```
// ******************************************************************************** mode0.h
// ****************************************************************************************
// ARNE-16 palette converted to RGB565 -- https://lospec.com/palette-list/arne-16
typedef enum {
    MODE0_BLACK,
    MODE0_BROWN,
    MODE0_RED,
    MODE0_BLUSH,
    MODE0_GRAY,
    MODE0_DESERT,
    MODE0_ORANGE,
    MODE0_YELLOW,
    MODE0_WHITE,
    MODE0_MIDNIGHT,
    MODE0_DARK_SLATE_GRAY,
    MODE0_GREEN,
    MODE0_YELLOW_GREEN,
    MODE0_BLUE,
    MODE0_PICTON_BLUE,
    MODE0_PALE_BLUE
} mode0_color_t;

void mode0_init();
void mode0_clear(mode0_color_t color);
void mode0_draw_screen();
void mode0_draw_region(uint8_t x, uint8_t y, uint8_t width, uint8_t height);
void mode0_scroll_vertical(int8_t amount);
void mode0_set_foreground(mode0_color_t color);
void mode0_set_background(mode0_color_t color);
void mode0_set_cursor(uint8_t x, uint8_t y);
uint8_t mode0_get_cursor_x();
uint8_t mode0_get_cursor_y();
void mode0_print(const char *s);
void mode0_write(const char *s, int len);
void mode0_putc(char c);
void mode0_show_cursor();
void mode0_hide_cursor();

// Won't redraw until the matching _end is invoked.
void mode0_begin();
void mode0_end();


// ******************************************************************************** mode0.cpp
/* Character graphics mode */

// Characters are 8x12 -- characters start at (x:1,y:1) and are 5x7 in size, so
// it is possible to not display the full area. This display mode actually treats
// them as 6x10, starting at (x:1,y:0)
static const uint8_t font_data[95][12] = {
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
        bla bla bla
    { 0x00, 0x28, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
};


#define TEXT_HEIGHT 24
#define TEXT_WIDTH 53

// #define SWAP_BYTES(color) ((uint16_t)(color>>8) | (uint16_t)(color<<8))

static mode0_color_t screen_bg_color = MODE0_BLACK;
static mode0_color_t screen_fg_color = MODE0_WHITE;  // TODO need to store a color per cell
static int cursor_x = 0;
static int cursor_y = 0;
static uint8_t screen[TEXT_HEIGHT * TEXT_WIDTH] = { 0 };
static uint8_t colors[TEXT_HEIGHT * TEXT_WIDTH] = { 0 };
static uint8_t show_cursor = 0;

static int depth = 0;
static uint16_t palette[16] = {
    SWAP_BYTES(0x0000),
    SWAP_BYTES(0x49E5),
    SWAP_BYTES(0xB926),
    SWAP_BYTES(0xE371),
    SWAP_BYTES(0x9CF3),
    SWAP_BYTES(0xA324),
    SWAP_BYTES(0xEC46),
    SWAP_BYTES(0xF70D),
    SWAP_BYTES(0xffff),
    SWAP_BYTES(0x1926),
    SWAP_BYTES(0x2A49),
    SWAP_BYTES(0x4443),
    SWAP_BYTES(0xA664),
    SWAP_BYTES(0x02B0),
    SWAP_BYTES(0x351E),
    SWAP_BYTES(0xB6FD)
};

void mode0_clear(mode0_color_t color) {
    mode0_begin();
    int size = TEXT_WIDTH*TEXT_HEIGHT;
    memset(screen, 0, size);
    memset(colors, color, size);
    mode0_set_cursor(0, 0);
    mode0_end();
}

void mode0_set_foreground(mode0_color_t color) {
    mode0_begin();
    screen_fg_color = color;
    mode0_end();
}

void mode0_set_background(mode0_color_t color) {
    mode0_begin();
    screen_bg_color = color;
    mode0_end();
}

void mode0_set_cursor(uint8_t x, uint8_t y) {
    cursor_x = x;
    cursor_y = y;
}

void mode0_show_cursor() {
    mode0_begin();
    show_cursor = 1;
    mode0_end();
}

void mode0_hide_cursor() {
    mode0_begin();
    show_cursor = 0;
    mode0_end();
}

uint8_t mode0_get_cursor_x() {
    return cursor_x;
}

uint8_t mode0_get_cursor_y() {
    return cursor_y;
}

void mode0_putc(char c) {
    mode0_begin();

    if (cursor_y >= TEXT_HEIGHT) {
        mode0_scroll_vertical(cursor_y-TEXT_HEIGHT+1);
        cursor_y = TEXT_HEIGHT-1;
    }

    int idx = cursor_y*TEXT_WIDTH + cursor_x;
    if (c == '\n') {
        // fill the rest of the line with empty content + the current bg color
        memset(screen+idx, 0, TEXT_WIDTH-cursor_x);
        memset(colors+idx, screen_bg_color, TEXT_WIDTH-cursor_x);
        cursor_y++;
        cursor_x = 0;
    } else if (c == '\r') {
        //cursor_x = 0;
    } else if (c>=32 && c<=127) {
        screen[idx] = c-32;
        colors[idx] = ((screen_fg_color & 0xf) << 4) | (screen_bg_color & 0xf);

        cursor_x++;
        if (cursor_x >= TEXT_WIDTH) {
            cursor_x = 0;
            cursor_y++;
        }
    }

    mode0_end();
}

void mode0_print(const char *str) {
    mode0_begin();
    char c;
    while (c = *str++) {
        mode0_putc(c);
    }
    mode0_end();
}

void mode0_write(const char *str, int len) {
    mode0_begin();
    for (int i=0; i<len; i++) {
        mode0_putc(*str++);
    }
    mode0_end();
}

inline void mode0_begin() {
    depth++;
}

inline void mode0_end() {
    if (--depth == 0) {
        mode0_draw_screen();
    }
}

void mode0_draw_region(uint8_t x, uint8_t y, uint8_t width, uint8_t height) {
    // TODO
    mode0_draw_screen();
}

void mode0_draw_screen() {
    // assert depth == 0?
    depth = 0;
    // setup to draw the whole screen

    // column address set
    ili.set_command(ILI9341_CASET);
    ili.command_param(0x00);
    ili.command_param(0x00);  // start column
    ili.command_param(0x00);
    ili.command_param(0xef);  // end column -> 239

    // page address set
    ili.set_command(ILI9341_PASET);
    ili.command_param(0x00);
    ili.command_param(0x00);  // start page
    ili.command_param(0x01);
    ili.command_param(0x3f);  // end page -> 319

    // start writing
    ili.set_command(ILI9341_RAMWR);

    uint16_t buffer[6*240];  // 'amount' pixels wide, 240 pixels tall

    int screen_idx = 0;
    for (int x=0; x<TEXT_WIDTH; x++) {
        // create one column of screen information

        uint16_t *buffer_idx = buffer;

        for (int bit=0; bit<6; bit++) {
            uint8_t mask = 64>>bit;
            for (int y=TEXT_HEIGHT-1; y>=0; y--) {
                uint8_t character = screen[y*53+x];
                uint16_t fg_color = palette[colors[y*53+x] >> 4];
                uint16_t bg_color = palette[colors[y*53+x] & 0xf];

                if (show_cursor && (cursor_x == x) && (cursor_y == y)) {
                    bg_color = MODE0_GREEN;
                }

                const uint8_t* pixel_data = font_data[character];

                // draw the character into the buffer
                for (int j=10; j>=1; j--) {
                    *buffer_idx++ = (pixel_data[j] & mask) ? fg_color : bg_color;
                }
            }
        }

        // now send the slice
        ili.write_data(buffer, 6*240*2);
    }

    uint16_t extra_buffer[2*240] = { 0 };
    ili.write_data(extra_buffer, 2*240*2);

}

void mode0_scroll_vertical(int8_t amount) {
    mode0_begin();


    if (amount > 0) {
        int size1 = TEXT_WIDTH*amount;
        int size2 = TEXT_WIDTH*TEXT_HEIGHT - size1;

        memmove(screen, screen+size1, size2);
        memmove(colors, colors+size1, size2);
        memset(screen+size2, 0, size1);
        memset(colors+size2, screen_bg_color, size1);
    } else if (amount < 0) {
        amount = -amount;
        int size1 = TEXT_WIDTH*amount;
        int size2 = TEXT_WIDTH*TEXT_HEIGHT - size1;

        memmove(screen+size1, screen, size2);
        memmove(colors+size1, colors, size2);
        memset(screen, 0, size1);
        memset(colors, screen_bg_color, size1);
    }

    mode0_end();
}

void mode0_init() {
    stdio_init_all();

    ili.init();
}


// ***********************************************************************************************
// ****************************************************************************** mode0_demo.cpp
// ***********************************************************************************************
int main() {
    mode0_init();

    mode0_set_cursor(0, 0);
    mode0_color_t fg = MODE0_WHITE;
    mode0_color_t bg = MODE0_BLACK;

    while (1) {
        mode0_print("Shawn Hyam (Larry was here 3)\n");
        sleep_ms(500);
        fg = (mode0_color_t)((fg+1) % 16);
        if (fg == 0) {
            bg = (mode0_color_t)((bg+1) % 16);
            mode0_set_background(bg);
        }
        mode0_set_foreground(fg);

    }
}
```

```
// ************************************************************************************ mode1.h
#define MAP_WIDTH 416
#define MAP_HEIGHT 32
// the Pico and ILI9341 are different endianness so we need to byte swap our
// 16-bit color values
#define SWAP_BYTES(color) ((uint16_t)(color>>8) | (uint16_t)(color<<8))


/* Tiled graphics mode */

typedef struct {
    uint8_t mem[24];
} Tile;

typedef struct {
    uint8_t mem[24];
} Sprite;

Tile tiles[256] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        bla bla bla
    0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

void mode1_init();

Tile* tile_at(int x, int y);
void set_tile_at(uint8_t tile_id, uint8_t palette, int x, int y);
uint8_t get_tile_palette_at(uint8_t x, uint8_t y);
void draw_slice(int x, int width);
void draw_background();
void scroll_background(int amount);

// *********************************************************************************** mode1.cpp
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240


Sprite sprites[256];

uint8_t background[MAP_HEIGHT * MAP_WIDTH] = { 0 };
uint8_t bg_palette[MAP_HEIGHT * MAP_WIDTH] = { 0 };
uint16_t scroll_offset;  // this is in 'map' pixels, not 'screen' pixels
uint8_t height_offset = 0;  // (maybe?) in map blocks (8 pixels)
uint16_t *palette[8];

uint16_t global_background = SWAP_BYTES(0x843E);

void mode1_init() {
    ili.init();
}

/* SPRITES */

void draw_sprite(Sprite *sprite, uint16_t x, uint16_t y) {
    // TODO
}

void erase_sprite(Sprite *sprite, uint16_t x, uint16_t y) {
    // TODO
}


/* BACKGROUND */

Tile* tile_at(int x, int y) {
    uint8_t tile_id = background[y*MAP_WIDTH + x];
    return &tiles[tile_id];
}

void set_tile_at(uint8_t tile_id, uint8_t palette, int x, int y) {
    background[y*MAP_WIDTH + x] = tile_id;
    bg_palette[y*MAP_WIDTH + x] = palette;
}

uint8_t get_tile_palette_at(uint8_t x, uint8_t y) {
    return bg_palette[y*MAP_WIDTH + x];
}

void draw_background() {
    draw_slice(scroll_offset, SCREEN_WIDTH);
}

void scroll_background(int amount) {

    scroll_offset += amount;
    uint16_t screen_offset = scroll_offset % SCREEN_WIDTH;

    ili.set_command(ILI9341_VSCRSADD);
    ili.command_param((uint8_t)(screen_offset / 256));
    ili.command_param((uint8_t)(screen_offset % 256));

    draw_slice(SCREEN_WIDTH + scroll_offset - amount, amount);
}


void draw_slice(int x, int width) {
    uint16_t buffer[width*SCREEN_HEIGHT];  // 'amount' pixels wide, 240 pixels tall
    int buffer_idx = 0;

    for (int h=0; h<width; h++) {
        int map_x = (x + h) % (MAP_WIDTH*8);
        int tile_x = map_x / 8;
        int tile_x_offset = map_x % 8;

        for (int tile_y=29; tile_y>=0; tile_y--) {
            Tile *tile = tile_at(tile_x, (tile_y+height_offset)%MAP_HEIGHT);
            //uint32_t foo =
            // (tile->mem[0][tile_x_offset] << 16) | (tile->mem[1][tile_x_offset] << 8) | | tile->mem[2][tile_x_offset]

            // the tile data is stored NES style, so we have to do some bit twiddling

            uint8_t tile_palette_idx = get_tile_palette_at(tile_x, (tile_y+height_offset)%MAP_HEIGHT);
            uint16_t *tile_palette = palette[tile_palette_idx];

            for (int i=0; i<8; i++) {
                uint8_t palette_index = 0;
                for (int bit=2; bit>=0; bit--) {
                    palette_index <<= 1;
                    palette_index |= ((tile->mem[bit*8+(7-i)] >> (7-tile_x_offset)) & 1);
                }
                // look up the color from the palette
                // color 0 is "transparent", will show global background color instead
                uint16_t color = palette_index ? tile_palette[palette_index] : global_background;

                // this color is pre-byteswapped and blue-red swapped
                buffer[buffer_idx++] = color;

            }
        }
    }

    // set the address to write to
    // page address set

    // TODO to write a full screen when the scroll_offset is set, we actually
    // need to write starting at page 0 -- in other words, according to the LCD's
    // memory layout, not starting at the VSCRADD scroll value
    uint16_t write_start = (scroll_offset - width) % SCREEN_WIDTH;
    uint16_t write_stop = (write_start + width - 1) % SCREEN_WIDTH;
    ili.set_command(ILI9341_PASET);
    ili.command_param(write_start / 256);
    ili.command_param(write_start % 256);  // start page
    ili.command_param(write_stop / 256);
    ili.command_param(write_stop % 256);  // end page -> 319

    // write out this data
    ili.set_command(ILI9341_RAMWR);
    ili.write_data(buffer, width*SCREEN_HEIGHT*2);
}
```

mode1_demo.cpp

```
// ***************************************************************************************
uint16_t palette0[8] = {  // .db $0f, $29, $1a, $0f
    0x0000,
    SWAP_BYTES(0x7600),  //CC23
    SWAP_BYTES(0x03C0),  // 52C0
    SWAP_BYTES(0x0000),
    0x0000,
    0x0000,
    0x0000,
    0x0000
};

/*
 $0CDB 0F 29 1A 0F - Pipes
 $0CDF 0F 36 17 0F - Blocks
 $0CE3 0F 30 21 0F - Sky
 $0CE7 0F 27 17 0F - ?block (The actual pallete for color 1 is elsewhere)
 */

// ground
static uint16_t palette1[8] = {  // .db $0f, $36, $17, $0f
    0x0000,
    SWAP_BYTES(0xE595),
    SWAP_BYTES(0x71C0),
    SWAP_BYTES(0x0000),
    0x0000,
    0x0000,
    0x0000,
    0x0000
};

uint16_t palette2[8] = {  // .db $0f, $30, $21, $0f
    0x0000,
    SWAP_BYTES(0xFFFF),
    SWAP_BYTES(0x4CDC),
    SWAP_BYTES(0x0000),
    0x0000,
    0x0000,
    0x0000,
    0x0000
};

uint16_t palette3[8] = {  // .db $0f, $27, $17, $0f
    0x0000,
    SWAP_BYTES(0xCC23),
    SWAP_BYTES(0x71C0),
    SWAP_BYTES(0x0000),
    0x0000,
    0x0000,
    0x0000,
    0x0000
```

```
};

/*
 .db $1e, $c2, $00, $6b, $06, $8b, $86, $63, $b7, $0f, $05
 .db $03, $06, $23, $06, $4b, $b7, $bb, $00, $5b, $b7
 .db $fb, $37, $3b, $b7, $0f, $0b, $1b, $37
 .db $ff
 */
uint8_t level_data[29] = {
    0x1e, 0xc2, 0x00, 0x6b, 0x8b, 0x86, 0x63, 0xb7, 0x0f, 0x05,
    0x03, 0x06, 0x23, 0x06, 0x4b, 0xb7, 0xbb, 0x00, 0x5b, 0xb7,
    0xfb, 0x37, 0x3b, 0xb7, 0x0f, 0x0b, 0x1b, 0x37,
    0xff
};


void place_metatile_at(int x, int y, uint8_t palette, const uint8_t tiles[4]) {
    set_tile_at(tiles[0], palette, x*2, y*2);
    set_tile_at(tiles[2], palette, x*2+1, y*2);
    set_tile_at(tiles[1], palette, x*2, y*2+1);
    set_tile_at(tiles[3], palette, x*2+1, y*2+1);

}

void place_bush_at(int x, int y, int width) {
    //  .db $24, $24, $24, $35 ;bush left
    //  .db $36, $25, $37, $25 ;bush middle
    //  .db $24, $38, $24, $24 ;bush right
    place_metatile_at(x, y, 0, (const uint8_t[4]){0x24, 0x24, 0x24, 0x35});
    for (int i=0; i<width; i++) {
        place_metatile_at(x+1+i, y, 0, (const uint8_t[4]){0x36, 0x25, 0x37, 0x25});
    }
    place_metatile_at(x+width+1, y, 0, (const uint8_t[4]){0x24, 0x38, 0x24, 0x24});
}

void place_pipe_at(int x, int y, int height) {
    /*
     .db $60, $64, $61, $65 ;decoration pipe end left, points up
     .db $62, $66, $63, $67 ;decoration pipe end right, points up
     .db $68, $68, $69, $69 ;pipe shaft left
     .db $26, $26, $6a, $6a ;pipe shaft right
     */

    for (int i=0; i<height; i++) {
        place_metatile_at(x, y-i, 0, (const uint8_t[4]){0x68, 0x68, 0x69, 0x69});
        place_metatile_at(x+1, y-i, 0, (const uint8_t[4]){0x26, 0x26, 0x6a, 0x6a});
    }
    place_metatile_at(x, y-height, 0, (const uint8_t[4]){0x60, 0x64, 0x61, 0x65});
    place_metatile_at(x+1, y-height, 0, (const uint8_t[4]){0x62, 0x66, 0x63, 0x67});

}


// x is the center, y is the top
void place_mountaintop_at(int x, int y) {
    /*
    .db $24, $30, $30, $26 ;mountain left
     .db $26, $26, $34, $26 ;mountain left bottom/middle center
     .db $24, $31, $24, $32 ;mountain middle top
     .db $33, $26, $24, $33 ;mountain right
     */
    place_metatile_at(x, y, 0, (const uint8_t[4]){0x24, 0x31, 0x24, 0x32});
    place_metatile_at(x, y+1, 0, (const uint8_t[4]){0x26, 0x26, 0x34, 0x26});
    place_metatile_at(x-1, y+1, 0, (const uint8_t[4]){0x24, 0x30, 0x30, 0x26});
    place_metatile_at(x+1, y+1, 0, (const uint8_t[4]){0x33, 0x26, 0x24, 0x33});
}

// x is the center, y is the top of the mountain (matches y above)
void place_mountainbase_at(int x, int y) {
    // mountain left metatile (0x05)
    place_metatile_at(x-2, y+2, 0, (const uint8_t[4]){0x24, 0x30, 0x30, 0x26});
    // mountain right
    place_metatile_at(x+2, y+2, 0, (const uint8_t[4]){0x33, 0x26, 0x24, 0x33});
    // 0x06   .db $26, $26, $34, $26 ;mountain left bottom/middle center
    place_metatile_at(x-1, y+2, 0, (const uint8_t[4]){0x26, 0x26, 0x34, 0x26});
    //  .db $34, $26, $26, $26 ;mountain right bottom
    place_metatile_at(x+1, y+2, 0, (const uint8_t[4]){0x34, 0x26, 0x26, 0x26});
    // .db $26, $26, $26, $26 ;mountain middle bottom
    place_metatile_at(x, y+2, 0, (const uint8_t[4]){0x26, 0x26, 0x26, 0x26});
}


void place_brick_at(int x, int y) {
    //   .db $45, $47, $45, $47 ;breakable brick w/ line
    place_metatile_at(x, y, 1, (const uint8_t[4]){0x45, 0x47, 0x45, 0x47});

}

void place_question_at(int x, int y, int power_up) {
//   .db $53, $55, $54, $56 ;question block (coin)
//   .db $53, $55, $54, $56 ;question block (power-up)
    place_metatile_at(x, y, 3, (const uint8_t[4]){0x53, 0x55, 0x54, 0x56});
}

void place_cloud_at(int x, int y, int width) {
    //   .db $24, $24, $24, $35 ;cloud left
    place_metatile_at(x, y, 2, (const uint8_t[4]){0x24, 0x24, 0x24, 0x35});

    //   .db $24, $24, $39, $24 ;cloud bottom left
    place_metatile_at(x, y+1, 2, (const uint8_t[4]){0x24, 0x24, 0x39, 0x24});
    //   .db $24, $38, $24, $24 ;cloud right
    place_metatile_at(x+width+1, y, 2, (const uint8_t[4]){0x24, 0x38, 0x24, 0x24});

    //   .db $3c, $24, $24, $24 ;cloud bottom right
    place_metatile_at(x+width+1, y+1, 2, (const uint8_t[4]){0x3c, 0x24, 0x24, 0x24});

    for (int i=0; i<width; i++) {
        // .db $36, $25, $37, $25 ;cloud middle
        place_metatile_at(x+i+1, y, 2, (const uint8_t[4]){0x36, 0x25, 0x37, 0x25});

        //   .db $3a, $24, $3b, $24 ;cloud bottom middle
        place_metatile_at(x+i+1, y+1, 2, (const uint8_t[4]){0x3a, 0x24, 0x3b, 0x24});
    }
}

void place_ground_at(int x, int y, int width, int height) {
    for (int v=0; v<height; v++) {
        for (int h=0; h<width; h++) {
            place_metatile_at(x+h, y+v, 1, (const uint8_t[4]){0xb4, 0xb6, 0xb5, 0xb7});
        }
    }
}


int main() {
    stdio_init_all();
    mode1_init();

    palette[0] = palette0;
    palette[1] = palette1;
    palette[2] = palette2;
    palette[3] = palette3;

    place_ground_at(0, 13, MAP_WIDTH/2, 2);

    place_cloud_at(8, 3, 1);
    place_cloud_at(19, 2, 1);
    place_cloud_at(27, 3, 3);
    place_cloud_at(36, 2, 2);

    place_bush_at(11, 12, 3);
    place_bush_at(23, 12, 1);

    place_pipe_at(28, 12, 1);
    place_pipe_at(38, 12, 2);

    place_brick_at(20, 9);
    place_brick_at(22, 9);
    place_brick_at(24, 9);

    place_question_at(16, 9, 0);
    place_question_at(21, 9, 1);
    place_question_at(23, 9, 0);
    place_question_at(22, 5, 0);

    place_mountaintop_at(2, 10);
    place_mountainbase_at(2, 10);

    place_mountaintop_at(17, 11);

    place_bush_at(41, 12, 2);
    place_pipe_at(46, 12, 3);
    place_mountaintop_at(50, 10);
    place_mountainbase_at(50, 10);
    place_pipe_at(57, 12, 3);

    place_bush_at(59, 12, 3);
    place_mountaintop_at(65, 11);
    place_bush_at(71, 12, 1);
    place_bush_at(89, 12, 2);
    place_mountaintop_at(98, 10);
    place_mountainbase_at(98, 10);
    place_bush_at(107, 12, 3);
    place_mountaintop_at(113, 11);
    place_bush_at(119, 12, 1);
    place_bush_at(137, 12, 2);
    place_mountaintop_at(146, 10);
    place_mountainbase_at(146, 10);
    place_bush_at(158, 12, 0);
    place_mountaintop_at(161, 11);
    place_pipe_at(163, 12, 1);
    place_bush_at(167, 12, 1);
    place_pipe_at(179, 12, 1);
    place_mountaintop_at(194, 10);
    place_mountainbase_at(194, 10);



    draw_background();

    while (1) {
        for (int i=0; i<32; i++) {
        scroll_background(1);
        sleep_ms(1);
        }
        //height_offset += 1;
        //draw_background();
    }

    const uint LED_PIN = 25;
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);

    // flash the LED and send SPI rate to the terminal
    uint8_t pin = 0;
    while (1) {
        pin = 1-pin;
        gpio_put(LED_PIN, pin);
        sleep_ms(1000);
    }
}
```