;
            b_short = false;
        }
        depress_time = 0;
        b_long = false;
        push_time = 0;
    }
}
return (res);
}

//=====
bool add_node(uint8_t xx, uint8_t yy, block *full_body) { //дообавить новый узел с коор
    block *tmp = full_body;
    bool collision = false, first = false;
    do {
        if (tmp->x == xx && tmp->y == yy) {
            collision = true; //если найдено хоть одно совпадение, поднять флаг столкновения
        }
        if (tmp->next) {
            tmp = tmp->next;
        }
    } while (tmp->next);
    if (tmp->x == xx && tmp->y == yy) {
        collision = true; //если найдено хоть одно совпадение, поднять флаг столкновения
    }
    if (!collision) { //если таких элементов нет в списке
        if (tmp->y) {
            tmp->next = calloc(1, sizeof(block));
            tmp = tmp->next;
        }
        tmp->x = xx;
        tmp->y = yy;
        full_fig_count++;
    }
    return collision;
}

//=====

```

```

// чтобы избежать циклического хождения по замкнутым фигурам
// используется флаг collision
// * * * * * * *
// * * * * * * *
// * * * * *
// * * * *
// подобные фигуры вызывают кольцевую рекурсию, поэтому нужно условие для выхода

void find_full_block(uint8_t xx, uint8_t yy, block *full_body, uint8_t dir) { //множест
    bool collision = false;
    bool res = false;
    //dir - запрещает обратное рекурсивное движение в сторону из которой пришли
    if (dir != 1 && xx + 1 < MAX_X && a_field[xx + 1][yy] == a_field[xx][yy]) { //вправо
        collision = add_node(xx, yy, full_body);
        res = true;
        find_full_block(xx + 1, yy, full_body, 0);
        add_node(xx + 1, yy, full_body);
    }

    if (dir != 0 && xx > 0 && a_field[xx - 1][yy] == a_field[xx][yy]) { //влево
        collision = add_node(xx, yy, full_body);
        res = true;
        find_full_block(xx - 1, yy, full_body, 1);
        add_node(xx - 1, yy, full_body);
    }

    if (dir != 3 && yy + 1 < MAX_Y && a_field[xx][yy + 1] == a_field[xx][yy]) { //вниз
        collision = add_node(xx, yy, full_body);
        res = true;
        find_full_block(xx, yy + 1, full_body, 2);
        add_node(xx, yy + 1, full_body);
    }

    if (dir != 2 && yy > 2 && a_field[xx][yy - 1] == a_field[xx][yy]) { //вверх
        collision = add_node(xx, yy, full_body);
        res = true;
        if (!collision) { // чтобы избежать хождения по кольцу, в любом из ветвлений нужно
            find_full_block(xx, yy - 1, full_body, 3);
        }
        add_node(xx, yy - 1, full_body);
    }
}

//=====
void free_block(block *full_body) { //освобождение памяти
    if (full_body->next) {
        free_block(full_body->next);

```

```

    }
    free(full_body);
}

//=====
void paint_black(block *full_body) { //заполнение черным всех найденных схожих элементов
    block *tmp = full_body;
    while (tmp) {
        a_field[tmp->x] [tmp->y] = 1;
        tmp = tmp->next;
    }
}

//=====
bool delete_black() { //удаление черных квадратов
    bool res = false;
    for (uint8_t xx = 0; xx < MAX_X; xx++) {
        for (uint8_t yy = MAX_Y - 1; yy > 2 && a_field[xx] [yy] != 0;) { //верхние ряды за эти
            if (a_field[xx] [yy] > 1) {
                --yy;
            } else if (a_field[xx] [yy] == 1) {
                a_field[xx] [yy] = a_field[xx] [yy - 1];
                for (uint8_t t_y = yy; a_field[xx] [t_y - 1] > 0 && t_y > 1; --t_y) {
                    if (yy < 4 && a_field[xx] [t_y - 2] != 0) {
                        a_field[xx] [t_y - 1] = 0;
                    } else {
                        a_field[xx] [t_y - 1] = a_field[xx] [t_y - 2];
                    }
                }
                res = true;
            }
        }
    }
    return res;
}

//=====
//colors[] = { ST77XX_WHITE, ST77XX_BLACK, ST77XX_RED, ST77XX_BLUE, ST77XX_GREEN, ST77XX_YELLOW };
//=====

bool search() { //поиск блоков одинакового цвета
    bool fnd = false;
    for (uint8_t yy = 3; yy < MAX_Y && !params.new_game && !params.menu; yy++) //перебор
        for (uint8_t xx = 0; xx < MAX_X && !params.new_game && !params.menu; xx++) {
            if (a_field[xx] [yy] > 1) { //белый - 0, черный -1, все что больше - цвета блоков
                full_fig_count = 0;
                block *full_body;
                full_body = calloc(1, sizeof(block)); //выделение памяти под список квадратов

```

```

        find_full_block(xx, yy, full_body, 5); // поиск блока по текущим координатам
        if (full_fig_count > 3) {
            TScore += full_fig_count;
            paint_black(full_body); // если квадратов 4 и больше, красим в черный
            fnd = true;
            changed.score = true;
        }
        free_block(full_body); // освободить память списка
    }
}

return fnd;
}

//=====
bool analyze() // ищем одноцветные цепочки
{
    bool res = false;
    params.allow_mv = false; // запрет движения во время паузы
    upd_field(); // перерисовать поле
    my_delay(200);
    if (search()) { // если поиск успешен
        upd_field(); // перерисовать поле
        my_delay(400);
        res = delete_black(); // удаление черных областей
        if (params.in_game) {
            my_tone(60, 100);
        }
    }
    return res;
}

//=====
void update() {
    TScore = 0;
    while (analyze()) { // анализ поля, подсчет очков
        if (TScore > 7) {
            score += TScore + (TScore - 8) * 2;
        } else {
            score += TScore;
        }
        if (score > top_score) {
            top_score = score;
            changed.top_score = true;
        }
        upd_info();
    }
    params.allow_mv = true; // разрешаем движение блоков во время паузы
}

```

```

}

//=====
void new_game() {
    params.tetris ? game_data.tetris_games++ : game_data.columns_games++;
    changed.games = true;
    eeprom_write_block((void *)&game_data, GDATA_ADDR, sizeof(game_data)); //записываем к
    memset(a_prev, 255, sizeof(a_prev)); // заполним бу
    memset(a_field, 0, sizeof(a_field));
    changed = (my_bools){ 1, 1, 1, 1, 1 };
    if (params.tetris) {
        begin_tetris();
    } else {
        begin_columns();
    }
}
//=====
void begin_tetris() {

    score = 0;
    fig_count = 0;
    params.in_game = true;
    draw_next();
    cur_level = 1;
    speed = a_speeds[0];
    top_score = game_data.tetris_record;
    upd_field();
    upd_info();
    params.new_game = false;
}
//=====
void begin_columns() {
    for (uint8_t xx = 0; xx < MAX_X - 1; xx++) {
        tft.drawFastVLine(3 + (PUZZLE_SZ - 1) * xx, 2, 255, ST77XX_BLACK); // направляющие
    }

    score = 0;
    fig_count = 0;
    params.in_game = true;
    draw_next();

    cur_level = 1;
    speed = a_speeds[0];
    top_score = game_data.columns_record;
    upd_info();
    params.new_game = false;
}

```

```

//=====
void gameover() {
    params.demo = true;
    params.in_game = false;
    params.menu = false;

    tft.drawRect(43, 100, 90, 50, ST77XX_BLACK);
    tft.fillRect(44, 101, 88, 48, ST77XX_WHITE);
    tft.setCursor(52, 107);
    tft.setTextSize(2);
    tft.setTextColor(ST77XX_RED);
    tft.print("GAME");
    tft.setCursor(77, 127);
    tft.print("OVER");
    changed = (my_bools){ 1, 1, 1, 1, 1 };
    if (params.tetris) {
        if (score > game_data.tetris_record) {
            game_data.tetris_record = score;
        }
        if (cur_level > game_data.tetris_level) {
            game_data.tetris_level = cur_level;
        }
    } else {
        if (score > game_data.columns_record) {
            game_data.columns_record = score;
        }
        if (cur_level > game_data.columns_level) {
            game_data.columns_level = cur_level;
        }
    }
    for (uint8_t i = 0; i < 4; i++) {
        my_tone(100, 300);
        my_delay(50);
        my_tone(50, 300);
        my_delay(50);
    }
    eeprom_write_block((void *)&game_data, GDATA_ADDR, sizeof(game_data)); //сохранить ре
    memset(a_prev, 255, sizeof(a_prev));
    memset(a_field, 0, sizeof(a_field));

    my_delay(500);
    tft.fillRect(3, 3, 178, 265, ST77XX_WHITE); //стереть экран, чтобы не было полос от н
}
//=====

void my_tone(uint8_t pwm, uint16_t delay) {
    if (params.sound) {

```

```

        analogWrite(PIN_SPK, pwm); // издаём звук
        my_delay(delay);
        analogWrite(PIN_SPK, 0);
    }
}

//=====
void screensaver() {
    if (params.demo) {
        score = 0;
        draw_next();
        if (!params.tetris) { // screensaver только для Columns
            for (uint8_t yy = 3; yy < MAX_Y; yy++)
                for (uint8_t xx = 0; xx < MAX_X; xx++) {
                    a_field[xx][yy] = random(6) + 2;
                }
        }
        upd_info();
        upd_field();
        params.tetris ? my_delay(250) : my_delay(800);
        update();
        if (top_score < score) {
            top_score = score;
            changed.top_score = true;
        }
    }
}

//=====
void go_to_sleep() { /* уход в сон или выключение
                     // если отпаять светодиоды с платы и перенести питание CH341 на пр
                     // то в режиме сна потребление устройства вместе со спящим экраном
                     */
    digitalWrite(PIN_LOCK, LOW); /*Отключение питания power off +/
}

/* Если не выключились, то засыпаем */
digitalWrite(TFT_BL, LOW);
tft.sendCommand(ST77XX_SLPIN);

set_sleep_mode(SLEEP_MODE_PWR_DOWN);
attachInterrupt(1, wakeup, LOW);
sleep_enable();
sleep_cpu();
sleep_disable();
detachInterrupt(1);

```

```

tft.sendCommand(ST77XX_SLPOUT);
digitalWrite(TFT_BL, HIGH);
if (params.menu) {
    params.menu_update = true;
} else {
    params.demo = true;
}

last_activity = millis();
}

//=====
void init_tft() {
    tft.init(240, 320); // initialize a ST7739S chip, black tab
    digitalWrite(TFT_BL, HIGH);
    tft.fillScreen(ST77XX_WHITE);

    tft.setTextColor(ST77XX_BLACK);
    tft.setTextWrap(true);
    tft.setTextSize(1);

    tft.setCursor(L_INFO, SHFT_Y);
    tft.print("NEXT");

    tft.setCursor(L_INFO, Y_INFO);
    tft.print("LEVEL");

    tft.setCursor(L_INFO, Y_INFO + 35);
    tft.print("SCORE");

    tft.setCursor(L_INFO, Y_INFO + 70);
    tft.print("TOP");

    tft.setCursor(L_INFO, Y_INFO + 80);
    tft.print("SCORE");

    tft.setCursor(L_INFO, Y_INFO + 105);
    tft.print("MAX");
    tft.setCursor(L_INFO, Y_INFO + 115);
    tft.print("LEVEL");

    tft.setCursor(L_INFO, Y_INFO + 145);
    tft.print("TOTAL");
    tft.setCursor(L_INFO, Y_INFO + 155);
    tft.print("GAMES");
}

```

```

tft.drawFastVLine(1, 2, B_FIELD - 1, ST77XX_BLACK);
tft.drawFastVLine(PUZZLE_SZ * MAX_X + 5 - MAX_X, 2, B_FIELD - 1, ST77XX_BLACK);
tft.drawFastHLine(1, B_FIELD, PUZZLE_SZ * MAX_X + 5 - MAX_X, ST77XX_BLACK);

/// Battery Frame
tft.setCursor(45, 290);
tft.print("===== BATTERY LEVEL =====");
tft.drawFastHLine(3, 300, 234, ST77XX_BLACK);
tft.drawFastHLine(3, 315, 234, ST77XX_BLACK);
tft.drawFastVLine(3, 300, 15, ST77XX_BLACK);
tft.drawFastVLine(236, 300, 15, ST77XX_BLACK);
}

//=====
void my_print_num(uint32_t num, uint8_t digits) { // Рекурсивный вывод, чтобы без исполь
                                                 // Библиотека для работы со строками
    if (--digits > 0) {
        my_print_num(num / 10, digits);
        num %= 10;
    }
    tft.print(num);
}

//=====
void show_battery_level() {
    int b_level = analogRead(A3), diff;
    diff = prev_batt - b_level;
    if (diff > 5 || diff < -5) {
        prev_batt = b_level;
        if (b_level > 444) { //5.1v значения измерены мультиметром, записана таблица соотве
            draw_battery_bars(19, ST77XX_CYAN);
        } else if (b_level > 434) { //5v
            draw_battery_bars(19, ST77XX_GREEN);
        } else if (b_level > 386) { //4.5v
            draw_battery_bars(18, ST77XX_GREEN);
        } else if (b_level > 377) { //4.4v
            draw_battery_bars(17, ST77XX_GREEN);
        } else if (b_level > 367) { //4.3v
            draw_battery_bars(16, ST77XX_GREEN);
        } else if (b_level > 359) { //4.2v
            draw_battery_bars(15, ST77XX_GREEN);
        } else if (b_level > 348) { //4.1v
            draw_battery_bars(14, ST77XX_GREEN);
        } else if (b_level > 340) { //4.0v
            draw_battery_bars(13, ST77XX_YELLOW);
        } else if (b_level > 329) { //3.9v
            draw_battery_bars(12, ST77XX_YELLOW);
        }
    }
}

```

```

} else if (b_level > 322) { //3.8v
    draw_battery_bars(11, ST77XX_YELLOW);
} else if (b_level > 312) { //3.7v
    draw_battery_bars(10, ST77XX_YELLOW);
} else if (b_level > 304) { //3.6v
    draw_battery_bars(9, ST77XX_YELLOW);
} else if (b_level > 295) { //3.5v
    draw_battery_bars(8, ST77XX_ORANGE);
} else if (b_level > 285) { //3.4v
    draw_battery_bars(7, ST77XX_ORANGE);
} else if (b_level > 278) { //3.3v
    draw_battery_bars(6, ST77XX_ORANGE);
} else if (b_level > 268) { //3.2v
    draw_battery_bars(5, ST77XX_ORANGE);
} else if (b_level > 261) { //3.1v
    draw_battery_bars(4, ST77XX_RED);
} else if (b_level > 251) { //3.0v
    draw_battery_bars(3, ST77XX_RED);
} else if (b_level > 244) { //2.90v
    draw_battery_bars(2, ST77XX_RED);
} else if (b_level > 240) { //2.85v
    draw_battery_bars(1, ST77XX_RED);
} else if (b_level > 235) { //2.8v
    draw_battery_bars(0, ST77XX_RED);
} else {
    draw_battery_bars(19, ST77XX_RED); //зажечь все красным, когда полный разряд
}
}

//=====
void draw_battery_bars(uint8_t num, uint16_t color) { //нарисовать шкалу батареи
    tft.fillRect(7, 302, 228, 12, ST77XX_WHITE);
    for (uint8_t i = 0; i < num; i++) {
        tft.fillRect(7 + 12 * i, 302, 10, 12, color);
    }
}

//=====
void eeprom_enable() {
    if (EEPROM_KEY == eeprom_read_byte(KEY_ADDR)) { //если ключ совпадает, считать значение
        eeprom_read_block((void *)&game_data, GDATA_ADDR, sizeof(game_data));
        eeprom_read_block((void *)&params, PARAM_ADDR, sizeof(params));
        tt.color = params.t_color;
    } else { // если ключ не совпадает, восстановить параметры по умолчанию
        game_data.columns_record = 0;
        game_data.columns_level = 1;
    }
}

```

```

game_data.columns_games = 0;
game_data.tetris_record = 0;
game_data.tetris_level = 1;
game_data.tetris_games = 0;
params.demo = 1;
params.sound = 1;
params.t_color = 4;      //зеленый цвет
params.screen_time = 5;  // 1-15 screen_time * 5 = время скринсейвера в секундах
eeprom_write_block((void *)&game_data, GDATA_ADDR, sizeof(game_data));
eeprom_write_block((void *)&params, PARAM_ADDR, sizeof(params));
eeprom_write_byte(KEY_ADDR, EEPROM_KEY); //запись ключа
}
}

//=====
void wakeup() {
}

//=====
// обработчик прерывания
void timerInterrupt() {
    encoder.scanState();
}

```

7. Дополнительные материалы

Демонстрация работы устройства расположена в папке на google диск по следующей ссылке:
Материалы к курсовому проекту