1. Device description

Tetris game on an STM32 board, displayed on a built-in LCD screen, controlled by UART and using a physical controller connected with a simple RJ12 cable.

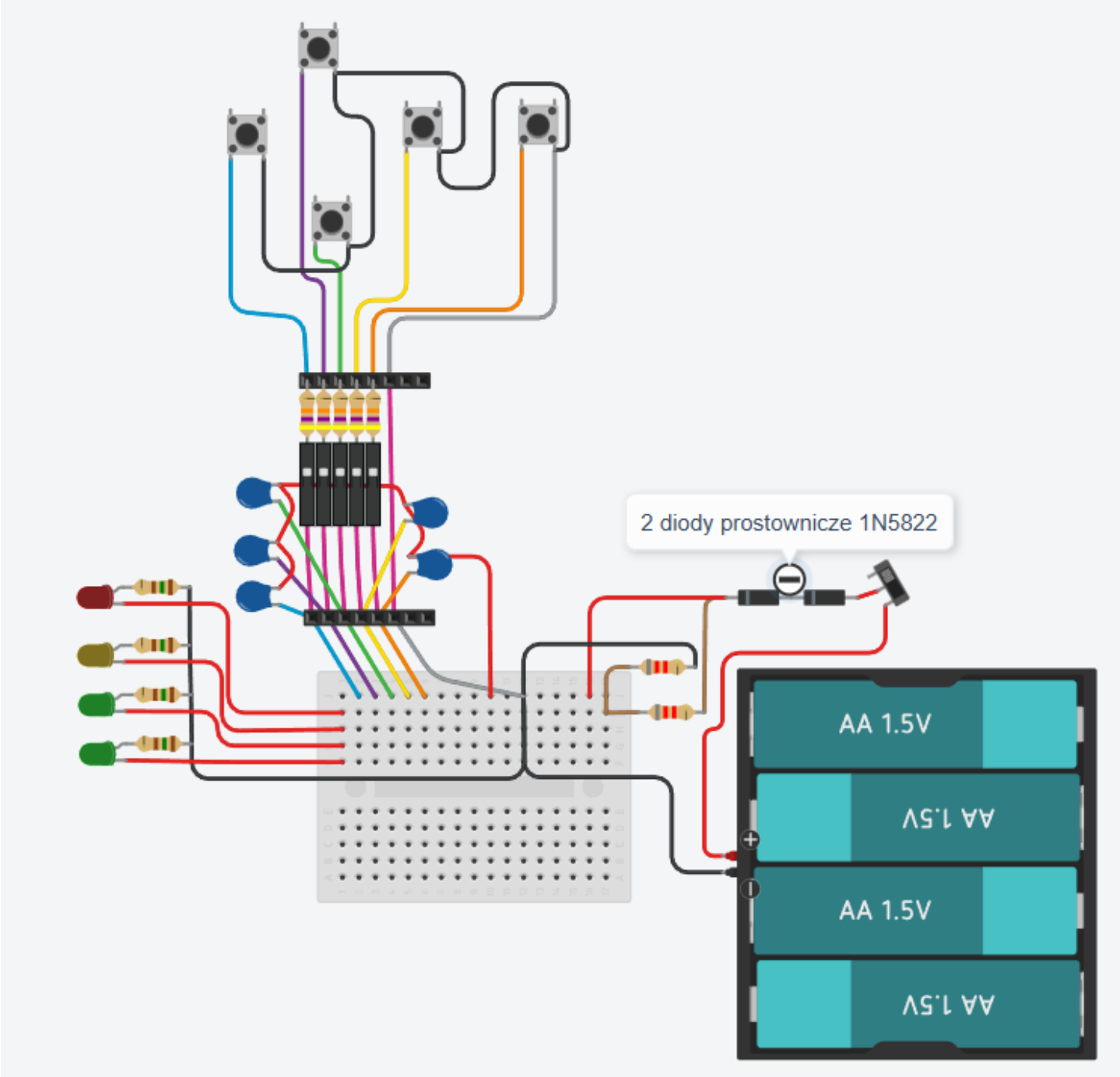
2. Components selected for device construction

STM32F429I-DISK1 board,
Monostable buttons,
Keystone Jack RJ45 sockets,
RJ12 cable,
4 AA battery holder,
4 AA batteries,
Slide switch,
LEDs, various colors,
1N5822 rectifier diodes,
8.2kOhm resistors,
150 Ohm resistors,
47 kOhm resistors,
100 nF ceramic capacitors,
Copper connecting cables,
Console housing components manufactured using 3D printers,
Controller housing components manufactured using 3D printers.

3. Device mechanics

The device runs a Tetris game based on the offpic/TETRIS-STM32 and KenKenMkIISR/picotetris projects, using the afiskon/stm32-ili9341 library for the ILI9341 display controller. The game code has undergone thorough changes and improvements, the most significant of which is the departure from displaying characters and character tables for individual pixels, which significantly affects every element of the game.
A controller with 5 buttons (Start, Up, Down, Left, Right) is connected to the console using an RJ12 cable. All necessary for the game, it is also possible to use the serial port by sending commands to the UART port using the E, W, S, A, D keys.
The console is powered by 4 AA batteries connected in series after voltage adjustment by 2 rectifier diodes. The supply voltage is monitored and the battery status is displayed on the LEDs each time the console is turned on.

4. Electronic diagram of the device



A breadboard is used as a replacement for the STM32 board.

5. Control software

```
void updateLEDs(void) {
  uint32_t sum = 0;
  for (int i = 0; i < 10; i++) {
    HAL_ADC_Start(&hadc3);
    HAL_ADC_PollForConversion(&hadc3, HAL_MAX_DELAY);
    sum += HAL_ADC_GetValue(&hadc3);
    HAL_Delay(1);
  }
  uint32_t adcValue = sum / 10;

  float battV = adcValue * 0.00161;         // V_in = ADC * (3.3/4095) * (R1+R2)/R2

  if (battV > 5.4) {                        // 100% // >3350 // PG3 PG2 PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 1);
  } else if (battV > 4.9) {                 // 75% // 3043-3350 // PG2 PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else if (battV > 4.4) {                 // 50% // 2730-3043 // PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else if (battV > 3.9) {                 // 25% // 2418-2730 // PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else {
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  }

  HAL_Delay(4000);

  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
  HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
  HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
}
```

Code fragment responsible for monitoring and displaying the battery status

```
void clearscreen(void)
{
    ILI9341_FillScreen(COLOR_BG);
}
void gameinit(void)
{
    highscore = 0;
    score = 0;
    total_lines = 0;
}
void gameinit2(void)
{
    score = 0;
    level = 0;
    lines = 0;
    total_lines = 0;
    fallspeed = 25;
    gamestatus = 1;
    clearscreen();
        //GAME AREA
    uint16_t board_frame_x = BOARD_X_OFFSET - 1;
    uint16_t board_frame_y = BOARD_Y_OFFSET - 1;
    uint16_t board_frame_w = BOARD_WIDTH * BLOCK_SIZE + 2; // Width including border pixels
    uint16_t board_frame_h = BOARD_HEIGHT * BLOCK_SIZE + 2; // Height including border pixels
        // Top line
    ILI9341_FillRectangle(board_frame_x, board_frame_y, board_frame_w, 1, COLOR_FRAME);
        // Bottom line
    ILI9341_FillRectangle(board_frame_x, board_frame_y + board_frame_h - 1, board_frame_w, 1, COLOR_FRAME);
        // Left line
    ILI9341_FillRectangle(board_frame_x, board_frame_y, 1, board_frame_h, COLOR_FRAME);
        // Right line
    ILI9341_FillRectangle(board_frame_x + board_frame_w - 1, board_frame_y, 1, board_frame_h, COLOR_FRAME);


        //SCORE AREA
    uint16_t score_frame_x = SCORE_X_OFFSET - 1;
    uint16_t score_frame_y = SCORE_Y_OFFSET - 1;
    uint16_t score_frame_w = 80;
    uint16_t score_frame_h = 5 * (Font_7x10.height + 2) + 4;
        // Top
    ILI9341_FillRectangle(score_frame_x, score_frame_y, score_frame_w, 1, COLOR_FRAME);
        // Bottom
    ILI9341_FillRectangle(score_frame_x, score_frame_y + score_frame_h - 1, score_frame_w, 1, COLOR_FRAME);
        // Left
    ILI9341_FillRectangle(score_frame_x, score_frame_y, 1, score_frame_h, COLOR_FRAME);
        // Right
    ILI9341_FillRectangle(score_frame_x + score_frame_w - 1, score_frame_y, 1, score_frame_h, COLOR_FRAME);
```

Code fragment responsible for the lowest levels of game logic, creating the game board, resetting and setting the most important variables.

```
void moveblock(void)
{
    _Block tempblock;
    const _Block *blockp;
        // Player Input
        // Rotate
    if (uart_up)
    {
        uart_up = 0;
        eraseblock();

        uint8_t next_angle = (blockangle + 1) % 4;
        if (falling.rot == 1 && next_angle > 1) next_angle = 0;
        if (falling.rot == 0) next_angle = 0;

        if (blockangle == falling.rot)
        {
            blockp = &block[blockno];
            tempblock = *blockp;
        } else {
            tempblock.x1 = -falling.y1; tempblock.y1 = falling.x1;
            tempblock.x2 = -falling.y2; tempblock.y2 = falling.x2;
            tempblock.x3 = -falling.y3; tempblock.y3 = falling.x3;
            tempblock.color = falling.color;
            tempblock.rot = falling.rot;
        }
            // Check rotate
        if (check(&tempblock, blockx, blocky) == 0)
        {
            falling = tempblock;
            blockangle = (blockangle == falling.rot) ? 0 : blockangle + 1;
        }
        putblock();
    }
        // Move Right
    else if (uart_right) {
        uart_right = 0;
        eraseblock();
        if (check(&falling, blockx + 1, blocky) == 0)
        {
            blockx++;
        }
        putblock();
    }
        // Move Left
```

Code fragment responsible for moving blocks on the game board

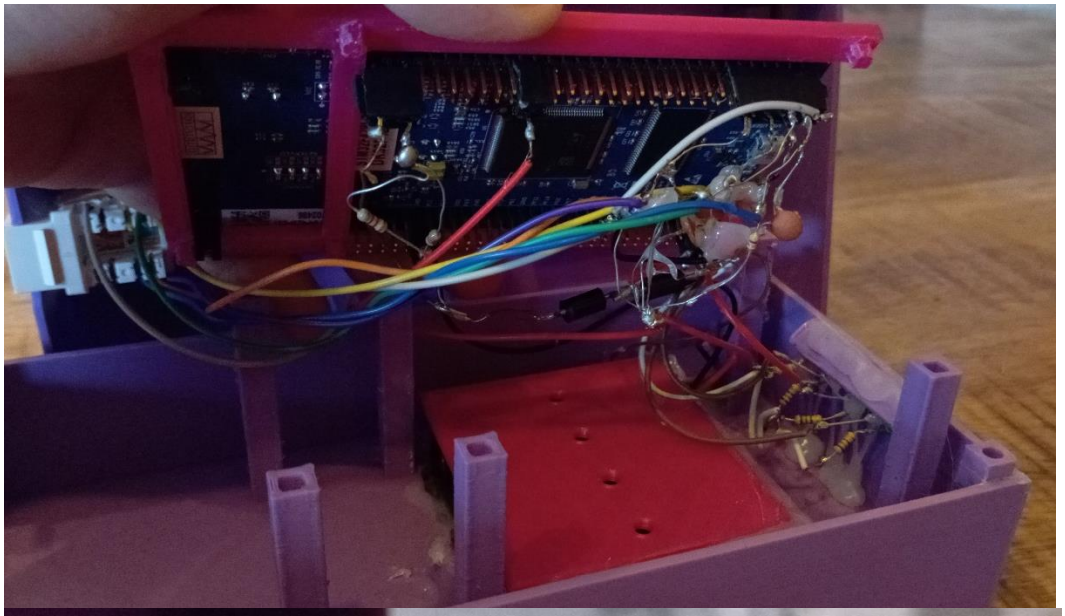
```
}
void game(void)
{
    gameinit2();

    while (gamestatus != 6)
    {
        switch (gamestatus)
        {
        case 1: // Initialize New Level
            gameinit3();
            displaylevel();
            gamestatus = 2;
            break;

        case 2: // Spawn New Block
            if (newblock() != 0)
            {
                gamestatus = 5;
            } else {
                gamestatus = 3;
            }
            break;

        case 3: // Block Falling, Input Check
            // Game tick delay happens here
            HAL_Delay(16); // ~60Hz loop speed

            show();          // Redraw changed parts of the board
            displayscore(); // Update score display
            eraseblock();    // Erase block at current position
            moveblock();     // Check input, apply natural fall, check landing
            putblock();      // Draw block at new position

            break;

        case 4: // Block Landed, Check for Line Clears
            show();
            linecheck(); // Check lines, clear them, update score/lines
            break;
        case 5: // Game Over
            gameover();
            gamestatus = 6;
            break;
        default:
            gamestatus = 6;
            break;
        }
```

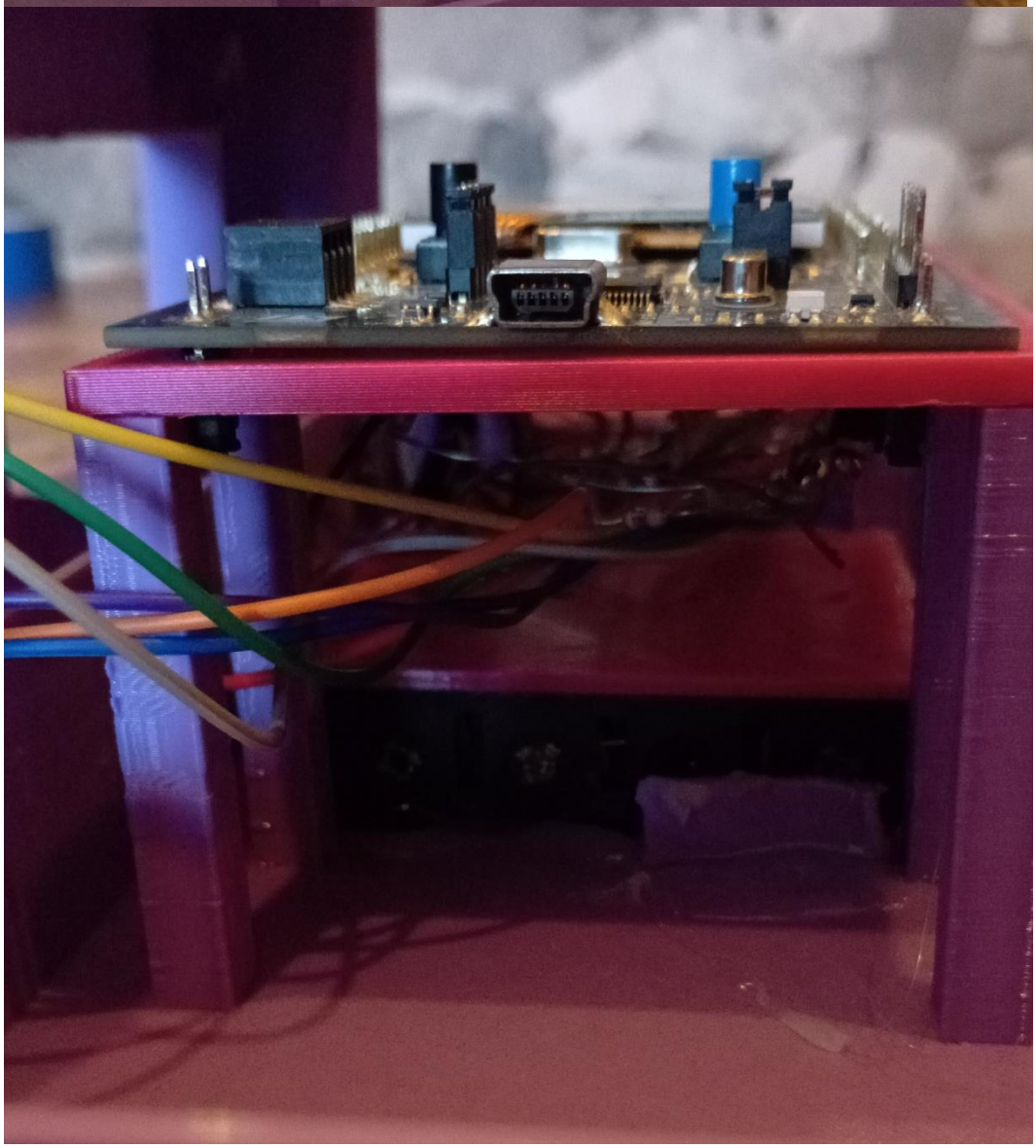
Code fragment responsible for the main game loop, where all the most important functions are called using the gamestatus variable.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    static uint32_t last_interrupt_time_up = 0;
    static uint32_t last_interrupt_time_left = 0;
    static uint32_t last_interrupt_time_right = 0;
    static uint32_t last_interrupt_time_down = 0;
    static uint32_t last_interrupt_time_start = 0;
    uint32_t current_time = HAL_GetTick();
    const uint32_t debounce_delay = 50;

    switch (GPIO_Pin)
    {
        case BTN_UP_Pin: // PC9
            if ((current_time - last_interrupt_time_up) > debounce_delay)
            {
                last_interrupt_time_up = current_time;
                uart_up = 1;
            }
            break;
        case BTN_LEFT_Pin: // PG5
            if ((current_time - last_interrupt_time_left) > debounce_delay)
            {
                last_interrupt_time_left = current_time;
                uart_left = 1;
            }
            break;
        case BTN_RIGHT_Pin: // PG6
            if ((current_time - last_interrupt_time_right) > debounce_delay)
            {
                last_interrupt_time_right = current_time;
                uart_right = 1;
            }
            break;
        case BTN_DOWN_Pin: // PG7
            if ((current_time - last_interrupt_time_down) > debounce_delay)
            {
                last_interrupt_time_down = current_time;
                uart_down = 1;
            }
            break;
        case BTN_START_Pin: // PG8
            if ((current_time - last_interrupt_time_start) > debounce_delay)
            {
                last_interrupt_time_start = current_time;

                if (gamestatus == 3 || gamestatus == 2) {
                    uart_up = 1;
                } else {
                    uart_start = 1;
                }
            }
            break;
        default:
```

Code fragment responsible for handling physical keys, including using the start key as a rotation when the game is running.

6. Photos of the developed device






TETRIS

The console housing was created in Autodesk Fusion around the board model
https://grabcad.com/library/stm32f429discovery-1
The model https://www.l-com.com/ethernet-category-3-keystone-jack-110-rj12-eia568-usoc-white was used to model the indentations on the Keystone module.