CSEE W4840 Final Report: Pac-Man

Jerry Lin (ml4686), Leo Qiao (flq2101)

May 13, 2022

1 Project Overview

In this project, we implement a variation of the Pac-Man on the FGPA. Pac-Man is a simple and popular game developed in the 1980s where a player-controlled yellow Pac-Man attempts to navigate a maze filled with food pellets that are eaten for points. Simultaneously, the player must avoid touching four ghosts who follow Pac-Man inside the maze. To implement the game, we used a FPGA to create a general purpose hardware-accelerated 2-D graphics API using sprite-and-tile graphics. Software will be used for all game logic and position tracking while a USB SNES controller will be used to play the game. The software and hardware will be connected by a driver program that sends data and corresponding data to the byte-addressable VRAM in the hardware. The overall block diagram of our system is shown in Figure 1.

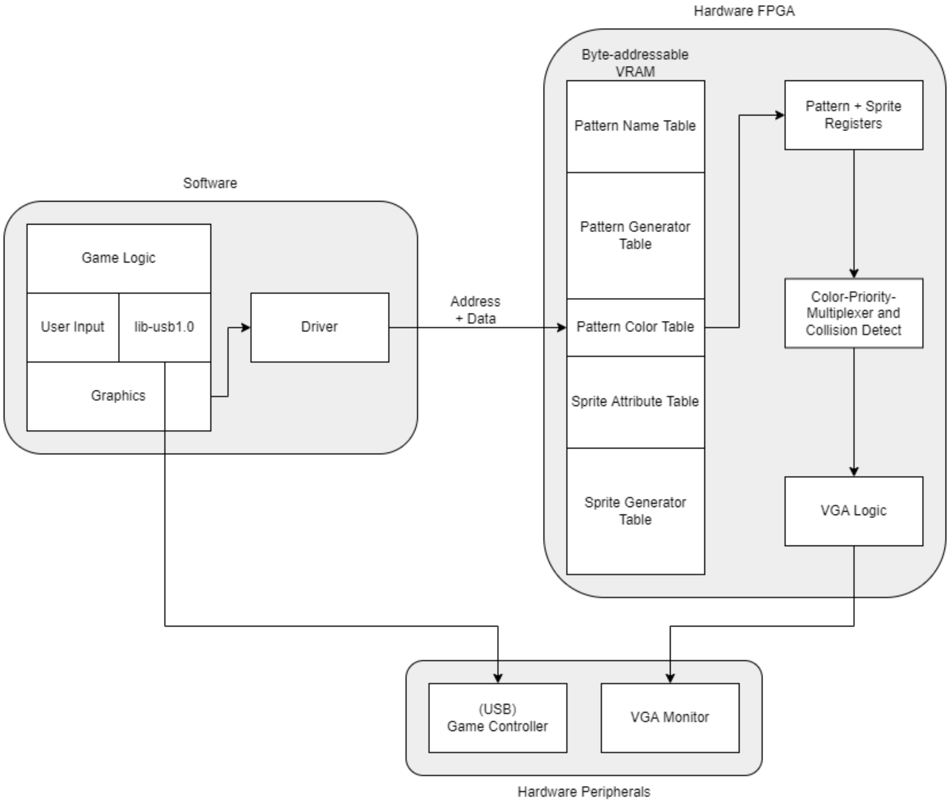

Figure 1: System Block Diagram

2 System Design

2.1 Hardware

The hardware of our system will be responsible for displaying graphics commanded by the software at a specified location. We implemented a tile-and-sprite graphics generator inspired by the architecture of the TMS9918 graphics processor. We designed our hardware to be general purpose so that it could support any game or even arbitrary graphics display with only a modification of the software. This is provided that the graphics data do no exceed our VRAM size

Since we require only 10 colors for our game, we created a color LUT that maps a 4 bit color code to a 24 bit RGB value, thus saving a significant amount of space. In our previous design document, included storage of the full 24 bit RGB value for each pixel in our resource budget. In addition to the color LUT which uses only combinational logic, we have 4 tables stored in RAM, each with a single read and write port.

For stationary graphics elements, such as the food pellets, the maze walls, and the text, tiles will be used for display. The tile pixel information is stored in the pattern generator table, which stores two pixels (1 byte) per row. A full 8x8 pixel pattern will require 32 bytes or rows in the pattern generator table. This table is addressed using 11 bit addresses and has 2048 rows, which allows for up to 64 unique patterns.

Our system is designed to display up to 64 by 64 8x8 pixel tiles. The pattern name table is used to index into the pattern generator table to determine the actual pattern displayed at each tile. Each 1 byte row of the table corresponds to a different tile and stores the address of the pattern to be displayed at that tile location. We made a space-saving optimization to drop the 5 LSB of the stored address of the pattern generator table. This is possible because a new pattern can only start in rows divisible by 32 in the generator table. The pattern name table is addressed with 12 bit addresses and contains 4096 rows, one for each tile location.

In terms of displaying the patterns, a finite state machine is used. During VGA horizontal sync, the hardware first uses the upcoming VGA vertical count value to index into the pattern name table to obtain an address for the pattern name table. The vertical count is used again to calculate the exact row of the pattern to be displayed, after which the pixel data is loaded into a shift register. This process is repeated until the pixel rows of all 64 tiles are loaded. As the next vertical count begins, 4 bit pixel values are read from the shift register one-by-one and mapped to a 24 bit RGB value for display.

For mobile graphics elements such as Pac-Man and the four ghosts, sprites were used. The sprite generator table is similar to the pattern generator table. A full 16x16 pixel sprite 128 bytes or rows in the sprite generator table, which is addressed using an 11 bit address. This table has 2048 rows, which allows for up to 16 unique sprites.

The sprite attribute table stores the location and addresses of sprites. For each sprite, the table stores an 1 byte vertical position, then an 1 byte horizontal position, and finally 1 byte for the base address of the sprite in the sprite generator table. Since we are only displaying 5 simultaneous sprites at a time, the sprite attribute table contains only 32 rows and uses a 5 bit address.

For a sprite to be displayed, each sprite has its own finite machine and waits for their designed , non-overlapping horizontal count value during VGA horizontal sync before memory access can begin. This is a limitation of our VRAM, which only allows for one memory access per cycle. For a given sprite, the vertical position in the attribute table is compared to VGA count, and if any part of the sprite must appear in the next line, the sprite generator table is read starting at the specified base address and horizontal position of the sprite is loaded into a down counter. The sprite pixels are then loaded into a shift register in a similar way to loading patterns. As the next vertical count starts, the down counter decrements until it reaches 0, after which 4 bit pixel values are read from the shift register and mapped to a 24 bit RGB value.

A top level module containing a color-priority-multiplexer controls the final VGA pixel value. Sprite pixels will appear over pattern pixels while sprite with a lower index in the sprite attribute table will show over sprite with higher indices. The top level module is also responsible for parsing data packets from the Avalon into the correct tables and address. In total hardware VRAM, consisting of the four previously mentioned tables, takes up **65792 bits** (8224 bytes) which is significantly less than the estimated 274944 bits in the design document.

2.2 Hardware/Software Communication

In order to load the hardware VRAM and change sprite positions, the hardware receives 32 bit data packets from the Avalon bus from the driver. The structure of the data packet is very simple, consisting of only 3 parts. Bits 0-1 selects which of the four tables to write data to. Bits 2-17 is the address in the selected table to write data to. Finally, bits 24-31 is the data to write to the specified address in the specified table. This simple interface allows for modular hardware.

2.3 Software

2.3.1 Peripheral: Gamepad

For our gaming peripheral, we purchased an SNES Classic USB Controller by Kiwitata from Amazon. Without any data sheet online, we resorted to reverse engineering and discovered the following communication protocol using libusb-1.0. The protocol uses a total of 7 bytes:

- 24 bits: (constant)
  - default: 0x017f7f
- 8 bits: (left/right)
  - default: 0x7f
  - left: 0x00
  - right: 0xff
- 8 bits: (up/down)
  - default: 0x7f
  - up: 0x00
  - down: 0xff
- 8 bits: (X/Y/A/B)
  - bit 7: Y
  - bit 6: B
  - bit 5: A
  - bit 4: X
  - bit 3: 1 (constant)
  - bit 2: 1 (constant)
  - bit 1: 1 (constant)
  - bit 0: 1 (constant)

- 8 bits: (SELECT/START/L/R)
  - bit 7: 0 (constant)
  - bit 6: 0 (constant)
  - bit 5: START
  - bit 4: SELECT
  - bit 3: 0 (constant)
  - bit 2: 0 (constant)
  - bit 1: R
  - bit 0: L

With this communication protocol understood, we setup up an event listener mechanism in C to set up a generic gamepad button event API.

2.3.2 Driver

The driver is separated into 2 parts: kernel space and user space.

The kernel space driver is very simple. It takes in 3 integer fields: table, addr, data. The integer fields are then shifted and OR'ed together to form teh 32-bit graphics command.

The user space driver builds on this simple kernel space driver and provides abstractions for drawing and loading patterns and sprites. The patterns and sprites are represented as 2-D arrays of bytes, and colors are represented by C macros to correspond to the values inside the hardware color LUT. The most notable functions that are provided by the user space driver are: set_sprite_bitmap, set_sprite, set_pattern_bitmap and set_pattern_at. The functions corresponds to loading entries into each of the 4 hardware tables.

2.3.3 Game Loop

The game loop is set to cycle once every 1ms. In order to provide varying rates for different events to occur, different counters are increment and modulo'ed at each cycle and heart beat functions return true at whenever the corresponding counter returns to 0.

The game is separated into 3 stages, namely, STAGE_MENU, STAGE_IN_GAME, STAGE_END_GAME.

2.3.4 Character Movements

The pacman movement is relatively straightforward. Whenever a gamepad direction key-down event is received, the set_pacman_dir function does the following simple check to see whether the direction should be applied immediately or buffered: if the new direction is perpendicular to current direction, the direction is buffered; else, the direction is immediately applied and buffered direction is cleared. This aligns with how pacman moves in the original Pac-Man game.

The ghost movements are much more complicated. There are a total of 5 movement modes: trapped, release, random, chase and scatter.

In trapped mode, the ghosts simply move up and down in the middle gated call.

In release mode, the ghosts start a 2-phase process that moves them to a designated starting point. The 2 phases refers to the x and y movements.

In random mode, the ghosts pick a random direction to move at each cycle. However, the backward direction is never considered because the ghosts are not supposed to move backwards.

In chase mode, a BFS search for the pacman is run for each direction, and the depths of each direction are recorded. The ghost then picks the direction with the lowest depth.

In scatter mode, the mechanism is similar to chase mode, except that the ghost picks the direction with the highest depth.

3 Challenges and Lessons Learned

One major challenge faced was debugging issues with the code. On the hardware side, it is very difficult to perform functional verification of a change unless we manually reload the FGPA with a new .rbf and .dtb and visually inspect the output. We found a workaround of this issue by writing a testbench for the hardware code and simulating the RTL using ModelSim. This solution greatly increased our development speed as pure RTL simulation is significantly faster compared to using the Qsys and Quartus workflow. We suggest that future project groups use a similar workflow based on RTL simulation for debugging. All files necessary for running the simulation is included in the code listing in the following section.

On a related note, we learned that having robust, functional hardware is critical to software and driver development. We often encountered undesirable graphics behavior that could have been attributed to both the software and hardware, which hampered troubleshooting. Therefore, developing a well-thought RTL testbench that covers edge cases is very important to eliminating hidden bugs in hardware code, and in turn, facilitating software development.

Lastly, game development usually requires a great deal of software engineering. Clean abstractions are a necessity, because the large number of variables can quickly slow down the development process if code base is not well-structured.

4 Code Listing

4.1 Hardware

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic        clk,
          input logic      reset,
    input logic [31:0]  writedata,
    input logic      write,
    input        chipselect,
    input logic [3:0]  address,

    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic     VGA_CLK, VGA_HS, VGA_VS,
                       VGA_BLANK_n,
    output logic     VGA_SYNC_n);

  logic [10:0]     hcount;
  logic [9:0]     vcount;

  logic [3:0] out_pixel[5:0]; //output pixels values from each of 5 sprites + 1
    pattern
  logic [3:0] final_out_pixel; //actual output pixel to display
  logic [7:0]      background_r, background_g, background_b;
  logic [23:0] rgb_val; //final RGB value to display


    //for pattern name table
    logic [11:0] ra_n, wa_n; //12 bits
  logic we_n;
  logic [7:0] din_n;
  logic [7:0] dout_n;

  //for pattern generator table
    logic [10:0] ra_pg, wa_pg; //change later
  logic we_pg;
  logic [7:0] din_pg;
  logic [7:0] dout_pg;

  //for sprite attribute table
    logic [4:0] ra_a, wa_a; //5  simultaneous sprites
  logic we_a;
  logic [7:0] din_a;
  logic [7:0] dout_a;

  //for sprite generator table
    logic [10:0] ra_g, wa_g; //10*128 sprite -> 11 bit addr
  logic we_g;
  logic [7:0] din_g;
  logic [7:0] dout_g;

  logic [4:0] sprite_base_addr[4:0]; //sprite attr table base address
  logic [10:0] h_start[4:0]; //hcount at which sprite_prep n starts
  logic [4:0] sprite_ra_a[4:0]; //requested read address for sprite attr table from sprite prep modules
  logic [10:0] sprite_ra_g[4:0]; //requested read address for sprite gen table from sprite prep modules

  //determines where each sprite prep instance will start reading the attr table from
  assign sprite_base_addr[0]=5'h0;
  assign sprite_base_addr[1]=5'h4;
  assign sprite_base_addr[2]=5'h8;
  assign sprite_base_addr[3]=5'hc;
  assign sprite_base_addr[4]=5'h10;

  //determines when each sprite prep instance will start processing sprites
  assign h_start[0]=11'b10100100000; //1312
  assign h_start[1]=11'b10100111010; //1338
  assign h_start[2]=11'b10101010100; //1364
  assign h_start[3]=11'b10101101110; //1390
  assign h_start[4]=11'b10110001000; //1416



  vga_counters counters(.clk50(clk), .*);
  patt_name_table pn1(.clk(clk), .ra(ra_n), .wa(wa_n), .we(we_n), .din(din_n), .dout(dout_n));
  patt_gen_table pg1(.clk(clk), .ra(ra_pg), .wa(wa_pg), .we(we_pg), .din(din_pg), .dout(dout_pg));

  sprite_attr_table sat1(.clk(clk), .ra(ra_a), .wa(wa_a), .we(we_a), .din(din_a), .dout(dout_a));
  sprite_gen_table sgt1(.clk(clk), .ra(ra_g), .wa(wa_g), .we(we_g), .din(din_g), .dout(dout_g));
  color_lut cl1(.color_code(final_out_pixel), .rgb_val(rgb_val));

  pattern_prep pp0(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n),
  .dout_n (dout_n), .dout_g (dout_pg), .ra_n (ra_n), .ra_g(ra_pg), .out_pixel(out_pixel[5]));

  sprite_prep sp0(.clk(clk), .reset(reset), .h_start(h_start[0]), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n), .base_addr(sprite_base_addr[0]),
  .dout_a (dout_a), .dout_g (dout_g), .ra_a (sprite_ra_a[0]), .ra_g(sprite_ra_g[0]), .out_pixel(out_pixel[0]));

  sprite_prep sp1(.clk(clk), .reset(reset), .h_start(h_start[1]), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n), .base_addr(sprite_base_addr[1]),
  .dout_a (dout_a), .dout_g (dout_g), .ra_a (sprite_ra_a[1]), .ra_g(sprite_ra_g[1]), .out_pixel(out_pixel[1]));

  sprite_prep sp2(.clk(clk), .reset(reset), .h_start(h_start[2]), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n), .base_addr(sprite_base_addr[2]),
  .dout_a (dout_a), .dout_g (dout_g), .ra_a (sprite_ra_a[2]), .ra_g(sprite_ra_g[2]), .out_pixel(out_pixel[2]));

  sprite_prep sp3(.clk(clk), .reset(reset), .h_start(h_start[3]), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n), .base_addr(sprite_base_addr[3]),
  .dout_a (dout_a), .dout_g (dout_g), .ra_a (sprite_ra_a[3]), .ra_g(sprite_ra_g[3]), .out_pixel(out_pixel[3]));

  sprite_prep sp4(.clk(clk), .reset(reset), .h_start(h_start[4]), .hcount(hcount), .vcount(vcount), .VGA_BLANK_n(VGA_BLANK_n), .base_addr(sprite_base_addr[4]),
  .dout_a (dout_a), .dout_g (dout_g), .ra_a (sprite_ra_a[4]), .ra_g(sprite_ra_g[4]), .out_pixel(out_pixel[4]));

    always_ff @(posedge clk) begin //Writing to VRAM
     if (reset) begin
      background_r <= 8'h0;
      background_g <= 8'h0;
      background_b <= 8'h20;
     end else if (chipselect && write) begin
      case (writedata[1:0])
        2'b0 : begin //pattern name table
            we_n<=1;
            we_pg<=0;
            we_a<=0;
            we_g<=0;
            din_n<=writedata[31:24];
            wa_n<=writedata[13:2];
          end
        2'b1 : begin //pattern gen table
            we_n<=0;
            we_pg<=1;
            we_a<=0;
            we_g<=0;
            din_pg<=writedata[31:24];
            wa_pg<=writedata[12:2];
          end
        2'b10 : begin //sprite attr table
            we_n<=0;
            we_pg<=0;
            we_a<=1;
            we_g<=0;
            din_a<=writedata[31:24];
            wa_a<=writedata[6:2];
          end
        2'b11 : begin //sprite gen table
            we_n<=0;
            we_pg<=0;
            we_a<=0;
            we_g<=1;
            din_g<=writedata[31:24];
            wa_g<=writedata[12:2];
          end
      endcase
     end
  end

  always_comb begin //Display logic
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n ) begin
      if (final_out_pixel!=4'b0) {VGA_R, VGA_G, VGA_B} = {rgb_val[23:16], rgb_val[15:8], rgb_val[7:0]};
```

```
      else  {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
    end
  end

  always_comb begin //color prioirity multiplexer (i.e. sprite 1 pixels precedes sprite 2, sprite 2 > sprite 3...)
    if (out_pixel[0]!=4'b0) final_out_pixel=out_pixel[0];
    else if (out_pixel[1]!=4'b0) final_out_pixel=out_pixel[1];
    else if (out_pixel[2]!=4'b0) final_out_pixel=out_pixel[2];
    else if (out_pixel[3]!=4'b0) final_out_pixel=out_pixel[3];
    else if (out_pixel[4]!=4'b0) final_out_pixel=out_pixel[4];
    else if (out_pixel[5]!=4'b0) final_out_pixel=out_pixel[5]; //pattern has lowest pixel priority
    else final_out_pixel=4'b0;
  end



  always_comb begin //VRAM read multiplexer
    //multiplex sprite attribute table reads
    if ((hcount>=h_start[0]) && (hcount<h_start[1])) begin
      ra_a=sprite_ra_a[0];
      ra_g=sprite_ra_g[0];
    end else if ((hcount>=h_start[1]) && (hcount<h_start[2])) begin
      ra_a=sprite_ra_a[1];
      ra_g=sprite_ra_g[1];
    end else if ((hcount>=h_start[2]) && (hcount<h_start[3])) begin
      ra_a=sprite_ra_a[2];
      ra_g=sprite_ra_g[2];
    end else if ((hcount>=h_start[3]) && (hcount<h_start[4])) begin
      ra_a=sprite_ra_a[3];
      ra_g=sprite_ra_g[3];
    end else if (hcount>=h_start[4]) begin
      ra_a=sprite_ra_a[4];
      ra_g=sprite_ra_g[4];
    end else begin //below should never run here
      ra_a=5'b0;
      ra_g=11'b0;
    end
  end

endmodule

module sprite_prep (input logic clk, reset,
  input logic [10:0] h_start,
  input logic [10:0] hcount,
  input logic [9:0] vcount,
  input logic VGA_BLANK_n,
  input logic [4:0] base_addr, //base address in sprite attr table
  input logic [7:0] dout_a,
  input logic [7:0] dout_g,
  output logic [4:0] ra_a,
  output logic [10:0] ra_g,
  output logic [3:0] out_pixel);

  logic [8:0] down_counter; //8 bit wide down counter
  logic [63:0] shift_reg; //64 bit wide shift register
  logic [7:0] shift_pos; //position in shift reg to read pixel value from
  logic [10:0] sprite_offset; //which row of a given sprite to display
  logic [63:0] display_pixel;// determines whether sprite or background pixel is shown
  logic [7:0] shift_reg_shift; //bit position in shift reg to write to (0-63, steps of 8)
  assign out_pixel=display_pixel[3:0];

  enum {IDLE, READ_VERT_POS,READ_VERT_POS_WAIT, READ_VERT_POS_WAIT2, READ_HORT_POS, READ_HORT_POS_WAIT,
  READ_SPRITE_ADDR, READ_SPRITE_ADDR_WAIT, READ_SPRITE_PIXELS_BASE, READ_SPRITE_PIXELS_BASE_WAIT,
  LOAD_SHIFT_REG, LOAD_SHIFT_REG_WAIT, SPRITES_LOADED, COUNT_DOWN, PREPARE_PIXELS }
  state, state_next;


  always_ff @(posedge clk) begin
    state<=state_next;
    if (reset) begin
      state<=IDLE;
      ra_g<=0;
      ra_a<=0;
    end

    case (state)
      IDLE: begin
        display_pixel<=64'b0;
        shift_reg<=64'b0;
        shift_reg_shift<=8'h40; //dec=64 (actual value used is 8 less)
        shift_pos<=8'h40; //dec=64 set shift position to start of shift regs (MSB) (actual value used is 4 less)
      end
      READ_VERT_POS: begin
        ra_a<=base_addr; //address of (starting) vertical position of sprite
      end
      READ_HORT_POS: begin
        ra_a<=base_addr+5'b1; //address of horizontal position of sprite
        sprite_offset<={2'b0, vcount[8:0]-{dout_a, 1'b0}}; //which of 16 rows of sprite to display //e.g. vcount=11, v_pos=5 -> 11-5=6th row
      end
      READ_SPRITE_ADDR: begin //base address need right shift of 3 bits
        ra_a<=base_addr+5'b10; //address of base address of sprite pixels in the generator table //test using 0
        down_counter<={dout_a, 1'b0}; //copy horizontal position into down counter
      end
      READ_SPRITE_PIXELS_BASE: begin //!!note: address no longer >> shifted by 3!!
        ra_g<={dout_a[3:0], 7'b0} + (sprite_offset<<3); //read left-most 8 pixels in gen table, 8x offset since 8 table rows needed per pixel line
      end
      LOAD_SHIFT_REG: begin
        shift_reg<= ({56'b0, dout_g}<<(shift_reg_shift-8'h8)) | shift_reg; //store left-most 8 pixels of sprite line
        shift_reg_shift<=shift_reg_shift-8'h8; //minus 8
        ra_g<=ra_g+1; //increment gen table address by one to read upcoming pixels
      end
      COUNT_DOWN: begin
        //only down count every 2 hcounts
        if (down_counter>9'b0 && VGA_BLANK_n && !hcount[0]) down_counter<=down_counter-1;
      end
      PREPARE_PIXELS: begin
        if (VGA_BLANK_n && !hcount[0]) begin
          display_pixel<=(shift_reg>>(shift_pos-8'h4)); //Only 4 LSB of display_pixel matter
          shift_pos<=shift_pos-8'h4; //minus 4
        end
      end
    endcase

  end

  always_comb begin
        case (state)
            IDLE:       state_next = (hcount==h_start) ? READ_VERT_POS: IDLE;
            READ_VERT_POS:      state_next = READ_VERT_POS_WAIT; //extra cycle for reading vertical position in attr table
      READ_VERT_POS_WAIT:   state_next = READ_VERT_POS_WAIT2; //ra_a update needs 2 cycles for some reason
      READ_VERT_POS_WAIT2: state_next = ((vcount [8:0]>={dout_a, 1'b0}) && (vcount[8:0]<({dout_a, 1'b0}+8'b10000)))? READ_HORT_POS: IDLE;  //check if any part of sprite is showing (don't need last 4 LSB)
      READ_HORT_POS:       state_next = READ_HORT_POS_WAIT;  //extra cycle for mem read
      READ_HORT_POS_WAIT:  state_next = READ_SPRITE_ADDR;
      READ_SPRITE_ADDR:    state_next = READ_SPRITE_ADDR_WAIT; //extra cycle for mem read
      READ_SPRITE_ADDR_WAIT:    state_next = READ_SPRITE_PIXELS_BASE;
      READ_SPRITE_PIXELS_BASE: state_next= READ_SPRITE_PIXELS_BASE_WAIT; //extra cycle for mem read
      READ_SPRITE_PIXELS_BASE_WAIT: state_next= LOAD_SHIFT_REG;
      LOAD_SHIFT_REG: state_next= LOAD_SHIFT_REG_WAIT;
      LOAD_SHIFT_REG_WAIT: state_next= (shift_reg_shift==8'b0) ? SPRITES_LOADED: LOAD_SHIFT_REG;

      //if new vertical line started, begin down counting
      SPRITES_LOADED: state_next= (hcount==11'b1111111) ? COUNT_DOWN : SPRITES_LOADED; //start at 127
      COUNT_DOWN: state_next= (down_counter==9'b0) ? PREPARE_PIXELS: COUNT_DOWN;
      PREPARE_PIXELS: state_next= (shift_pos==8'b0) ? IDLE : PREPARE_PIXELS;
            default:    state_next = IDLE;
        endcase
    end
endmodule

module pattern_prep (input logic clk, reset,
  input logic [10:0] hcount,
  input logic [9:0] vcount,
  input logic VGA_BLANK_n,
  input logic [7:0] dout_n,
  input logic [7:0] dout_g,
  output logic [11:0] ra_n,
  output logic [10:0] ra_g,
  output logic [3:0] out_pixel);

  logic [2047:0] shift_reg; //8*64*4 bit wide shift register
  logic [11:0] shift_pos; //position in shift reg to read pixel value from
  logic [10:0] pattern_row_offset; //which of 8 of a given pattern to display
  logic [2047:0] display_pixel;// determines whether sprite or background pixel is shown
  logic [11:0] shift_reg_shift; //bit position in shift reg to write to (0-63, steps of 8)
  logic [7:0] tile_total_counter; //counts the total number of tiles that has been loaded into shift reg
  logic [7:0] tile_pixel_counter; //counts the number of tile pixel rows that has been loaded
  assign out_pixel=display_pixel[3:0];

  parameter [11:0] v_start=12'h0; //vertical position where first pattern begins
  parameter [7:0] tiles_per_row=8'd64; //number of tiles per row
  //parameter [7:0] tiles_per_col=8'h18; //number of tiles per column
  parameter [11:0] name_table_addr_mask={6'b111111, 6'b0};

  enum {IDLE, READ_TILE_ADDR_BASE, READ_TILE_ADDR_BASE_WAIT, READ_PATT_PIXELS_BASE, READ_PATT_PIXELS_BASE_WAIT,
  LOAD_SHIFT_REG, LOAD_SHIFT_REG_WAIT, READ_TILE_NEXT, READ_TILE_NEXT_WAIT, PATT_LOADED, PREPARE_PIXELS }
  state, state_next;


  always_ff @(posedge clk) begin
    state<=state_next;
    if (reset) begin
      state<=IDLE;
      ra_n<=0;
      ra_g<=0;
    end

    case (state)
      IDLE: begin
        tile_total_counter<=8'b0;
        tile_pixel_counter<=8'b0;
        display_pixel<=2048'b0;
        shift_reg<=2048'b0;
        shift_reg_shift<=12'b100000000000; //dec=2048 (actual value used is 8 less)
        shift_pos<=12'b100000000000; // dec=2048 set shift position to start of shift regs (MSB) (actual value used is 4 less)

      end
      READ_TILE_ADDR_BASE: begin
        ra_n<=(({2'b0, vcount}-v_start)<<3) & name_table_addr_mask; //get address of (starting) tile pixel address in name table
        pattern_row_offset<={8'b0, vcount[2:0]-v_start[2:0]}; //which of 8 pixel rows to access
      end

      READ_PATT_PIXELS_BASE: begin //!!note: address no longer >> shifted by 3!!
        ra_g<={dout_n[5:0], 5'b0} + (pattern_row_offset<<2); //read base 8 pixels in gen table,4x offset since 4 table rows needed per pixel line
      end

      READ_PATT_PIXELS_BASE_WAIT: begin //!!note: address no longer >> shifted by 3!!
        ra_g<=ra_g+1;
      end

      LOAD_SHIFT_REG: begin //first time: gets ra_g pixels_base stage and not base_wait stage
        shift_reg<= ({2040'b0, dout_g}<<(shift_reg_shift-12'h8)) | shift_reg; //store left-most 8 pixels of sprite line
        shift_reg_shift<=shift_reg_shift-12'h8; //minus 8
        ra_g<=ra_g+1; //increment gen table address by one to read upcoming pixels
        tile_pixel_counter<=tile_pixel_counter+8'b1;
      end
      READ_TILE_NEXT: begin
        ra_n<=ra_n+1; //increment name table address
        tile_pixel_counter<=8'b0;
        tile_total_counter<=tile_total_counter+8'b1;
      end

      PREPARE_PIXELS: begin
        if (VGA_BLANK_n && !hcount[0]) begin
          display_pixel<=(shift_reg>>(shift_pos-12'h4)); //Only 4 LSB of display_pixel matter
          shift_pos<=shift_pos-12'h4; //minus 4
        end
      end
    endcase

  end

  always_comb begin
        case (state)
            IDLE:       state_next = ((hcount==11'd1152) && (vcount>=v_start[9:0]) && (vcount<10'd480)) ? READ_TILE_ADDR_BASE: IDLE; //start at h=1152 and vcount=0
            READ_TILE_ADDR_BASE:      state_next = READ_TILE_ADDR_BASE_WAIT; //extra cycle for reading vertical position in attr table
      READ_TILE_ADDR_BASE_WAIT:   state_next = READ_PATT_PIXELS_BASE; //check if true: ra_a update needs 2 cycles for some reason
      READ_PATT_PIXELS_BASE:    state_next = READ_PATT_PIXELS_BASE_WAIT;
      READ_PATT_PIXELS_BASE_WAIT: state_next= LOAD_SHIFT_REG;
      LOAD_SHIFT_REG: state_next= (tile_pixel_counter==8'h3) ? READ_TILE_NEXT: LOAD_SHIFT_REG;
      READ_TILE_NEXT: state_next=READ_TILE_NEXT_WAIT;
      READ_TILE_NEXT_WAIT: state_next=(tile_total_counter==tiles_per_row)? PATT_LOADED : READ_PATT_PIXELS_BASE;

      //if new vertical line started, begin down counting
      PATT_LOADED: state_next= (hcount==11'd127) ? PREPARE_PIXELS : PATT_LOADED;
      PREPARE_PIXELS: state_next= (shift_pos==12'b0 || vcount>10'd480) ? IDLE : PREPARE_PIXELS;
            default:    state_next = IDLE;
        endcase
    end
endmodule

module sprite_attr_table( //stores sprite information (x, y, name, color)
  //x and y position has to be a multiple (2x) of hcount/vcount since only 8 bits
  input logic clk,
  input logic [4:0] ra, wa, //change later
  input logic we,
  input logic [7:0] din,
  output logic [7:0] dout);

  logic[7:0] mem[31:0];

  always_ff @(posedge clk) begin
      if (we) mem[wa] <= din;
      dout <= mem[ra];
  end
endmodule

module sprite_gen_table( //stores 16x 16x16 sprites (only 10 needed)
  input logic clk,
  input logic [10:0] ra, wa, //change later
  input logic we,
  input logic [7:0] din,
  output logic [7:0] dout);

  logic[7:0] mem[2047:0]; //128 8 bit words need per sprite: 64 bits per pixel row

  always_ff @(posedge clk) begin
      if (we) mem[wa] <= din;
      dout <= mem[ra];
  end
endmodule

module patt_name_table( //stores 8 bit address of tiles on each row
  input logic clk,
  input logic [11:0] ra, wa, //12 bit addr
  input logic we,
  input logic [7:0] din,
  output logic [7:0] dout);

  logic[7:0] mem[4095:0];

  always_ff @(posedge clk) begin
      if (we) mem[wa] <= din;
      dout <= mem[ra];
  end
endmodule

module patt_gen_table( //stores 8x8 patterns
  input logic clk,
  input logic [10:0] ra, wa,
  input logic we,
  input logic [7:0] din,
  output logic [7:0] dout);

  logic[7:0] mem[2047:0]; //32 8 bit words need per pattern: 4 table rows (32 bits) per pixel row

  always_ff @(posedge clk) begin
      if (we) mem[wa] <= din;
      dout <= mem[ra];
  end
endmodule

module color_lut(input logic  [3:0] color_code,
         output logic [23:0] rgb_val);
   always_comb
    case(color_code)
      4'h1: rgb_val=24'hfef104; //yellow pac-man
      4'h2: rgb_val=24'hfe0e03; //red ghost
      4'h3: rgb_val=24'hfeb846; //orange ghost
      4'h4: rgb_val=24'h00ecfe; //cyan ghost
      4'h5: rgb_val=24'hfdbff9; //pink ghost
      4'h6: rgb_val=24'he5dfee; //ghost eyes (whites)
      4'h7: rgb_val=24'h1e26b8; //blue: ghost eye iris, maze walls, blue ghosts
      4'h8: rgb_val=24'hffc0b7; //salmon food pellets
      4'h9: rgb_val=24'hffffff; //white text
      default: rgb_val=24'hffffff; //if something goes wrong, use white to make it
    obvious
    endcase

endmodule



module vga_counters(
 input logic       clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      ____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                            VBACK_PORCH; // 525

   logic endOfLine;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)          hcount <= 0;
     else if (endOfLine) hcount <= 0;
     else            hcount <= hcount + 11'd 1;

   assign endOfLine = hcount == HTOTAL - 1;

   logic endOfField;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)          vcount <= 0;
     else if (endOfLine)
       if (endOfField)   vcount <= 0;
       else              vcount <= vcount + 10'd 1;

   assign endOfField = vcount == VTOTAL - 1;

   // Horizontal sync: from 0x520 to 0x5DF (0x57F)
   // 101 0010 0000 to 101 1101 1111 (active LOW during 1312-1503) (192 cycles)
   assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
   assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

   assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

   // Horizontal active: 0 to 1279      Vertical active: 0 to 479
   // 101 0000 0000   1280        01 1110 0000   480
   // 110 0011 1111   1599        10 0000 1100   524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
      !( vcount[9] | (vcount[8:5] == 4’b1111) );

   /* VGA_CLK is 25 MHz
    *             --    --    --
    * clk50    __|  |__|  |__|
    *
    *             -----       --
    * hcount[0]__|     |_____|
    */
   assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

Listing 1: vga-ball.sv

4.2 Hardware Testbench

```
‘timescale 1ns/1ps
‘define HALF_CLOCK_PERIOD #20 //design uses 50MHz clock

module testbench();
  reg clk;
  reg rst;
  reg [31:0] writedata;
  reg write;
  reg chipselect;
  reg [3:0] address;
  wire [7:0] VGA_R, VGA_G, VGA_B;
  reg [15:0] writedata_address;
  reg [15:0] name_writedata_address;
  reg name_value;
  reg [3:0] color_counter;
  reg [7:0] pg_table_data;

  reg [1:0] table_select;
  reg [7:0] gen_table_base_addr;
  reg [7:0] v_pos;
  reg [7:0] h_pos;
  reg [15:0] j_counter1;
  reg [15:0] i_counter1;
  reg [7:0] table_data;



  integer i, j, k;

   vga_ball vga_ball0 ( .clk(clk), .reset(rst), .writedata(writedata), .write(write), .chipselect(chipselect), .address(address),
    .VGA_R(VGA_R), .VGA_G(VGA_G), .VGA_B(VGA_B), .VGA_CLK(), .VGA_HS(), .VGA_VS(), .VGA_BLANK_n(), .VGA_SYNC_n());

  always begin
    ‘HALF_CLOCK_PERIOD;
    clk = ~clk;
  end

  initial begin
    // register setup
    clk = 0;
    rst = 1;
    chipselect=0;
    write=0;
    writedata_address=0;
    name_writedata_address=0;
    color_counter=0;
    name_value=1;
    pg_table_data=0;

    table_select=0;
    gen_table_base_addr=0;
    v_pos=0;
    h_pos=0;
    j_counter1=0;
    i_counter1=0;
    table_data=0;
    @(posedge clk);

    @(negedge clk);   // release rst
    rst = 0;

    @(posedge clk);   // start the first cycle

    //start TB
    //Pattern TB
    //write pattern name table
    chipselect=1;
    write=1;
    table_select=2'b0;

    //Load pattern name table
    for (j=0 ; j<60; j=j+1) begin //# of cols
      for (i=0 ; i<64; i=i+1) begin
        writedata={name_value, 6'b0, name_writedata_address, table_select};
        name_value=~name_value;
        @(posedge clk);
        name_writedata_address=name_writedata_address+1;
      end
      name_value=~name_value; //extra toggle so each adjacent row has different patterns
    end

    table_select=2'b1;
    //write pattern generator table values
    for (j=0 ; j<2; j=j+1) begin //one of two patterns
      for (i=0 ; i<32; i=i+1) begin
        if (j==0) begin
          if (i<7) pg_table_data=8'b0011_0011; //orange sprite pixel data
          else if (i>15) pg_table_data=8'b0101_0101; //pink sprite pixel data
          else pg_table_data=8'b0000_0000; //transparent
        end else
          if (i%3) pg_table_data=8'b0000_0000; //transparent
          else pg_table_data= 8'b0001_0001; //yellow
        @(posedge clk);
        writedata={pg_table_data, 6'b0, writedata_address, table_select};
        writedata_address=writedata_address+1;
      end
    end

    //full 5 sprites tb
    @(posedge clk);

    chipselect=1;
    write=1;

    //write to sprite attribute table
    table_select=2'b10;
    for (j=0 ; j<5; j=j+1) begin
      i_counter1=0;
      for (i=0 ; i<4; i=i+1) begin
        case (i)
          0 : begin //sprite vertical position
            writedata={v_pos, 6'b0, (j_counter1<<2) + i_counter1,  table_select};
          end
          1 : begin //sprite horizontal position
            writedata={h_pos, 6'b0, (j_counter1<<2) + i_counter1,  table_select};
          end
          2 : begin //gen table base addr
            writedata={gen_table_base_addr, 6'b0, (j_counter1<<2) + i_counter1, table_select};
          end
          3 : begin //unused
            writedata={8'b0, 6'b0, (j_counter1<<2) + i_counter1,  table_select};
          end
        endcase
        @(posedge clk); //wait 1 cycle
        i_counter1=i_counter1+1;
      end
      if (j==3) begin //one more increment to get to edge (630, 460)
        v_pos=v_pos+8'd46;
        h_pos=h_pos+8'd50;
      end
      gen_table_base_addr=gen_table_base_addr+1;
      j_counter1=j_counter1+1;
      v_pos=v_pos+8'd46;
      h_pos=h_pos+8'd50;
    end


    //write to sprite generator table
    writedata_address=0;
    table_select=2'b11;
    for (k=0 ; k<5; k=k+1) begin
      for (j=0 ; j<16; j=j+1) begin
        for (i=0 ; i<8; i=i+1) begin
          @(posedge clk); //wait 1 cycle
          if (k%2) begin
            if (i<4) table_data=8'b0011_0011; //orange sprite pixel data
            else if (i>3 && i <6) table_data=8'b0000_0000; //transparent
            else table_data=8'b0101_0101; //pink sprite pixel data
          end else
            if (i<4) table_data=8'b0000_0000; //transparent
            else if (i>3 && i <6) table_data=8'b0001_0001; //yellow
            else table_data=8'b0010_0010; //red
          writedata={table_data, 6'b0, writedata_address, table_select};
          writedata_address=writedata_address+1;
        end
      end
    end

    for (i=0 ; i<850000; i=i+1) begin
      @(posedge clk);  // next cycle
    end
    $finish;
  end
endmodule // testbench
```

Listing 2: vga-ball-tb.sv

```
run:
  vsim -do "runsim.do"

clean:
  rm -rf work modelsim.ini
  rm -rf *.log *.syn *.rpt *.mr *.nl.v *.sdf *.svf *.vcd transcript *.wlf
  rm -rf *.result
```

Listing 3: Testbench Makefile

```
##################################################
#  Modelsim do file to run simuilation
#  MS 7/2015
##################################################

vlib work
vmap work work

# include netlist and testbench files
vlog +acc -incr ./vga_ball.sv
vlog +acc -incr ./vga_ball_tb.sv

# run simulation
```

```
vsim -t ps -lib work testbench
do waveformat.do
run -all
```

Listing 4: runsim.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/rst
add wave -noupdate /testbench/i
add wave -noupdate /testbench/writedata
add wave -noupdate /testbench/table_data
add wave -noupdate /testbench/vga_ball0/din_a
add wave -noupdate /testbench/vga_ball0/wa_a
add wave -noupdate /testbench/vga_ball0/din_g
add wave -noupdate /testbench/vga_ball0/wa_g
add wave -noupdate /testbench/vga_ball0/sp0/ra_a
add wave -noupdate /testbench/vga_ball0/sp0/dout_a
add wave -noupdate /testbench/vga_ball0/sp0/ra_g
add wave -noupdate /testbench/vga_ball0/sp0/dout_g
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/down_counter
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/shift_pos
add wave -noupdate /testbench/vga_ball0/sp0/sprite_offset
add wave -noupdate /testbench/vga_ball0/sp0/shift_reg
add wave -noupdate /testbench/vga_ball0/sp0/display_pixel
add wave -noupdate /testbench/vga_ball0/sp0/state
add wave -noupdate /testbench/vga_ball0/sp0/state_next


add wave -noupdate /testbench/vga_ball0/sp4/ra_a
add wave -noupdate /testbench/vga_ball0/sp4/dout_a
add wave -noupdate /testbench/vga_ball0/sp4/ra_g
add wave -noupdate /testbench/vga_ball0/sp4/dout_g
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp4/down_counter
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp4/shift_pos
add wave -noupdate /testbench/vga_ball0/sp4/sprite_offset
add wave -noupdate /testbench/vga_ball0/sp4/shift_reg
add wave -noupdate /testbench/vga_ball0/sp4/display_pixel
add wave -noupdate /testbench/vga_ball0/sp4/state
add wave -noupdate /testbench/vga_ball0/sp4/state_next

add wave -noupdate /testbench/vga_ball0/din_n
add wave -noupdate /testbench/vga_ball0/wa_n
add wave -noupdate /testbench/vga_ball0/din_pg
add wave -noupdate /testbench/vga_ball0/wa_pg
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/ra_n
add wave -noupdate /testbench/vga_ball0/pp0/dout_n
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/ra_g
add wave -noupdate /testbench/vga_ball0/pp0/dout_g
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/shift_pos
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/shift_reg_shift
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/tile_total_counter
add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/tile_pixel_counter
add wave -noupdate /testbench/vga_ball0/pp0/pattern_row_offset
add wave -noupdate /testbench/vga_ball0/pp0/shift_reg
add wave -noupdate /testbench/vga_ball0/pp0/out_pixel
add wave -noupdate /testbench/vga_ball0/pp0/state
add wave -noupdate /testbench/vga_ball0/pp0/state_next

add wave -noupdate -radix unsigned /testbench/write
add wave -noupdate -radix unsigned /testbench/chipselect
add wave -noupdate -radix unsigned /testbench/address
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/vcount
add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/hcount
add wave -noupdate /testbench/VGA_R
add wave -noupdate /testbench/VGA_G
add wave -noupdate /testbench/VGA_B
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3 ns} 0}
quietly wave cursor active 1
```

```
configure wave -namecolwidth 223
configure wave -valuecolwidth 89
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {0 ns} {12 ns}
```

Listing 5: waveformat.do

4.3 Software

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
        obj-m := vga_ball.o

else

# We are being compiled as a module: use the Kernel build system

KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
       PWD := $(shell pwd)
#
default: module pacman
# default: pacman

CFLAGS = -Wall

OBJECTS = pacman.o gamepad.o pattern.o sprite.o vga_ball_user.o map.o gameplay.o

pacman: $(OBJECTS)
  cc $(CFLAGS) -o pacman $(OBJECTS) -lusb-1.0 -pthread

pacman.o: pacman.c gamepad.h pattern.h map.h vga_ball_user.h

gamepad.o: gamepad.h

pattern.o: pattern.c pattern.h color.h vga_ball_user.h

sprite.o: sprite.c sprite.h

vga_ball_user.o: vga_ball_user.c vga_ball_user.h

map.o: map.c map.h pattern.h

gameplay.o: gameplay.c gameplay.h map.h pattern.h


module:
  ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
  ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
  ${RM} -rf *.o pacman

# TARFILES = Makefile README vga_ball.h vga_ball.c hello.c
# TARFILE = lab3-sw.tar.gz
# .PHONY : tar
# tar : $(TARFILE)
#
# $(TARFILE) : $(TARFILES)
#   tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)

endif
```

Listing 6: Makefile

```c
#include "gamepad.h"
#include "gameplay.h"
#include "map.h"
#include "pattern.h"
#include "sprite.h"
#include "vga_ball_user.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

void gameplay_listener(gamepad_button_event_t e, gamepad_button_t b) {
  switch (get_game_stage()) {
  case STAGE_MENU:
    if (e == GAMEPAD_KEY_UP && b == GAMEPAD_A) {
      press_start_game();
    }
    break;
  case STAGE_IN_GAME:
    if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_LEFT) {
      set_pacman_dir(DIR_LEFT);
    } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_RIGHT) {
      set_pacman_dir(DIR_RIGHT);
    } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_UP) {
      set_pacman_dir(DIR_UP);
    } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_DOWN) {
      set_pacman_dir(DIR_DOWN);
    }
    break;
  case STAGE_END_GAME:
    break;
  }
}

void show_ui() {
  set_map_at(25, 1, PAT_W);
  set_map_at(25, 2, PAT_E);
  set_map_at(25, 3, PAT_L);
  set_map_at(25, 4, PAT_C);
  set_map_at(25, 5, PAT_O);
  set_map_at(25, 6, PAT_M);
  set_map_at(25, 7, PAT_E);

  set_map_at(27, 1, PAT_T);
  set_map_at(27, 2, PAT_O);

  set_map_at(29, 1, PAT_D);
  set_map_at(29, 2, PAT_E);
  set_map_at(29, 3, PAT_1);
  set_map_at(29, 5, PAT_S);
  set_map_at(29, 6, PAT_O);
  set_map_at(29, 7, PAT_C);

  set_map_at(31, 1, PAT_P);
  set_map_at(31, 2, PAT_A);
  set_map_at(31, 3, PAT_C);
  set_map_at(31, 4, PAT_M);
  set_map_at(31, 5, PAT_A);
  set_map_at(31, 6, PAT_N);
}

void hide_side() {
  for (int r = 25; r <= 31; r++) {
    for (int c = 50; c <= 55; c++) {
      set_map_at(r, c, PAT_BACKGROUND);
    }
  }
}

void show_help() {
  static int counter = 0;
  static int flip = 1;
  counter = (counter + 1) % 800;

  if (counter == 0)
    flip *= -1;

  if (flip == 1) {
    set_map_at(25, 51, PAT_P);
    set_map_at(25, 52, PAT_R);
    set_map_at(25, 53, PAT_E);
    set_map_at(25, 54, PAT_S);
    set_map_at(25, 55, PAT_S);

    set_map_at(27, 51, PAT_A);

    set_map_at(29, 51, PAT_T);
    set_map_at(29, 52, PAT_O);

    set_map_at(31, 51, PAT_S);
    set_map_at(31, 52, PAT_T);
    set_map_at(31, 53, PAT_A);
    set_map_at(31, 54, PAT_R);
    set_map_at(31, 55, PAT_T);
  } else {
    hide_side();
  }
}

void show_game_over() {
  static int counter = 0;
  static int flip = 1;
  counter = (counter + 1) % 800;

  if (counter == 0)
    flip *= -1;

  if (flip == 1) {
    set_map_at(27, 51, PAT_G);
    set_map_at(27, 52, PAT_A);
    set_map_at(27, 53, PAT_M);
    set_map_at(27, 54, PAT_E);

    set_map_at(29, 51, PAT_O);
    set_map_at(29, 52, PAT_V);
    set_map_at(29, 53, PAT_E);
    set_map_at(29, 54, PAT_R);
  } else {
    hide_side();
  }
}

void show_congrats() {
  static int counter = 0;
  static int flip = 1;
  counter = (counter + 1) % 800;

  if (counter == 0)
    flip *= -1;

  if (flip == 1) {
    set_map_at(27, 51, PAT_N);
    set_map_at(27, 52, PAT_E);
    set_map_at(27, 53, PAT_W);

    set_map_at(29, 51, PAT_H);
    set_map_at(29, 52, PAT_I);
    set_map_at(29, 53, PAT_G);
    set_map_at(29, 54, PAT_H);

    set_map_at(31, 51, PAT_S);
    set_map_at(31, 52, PAT_C);
    set_map_at(31, 53, PAT_O);
    set_map_at(31, 54, PAT_R);
    set_map_at(31, 55, PAT_E);
  } else {
    hide_side();
  }
}

int main() {
  srand(time(NULL));
  vga_ball_init();
  gamepad_init();

  load_pattern_bitmaps();
  load_sprite_bitmaps();

  clear_screen();
  show_ui();
  setup_map();
  setup_map_foods();
  setup_map_score();

  setup_game();

  gamepad_set_listener(&gameplay_listener);

  while (1) {
    switch (get_game_stage()) {
    case STAGE_MENU:
      show_help();
      if (ghost_trapped_move_timer()) {
        move_ghosts_trapped();
      }
      break;
    case STAGE_IN_GAME:
      hide_side();

      if (pacman_move_timer()) {
        move_pacman();
      }
      if (blink_timer()) {
        animate_pacman();
      }
      if (ghost_trapped_move_timer()) {
        move_ghosts_trapped();
        move_ghosts_release();
      }
      if (ghost_move_timer()) {
        move_ghosts();
      }

      if (ghost_release_timer()) {
        release_next_ghost();
      }

      eat_food();
      ghosts_catch_pacman();

      scatter_timer();
      break;
    case STAGE_END_GAME: {
      // show game-over
      uint32_t counter = 0;
      while (counter < 3000) {
        counter++;
        show_game_over();
        usleep(1000);
      }
    }

      hide_side();

      if (beat_best_score()) {
        // show congrats
        uint32_t counter = 0;
        while (counter < 3000) {
          counter++;
          show_congrats();
          usleep(1000);
        }
      }
      update_scores();
      reset_game();
      break;
    }

    usleep(1000);
  }
  return 1;
}
```

Listing 7: pacman.c

```
#ifndef _PACMAN_GAMEPAD_H
#define _PACMAN_GAMEPAD_H

#include <stdint.h>

struct gamepad_packet {
  uint8_t reserved0;
  uint8_t reserved1;
  uint8_t reserved2;
  uint8_t dir_x;
  uint8_t dir_y;
  uint8_t primary;
  uint8_t secondary;
};

typedef uint16_t gamepad_button_t;
#define GAMEPAD_LEFT (((gamepad_button_t)1) << 0)
#define GAMEPAD_RIGHT (((gamepad_button_t)1) << 1)
#define GAMEPAD_UP (((gamepad_button_t)1) << 2)
#define GAMEPAD_DOWN (((gamepad_button_t)1) << 3)
#define GAMEPAD_X (((gamepad_button_t)1) << 4)
#define GAMEPAD_Y (((gamepad_button_t)1) << 5)
#define GAMEPAD_A (((gamepad_button_t)1) << 6)
#define GAMEPAD_B (((gamepad_button_t)1) << 7)
#define GAMEPAD_L (((gamepad_button_t)1) << 8)
#define GAMEPAD_R (((gamepad_button_t)1) << 9)
#define GAMEPAD_SELECT (((gamepad_button_t)1) << 10)
#define GAMEPAD_START (((gamepad_button_t)1) << 11)

#define GAMEPAD_DEFAULT ((gamepad_button_t)0)

typedef enum { GAMEPAD_KEY_DOWN, GAMEPAD_KEY_UP } gamepad_button_event_t;

void gamepad_init();
void gamepad_destroy();
void gamepad_set_listener(void (*listener)(gamepad_button_event_t,
                                           gamepad_button_t));

#endif
```

Listing 8: gamepad.h

```
#include "gamepad.h"
```

```c
#include <libusb-1.0/libusb.h>
#include <pthread.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

/* ---------- Gamepad USB info ---------- */
#define GAMEPAD_ID_VENDOR 0x79
#define GAMEPAD_ID_PRODUCT 0x11

/* Private function declarations */
void *gamepad_worker(void *arg);
void gamepad_generate_events(gamepad_button_t prev, gamepad_button_t next);
void gamepad_set_buttons(gamepad_button_t buttons);
struct libusb_device_handle *gamepad_open(uint8_t *endpoint_address);
gamepad_button_t gamepad_decode_packet(struct gamepad_packet packet);

/* ---------- States ---------- */
typedef struct {
  /* control */
  pthread_mutex_t mu;
  pthread_t tid;
  bool dead;

  /* current button state */
  gamepad_button_t buttons;

  /* usb */
  uint8_t endpoint;
  struct libusb_device_handle *handle;

  /* called by gamepad_worker() */
  void (*listener)(gamepad_button_event_t e, gamepad_button_t bs);
} gamepad_state_t;

static gamepad_state_t gp;

/* ---------- Implementations ---------- */
void gamepad_init() {
  int error;

  pthread_mutex_init(&gp.mu, NULL);
  pthread_mutex_lock(&gp.mu);

  gp.dead = false;
  gp.listener = NULL;
  gp.buttons = GAMEPAD_DEFAULT;

  if ((gp.handle = gamepad_open(&gp.endpoint)) == NULL) {
    fprintf(stderr, "Did not find a gamepad\n");
    exit(1);
  }

  if ((error = pthread_create(&gp.tid, NULL, &gamepad_worker, NULL)) != 0) {
    fprintf(stderr, "Gamepad worker could not be created: %s\n",
            strerror(error));
    exit(1);
  }

  pthread_mutex_unlock(&gp.mu);

  printf("Gamepad initialized\n");
}

void gamepad_destroy() {
  pthread_mutex_lock(&gp.mu);
  gp.dead = true;
  pthread_mutex_unlock(&gp.mu);

  pthread_join(gp.tid, NULL);
  pthread_mutex_destroy(&gp.mu);

  printf("Gamepad destroyed\n");
}

void gamepad_set_listener(void (*listener)(gamepad_button_event_t,
                                           gamepad_button_t)) {
  pthread_mutex_lock(&gp.mu);
  gp.listener = listener;
  pthread_mutex_unlock(&gp.mu);

  printf("Set gamepad listener\n");
}

void *gamepad_worker(void *arg) {
  struct gamepad_packet packet;
  gamepad_button_t buttons;
  int transferred;

  /* Handle button data */
  while (true) {
    pthread_mutex_lock(&gp.mu);

    /* exit worker if dead */
    if (gp.dead) {
      pthread_mutex_unlock(&gp.mu);
      break;
    }

    /* retrieve */
    libusb_interrupt_transfer(gp.handle, gp.endpoint, (unsigned char *)&packet,
                              sizeof(packet), &transferred, 0);
    buttons = gamepad_decode_packet(packet);

    /* process */
    gamepad_generate_events(gp.buttons, buttons);
    gp.buttons = buttons;

    pthread_mutex_unlock(&gp.mu);
    usleep(100);
  }

  printf("Gamepad worker exited\n");
  return NULL;
}

void gamepad_generate_events(gamepad_button_t prev, gamepad_button_t next) {
  /* no need to generate event if no one cares */
  if (gp.listener == NULL)
    return;

  /* 1. KEY_DOWN */
  if (next & GAMEPAD_LEFT)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_LEFT);
  if (next & GAMEPAD_RIGHT)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_RIGHT);
  if (next & GAMEPAD_UP)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_UP);
  if (next & GAMEPAD_DOWN)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_DOWN);
  if (next & GAMEPAD_X)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_X);
  if (next & GAMEPAD_Y)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_LEFT);
  if (next & GAMEPAD_A)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_A);
  if (next & GAMEPAD_B)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_B);
  if (next & GAMEPAD_L)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_L);
  if (next & GAMEPAD_R)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_R);
  if (next & GAMEPAD_SELECT)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_SELECT);
  if (next & GAMEPAD_START)
    gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_START);

  /* 2. KEY_UP */
  if (!(next & GAMEPAD_LEFT) && (prev & GAMEPAD_LEFT))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_LEFT);
  if (!(next & GAMEPAD_RIGHT) && (prev & GAMEPAD_RIGHT))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_RIGHT);
  if (!(next & GAMEPAD_UP) && (prev & GAMEPAD_UP))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_UP);
  if (!(next & GAMEPAD_DOWN) && (prev & GAMEPAD_DOWN))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_DOWN);
  if (!(next & GAMEPAD_X) && (prev & GAMEPAD_X))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_X);
  if (!(next & GAMEPAD_Y) && (prev & GAMEPAD_Y))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_LEFT);
  if (!(next & GAMEPAD_A) && (prev & GAMEPAD_A))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_A);
  if (!(next & GAMEPAD_B) && (prev & GAMEPAD_B))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_B);
  if (!(next & GAMEPAD_L) && (prev & GAMEPAD_L))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_L);
  if (!(next & GAMEPAD_R) && (prev & GAMEPAD_R))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_R);
  if (!(next & GAMEPAD_SELECT) && (prev & GAMEPAD_SELECT))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_SELECT);
  if (!(next & GAMEPAD_START) && (prev & GAMEPAD_START))
    gp.listener(GAMEPAD_KEY_UP, GAMEPAD_START);
}

struct libusb_device_handle *gamepad_open(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *handle = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if (libusb_init(NULL) < 0) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
     the keyboard protocol */

  for (d = 0; d < num_devs; d++) {
    libusb_device *dev = devs[d];
    if (libusb_get_device_descriptor(dev, &desc) < 0) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
    }

    if (desc.idVendor == GAMEPAD_ID_VENDOR &&
        desc.idProduct == GAMEPAD_ID_PRODUCT) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);

      for (i = 0; i < config->bNumInterfaces; i++) {
        for (k = 0; k < config->interface[i].num_altsetting; k++) {
          int r;
```

```c
          const struct libusb_interface_descriptor *inter =
              config->interface[i].altsetting + k;
          if ((r = libusb_open(dev, &handle)) != 0) {
            fprintf(stderr, "Error: libusb_open failed: %d\n", r);
            exit(1);
          }
          if (libusb_kernel_driver_active(handle, i)) {
            libusb_detach_kernel_driver(handle, i);
          }
          libusb_set_auto_detach_kernel_driver(handle, i);
          if ((r = libusb_claim_interface(handle, i)) != 0) {
            fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
            exit(1);
          }
          *endpoint_address = inter->endpoint[0].bEndpointAddress;
          goto found;
        }
      }
    }
  }

found:
  libusb_free_device_list(devs, 1);

  return handle;
}

gamepad_button_t gamepad_decode_packet(struct gamepad_packet packet) {
  gamepad_button_t buttons = 0;

  if (packet.dir_x == 0x00)
    buttons |= GAMEPAD_LEFT;
  if (packet.dir_x == 0xff)
    buttons |= GAMEPAD_RIGHT;

  if (packet.dir_y == 0x00)
    buttons |= GAMEPAD_UP;
  if (packet.dir_y == 0xff)
    buttons |= GAMEPAD_DOWN;

  if (packet.primary & (1 << 7))
    buttons |= GAMEPAD_Y;
  if (packet.primary & (1 << 6))
    buttons |= GAMEPAD_B;
  if (packet.primary & (1 << 5))
    buttons |= GAMEPAD_A;
  if (packet.primary & (1 << 4))
    buttons |= GAMEPAD_X;

  if (packet.secondary & (1 << 5))
    buttons |= GAMEPAD_START;
  if (packet.secondary & (1 << 4))
    buttons |= GAMEPAD_SELECT;
  if (packet.secondary & (1 << 1))
    buttons |= GAMEPAD_R;
  if (packet.secondary & (1 << 0))
    buttons |= GAMEPAD_L;

  return buttons;
}
```

Listing 9: gamepad.c

```c
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>
#include <linux/types.h>

typedef struct {
  u8 table;
  u16 addr;
  u8 data;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)

#endif
```

Listing 10: vga_ball.h

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.c
 */

#include "vga_ball.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "vga_ball"

/*
 * Information about our device
 */
struct vga_ball_dev {
  struct resource res;    /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void vga_ball_write(vga_ball_arg_t *arg) {
  u32 command = (((u32)(arg->data)) << 24) | (((u32)(arg->addr)) << 2) |
                (((u32)(arg->table)) & 0x3);
  iowrite32(command, dev.virtbase);
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
```

```c
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd,
                           unsigned long arg) {
  vga_ball_arg_t vla;

  switch (cmd) {
  case VGA_BALL_WRITE:
    if (copy_from_user(&vla, (vga_ball_arg_t *)arg, sizeof(vga_ball_arg_t)))
      return -EACCES;
    vga_ball_write(&vla);
    break;

  default:
    return -EINVAL;
  }

  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DRIVER_NAME,
    .fops = &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev) {
  int ret;

  /* Register ourselves as a misc device: creates /dev/vga_ball */
  ret = misc_register(&vga_ball_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
  if (ret) {
    ret = -ENOENT;
    goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(dev.res.start, resource_size(&dev.res), DRIVER_NAME) ==
      NULL) {
    ret = -EBUSY;
    goto out_deregister;
  }

  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
    ret = -ENOMEM;
    goto out_release_mem_region;
  }

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&vga_ball_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev) {
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&vga_ball_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
    {.compatible = "csee4840,vga_ball-1.0"},
    {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
    .driver =
        {
            .name = DRIVER_NAME,
            .owner = THIS_MODULE,
            .of_match_table = of_match_ptr(vga_ball_of_match),
        },
    .remove = __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void) {
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void) {
  platform_driver_unregister(&vga_ball_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

Listing 11: vga_ball.c

```c
#ifndef _VGA_BALL_USER_H
#define _VGA_BALL_USER_H

#include <sys/ioctl.h>
#include <stdint.h>

#define PATTERN_NAME_TABLE 0
#define PATTERN_GENERATOR_TABLE 1
#define SPRITE_ATTRIBUTE_TABLE 2
#define SPRITE_GENERATOR_TABLE 3

typedef struct {
  uint8_t table;
  uint16_t addr;
  uint8_t data;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

#define VGA_BALL_WRITE _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
```

```

void vga_ball_init();

void vga_ball_write(vga_ball_arg_t *arg);

#endif
```

Listing 12: vga_ball_user.h

```
#include "vga_ball_user.h"
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

static const char filename[] = "/dev/vga_ball";
static int vga_ball_fd;

void vga_ball_init() {
  if ((vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    exit(-1);
  }
}

void vga_ball_write(vga_ball_arg_t *arg) {
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE, arg)) {
    perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
    return;
  }
}
```

Listing 13: vga_ball_user.c

```
#ifndef _PATTERN_H
#define _PATTERN_H

#include <stdint.h>

#define PATTERN_BITMAP_SIZE 32
#define PATTERN_BITMAP_NROW 8
#define PATTERN_BITMAP_NCOL 8

#define PATTERN_NROW 60
#define PATTERN_NCOL 64

#define pattern_pixel(x) ((x)&0xf)

void load_pattern_bitmaps();
void set_pattern_bitmap(int i, const uint8_t *pat);
void set_pattern_at(uint8_t r, uint8_t c, uint8_t name);

typedef enum {
  PAT_BACKGROUND = 0,
  PAT_0,
  PAT_1,
  PAT_2,
  PAT_3,
  PAT_4,
  PAT_5,
  PAT_6,
  PAT_7,
  PAT_8,
  PAT_9,
  PAT_A,
  PAT_B,
  PAT_C,
  PAT_D,
  PAT_E,
  PAT_F,
  PAT_G,
  PAT_H,
  PAT_I,
  PAT_J,
  PAT_K,
  PAT_L,
  PAT_M,
  PAT_N,
  PAT_O,
  PAT_P,
  PAT_Q,
  PAT_R,
  PAT_S,
  PAT_T,
  PAT_U,
  PAT_V,
  PAT_W,
  PAT_X,
  PAT_Y,
  PAT_Z,
  PAT_FOOD_SM,
  PAT_FOOD_LG,
  PAT_WALL_0,
  PAT_WALL_1,
  PAT_WALL_2,
  PAT_WALL_3,
  PAT_WALL_4,
  PAT_WALL_5,
  PAT_WALL_6,
  PAT_WALL_7,
  PAT_WALL_8,
  PAT_WALL_9,
  PAT_WALL_10,
  PAT_WALL_11,
  PAT_WALL_12,
  PAT_WALL_13,
  PAT_WALL_14,
  PAT_WALL_15,
  PAT_WALL_16,
  PAT_WALL_17,
  PAT_WALL_18,
  PAT_WALL_19,
  PAT_WALL_20,
  PAT_WALL_21,
  PAT_WALL_22,
  PAT_WALL_23,
  PAT_GATE,
} pattern_name_t;

#endif
```

Listing 14: pattern.h

```
#include "pattern.h"
#include "color.h"
#include "vga_ball_user.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

const uint8_t pat_background[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_0[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, Transp, Transp, White, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
};

const uint8_t pat_1[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
};

const uint8_t pat_2[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, White},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, White, White, White, White, Transp, Transp},
    {Transp, White, White, White, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
};

const uint8_t pat_3[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_4[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
    {Transp, Transp, Transp, Transp, White, White, White, Transp},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, White, White, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
};

const uint8_t pat_5[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_6[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_7[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
};

const uint8_t pat_8[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, Transp},
    {Transp, White, White, White, Transp, Transp, White, Transp},
    {Transp, Transp, White, White, White, White, Transp, Transp},
    {Transp, White, Transp, Transp, White, White, White, White},
    {Transp, White, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_9[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
    {Transp, Transp, White, White, White, White, Transp, Transp},
};

const uint8_t pat_A[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, White, White, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_B[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, White, White, White, Transp},
};

const uint8_t pat_C[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, Transp, Transp, White, White, White, White, Transp},
};

const uint8_t pat_D[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, Transp, Transp},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, White, White, White, Transp, Transp},
};

const uint8_t pat_E[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
};

const uint8_t pat_F[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_G[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, White, White, White, White, White},
    {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, Transp, Transp, White, White, White, White, White},
};

const uint8_t pat_H[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_I[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
};

const uint8_t pat_J[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_K[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, Transp, White, White, Transp, Transp},
    {Transp, White, White, White, White, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, Transp, Transp},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_L[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
};

const uint8_t pat_M[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, Transp, White, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, White, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_N[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, Transp, Transp, White, White},
    {Transp, White, White, White, White, Transp, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, Transp, White, White, White, White},
    {Transp, White, White, Transp, Transp, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_O[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_P[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
};

```

```
const uint8_t pat_Q[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, White, White, White, White},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, Transp, White, White, White, White, Transp, White},
};

const uint8_t pat_R[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, White, White, White},
    {Transp, White, White, White, White, White, Transp, Transp},
    {Transp, White, White, Transp, White, White, White, Transp},
    {Transp, White, White, Transp, Transp, White, White, White},
};

const uint8_t pat_S[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, Transp, Transp},
    {Transp, White, White, Transp, Transp, White, White, Transp},
    {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_T[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
};

const uint8_t pat_U[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
};

const uint8_t pat_V[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, Transp, White, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, Transp, Transp, Transp},
};

const uint8_t pat_W[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, Transp, White, Transp, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, White, White, White, White, White},
    {Transp, White, White, White, Transp, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_X[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, Transp, Transp, Transp, White, White},
    {Transp, White, White, White, Transp, White, White, White},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, White, White, White, White, White, Transp},
    {Transp, White, White, White, Transp, White, White, White},
    {Transp, White, White, Transp, Transp, Transp, White, White},
};

const uint8_t pat_Y[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, Transp, White, White, Transp, Transp, White, White},
    {Transp, Transp, Transp, White, White, White, White, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
};

const uint8_t pat_Z[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
    {Transp, Transp, Transp, Transp, Transp, White, White, White},
    {Transp, Transp, Transp, Transp, White, White, White, Transp},
    {Transp, Transp, Transp, White, White, White, Transp, Transp},
    {Transp, Transp, White, White, White, Transp, Transp, Transp},
    {Transp, White, White, White, Transp, Transp, Transp, Transp},
    {Transp, White, White, White, White, White, White, White},
};

const uint8_t pat_food_sm[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Salmon, Salmon, Transp, Transp, Transp},
    {Transp, Transp, Transp, Salmon, Salmon, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_food_lg[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Salmon, Salmon, Salmon, Salmon, Transp, Transp},
    {Transp, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Transp},
    {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
    {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
    {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
    {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
    {Transp, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Transp},
    {Transp, Transp, Salmon, Salmon, Salmon, Salmon, Transp, Transp},
};

const uint8_t pat_wall_0[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};
```

```c

const uint8_t pat_wall_1[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
};

const uint8_t pat_wall_2[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
};

const uint8_t pat_wall_3[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_4[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue},
    {Transp, Transp, Blue, Blue, Transp, Transp, Transp, Transp},
    {Transp, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Blue},
    {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_5[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Blue, Blue, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Blue, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
    {Blue, Blue, Blue, Transp, Transp, Transp, Blue, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
};

const uint8_t pat_wall_6[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
    {Blue, Blue, Blue, Transp, Transp, Transp, Blue, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
    {Transp, Transp, Transp, Transp, Blue, Blue, Transp, Transp},
    {Blue, Blue, Blue, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_7[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Blue},
    {Transp, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Blue, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue},
};

const uint8_t pat_wall_8[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_9[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Blue, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_10[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Blue, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
};

const uint8_t pat_wall_11[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_12[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
};

const uint8_t pat_wall_13[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_14[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
    {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
};

const uint8_t pat_wall_15[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
    {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
};

const uint8_t pat_wall_16[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
};

const uint8_t pat_wall_17[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
};

const uint8_t pat_wall_18[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
    {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_19[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_20[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
};

const uint8_t pat_wall_21[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_22[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
};

const uint8_t pat_wall_23[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};
//
// const uint8_t pat_wall_24[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
//     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
//     {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
// };
//
// const uint8_t pat_wall_25[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
//     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Blue, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
// };
//
// const uint8_t pat_wall_26[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
//     {Transp, Transp, Transp, Transp, Transp, Blue, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
// };
//
// const uint8_t pat_wall_27[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
//     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
// };
//
const uint8_t pat_gate[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t *patterns[] = {
    (uint8_t *)pat_background, (uint8_t *)pat_0,        (uint8_t *)pat_1,
    (uint8_t *)pat_2,          (uint8_t *)pat_3,        (uint8_t *)pat_4,
    (uint8_t *)pat_5,          (uint8_t *)pat_6,        (uint8_t *)pat_7,
    (uint8_t *)pat_8,          (uint8_t *)pat_9,        (uint8_t *)pat_A,
    (uint8_t *)pat_B,          (uint8_t *)pat_C,        (uint8_t *)pat_D,
    (uint8_t *)pat_E,          (uint8_t *)pat_F,        (uint8_t *)pat_G,
    (uint8_t *)pat_H,          (uint8_t *)pat_I,        (uint8_t *)pat_J,
    (uint8_t *)pat_K,          (uint8_t *)pat_L,        (uint8_t *)pat_M,
    (uint8_t *)pat_N,          (uint8_t *)pat_O,        (uint8_t *)pat_P,
    (uint8_t *)pat_Q,          (uint8_t *)pat_R,        (uint8_t *)pat_S,
    (uint8_t *)pat_T,          (uint8_t *)pat_U,        (uint8_t *)pat_V,
    (uint8_t *)pat_W,          (uint8_t *)pat_X,        (uint8_t *)pat_Y,
    (uint8_t *)pat_Z,          (uint8_t *)pat_food_sm,  (uint8_t *)pat_food_lg,
    (uint8_t *)pat_wall_0,     (uint8_t *)pat_wall_1,   (uint8_t *)pat_wall_2,
    (uint8_t *)pat_wall_3,     (uint8_t *)pat_wall_4,   (uint8_t *)pat_wall_5,
    (uint8_t *)pat_wall_6,     (uint8_t *)pat_wall_7,   (uint8_t *)pat_wall_8,
    (uint8_t *)pat_wall_9,     (uint8_t *)pat_wall_10,  (uint8_t *)pat_wall_11,
    (uint8_t *)pat_wall_12,    (uint8_t *)pat_wall_13,  (uint8_t *)pat_wall_14,
    (uint8_t *)pat_wall_15,    (uint8_t *)pat_wall_16,  (uint8_t *)pat_wall_17,
    (uint8_t *)pat_wall_18,    (uint8_t *)pat_wall_19,  (uint8_t *)pat_wall_20,
    (uint8_t *)pat_wall_21,    (uint8_t *)pat_wall_22,  (uint8_t *)pat_wall_23,
    (uint8_t *)pat_gate,
};

void load_pattern_bitmaps() {
  for (int i = 0; i < sizeof(patterns) / sizeof(const uint8_t *); i++) {
    const uint8_t *pat = patterns[i];
    set_pattern_bitmap(i, pat);
  }
}

void set_pattern_bitmap(int pati, const uint8_t *pat) {
  vga_ball_arg_t arg;
  int start;

  arg.table = PATTERN_GENERATOR_TABLE;
  start = pati * PATTERN_BITMAP_SIZE;
  for (int i = 0; i < PATTERN_BITMAP_SIZE; i++) {
    arg.addr = start + i;
    arg.data = pattern_pixel(pat[2 * i]) << 4 | pattern_pixel(pat[2 * i + 1]);
    vga_ball_write(&arg);
  }
}

void set_pattern_at(uint8_t r, uint8_t c, uint8_t name) {
```

```c
  if (r >= PATTERN_NROW) {
    fprintf(stderr, "Row %d is too large\n", r);
    exit(-1);
  }

  if (c >= PATTERN_NCOL) {
    fprintf(stderr, "Column %d is too large\n", r);
    exit(-1);
  }

  vga_ball_arg_t arg;

  arg.table = PATTERN_NAME_TABLE;
  arg.addr = r * PATTERN_NCOL + c;
  arg.data = name;

  vga_ball_write(&arg);
}
```

Listing 15: pattern.c

```c
#ifndef _SPRITE_H
#define _SPRITE_H

#include <stdint.h>

typedef struct {
  uint8_t i;
  uint16_t y;
  uint16_t x;
  uint8_t name;
} sprite_attr_t;

#define SPRITE_BITMAP_SIZE 128
#define SPRITE_BITMAP_NROW 16
#define SPRITE_BITMAP_NCOL 16
#define sprite_pixel(x) ((x)&0xf)

void load_sprite_bitmaps();
void set_sprite_bitmap(int spriti, const uint8_t *sprite);
void set_sprite(sprite_attr_t attr);

typedef enum {
  SPRITE_PACMAN_CLOSED = 0,
  SPRITE_PACMAN_LEFT,
  SPRITE_PACMAN_RIGHT,
  SPRITE_PACMAN_UP,
  SPRITE_PACMAN_DOWN,
  SPRITE_GHOST_RED,
  SPRITE_GHOST_CYAN,
  SPRITE_GHOST_PINK,
  SPRITE_GHOST_ORANGE,
  SPRITE_GHOST_SCATTER,
} sprite_name_t;

#endif
```

Listing 16: sprite.h

```c
#include "sprite.h"
#include "color.h"
#include "vga_ball_user.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

const uint8_t sprite_pacman_closed[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_pacman_left[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_pacman_right[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow,
     Yellow, Yellow, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow,
     Yellow, Yellow, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_pacman_up[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Transp, Transp, Transp, Transp,
     Transp, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp,
     Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_pacman_down[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
     Yellow, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp,
     Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp},
    {Transp, Transp, Transp, Yellow, Yellow, Transp, Transp, Transp, Transp,
     Transp, Yellow, Yellow, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_ghost_red[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Red, Red, Red, Red, Transp,
     Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp, Transp},
    {Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp},
    {Transp, Transp, Red, Red, White, White, Red, Red, Red, Red, White, White,
     Red, Red, Transp, Transp},
    {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
     White, White, Red, Red, Transp},
    {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
     White, White, Red, Red, Transp},
    {Transp, Red, Red, White, Blue, Blue, White, Red, Red, White, Blue, Blue,
     White, Red, Red, Transp},
    {Transp, Red, Red, Red, Blue, Blue, Red, Red, Red, Red, Blue, Blue, Red,
     Red, Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Transp, Red, Red, Red, Transp, Transp, Red, Red, Red,
     Transp, Red, Red, Transp},
    {Transp, Red, Transp, Transp, Transp, Red, Red, Transp, Transp, Red, Red,
     Transp, Transp, Transp, Red, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_ghost_orange[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Orange, Orange, Orange,
     Orange, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Orange, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Orange, Transp, Transp, Transp},
    {Transp, Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Orange, Transp, Transp},
    {Transp, Transp, Orange, Orange, White, White, Orange, Orange, Orange,
     Orange, White, White, Orange, Orange, Transp, Transp},
    {Transp, Orange, Orange, White, White, White, White, Orange, Orange, White,
     White, White, White, Orange, Orange, Transp},
    {Transp, Orange, Orange, White, White, White, White, Orange, Orange, White,
     White, White, White, Orange, Orange, Transp},
    {Transp, Orange, Orange, White, Blue, Blue, White, Orange, Orange, White,
     Blue, Blue, White, Orange, Orange, Transp},
    {Transp, Orange, Orange, Orange, Blue, Blue, Orange, Orange, Orange, Orange,
     Blue, Blue, Orange, Orange, Orange, Transp},
    {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Orange, Orange, Orange, Transp},
    {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Orange, Orange, Orange, Transp},
    {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
     Orange, Orange, Orange, Orange, Orange, Orange, Transp},
    {Transp, Orange, Orange, Transp, Orange, Orange, Orange, Transp, Transp,
     Orange, Orange, Orange, Transp, Orange, Orange, Transp},
    {Transp, Orange, Transp, Transp, Transp, Orange, Orange, Transp, Transp,
     Orange, Orange, Transp, Transp, Transp, Orange, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_ghost_cyan[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan,
     Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Cyan, Transp, Transp, Transp},
    {Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Cyan, Transp, Transp},
    {Transp, Transp, Cyan, Cyan, White, White, Cyan, Cyan, Cyan, Cyan, White,
     White, Cyan, Cyan, Transp, Transp},
    {Transp, Cyan, Cyan, White, White, White, White, Cyan, Cyan, White, White,
     White, White, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, White, White, White, White, Cyan, Cyan, White, White,
     White, White, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, White, Blue, Blue, White, Cyan, Cyan, White, Blue,
     Blue, White, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, Cyan, Blue, Blue, Cyan, Cyan, Cyan, Cyan, Blue, Blue,
     Cyan, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
     Cyan, Cyan, Cyan, Transp},
    {Transp, Cyan, Cyan, Transp, Cyan, Cyan, Cyan, Transp, Transp, Cyan, Cyan,
     Cyan, Transp, Cyan, Cyan, Transp},
    {Transp, Cyan, Transp, Transp, Transp, Cyan, Cyan, Transp, Transp, Cyan,
     Cyan, Transp, Transp, Transp, Cyan, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_ghost_pink[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Pink, Pink, Pink, Pink,
     Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Pink, Transp, Transp, Transp},
    {Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Pink, Transp, Transp},
    {Transp, Transp, Pink, Pink, White, White, Pink, Pink, Pink, Pink, White,
     White, Pink, Pink, Transp, Transp},
    {Transp, Pink, Pink, White, White, White, White, Pink, Pink, White, White,
     White, White, Pink, Pink, Transp},
    {Transp, Pink, Pink, White, White, White, White, Pink, Pink, White, White,
     White, White, Pink, Pink, Transp},
    {Transp, Pink, Pink, White, Blue, Blue, White, Pink, Pink, White, Blue,
     Blue, White, Pink, Pink, Transp},
    {Transp, Pink, Pink, Pink, Blue, Blue, Pink, Pink, Pink, Pink, Blue, Blue,
     Pink, Pink, Pink, Transp},
    {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Pink, Pink, Transp},
    {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Pink, Pink, Transp},
    {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
     Pink, Pink, Pink, Transp},
    {Transp, Pink, Pink, Transp, Pink, Pink, Pink, Transp, Transp, Pink, Pink,
     Pink, Transp, Pink, Pink, Transp},
    {Transp, Pink, Transp, Transp, Transp, Pink, Pink, Transp, Transp, Pink,
     Pink, Transp, Transp, Transp, Pink, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t sprite_ghost_scatter[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue,
     Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Transp, Transp, Transp},
    {Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Transp, Transp},
    {Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Transp, Transp},
    {Transp, Blue, Blue, Blue, Blue, White, White, Blue, Blue, White, White,
     Blue, Blue, Blue, Blue, Transp},
    {Transp, Blue, Blue, Blue, Blue, White, White, Blue, Blue, White, White,
     Blue, Blue, Blue, Blue, Transp},
    {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Blue, Transp},
    {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Blue, Transp},
    {Transp, Blue, Blue, White, White, Blue, Blue, White, White, Blue, Blue,
     White, White, Blue, Blue, Transp},
    {Transp, Blue, White, Blue, Blue, White, White, Blue, Blue, White, White,
     Blue, Blue, White, Blue, Transp},
    {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
     Blue, Blue, Blue, Transp},
    {Transp, Blue, Blue, Transp, Blue, Blue, Blue, Transp, Transp, Blue, Blue,
     Blue, Transp, Blue, Blue, Transp},
    {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Transp, Transp, Blue,
     Blue, Transp, Transp, Transp, Blue, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};

const uint8_t *sprites[] = {
    (uint8_t *)sprite_pacman_closed, (uint8_t *)sprite_pacman_left,
    (uint8_t *)sprite_pacman_right,  (uint8_t *)sprite_pacman_up,
    (uint8_t *)sprite_pacman_down,   (uint8_t *)sprite_ghost_red,
    (uint8_t *)sprite_ghost_cyan,    (uint8_t *)sprite_ghost_pink,
    (uint8_t *)sprite_ghost_orange,  (uint8_t *)sprite_ghost_scatter,
};

void load_sprite_bitmaps() {
  for (int i = 0; i < sizeof(sprites) / sizeof(const uint8_t *); i++) {
    const uint8_t *pat = sprites[i];
    set_sprite_bitmap(i, pat);
  }
}

void set_sprite_bitmap(int spritei, const uint8_t *pat) {
  vga_ball_arg_t arg;
  int start;

  arg.table = SPRITE_GENERATOR_TABLE;
  start = spritei * SPRITE_BITMAP_SIZE;
  for (int i = 0; i < SPRITE_BITMAP_SIZE; i++) {
    arg.addr = start + i;
    arg.data = sprite_pixel(pat[2 * i]) << 4 | sprite_pixel(pat[2 * i + 1]);
    vga_ball_write(&arg);
  }
}

void set_sprite(sprite_attr_t attr) {
  vga_ball_arg_t arg;
  int start;

  start = 4 * attr.i;
  arg.table = SPRITE_ATTRIBUTE_TABLE;

  arg.addr = start;
  arg.data = (uint8_t)(attr.y / 2);
  vga_ball_write(&arg);

  arg.addr = start + 1;
  arg.data = (uint8_t)(attr.x / 2);
  vga_ball_write(&arg);

  arg.addr = start + 2;
  arg.data = attr.name;
  vga_ball_write(&arg);
}
```

Listing 17: sprite.c

```
#ifndef _MAP_H
#define _MAP_H

#include <stdint.h>

#define MAP_NROW 36
#define MAP_NCOL 28

#define MAP_ROW_OFFSET 12
#define MAP_COL_OFFSET 16

void clear_screen();
void set_map_at(int r, int c, uint8_t name);
uint8_t get_map_at(int r, int c);
void set_local_map_at(int r, int c, uint8_t name);
void setup_map();
void setup_map_foods();
void setup_map_score();
void set_map_best_score(uint32_t score);
void set_map_player_score(uint32_t score);
void set_map_lives(uint8_t lives);

#endif
```

Listing 18: map.h

```
#include "map.h"
#include "pattern.h"
#include "sprite.h"

// static uint8_t map[31][28];
```

```c

static uint8_t map[PATTERN_NROW][PATTERN_NCOL];

void clear_screen() {
  int r, c;
  for (r = 0; r < PATTERN_NROW; r++) {
    for (c = 0; c < PATTERN_NCOL; c++) {
      set_map_at(r, c, PAT_BACKGROUND);
    }
  }
}

void set_map_best_score(uint32_t score) {
  // clear
  int i;
  for (i = 22; i <= 27; i++) {
    set_local_map_at(1, i, PAT_BACKGROUND);
  }

  // digits
  for (i = 27; i >= 22; i--) {
    set_local_map_at(1, i, PAT_0 + score % 10);
    score /= 10;
    if (score == 0)
      return;
  }
}

void set_map_player_score(uint32_t score) {
  // clear
  int i;
  for (i = 0; i <= 5; i++) {
    set_local_map_at(1, i, PAT_BACKGROUND);
  }

  // digits
  for (i = 5; i >= 0; i--) {
    set_local_map_at(1, i, PAT_0 + score % 10);
    score /= 10;
    if (score == 0)
      return;
  }
}

void set_map_lives(uint8_t lives) {
  set_local_map_at(35, 0, PAT_L);
  set_local_map_at(35, 1, PAT_I);
  set_local_map_at(35, 2, PAT_V);
  set_local_map_at(35, 3, PAT_E);
  set_local_map_at(35, 4, PAT_S);

  set_local_map_at(35, 6, PAT_0 + lives);
}

void set_map_at(int r, int c, uint8_t name) {
  map[r][c] = name;
  set_pattern_at(r, c, name);
}

uint8_t get_map_at(int r, int c) { return map[r][c]; }

void set_local_map_at(int r, int c, uint8_t name) {
  set_map_at(MAP_ROW_OFFSET + r, MAP_COL_OFFSET + c, name);
}

void setup_map_score() {
  set_local_map_at(0, 0, PAT_P);
  set_local_map_at(0, 1, PAT_L);
  set_local_map_at(0, 2, PAT_A);
  set_local_map_at(0, 3, PAT_Y);
  set_local_map_at(0, 4, PAT_E);
  set_local_map_at(0, 5, PAT_R);

  set_local_map_at(0, 18, PAT_H);
  set_local_map_at(0, 19, PAT_I);
  set_local_map_at(0, 20, PAT_G);
  set_local_map_at(0, 21, PAT_H);

  set_local_map_at(0, 23, PAT_S);
  set_local_map_at(0, 24, PAT_C);
  set_local_map_at(0, 25, PAT_O);
  set_local_map_at(0, 26, PAT_R);
  set_local_map_at(0, 27, PAT_E);
}

void setup_map() {
  int r;
  int i;

  // row 3
  r = 3;
  set_local_map_at(r, 0, PAT_WALL_4);
  for (i = 1; i < 13; i++)
    set_local_map_at(3, i, PAT_WALL_0);
  set_local_map_at(r, 13, PAT_WALL_12);
  set_local_map_at(r, 14, PAT_WALL_13);
  for (i = 15; i < 27; i++)
    set_local_map_at(3, i, PAT_WALL_0);
  set_local_map_at(r, 27, PAT_WALL_5);

  // row 4
  r = 4;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 5
  r = 5;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 2, PAT_WALL_10);
  set_local_map_at(r, 3, PAT_WALL_21);
  set_local_map_at(r, 4, PAT_WALL_21);
  set_local_map_at(r, 5, PAT_WALL_11);

  set_local_map_at(r, 7, PAT_WALL_10);
  set_local_map_at(r, 8, PAT_WALL_21);
  set_local_map_at(r, 9, PAT_WALL_21);
  set_local_map_at(r, 10, PAT_WALL_21);
  set_local_map_at(r, 11, PAT_WALL_11);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 16, PAT_WALL_10);
  set_local_map_at(r, 17, PAT_WALL_21);
  set_local_map_at(r, 18, PAT_WALL_21);
  set_local_map_at(r, 19, PAT_WALL_21);
  set_local_map_at(r, 20, PAT_WALL_11);

  set_local_map_at(r, 22, PAT_WALL_10);
  set_local_map_at(r, 23, PAT_WALL_21);
  set_local_map_at(r, 24, PAT_WALL_21);
  set_local_map_at(r, 25, PAT_WALL_11);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 6
  r = 6;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 2, PAT_WALL_20);
  set_local_map_at(r, 5, PAT_WALL_22);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 11, PAT_WALL_22);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 16, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_20);
  set_local_map_at(r, 25, PAT_WALL_22);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 7
  r = 7;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 2, PAT_WALL_9);
  set_local_map_at(r, 3, PAT_WALL_23);
  set_local_map_at(r, 4, PAT_WALL_23);
  set_local_map_at(r, 5, PAT_WALL_8);

  set_local_map_at(r, 7, PAT_WALL_9);
  set_local_map_at(r, 8, PAT_WALL_23);
  set_local_map_at(r, 9, PAT_WALL_23);
  set_local_map_at(r, 10, PAT_WALL_23);
  set_local_map_at(r, 11, PAT_WALL_8);

  set_local_map_at(r, 13, PAT_WALL_9);
  set_local_map_at(r, 14, PAT_WALL_8);

  set_local_map_at(r, 16, PAT_WALL_9);
  set_local_map_at(r, 17, PAT_WALL_23);
  set_local_map_at(r, 18, PAT_WALL_23);
  set_local_map_at(r, 19, PAT_WALL_23);
  set_local_map_at(r, 20, PAT_WALL_8);

  set_local_map_at(r, 22, PAT_WALL_9);
  set_local_map_at(r, 23, PAT_WALL_23);
  set_local_map_at(r, 24, PAT_WALL_23);
  set_local_map_at(r, 25, PAT_WALL_8);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 8
  r = 8;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 9
  r = 9;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 2, PAT_WALL_10);
  set_local_map_at(r, 3, PAT_WALL_21);
  set_local_map_at(r, 4, PAT_WALL_21);
  set_local_map_at(r, 5, PAT_WALL_11);

  set_local_map_at(r, 7, PAT_WALL_10);
  set_local_map_at(r, 8, PAT_WALL_11);

  set_local_map_at(r, 10, PAT_WALL_10);
  set_local_map_at(r, 11, PAT_WALL_21);
  set_local_map_at(r, 12, PAT_WALL_21);
  set_local_map_at(r, 13, PAT_WALL_21);
  set_local_map_at(r, 14, PAT_WALL_21);
  set_local_map_at(r, 15, PAT_WALL_21);
  set_local_map_at(r, 16, PAT_WALL_21);
  set_local_map_at(r, 17, PAT_WALL_11);

  set_local_map_at(r, 19, PAT_WALL_10);
  set_local_map_at(r, 20, PAT_WALL_11);

  set_local_map_at(r, 22, PAT_WALL_10);
  set_local_map_at(r, 23, PAT_WALL_21);
  set_local_map_at(r, 24, PAT_WALL_21);
  set_local_map_at(r, 25, PAT_WALL_11);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 10
  r = 10;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 2, PAT_WALL_9);
  set_local_map_at(r, 3, PAT_WALL_23);
  set_local_map_at(r, 4, PAT_WALL_23);
  set_local_map_at(r, 5, PAT_WALL_8);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 10, PAT_WALL_9);
  set_local_map_at(r, 11, PAT_WALL_23);
  set_local_map_at(r, 12, PAT_WALL_23);
  set_local_map_at(r, 13, PAT_WALL_11);
  set_local_map_at(r, 14, PAT_WALL_10);
  set_local_map_at(r, 15, PAT_WALL_23);
  set_local_map_at(r, 16, PAT_WALL_23);
  set_local_map_at(r, 17, PAT_WALL_8);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_9);
  set_local_map_at(r, 23, PAT_WALL_23);
  set_local_map_at(r, 24, PAT_WALL_23);
  set_local_map_at(r, 25, PAT_WALL_8);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 11
  r = 11;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 12
  r = 12;
  set_local_map_at(r, 0, PAT_WALL_7);
  set_local_map_at(r, 1, PAT_WALL_2);
  set_local_map_at(r, 2, PAT_WALL_2);
  set_local_map_at(r, 3, PAT_WALL_2);
  set_local_map_at(r, 4, PAT_WALL_2);
  set_local_map_at(r, 5, PAT_WALL_11);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_9);
  set_local_map_at(r, 9, PAT_WALL_21);
  set_local_map_at(r, 10, PAT_WALL_21);
  set_local_map_at(r, 11, PAT_WALL_11);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 16, PAT_WALL_10);
  set_local_map_at(r, 17, PAT_WALL_21);
  set_local_map_at(r, 18, PAT_WALL_21);
  set_local_map_at(r, 19, PAT_WALL_8);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_10);
  set_local_map_at(r, 23, PAT_WALL_2);
  set_local_map_at(r, 24, PAT_WALL_2);
  set_local_map_at(r, 25, PAT_WALL_2);
  set_local_map_at(r, 26, PAT_WALL_2);
  set_local_map_at(r, 27, PAT_WALL_6);

  // row 13
  r = 13;

  set_local_map_at(r, 5, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_10);
  set_local_map_at(r, 9, PAT_WALL_23);
  set_local_map_at(r, 10, PAT_WALL_23);
  set_local_map_at(r, 11, PAT_WALL_8);

  set_local_map_at(r, 13, PAT_WALL_9);
  set_local_map_at(r, 14, PAT_WALL_8);

  set_local_map_at(r, 16, PAT_WALL_9);
  set_local_map_at(r, 17, PAT_WALL_23);
  set_local_map_at(r, 18, PAT_WALL_23);
  set_local_map_at(r, 19, PAT_WALL_11);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_1);

  // row 14
  r = 14;
  set_local_map_at(r, 5, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_1);

  // row 15
  r = 15;
  set_local_map_at(r, 5, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 10, PAT_WALL_10);
  set_local_map_at(r, 11, PAT_WALL_2);
  set_local_map_at(r, 12, PAT_WALL_2);
  set_local_map_at(r, 13, PAT_GATE);
  set_local_map_at(r, 14, PAT_GATE);
  set_local_map_at(r, 15, PAT_WALL_2);
  set_local_map_at(r, 16, PAT_WALL_2);
  set_local_map_at(r, 17, PAT_WALL_11);
```

```

set_local_map_at(r, 19, PAT_WALL_20);
set_local_map_at(r, 20, PAT_WALL_22);

set_local_map_at(r, 22, PAT_WALL_1);

// row 16
r = 16;
set_local_map_at(r, 5, PAT_WALL_3);

set_local_map_at(r, 7, PAT_WALL_9);
set_local_map_at(r, 8, PAT_WALL_8);

set_local_map_at(r, 10, PAT_WALL_1);

set_local_map_at(r, 17, PAT_WALL_3);

set_local_map_at(r, 19, PAT_WALL_9);
set_local_map_at(r, 20, PAT_WALL_8);

set_local_map_at(r, 22, PAT_WALL_1);

// row 17
r = 17;
set_local_map_at(r, 5, PAT_WALL_3);

set_local_map_at(r, 10, PAT_WALL_1);

set_local_map_at(r, 17, PAT_WALL_3);

set_local_map_at(r, 22, PAT_WALL_1);

// row 18
r = 18;
set_local_map_at(r, 5, PAT_WALL_3);

set_local_map_at(r, 7, PAT_WALL_10);
set_local_map_at(r, 8, PAT_WALL_11);

set_local_map_at(r, 10, PAT_WALL_1);

set_local_map_at(r, 17, PAT_WALL_3);

set_local_map_at(r, 19, PAT_WALL_10);
set_local_map_at(r, 20, PAT_WALL_11);

set_local_map_at(r, 22, PAT_WALL_1);

// row 19
r = 19;
set_local_map_at(r, 5, PAT_WALL_3);

set_local_map_at(r, 7, PAT_WALL_20);
set_local_map_at(r, 8, PAT_WALL_22);

set_local_map_at(r, 10, PAT_WALL_9);
set_local_map_at(r, 11, PAT_WALL_0);
set_local_map_at(r, 12, PAT_WALL_0);
set_local_map_at(r, 13, PAT_WALL_0);
set_local_map_at(r, 14, PAT_WALL_0);
set_local_map_at(r, 15, PAT_WALL_0);
set_local_map_at(r, 16, PAT_WALL_0);
set_local_map_at(r, 17, PAT_WALL_8);

set_local_map_at(r, 19, PAT_WALL_20);
set_local_map_at(r, 20, PAT_WALL_22);

set_local_map_at(r, 22, PAT_WALL_1);

// row 20
r = 20;
```

```
  set_local_map_at(r, 5, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_1);

  // row 21
  r = 21;
  set_local_map_at(r, 5, PAT_WALL_3);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 10, PAT_WALL_10);
  set_local_map_at(r, 11, PAT_WALL_21);
  set_local_map_at(r, 12, PAT_WALL_21);
  set_local_map_at(r, 13, PAT_WALL_21);
  set_local_map_at(r, 14, PAT_WALL_21);
  set_local_map_at(r, 15, PAT_WALL_21);
  set_local_map_at(r, 16, PAT_WALL_21);
  set_local_map_at(r, 17, PAT_WALL_11);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_1);

  // row 22
  r = 22;
  set_local_map_at(r, 0, PAT_WALL_4);
  set_local_map_at(r, 1, PAT_WALL_0);
  set_local_map_at(r, 2, PAT_WALL_0);
  set_local_map_at(r, 3, PAT_WALL_0);
  set_local_map_at(r, 4, PAT_WALL_0);
  set_local_map_at(r, 5, PAT_WALL_8);

  set_local_map_at(r, 7, PAT_WALL_9);
  set_local_map_at(r, 8, PAT_WALL_8);

  set_local_map_at(r, 10, PAT_WALL_9);
  set_local_map_at(r, 11, PAT_WALL_23);
  set_local_map_at(r, 12, PAT_WALL_23);
  set_local_map_at(r, 13, PAT_WALL_11);
  set_local_map_at(r, 14, PAT_WALL_10);
  set_local_map_at(r, 15, PAT_WALL_23);
  set_local_map_at(r, 16, PAT_WALL_23);
  set_local_map_at(r, 17, PAT_WALL_8);

  set_local_map_at(r, 19, PAT_WALL_9);
  set_local_map_at(r, 20, PAT_WALL_8);

  set_local_map_at(r, 22, PAT_WALL_9);
  set_local_map_at(r, 23, PAT_WALL_0);
  set_local_map_at(r, 24, PAT_WALL_0);
  set_local_map_at(r, 25, PAT_WALL_0);
  set_local_map_at(r, 26, PAT_WALL_0);
  set_local_map_at(r, 27, PAT_WALL_5);

  // row 23
  r = 23;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 13, PAT_WALL_20);
  set_local_map_at(r, 14, PAT_WALL_22);

  set_local_map_at(r, 27, PAT_WALL_1);

```

```
// row 24
r = 24;
set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 2, PAT_WALL_10);
set_local_map_at(r, 3, PAT_WALL_21);
set_local_map_at(r, 4, PAT_WALL_21);
set_local_map_at(r, 5, PAT_WALL_11);

set_local_map_at(r, 7, PAT_WALL_10);
set_local_map_at(r, 8, PAT_WALL_21);
set_local_map_at(r, 9, PAT_WALL_21);
set_local_map_at(r, 10, PAT_WALL_21);
set_local_map_at(r, 11, PAT_WALL_11);

set_local_map_at(r, 13, PAT_WALL_20);
set_local_map_at(r, 14, PAT_WALL_22);

set_local_map_at(r, 16, PAT_WALL_10);
set_local_map_at(r, 17, PAT_WALL_21);
set_local_map_at(r, 18, PAT_WALL_21);
set_local_map_at(r, 19, PAT_WALL_21);
set_local_map_at(r, 20, PAT_WALL_11);

set_local_map_at(r, 22, PAT_WALL_10);
set_local_map_at(r, 23, PAT_WALL_21);
set_local_map_at(r, 24, PAT_WALL_21);
set_local_map_at(r, 25, PAT_WALL_11);

set_local_map_at(r, 27, PAT_WALL_1);

// row 25
r = 25;

set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 2, PAT_WALL_9);
set_local_map_at(r, 3, PAT_WALL_23);
set_local_map_at(r, 4, PAT_WALL_11);
set_local_map_at(r, 5, PAT_WALL_22);

set_local_map_at(r, 7, PAT_WALL_9);
set_local_map_at(r, 8, PAT_WALL_23);
set_local_map_at(r, 9, PAT_WALL_23);
set_local_map_at(r, 10, PAT_WALL_23);
set_local_map_at(r, 11, PAT_WALL_8);

set_local_map_at(r, 13, PAT_WALL_9);
set_local_map_at(r, 14, PAT_WALL_8);

set_local_map_at(r, 16, PAT_WALL_9);
set_local_map_at(r, 17, PAT_WALL_23);
set_local_map_at(r, 18, PAT_WALL_23);
set_local_map_at(r, 19, PAT_WALL_23);
set_local_map_at(r, 20, PAT_WALL_8);

set_local_map_at(r, 22, PAT_WALL_20);
set_local_map_at(r, 23, PAT_WALL_10);
set_local_map_at(r, 24, PAT_WALL_23);
set_local_map_at(r, 25, PAT_WALL_8);

set_local_map_at(r, 27, PAT_WALL_1);

// row 26
r = 26;

set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 4, PAT_WALL_20);
set_local_map_at(r, 5, PAT_WALL_22);

```

```
  set_local_map_at(r, 22, PAT_WALL_20);
  set_local_map_at(r, 23, PAT_WALL_22);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 27
  r = 27;

  set_local_map_at(r, 0, PAT_WALL_19);
  set_local_map_at(r, 1, PAT_WALL_21);
  set_local_map_at(r, 2, PAT_WALL_11);

  set_local_map_at(r, 4, PAT_WALL_20);
  set_local_map_at(r, 5, PAT_WALL_22);

  set_local_map_at(r, 7, PAT_WALL_10);
  set_local_map_at(r, 8, PAT_WALL_11);

  set_local_map_at(r, 10, PAT_WALL_10);
  set_local_map_at(r, 11, PAT_WALL_21);
  set_local_map_at(r, 12, PAT_WALL_21);
  set_local_map_at(r, 13, PAT_WALL_21);
  set_local_map_at(r, 14, PAT_WALL_21);
  set_local_map_at(r, 15, PAT_WALL_21);
  set_local_map_at(r, 16, PAT_WALL_21);
  set_local_map_at(r, 17, PAT_WALL_11);

  set_local_map_at(r, 19, PAT_WALL_10);
  set_local_map_at(r, 20, PAT_WALL_11);

  set_local_map_at(r, 22, PAT_WALL_20);
  set_local_map_at(r, 23, PAT_WALL_22);

  set_local_map_at(r, 25, PAT_WALL_10);
  set_local_map_at(r, 26, PAT_WALL_21);
  set_local_map_at(r, 27, PAT_WALL_14);

  // row 28
  r = 28;

  set_local_map_at(r, 0, PAT_WALL_18);
  set_local_map_at(r, 1, PAT_WALL_23);
  set_local_map_at(r, 2, PAT_WALL_8);

  set_local_map_at(r, 4, PAT_WALL_9);
  set_local_map_at(r, 5, PAT_WALL_8);

  set_local_map_at(r, 7, PAT_WALL_20);
  set_local_map_at(r, 8, PAT_WALL_22);

  set_local_map_at(r, 10, PAT_WALL_9);
  set_local_map_at(r, 11, PAT_WALL_23);
  set_local_map_at(r, 12, PAT_WALL_23);
  set_local_map_at(r, 13, PAT_WALL_11);
  set_local_map_at(r, 14, PAT_WALL_10);
  set_local_map_at(r, 15, PAT_WALL_23);
  set_local_map_at(r, 16, PAT_WALL_23);
  set_local_map_at(r, 17, PAT_WALL_8);

  set_local_map_at(r, 19, PAT_WALL_20);
  set_local_map_at(r, 20, PAT_WALL_22);

  set_local_map_at(r, 22, PAT_WALL_9);
  set_local_map_at(r, 23, PAT_WALL_8);

  set_local_map_at(r, 25, PAT_WALL_9);
  set_local_map_at(r, 26, PAT_WALL_23);
  set_local_map_at(r, 27, PAT_WALL_15);

  // row 29
  r = 29;
```

```

set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 7, PAT_WALL_20);
set_local_map_at(r, 8, PAT_WALL_22);

set_local_map_at(r, 13, PAT_WALL_20);
set_local_map_at(r, 14, PAT_WALL_22);

set_local_map_at(r, 19, PAT_WALL_20);
set_local_map_at(r, 20, PAT_WALL_22);

set_local_map_at(r, 27, PAT_WALL_1);

// row 30
r = 30;

set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 2, PAT_WALL_10);
set_local_map_at(r, 3, PAT_WALL_21);
set_local_map_at(r, 4, PAT_WALL_21);
set_local_map_at(r, 5, PAT_WALL_21);
set_local_map_at(r, 6, PAT_WALL_21);
set_local_map_at(r, 7, PAT_WALL_8);
set_local_map_at(r, 8, PAT_WALL_9);
set_local_map_at(r, 9, PAT_WALL_21);
set_local_map_at(r, 10, PAT_WALL_21);
set_local_map_at(r, 11, PAT_WALL_11);

set_local_map_at(r, 13, PAT_WALL_20);
set_local_map_at(r, 14, PAT_WALL_22);

set_local_map_at(r, 16, PAT_WALL_10);
set_local_map_at(r, 17, PAT_WALL_21);
set_local_map_at(r, 18, PAT_WALL_21);
set_local_map_at(r, 19, PAT_WALL_8);
set_local_map_at(r, 20, PAT_WALL_9);
set_local_map_at(r, 21, PAT_WALL_21);
set_local_map_at(r, 22, PAT_WALL_21);
set_local_map_at(r, 23, PAT_WALL_21);
set_local_map_at(r, 24, PAT_WALL_21);
set_local_map_at(r, 25, PAT_WALL_11);

set_local_map_at(r, 27, PAT_WALL_1);

// row 31
r = 31;

set_local_map_at(r, 0, PAT_WALL_3);

set_local_map_at(r, 2, PAT_WALL_9);
set_local_map_at(r, 3, PAT_WALL_23);
set_local_map_at(r, 4, PAT_WALL_23);
set_local_map_at(r, 5, PAT_WALL_23);
set_local_map_at(r, 6, PAT_WALL_23);
set_local_map_at(r, 7, PAT_WALL_23);
set_local_map_at(r, 8, PAT_WALL_23);
set_local_map_at(r, 9, PAT_WALL_23);
set_local_map_at(r, 10, PAT_WALL_23);
set_local_map_at(r, 11, PAT_WALL_8);

set_local_map_at(r, 13, PAT_WALL_9);
set_local_map_at(r, 14, PAT_WALL_8);

set_local_map_at(r, 16, PAT_WALL_9);
set_local_map_at(r, 17, PAT_WALL_23);
set_local_map_at(r, 18, PAT_WALL_23);
set_local_map_at(r, 19, PAT_WALL_23);
set_local_map_at(r, 20, PAT_WALL_23);
set_local_map_at(r, 21, PAT_WALL_23);
```

```
  set_local_map_at(r, 22, PAT_WALL_23);
  set_local_map_at(r, 23, PAT_WALL_23);
  set_local_map_at(r, 24, PAT_WALL_23);
  set_local_map_at(r, 25, PAT_WALL_8);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 32
  r = 32;
  set_local_map_at(r, 0, PAT_WALL_3);

  set_local_map_at(r, 27, PAT_WALL_1);

  // row 33
  r = 33;
  set_local_map_at(r, 0, PAT_WALL_7);
  for (i = 1; i < 27; i++)
    set_local_map_at(r, i, PAT_WALL_2);
  set_local_map_at(r, 27, PAT_WALL_6);
}

void setup_map_foods() {
  int i, r;

  // row 4
  r = 4;
  for (i = 1; i < 13; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 15; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);

  // row 5
  r = 5;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 6
  r = 6;
  set_local_map_at(r, 1, PAT_FOOD_LG);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_LG);

  // row 7
  r = 7;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 8
  r = 8;
  for (i = 1; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);

  // row 9
  r = 9;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 9, PAT_FOOD_SM);
  set_local_map_at(r, 18, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 10
  r = 10;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 9, PAT_FOOD_SM);
  set_local_map_at(r, 18, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 11
  r = 11;
  for (i = 1; i < 7; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 9; i < 13; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 15; i < 19; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 21; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);

  // row 12
  r = 12;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 13
  r = 13;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 14
  r = 14;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 15
  r = 15;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 16
  r = 16;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 17
  r = 17;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 18
  r = 18;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 19
  r = 19;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 20
  r = 20;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 21
  r = 21;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 22
  r = 22;
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);

  // row 23
  r = 23;
  for (i = 1; i < 13; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 15; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);

  // row 24
  r = 24;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 25
  r = 25;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 26
  r = 26;
  set_local_map_at(r, 1, PAT_FOOD_LG);
  set_local_map_at(r, 2, PAT_FOOD_SM);
  set_local_map_at(r, 3, PAT_FOOD_SM);
  for (i = 6; i < 13; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 15; i < 22; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  set_local_map_at(r, 24, PAT_FOOD_SM);
  set_local_map_at(r, 25, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_LG);

  // row 27
  r = 27;
  set_local_map_at(r, 3, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 9, PAT_FOOD_SM);
  set_local_map_at(r, 18, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 24, PAT_FOOD_SM);

  // row 28
  r = 28;
  set_local_map_at(r, 3, PAT_FOOD_SM);
  set_local_map_at(r, 6, PAT_FOOD_SM);
  set_local_map_at(r, 9, PAT_FOOD_SM);
  set_local_map_at(r, 18, PAT_FOOD_SM);
  set_local_map_at(r, 21, PAT_FOOD_SM);
  set_local_map_at(r, 24, PAT_FOOD_SM);

  // row 29
  r = 29;
  for (i = 1; i < 7; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 9; i < 13; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 15; i < 19; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
  for (i = 21; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
```

```c

  // row 30
  r = 30;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 31
  r = 31;
  set_local_map_at(r, 1, PAT_FOOD_SM);
  set_local_map_at(r, 12, PAT_FOOD_SM);
  set_local_map_at(r, 15, PAT_FOOD_SM);
  set_local_map_at(r, 26, PAT_FOOD_SM);

  // row 32
  r = 32;
  for (i = 1; i < 27; i++)
    set_local_map_at(r, i, PAT_FOOD_SM);
}
```

Listing 19: map.c

```c
#ifndef _GAMEPLAY_H
#define _GAMEPLAY_H

#include "sprite.h"
#include <stdbool.h>

typedef enum {
  STAGE_MENU,
  STAGE_IN_GAME,
  STAGE_END_GAME,
} game_stage_t;

typedef enum {
  DIR_NONE,
  DIR_LEFT,
  DIR_RIGHT,
  DIR_UP,
  DIR_DOWN,
} dir_t;

typedef struct {
  dir_t dir0;
  dir_t dir1;
  sprite_attr_t attr;
} pacman_t;

typedef struct {
  dir_t dir;
  sprite_attr_t attr;
  int release;
  int trapped_dir;
  int scatter;
} ghost_t;

void setup_game();

bool can_turn(int r, int c, dir_t);

bool will_collide(int r, int c, dir_t);

bool is_wall(uint8_t pat);

void reset_characters();

void reset_lives();

void reset_scores();

void set_pacman_dir(dir_t dir);

bool blink_timer();

bool pacman_move_timer();

bool ghost_move_timer();

bool ghost_release_timer();

bool ghost_trapped_move_timer();

void move_pacman();

void eat_food();

void ghosts_catch_pacman();

void update_player_score(int s);

void animate_pacman();

void release_ghost();

void move_ghosts();

void move_ghosts_trapped();

void move_ghosts_release();

void move_ghost_random();

void release_next_ghost();

bool need_turn(int r, int c, dir_t dir);

void reset_game();

game_stage_t get_game_stage();

void press_start_game();

void end_game();

void update_scores();

bool beat_best_score();

void scatter_timer();

void next_life();

#endif
```

Listing 20: gameplay.h

```c
#include "gameplay.h"
#include "map.h"
#include "pattern.h"
#include <limits.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/queue.h>
#include <unistd.h>

typedef struct {
  pthread_mutex_t mu;
  game_stage_t stage;
  int scatter_time;
  bool scatter_mode;
  uint32_t player_score;
  uint32_t best_score;
  uint8_t lives;
  uint8_t nreleased;
  uint32_t release_timer;
  pacman_t pacman;
  ghost_t ghost_red;
  ghost_t ghost_cyan;
  ghost_t ghost_pink;
  ghost_t ghost_orange;
} game_state_t;

static game_state_t game;

void setup_game() {
  pthread_mutex_init(&game.mu, NULL);
  pthread_mutex_lock(&game.mu);

  game.stage = STAGE_MENU;
  game.release_timer = 0;
  game.nreleased = 1;

  game.scatter_time = 0;
  game.scatter_mode = false;
  game.ghost_red.scatter = 0;
  game.ghost_cyan.scatter = 0;
  game.ghost_pink.scatter = 0;
  game.ghost_orange.scatter = 0;

  setup_map_foods();

  reset_scores();
  reset_lives();
  reset_characters();

  printf("Game is ready\n");
  pthread_mutex_unlock(&game.mu);
}

void reset_game() {
  pthread_mutex_lock(&game.mu);

  game.stage = STAGE_MENU;
  game.release_timer = 0;
  game.nreleased = 1;

  game.scatter_time = 0;
  game.scatter_mode = false;
  game.ghost_red.scatter = 0;
  game.ghost_cyan.scatter = 0;
  game.ghost_pink.scatter = 0;
  game.ghost_orange.scatter = 0;

  setup_map_foods();
  reset_lives();
  reset_characters();

  pthread_mutex_unlock(&game.mu);
}

void hide_lives() {
  for (int c = 0; c <= 6; c++) {
    set_local_map_at(35, c, PAT_BACKGROUND);
  }
}

void show_lives() {
  static int counter = 0;
  static int flip = 1;
  counter = (counter + 1) % 800;

  if (counter == 0)
    flip *= -1;

  if (flip == 1) {
    set_map_lives(game.lives);
  } else {
    hide_lives();
  }
}

void next_life() {
  pthread_mutex_lock(&game.mu);

  game.lives--;
  if (game.lives >= 1) {

    // blink lives
    {
      uint32_t counter = 0;
      while (counter < 2500) {
        counter++;
        show_lives();
        usleep(1000);
      }
    }
    set_map_lives(game.lives);

    // reset positions
    game.release_timer = 0;
    game.nreleased = 1;

    game.scatter_time = 0;
    game.scatter_mode = false;
    game.ghost_red.scatter = 0;
    game.ghost_cyan.scatter = 0;
    game.ghost_pink.scatter = 0;
    game.ghost_orange.scatter = 0;

    reset_characters();

    {
      uint32_t counter = 0;
      while (counter < 2000) {
        counter++;
        usleep(1000);
      }
    }

    pthread_mutex_unlock(&game.mu);
  } else {

    // blink lives
    {
      uint32_t counter = 0;
      while (counter < 2500) {
        counter++;
        show_lives();
        usleep(1000);
      }
    }
    set_map_lives(game.lives);

    pthread_mutex_unlock(&game.mu);
    end_game();
  }
}

game_stage_t get_game_stage() { return game.stage; }

void press_start_game() {
  pthread_mutex_lock(&game.mu);
  game.stage = STAGE_IN_GAME;
  pthread_mutex_unlock(&game.mu);
}

void start_scatter_mode() {
  game.scatter_mode = true;

  game.ghost_red.attr.name = SPRITE_GHOST_SCATTER;
  game.ghost_red.dir = DIR_NONE;
  game.ghost_red.scatter = 1;
  game.ghost_cyan.attr.name = SPRITE_GHOST_SCATTER;
  game.ghost_cyan.dir = DIR_NONE;
  game.ghost_cyan.scatter = 1;
  game.ghost_pink.attr.name = SPRITE_GHOST_SCATTER;
  game.ghost_pink.dir = DIR_NONE;
  game.ghost_pink.scatter = 1;
  game.ghost_orange.attr.name = SPRITE_GHOST_SCATTER;
  game.ghost_orange.dir = DIR_NONE;
  game.ghost_orange.scatter = 1;
}

void eat_food() {
  uint8_t pat;
  int r, c;
  int tiler, tilec;

  r = game.pacman.attr.y;
  c = game.pacman.attr.x;

  if (r % 8 == 4 && c % 8 == 4) {
    tiler = 1 + r / 8;
    tilec = 1 + c / 8;
    pat = get_map_at(tiler, tilec);
    if (pat == PAT_FOOD_SM) {
      set_map_at(tiler, tilec, PAT_BACKGROUND);
      update_player_score(10);
    } else if (pat == PAT_FOOD_LG) {
      set_map_at(tiler, tilec, PAT_BACKGROUND);
      start_scatter_mode();
      game.scatter_time = 0;
      update_player_score(50);
    }
  }
}

void update_scores() {
  if (game.player_score > game.best_score) {
    game.best_score = game.player_score;
  }
  game.player_score = 0;
  set_map_player_score(game.player_score);
  set_map_best_score(game.best_score);
}

bool beat_best_score() { return game.player_score > game.best_score; }

void end_game() {
  pthread_mutex_lock(&game.mu);
  game.stage = STAGE_END_GAME;
  pthread_mutex_unlock(&game.mu);
}

bool __ghost_catch_pacman(ghost_t *ghost) {
  int r = ghost->attr.y;
  int c = ghost->attr.x;

  int aleft, aright, atop, abottom;
  int bleft, bright, btop, bbottom;

  aleft = c;
  aright = c + 12;
  atop = r + 12;
  abottom = r;

  bleft = game.pacman.attr.x;
  bright = game.pacman.attr.x + 12;
  btop = game.pacman.attr.y + 12;
  bbottom = game.pacman.attr.y;

  return aleft < bright && aright > bleft && atop > bbottom && abottom < btop;
}

void caught_ghost(int g, ghost_t *ghost) {
  update_player_score(400);
  ghost->scatter = 0;
  ghost->release = 1;
  ghost->attr.name = SPRITE_GHOST_RED + g;
  ghost->attr.y = (MAP_ROW_OFFSET + 16) * 8;
  ghost->attr.x = (MAP_COL_OFFSET + 13) * 8;

  set_sprite(ghost->attr);
}

bool ghost_catch_pacman(int g, ghost_t *ghost) {
  if (game.stage != STAGE_IN_GAME)
    return false;

  if (__ghost_catch_pacman(ghost)) {
    if (ghost->scatter != 0) {
      caught_ghost(g, ghost);
      return true;
    } else {
      next_life();
      return false;
    }
  }

  return true;
}

void ghosts_catch_pacman() {
  if (ghost_catch_pacman(0, &game.ghost_red)) {
    if (ghost_catch_pacman(1, &game.ghost_cyan)) {
      if (ghost_catch_pacman(2, &game.ghost_pink)) {
        ghost_catch_pacman(3, &game.ghost_orange);
      }
    }
  }
}

void flip_ghost_scatter(int g, ghost_t *ghost) {
  if (ghost->scatter == 0)
    return;

  if (ghost->attr.name == SPRITE_GHOST_SCATTER) {
    ghost->attr.name = SPRITE_GHOST_RED + g;
  } else {
    ghost->attr.name = SPRITE_GHOST_SCATTER;
  }

  set_sprite(ghost->attr);
}

void end_ghost_scatter(int g, ghost_t *ghost) {
  if (ghost->scatter == 0)
    return;

  ghost->scatter = 0;
  ghost->attr.name = SPRITE_GHOST_RED + g;

  set_sprite(ghost->attr);
}
```

```

void scatter_timer() {
  if (game.scatter_mode) {
    game.scatter_time++;
    if (game.scatter_time >= 4000) {
      if (game.scatter_time % 200 == 0) {
        flip_ghost_scatter(0, &game.ghost_red);
        flip_ghost_scatter(1, &game.ghost_cyan);
        flip_ghost_scatter(2, &game.ghost_pink);
        flip_ghost_scatter(3, &game.ghost_orange);
      }

      if (game.scatter_time >= 6000) {
        end_ghost_scatter(0, &game.ghost_red);
        end_ghost_scatter(1, &game.ghost_cyan);
        end_ghost_scatter(2, &game.ghost_pink);
        end_ghost_scatter(3, &game.ghost_orange);

        game.scatter_mode = false;
      }
    }
  }
}

void update_player_score(int s) {
  game.player_score += s;
  set_map_player_score(game.player_score);
}

bool is_perpendicular(dir_t dir1, dir_t dir2) {
  if (dir1 == DIR_LEFT || dir1 == DIR_RIGHT) {
    return dir2 == DIR_UP || dir2 == DIR_DOWN;
  }

  if (dir1 == DIR_UP || dir1 == DIR_DOWN) {
    return dir2 == DIR_LEFT || dir2 == DIR_RIGHT;
  }

  return false;
}

bool is_not_backward(dir_t dir1, dir_t dir2) {
  return is_perpendicular(dir1, dir2) || dir1 == dir2;
}

void set_pacman_dir(dir_t dir) {
  pthread_mutex_lock(&game.mu);

  if (is_perpendicular(game.pacman.dir0, dir)) {
    game.pacman.dir1 = dir;
  } else {
    game.pacman.dir0 = dir;
    game.pacman.dir1 = DIR_NONE;
  }

  pthread_mutex_unlock(&game.mu);
}

void reset_lives() {
  game.lives = 3;

  set_map_lives(game.lives);
}

void reset_scores() {
  game.player_score = 0;
  game.best_score = 0;

  set_map_player_score(game.player_score);
  set_map_best_score(game.best_score);
}
```

```

void animate_pacman() {
  if (game.pacman.attr.name == SPRITE_PACMAN_CLOSED) {
    game.pacman.attr.name = SPRITE_PACMAN_CLOSED + game.pacman.dir0;
  } else {
    game.pacman.attr.name = SPRITE_PACMAN_CLOSED;
  }
}

void reset_characters() {
  game.pacman.dir0 = DIR_NONE;
  game.pacman.dir1 = DIR_NONE;
  game.pacman.attr.i = 0;
  game.pacman.attr.y = (MAP_ROW_OFFSET + 25) * 8 + 4;
  game.pacman.attr.x = (MAP_COL_OFFSET + 13) * 8;
  game.pacman.attr.name = SPRITE_PACMAN_CLOSED;

  game.ghost_red.release = 2;
  game.ghost_red.trapped_dir = -1;
  game.ghost_red.dir = DIR_NONE;
  game.ghost_red.attr.i = 1;
  game.ghost_red.attr.y = (MAP_ROW_OFFSET + 13) * 8 + 4;
  game.ghost_red.attr.x = (MAP_COL_OFFSET + 13) * 8;
  game.ghost_red.attr.name = SPRITE_GHOST_RED;

  game.ghost_cyan.release = 0;
  game.ghost_cyan.trapped_dir = -1;
  game.ghost_cyan.dir = DIR_NONE;
  game.ghost_cyan.attr.i = 2;
  game.ghost_cyan.attr.y = (MAP_ROW_OFFSET + 17) * 8;
  game.ghost_cyan.attr.x = (MAP_COL_OFFSET + 11) * 8;
  game.ghost_cyan.attr.name = SPRITE_GHOST_CYAN;

  game.ghost_pink.release = 0;
  game.ghost_pink.trapped_dir = 1;
  game.ghost_pink.dir = DIR_NONE;
  game.ghost_pink.attr.i = 3;
  game.ghost_pink.attr.y = (MAP_ROW_OFFSET + 16) * 8;
  game.ghost_pink.attr.x = (MAP_COL_OFFSET + 13) * 8;
  game.ghost_pink.attr.name = SPRITE_GHOST_PINK;

  game.ghost_orange.release = 0;
  game.ghost_orange.trapped_dir = -1;
  game.ghost_orange.dir = DIR_NONE;
  game.ghost_orange.attr.i = 4;
  game.ghost_orange.attr.y = (MAP_ROW_OFFSET + 17) * 8;
  game.ghost_orange.attr.x = (MAP_COL_OFFSET + 15) * 8;
  game.ghost_orange.attr.name = SPRITE_GHOST_ORANGE;

  set_sprite(game.pacman.attr);
  set_sprite(game.ghost_red.attr);
  set_sprite(game.ghost_cyan.attr);
  set_sprite(game.ghost_pink.attr);
  set_sprite(game.ghost_orange.attr);
}

bool blink_timer() {
  static int counter = 0;
  counter = (counter + 1) % 120;
  return counter == 0;
}

bool pacman_move_timer() {
  static int counter = 0;
  counter = (counter + 1) % 15;
  return counter == 0;
}

bool ghost_move_timer() {
  static int counter = 0;
  counter = (counter + 1) % 20;
```

```c
  return counter == 0;
}

bool ghost_trapped_move_timer() {
  static int counter = 0;
  counter = (counter + 1) % 50;
  return counter == 0;
}

bool ghost_release_timer() {
  game.release_timer = (game.release_timer + 1) % 2000;
  return game.release_timer == 0;
}

void release_next_ghost() {
  if (game.ghost_pink.release == 0) {
    game.ghost_pink.release = 1;
    return;
  }
  if (game.ghost_cyan.release == 0) {
    game.ghost_cyan.release = 1;
    return;
  }
  if (game.ghost_orange.release == 0) {
    game.ghost_orange.release = 1;
    return;
  }
}

void move_ghost_with_dir(ghost_t *ghost) {
  switch (ghost->dir) {
  case DIR_NONE:
    break;
  case DIR_LEFT:
    ghost->attr.x--;
    break;
  case DIR_RIGHT:
    ghost->attr.x++;
    break;
  case DIR_UP:
    ghost->attr.y--;
    break;
  case DIR_DOWN:
    ghost->attr.y++;
    break;
  }

  set_sprite(ghost->attr);
}

void release_ghost(ghost_t *ghost) {
  if (ghost->release == 0) {
    ghost->release = 1;
  }
}

void move_ghost_trapped(ghost_t *ghost) {
  if (ghost->release != 0)
    return;

  if (ghost->attr.y >= (MAP_ROW_OFFSET + 17) * 8) {
    ghost->trapped_dir = -1;
  }
  if (ghost->attr.y <= (MAP_ROW_OFFSET + 16) * 8) {
    ghost->trapped_dir = 1;
  }

  ghost->attr.y += ghost->trapped_dir;
  set_sprite(ghost->attr);
}

```

```c
void move_ghosts_trapped() {
  move_ghost_trapped(&game.ghost_red);
  move_ghost_trapped(&game.ghost_cyan);
  move_ghost_trapped(&game.ghost_pink);
  move_ghost_trapped(&game.ghost_orange);
}

void move_ghost_release(ghost_t *ghost) {
  if (ghost->release != 1)
    return;

  int target_r = (MAP_ROW_OFFSET + 13) * 8 + 4;
  int target_c = (MAP_COL_OFFSET + 13) * 8;

  if (ghost->attr.x != target_c) {
    int d = 1;
    if (ghost->attr.x > target_c)
      d = -1;

    ghost->attr.x += d;
    set_sprite(ghost->attr);
    return;
  }

  if (ghost->attr.y != target_r) {
    int d = 1;
    if (ghost->attr.y > target_r) {
      d = -1;
    }
    ghost->attr.y += d;
    set_sprite(ghost->attr);
    return;
  }

  ghost->release = 2;
}

void move_ghosts_release() {
  move_ghost_release(&game.ghost_red);
  move_ghost_release(&game.ghost_cyan);
  move_ghost_release(&game.ghost_pink);
  move_ghost_release(&game.ghost_orange);
}

typedef struct {
  int r;
  int c;
} coordinate_t;

int search_depth_bfs(coordinate_t pos0, coordinate_t pacman_pos, dir_t dir0,
                     coordinate_t visited[], int nvisited) {
  int found_depth;

  if (pos0.r == pacman_pos.r && pos0.c == pacman_pos.c) {
    return 0;
  }

  TAILQ_HEAD(tailhead, entry) head;
  struct entry {
    coordinate_t pos;
    dir_t dir;
    int depth;
    TAILQ_ENTRY(entry) entries;
  };
  TAILQ_INIT(&head);

  // add initial pos to visited
  visited[nvisited++] = pos0;

  // add initial pos to queue
  struct entry *n0 = malloc(sizeof(struct entry));
  n0->pos = pos0;
  n0->dir = dir0;
  n0->depth = 0;
  TAILQ_INSERT_HEAD(&head, n0, entries);

  while (!TAILQ_EMPTY(&head)) {
    // dequeue
    struct entry *e = TAILQ_FIRST(&head);

    if (e->pos.r % 8 != 4 || e->pos.c % 8 != 4) {
      fprintf(stderr, "WHAT\n");
      exit(1);
    }

    // get neighbor directions
    int n = 0;
    dir_t candidates[4];
    dir_t all_dirs[4] = {
        DIR_LEFT,
        DIR_RIGHT,
        DIR_UP,
        DIR_DOWN,
    };
    for (int i = 0; i < 4; i++) {
      if (e->dir == DIR_NONE || is_not_backward(e->dir, all_dirs[i])) {
        if (!will_collide(e->pos.r, e->pos.c, all_dirs[i])) {
          candidates[n] = all_dirs[i];
          n++;
        }
      }
    }

    if (n == 0) {
      fprintf(stderr, "Can't have zero directions\n");
      exit(1);
    }

    // loop through possible directions
    for (int i = 0; i < n; i++) {
      coordinate_t new_pos;
      switch (candidates[i]) {
      case DIR_LEFT:
        new_pos.r = e->pos.r;
        new_pos.c = e->pos.c - 8;
        break;
      case DIR_RIGHT:
        new_pos.r = e->pos.r;
        new_pos.c = e->pos.c + 8;
        break;
      case DIR_UP:
        new_pos.r = e->pos.r - 8;
        new_pos.c = e->pos.c;
        break;
      case DIR_DOWN:
        new_pos.r = e->pos.r + 8;
        new_pos.c = e->pos.c;
        break;
      case DIR_NONE:
        fprintf(stderr, "Search depth can't have a DIR_NONE.\n");
        exit(1);
      }

      // found pacman
      if (new_pos.r == pacman_pos.r && new_pos.c == pacman_pos.c) {
        found_depth = e->depth + 1;
        goto found_target;
      }

      // test visited
      bool has_visited = false;
      for (int i = 0; i < nvisited; i++) {
        coordinate_t v = visited[i];
        if (v.r == new_pos.r && v.c == new_pos.c) {
          has_visited = true;
        }
      }

      if (!has_visited) {
        struct entry *n = malloc(sizeof(struct entry));
        n->pos = new_pos;
        n->dir = candidates[i];
        n->depth = e->depth + 1;
        TAILQ_INSERT_TAIL(&head, n, entries);
      }
    }
    TAILQ_REMOVE(&head, e, entries);
    free(e);
  }

found_target:
  // cleanup
  while (!TAILQ_EMPTY(&head)) {
    struct entry *e = TAILQ_FIRST(&head);
    TAILQ_REMOVE(&head, e, entries);
    free(e);
  }

  return found_depth;
}

int search_depth_with_dir(coordinate_t pos, coordinate_t pacman_pos,
                          dir_t dir) {
  coordinate_t visited[2500];

  // printf("search_depth_with_dir\n");
  if (pos.r == pacman_pos.r && pos.c == pacman_pos.c)
    return 0;

  visited[0] = pos;

  if (will_collide(pos.r, pos.c, dir)) {
    // printf("search_depth_with_dir: will_collide\n");
    return INT_MAX;
  }

  coordinate_t new_pos;

  switch (dir) {
  case DIR_LEFT:
    new_pos.r = pos.r;
    new_pos.c = pos.c - 8;
    break;
  case DIR_RIGHT:
    new_pos.r = pos.r;
    new_pos.c = pos.c + 8;
    break;
  case DIR_UP:
    new_pos.r = pos.r - 8;
    new_pos.c = pos.c;
    break;
  case DIR_DOWN:
    new_pos.r = pos.r + 8;
    new_pos.c = pos.c;
    break;
  case DIR_NONE:
    fprintf(stderr, "Search depth can't have a DIR_NONE.");
    exit(1);
  }

  return search_depth_bfs(new_pos, pacman_pos, dir, visited, 1);
}
```

```c
void move_ghost_scatter(ghost_t *ghost) {
  dir_t final_dir;
  int results[4];

  if (!need_turn(ghost->attr.y, ghost->attr.x, ghost->dir)) {
    move_ghost_with_dir(ghost);
    return;
  }

  coordinate_t pos, pacman_pos;

  pos.r = (ghost->attr.y / 8) * 8 + 4;
  pos.c = (ghost->attr.x / 8) * 8 + 4;

  pacman_pos.r = (game.pacman.attr.y / 8) * 8 + 4;
  pacman_pos.c = (game.pacman.attr.x / 8) * 8 + 4;

  if (pos.r == pacman_pos.r && pos.c == pacman_pos.c) {
    final_dir = DIR_NONE;
  } else if (ghost->dir == DIR_NONE ||
             (ghost->attr.x % 8 == 4 && ghost->attr.y % 8 == 4)) {
    results[0] = search_depth_with_dir(pos, pacman_pos, DIR_LEFT);
    results[1] = search_depth_with_dir(pos, pacman_pos, DIR_RIGHT);
    results[2] = search_depth_with_dir(pos, pacman_pos, DIR_UP);
    results[3] = search_depth_with_dir(pos, pacman_pos, DIR_DOWN);

    int max_dir = -1;
    int max_depth = -1;
    for (int i = 0; i < 4; i++) {
      if (results[i] > max_depth && results[i] != INT_MAX) {
        max_dir = i;
        max_depth = results[i];
      }
    }

    if (max_dir == -1) {
      final_dir = DIR_NONE;
    } else {
      final_dir = DIR_LEFT + max_dir;
    }
  } else {
    final_dir = ghost->dir;
  }

  if (final_dir == DIR_NONE) {
    move_ghost_random(ghost);
  } else if (will_collide(ghost->attr.y, ghost->attr.x, final_dir)) {
    final_dir = DIR_NONE;
    move_ghost_random(ghost);
  } else {
    ghost->dir = final_dir;
    move_ghost_with_dir(ghost);
  }
}

void move_ghosts_scatter() {
  move_ghost_scatter(&game.ghost_red);
  move_ghost_scatter(&game.ghost_cyan);
  move_ghost_scatter(&game.ghost_pink);
  move_ghost_scatter(&game.ghost_orange);
}

void move_ghost_random(ghost_t *ghost) {
  int n = 0;
  dir_t candidates[4];
  dir_t all_dirs[4] = {
      DIR_LEFT,
      DIR_RIGHT,
      DIR_UP,
      DIR_DOWN,
  };

  // get candidates
  for (int i = 0; i < 4; i++) {
    if (ghost->dir == DIR_NONE || is_not_backward(ghost->dir, all_dirs[i])) {
      if (!will_collide(ghost->attr.y, ghost->attr.x, all_dirs[i])) {
        candidates[n] = all_dirs[i];
        n++;
      }
    }
  }

  // choose direction
  if (n == 0) {
    fprintf(stderr, "Ghost has nowhere to go. Can't happen\n");
  }
  int choice = rand() % n;
  ghost->dir = candidates[choice];

  move_ghost_with_dir(ghost);
}

void move_ghost_targeted(ghost_t *ghost) {
  dir_t final_dir;
  int results[4];

  if (!need_turn(ghost->attr.y, ghost->attr.x, ghost->dir)) {
    move_ghost_with_dir(ghost);
    return;
  }

  coordinate_t pos, pacman_pos;

  pos.r = (ghost->attr.y / 8) * 8 + 4;
  pos.c = (ghost->attr.x / 8) * 8 + 4;

  pacman_pos.r = (game.pacman.attr.y / 8) * 8 + 4;
  pacman_pos.c = (game.pacman.attr.x / 8) * 8 + 4;

  if (pos.r == pacman_pos.r && pos.c == pacman_pos.c) {
    final_dir = DIR_NONE;
  } else if (ghost->dir == DIR_NONE ||
             (ghost->attr.x % 8 == 4 && ghost->attr.y % 8 == 4)) {
    results[0] = search_depth_with_dir(pos, pacman_pos, DIR_LEFT);
    results[1] = search_depth_with_dir(pos, pacman_pos, DIR_RIGHT);
    results[2] = search_depth_with_dir(pos, pacman_pos, DIR_UP);
    results[3] = search_depth_with_dir(pos, pacman_pos, DIR_DOWN);

    int min_dir = -1;
    int min_depth = INT_MAX;
    for (int i = 0; i < 4; i++) {
      if (results[i] < min_depth) {
        min_dir = i;
        min_depth = results[i];
      }
    }

    if (min_dir == -1) {
      final_dir = DIR_NONE;
    } else {
      final_dir = DIR_LEFT + min_dir;
    }
  } else {
    final_dir = ghost->dir;
  }

  if (final_dir == DIR_NONE) {
    move_ghost_random(ghost);
  } else if (will_collide(ghost->attr.y, ghost->attr.x, final_dir)) {
    final_dir = DIR_NONE;
    move_ghost_random(ghost);
  } else {
```

```c
    ghost->dir = final_dir;
    move_ghost_with_dir(ghost);
  }
}

void move_ghost(ghost_t *ghost) {
  if (ghost->release != 2)
    return;

  int32_t deltar = ((int32_t)ghost->attr.y) - ((int32_t)game.pacman.attr.y);
  int32_t deltac = ((int32_t)ghost->attr.x) - ((int32_t)game.pacman.attr.x);

  // move_ghost_random(ghost);
  if (ghost->scatter == 0) {
    if (deltar * deltar + deltac * deltac <= (10 * 8) * (10 * 8)) {
      move_ghost_targeted(ghost);
    } else {
      move_ghost_random(ghost);
    }
  } else if (ghost->scatter == 1) {
    if (deltar * deltar + deltac * deltac <= (10 * 8) * (10 * 8)) {
      move_ghost_scatter(ghost);
    } else {
      move_ghost_random(ghost);
    }
  }
}

void move_ghosts() {
  move_ghost(&game.ghost_red);
  move_ghost(&game.ghost_cyan);
  move_ghost(&game.ghost_pink);
  move_ghost(&game.ghost_orange);
}

void move_pacman() {
  pthread_mutex_lock(&game.mu);

  // try turn
  if (can_turn(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir1)) {
    game.pacman.dir0 = game.pacman.dir1;
    game.pacman.dir1 = DIR_NONE;
  }

  // try dir0
  if (will_collide(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir0)) {
    game.pacman.dir0 = game.pacman.dir1;
    game.pacman.dir1 = DIR_NONE;
  }

  // try dir1
  if (will_collide(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir0)) {
    game.pacman.dir0 = game.pacman.dir1;
    game.pacman.dir1 = DIR_NONE;
  }

  switch (game.pacman.dir0) {
  case DIR_NONE:
    break;
  case DIR_LEFT:
    game.pacman.attr.x--;
    break;
  case DIR_RIGHT:
    game.pacman.attr.x++;
    break;
  case DIR_UP:
    game.pacman.attr.y--;
    break;
  case DIR_DOWN:
    game.pacman.attr.y++;
    break;
  }

  set_sprite(game.pacman.attr);

  pthread_mutex_unlock(&game.mu);
}

bool can_turn(int r, int c, dir_t dir) {
  uint8_t pat;

  if (r % 8 != 4 || c % 8 != 4)
    return false;

  switch (dir) {
  case DIR_NONE:
    return false;
  case DIR_LEFT:
    pat = get_map_at(1 + r / 8, c / 8);
    return !is_wall(pat);
  case DIR_RIGHT:
    pat = get_map_at(1 + r / 8, 2 + c / 8);
    return !is_wall(pat);
  case DIR_UP:
    pat = get_map_at(r / 8, 1 + c / 8);
    return !is_wall(pat);
  case DIR_DOWN:
    pat = get_map_at(2 + r / 8, 1 + c / 8);
    return !is_wall(pat);
  }

  return false;
}

bool need_turn(int r, int c, dir_t dir) {
  dir_t all_dirs[4] = {
      DIR_LEFT,
      DIR_RIGHT,
      DIR_UP,
      DIR_DOWN,
  };

  if (dir == DIR_NONE)
    return true;

  // get candidates
  for (int i = 0; i < 4; i++) {
    if (all_dirs[i] != dir && is_not_backward(dir, all_dirs[i])) {
      if (!will_collide(r, c, all_dirs[i])) {
        return true;
      }
    }
  }

  return false;
}

bool will_collide(int r, int c, dir_t dir) {
  uint8_t pat;
  bool is_h, is_v;

  is_h = (r % 8 == 4);
  is_v = (c % 8 == 4);

  switch (dir) {
  case DIR_NONE:
    return false;
  case DIR_LEFT:
    if (!is_h)
      return true;
    if (!is_v)
      return false;
    pat = get_map_at(1 + r / 8, c / 8);
    return is_wall(pat);
  case DIR_RIGHT:
    if (!is_h)
      return true;
    if (!is_v)
      return false;
    pat = get_map_at(1 + r / 8, 2 + c / 8);
    return is_wall(pat);
  case DIR_UP:
    if (!is_v)
      return true;
    if (!is_h)
      return false;
    pat = get_map_at(r / 8, 1 + c / 8);
    return is_wall(pat);
  case DIR_DOWN:
    if (!is_v)
      return true;
    if (!is_h)
      return false;
    pat = get_map_at(2 + r / 8, 1 + c / 8);
    return is_wall(pat);
  }

  return true; // shouldn't reach here
}

bool is_eating_small(sprite_attr_t s) {
  uint8_t pat;

  if (s.y % 8 == 4 && s.x % 8 == 4) {
    pat = get_map_at(1 + s.y / 8, 1 + s.x / 8);
    return pat == PAT_FOOD_SM;
  }

  return false;
}

bool is_eating_large(sprite_attr_t s) {
  uint8_t pat;

  if (s.y % 8 == 4 && s.x % 8 == 4) {
    pat = get_map_at(1 + s.y / 8, 1 + s.x / 8);
    return pat == PAT_FOOD_LG;
  }

  return false;
}

bool is_wall(uint8_t pat) {
  if (PAT_WALL_0 <= pat && pat <= PAT_GATE) {
    return true;
  }
  return false;
}
```

Listing 21: gameplay.c

```c
#ifndef _COLOR_H
#define _COLOR_H

#define Transp 0x0
#define Yellow 0x1
#define Red 0x2
#define Orange 0x3
#define Cyan 0x4
#define Pink 0x5
#define Ivory 0x6
#define Blue 0x7
#define Salmon 0x8
```

```
#define White 0x9

#endif
```

Listing 22: color.h