

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.

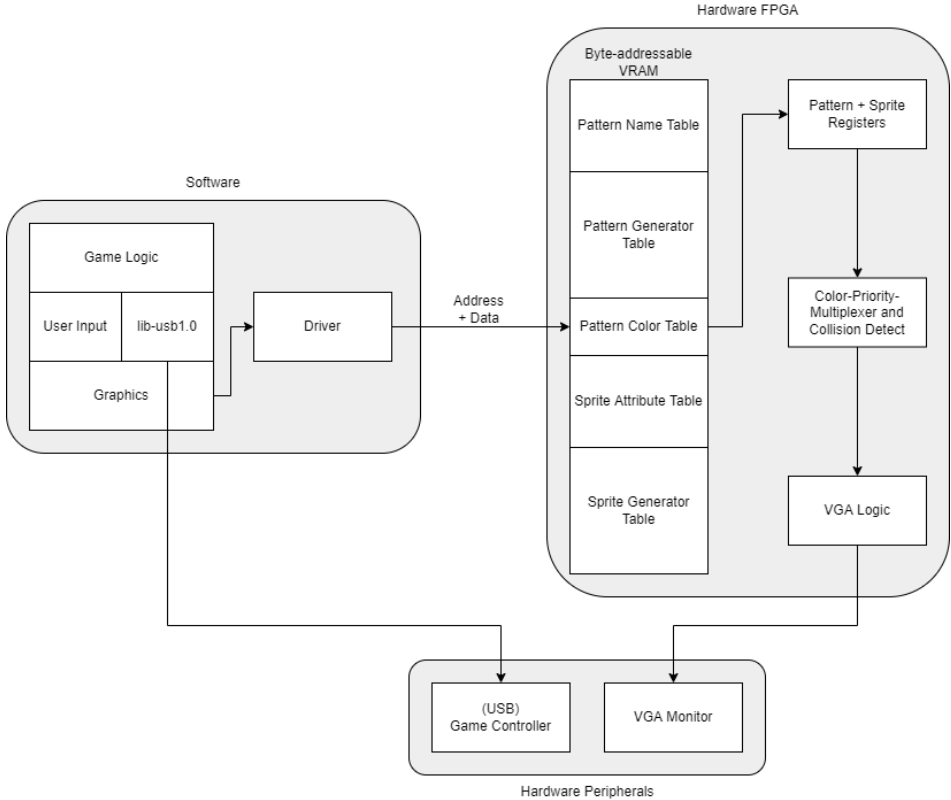


Figure 1: System Block Diagram

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 not 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 the 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 FPGA 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

```
1 /*
2  * Avalon memory-mapped peripheral that generates VGA
3  *
4  * Stephen A. Edwards
5  * Columbia University
6  */
7
8 module vga_ball(input logic      clk,
9                input logic      reset,
10               input logic [31:0] writedata,
11               input logic       write,
12               input logic       chipselect,
13               input logic [3:0]  address,
14
15               output logic [7:0] VGA_R, VGA_G, VGA_B,
16               output logic      VGA_CLK, VGA_HS, VGA_VS,
17                               VGA_BLANK_n,
18               output logic      VGA_SYNC_n);
19
20 logic [10:0]    hcount;
21 logic [9:0]     vcount;
22
23 logic [3:0] out_pixel[5:0]; //output pixels values from each of 5 sprites + 1
24   pattern
```

```

24 logic [3:0] final_out_pixel; //actual output pixel to display
25 logic [7:0] background_r, background_g, background_b;
26 logic [23:0] rgb_val; //final RGB value to display
27
28
29 //for pattern name table
30 logic [11:0] ra_n, wa_n; //12 bits
31 logic we_n;
32 logic [7:0] din_n;
33 logic [7:0] dout_n;
34
35 //for pattern generator table
36 logic [10:0] ra_pg, wa_pg; //change later
37 logic we_pg;
38 logic [7:0] din_pg;
39 logic [7:0] dout_pg;
40
41 //for sprite attribute table
42 logic [4:0] ra_a, wa_a; //5 simultaneous sprites
43 logic we_a;
44 logic [7:0] din_a;
45 logic [7:0] dout_a;
46
47 //for sprite generator table
48 logic [10:0] ra_g, wa_g; //10*128 sprite -> 11 bit addr
49 logic we_g;
50 logic [7:0] din_g;
51 logic [7:0] dout_g;
52
53 logic [4:0] sprite_base_addr[4:0]; //sprite attr table base address
54 logic [10:0] h_start[4:0]; //hcount at which sprite_prep n starts
55 logic [4:0] sprite_ra_a[4:0]; //requested read address for sprite attr table from
    sprite_prep modules
56 logic [10:0] sprite_ra_g[4:0]; //requested read address for sprite gen table from
    sprite_prep modules
57
58 //determines where each sprite prep instance will start reading the attr table from
59 assign sprite_base_addr[0]=5'h0;
60 assign sprite_base_addr[1]=5'h4;
61 assign sprite_base_addr[2]=5'h8;
62 assign sprite_base_addr[3]=5'hc;
63 assign sprite_base_addr[4]=5'h10;
64
65 //determines when each sprite prep instance will start processing sprites
66 assign h_start[0]=11'b10100100000; //1312
67 assign h_start[1]=11'b10100111010; //1338
68 assign h_start[2]=11'b10101010100; //1364
69 assign h_start[3]=11'b10101101110; //1390
70 assign h_start[4]=11'b10110001000; //1416
71
72
73
74 vga_counters counters(.clk50(clk), .*);
75 patt_name_table pn1(.clk(clk), .ra(ra_n), .wa(wa_n), .we(we_n), .din(din_n), .dout(
    dout_n));
76 patt_gen_table pg1(.clk(clk), .ra(ra_pg), .wa(wa_pg), .we(we_pg), .din(din_pg), .
    dout(dout_pg));
77
78 sprite_attr_table sat1(.clk(clk), .ra(ra_a), .wa(wa_a), .we(we_a), .din(din_a), .
    dout(dout_a));
79 sprite_gen_table sgt1(.clk(clk), .ra(ra_g), .wa(wa_g), .we(we_g), .din(din_g), .dout
    (dout_g));
80 color_lut cl1(.color_code(final_out_pixel), .rgb_val(rgb_val));
81
82 pattern_prep pp0(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .
    VGA_BLANK_n(VGA_BLANK_n),
83 .dout_n(dout_n), .dout_g(dout_pg), .ra_n(ra_n), .ra_g(ra_pg), .out_pixel(
    out_pixel[5]));
84
85 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]),

```

```

86  .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]));
87
88  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]),
89  .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]));
90
91  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]),
92  .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]));
93
94  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]),
95  .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]));
96
97  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]),
98  .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]));
99
100  always_ff @(posedge clk) begin //Writing to VRAM
101  if (reset) begin
102  background_r <= 8'h0;
103  background_g <= 8'h0;
104  background_b <= 8'h20;
105  end else if (chipselct && write) begin
106  case (writedata[1:0])
107  2'b0 : begin //pattern name table
108  we_n<=1;
109  we_pg<=0;
110  we_a<=0;
111  we_g<=0;
112  din_n<=writedata[31:24];
113  wa_n<=writedata[13:2];
114  end
115  2'b1 : begin //pattern gen table
116  we_n<=0;
117  we_pg<=1;
118  we_a<=0;
119  we_g<=0;
120  din_pg<=writedata[31:24];
121  wa_pg<=writedata[12:2];
122  end
123  2'b10 : begin //sprite attr table
124  we_n<=0;
125  we_pg<=0;
126  we_a<=1;
127  we_g<=0;
128  din_a<=writedata[31:24];
129  wa_a<=writedata[6:2];
130  end
131  2'b11 : begin //sprite gen table
132  we_n<=0;
133  we_pg<=0;
134  we_a<=0;
135  we_g<=1;
136  din_g<=writedata[31:24];
137  wa_g<=writedata[12:2];
138  end
139  endcase
140  end
141  end
142
143  always_comb begin //Display logic
144  {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
145  if (VGA_BLANK_n ) begin
146  if (final_out_pixel!=4'b0) {VGA_R, VGA_G, VGA_B} = {rgb_val[23:16], rgb_val
    [15:8], rgb_val[7:0]};

```

```

147     else {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
148     end
149 end
150
151 always_comb begin //color priority multiplexer (i.e. sprite 1 pixels precedes
152     sprite 2, sprite 2 > sprite 3...)
153     if (out_pixel[0]!=4'b0) final_out_pixel=out_pixel[0];
154     else if (out_pixel[1]!=4'b0) final_out_pixel=out_pixel[1];
155     else if (out_pixel[2]!=4'b0) final_out_pixel=out_pixel[2];
156     else if (out_pixel[3]!=4'b0) final_out_pixel=out_pixel[3];
157     else if (out_pixel[4]!=4'b0) final_out_pixel=out_pixel[4];
158     else if (out_pixel[5]!=4'b0) final_out_pixel=out_pixel[5]; //pattern has lowest
159     pixel priority
160     else final_out_pixel=4'b0;
161 end
162
163 always_comb begin //VRAM read multiplexer
164     //multiplex sprite attribute table reads
165     if ((hcount>=h_start[0]) && (hcount<h_start[1])) begin
166         ra_a=sprite_ra_a[0];
167         ra_g=sprite_ra_g[0];
168     end else if ((hcount>=h_start[1]) && (hcount<h_start[2])) begin
169         ra_a=sprite_ra_a[1];
170         ra_g=sprite_ra_g[1];
171     end else if ((hcount>=h_start[2]) && (hcount<h_start[3])) begin
172         ra_a=sprite_ra_a[2];
173         ra_g=sprite_ra_g[2];
174     end else if ((hcount>=h_start[3]) && (hcount<h_start[4])) begin
175         ra_a=sprite_ra_a[3];
176         ra_g=sprite_ra_g[3];
177     end else if (hcount>=h_start[4]) begin
178         ra_a=sprite_ra_a[4];
179         ra_g=sprite_ra_g[4];
180     end else begin //below should never run here
181         ra_a=5'b0;
182         ra_g=11'b0;
183     end
184 end
185
186 endmodule
187
188 module sprite_prep (input logic clk, reset,
189     input logic [10:0] h_start,
190     input logic [10:0] hcount,
191     input logic [9:0] vcount,
192     input logic VGA_BLANK_n,
193     input logic [4:0] base_addr, //base address in sprite attr table
194     input logic [7:0] dout_a,
195     input logic [7:0] dout_g,
196     output logic [4:0] ra_a,
197     output logic [10:0] ra_g,
198     output logic [3:0] out_pixel);
199
200     logic [8:0] down_counter; //8 bit wide down counter
201     logic [63:0] shift_reg; //64 bit wide shift register
202     logic [7:0] shift_pos; //position in shift reg to read pixel value from
203     logic [10:0] sprite_offset; //which row of a given sprite to display
204     logic [63:0] display_pixel; // determines whether sprite or background pixel is shown
205     logic [7:0] shift_reg_shift; //bit position in shift reg to write to (0-63, steps of
206         8)
207     assign out_pixel=display_pixel[3:0];
208
209     enum {IDLE, READ_VERT_POS, READ_VERT_POS_WAIT, READ_VERT_POS_WAIT2, READ_HORT_POS,
210         READ_HORT_POS_WAIT,
211         READ_SPRITE_ADDR, READ_SPRITE_ADDR_WAIT, READ_SPRITE_PIXELS_BASE,
212         READ_SPRITE_PIXELS_BASE_WAIT,
213         LOAD_SHIFT_REG, LOAD_SHIFT_REG_WAIT, SPRITES_LOADED, COUNT_DOWN, PREPARE_PIXELS }
214     state, state_next;
215
216

```



```

213 always_ff @(posedge clk) begin
214     state<=state_next;
215     if (reset) begin
216         state<=IDLE;
217         ra_g<=0;
218         ra_a<=0;
219     end
220
221
222     case (state)
223     IDLE: begin
224         display_pixel<=64'b0;
225         shift_reg<=64'b0;
226         shift_reg_shift<=8'h40; //dec=64 (actual value used is 8 less)
227         shift_pos<=8'h40; //dec=64 set shift position to start of shift regs (MSB) (
actual value used is 4 less)
228     end
229     READ_VERT_POS: begin
230         ra_a<=base_addr; //address of (starting) vertical position of sprite
231     end
232     READ_HORT_POS: begin
233         ra_a<=base_addr+5'b1; //address of horizontal position of sprite
234         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
235     end
236     READ_SPRITE_ADDR: begin //base address need right shift of 3 bits
237         ra_a<=base_addr+5'b10; //address of base address of sprite pixels in the
generator table //test using 0
238         down_counter<={dout_a, 1'b0}; //copy horizontal position into down counter
239     end
240     READ_SPRITE_PIXELS_BASE: begin //!!note: address no longer >> shifted by 3!!
241         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
242     end
243     LOAD_SHIFT_REG: begin
244         shift_reg<= ({56'b0, dout_g}<<(shift_reg_shift-8'h8)) | shift_reg; //store
left-most 8 pixels of sprite line
245         shift_reg_shift<=shift_reg_shift-8'h8; //minus 8
246         ra_g<=ra_g+1; //increment gen table address by one to read upcoming pixels
247     end
248     COUNT_DOWN: begin
249         //only down count every 2 hcounts
250         if (down_counter>9'b0 && VGA_BLANK_n && !hcount[0]) down_counter<=down_counter
-1;
251     end
252     PREPARE_PIXELS: begin
253         if (VGA_BLANK_n && !hcount[0]) begin
254             display_pixel<=(shift_reg>>(shift_pos-8'h4)); //Only 4 LSB of display_pixel
matter
255             shift_pos<=shift_pos-8'h4; //minus 4
256         end
257     end
258     endcase
259
260 end
261
262 always_comb begin
263     case (state)
264     IDLE: state_next = (hcount==h_start) ? READ_VERT_POS: IDLE;
265     READ_VERT_POS: state_next = READ_VERT_POS_WAIT; //extra cycle for
reading vertical position in attr table
266     READ_VERT_POS_WAIT: state_next = READ_VERT_POS_WAIT2; //ra_a update needs 2
cycles for some reason
267     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)
268     READ_HORT_POS: state_next = READ_HORT_POS_WAIT; //extra cycle for mem
read
269     READ_HORT_POS_WAIT: state_next = READ_SPRITE_ADDR;
270     READ_SPRITE_ADDR: state_next = READ_SPRITE_ADDR_WAIT; //extra cycle for mem
read

```

```

271     READ_SPRITE_ADDR_WAIT:    state_next = READ_SPRITE_PIXELS_BASE;
272     READ_SPRITE_PIXELS_BASE: state_next= READ_SPRITE_PIXELS_BASE_WAIT; //extra cycle
    for mem read
273     READ_SPRITE_PIXELS_BASE_WAIT: state_next= LOAD_SHIFT_REG;
274     LOAD_SHIFT_REG: state_next= LOAD_SHIFT_REG_WAIT;
275     LOAD_SHIFT_REG_WAIT: state_next= (shift_reg_shift==8'b0) ? SPRITES_LOADED:
LOAD_SHIFT_REG;
276
277     //if new vertical line started, begin down counting
278     SPRITES_LOADED: state_next= (hcount==11'b1111111) ? COUNT_DOWN : SPRITES_LOADED;
    //start at 127
279     COUNT_DOWN: state_next= (down_counter==9'b0) ? PREPARE_PIXELS: COUNT_DOWN;
280     PREPARE_PIXELS: state_next= (shift_pos==8'b0) ? IDLE : PREPARE_PIXELS;
281         default:    state_next = IDLE;
282     endcase
283     end
284 endmodule
285
286 module pattern_prep (input logic clk, reset,
287     input logic [10:0] hcount,
288     input logic [9:0] vcount,
289     input logic VGA_BLANK_n,
290     input logic [7:0] dout_n,
291     input logic [7:0] dout_g,
292     output logic [11:0] ra_n,
293     output logic [10:0] ra_g,
294     output logic [3:0] out_pixel);
295
296     logic [2047:0] shift_reg; //8*64*4 bit wide shift register
297     logic [11:0] shift_pos; //position in shift reg to read pixel value from
298     logic [10:0] pattern_row_offset; //which of 8 of a given pattern to display
299     logic [2047:0] display_pixel; // determines whether sprite or background pixel is
    shown
300     logic [11:0] shift_reg_shift; //bit position in shift reg to write to (0-63, steps
    of 8)
301     logic [7:0] tile_total_counter; //counts the total number of tiles that has been
    loaded into shift reg
302     logic [7:0] tile_pixel_counter; //counts the number of tile pixel rows that has been
    loaded
303     assign out_pixel=display_pixel[3:0];
304
305     parameter [11:0] v_start=12'h0; //vertical position where first pattern begins
306     parameter [7:0] tiles_per_row=8'd64; //number of tiles per row
307     //parameter [7:0] tiles_per_col=8'h18; //number of tiles per column
308     parameter [11:0] name_table_addr_mask={6'b111111, 6'b0};
309
310     enum {IDLE, READ_TILE_ADDR_BASE, READ_TILE_ADDR_BASE_WAIT, READ_PATT_PIXELS_BASE,
    READ_PATT_PIXELS_BASE_WAIT,
311     LOAD_SHIFT_REG, LOAD_SHIFT_REG_WAIT, READ_TILE_NEXT, READ_TILE_NEXT_WAIT,
    PATT_LOADED, PREPARE_PIXELS }
312     state, state_next;
313
314
315     always_ff @(posedge clk) begin
316         state<=state_next;
317         if (reset) begin
318             state<=IDLE;
319             ra_n<=0;
320             ra_g<=0;
321         end
322
323         case (state)
324             IDLE: begin
325                 tile_total_counter<=8'b0;
326                 tile_pixel_counter<=8'b0;
327                 display_pixel<=2048'b0;
328                 shift_reg<=2048'b0;
329                 shift_reg_shift<=12'b100000000000; //dec=2048 (actual value used is 8 less)
330                 shift_pos<=12'b100000000000; // dec=2048 set shift position to start of shift
    regs (MSB) (actual value used is 4 less)
331

```

```

332     end
333     READ_TILE_ADDR_BASE: begin
334         ra_n<={{2'b0, vcount}-v_start}<<3) & name_table_addr_mask; //get address of (
starting) tile pixel address in name table
335         pattern_row_offset<={{8'b0, vcount[2:0]-v_start[2:0]}; //which of 8 pixel rows
to access
336     end
337
338     READ_PATT_PIXELS_BASE: begin //!!note: address no longer >> shifted by 3!!
339         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
340     end
341
342     READ_PATT_PIXELS_BASE_WAIT: begin //!!note: address no longer >> shifted by 3!!
343         ra_g<=ra_g+1;
344     end
345
346     LOAD_SHIFT_REG: begin //first time: gets ra_g pixels_base stage and not
base_wait stage
347         shift_reg<={{2040'b0, dout_g}<<(shift_reg_shift-12'h8)} | shift_reg; //store
left-most 8 pixels of sprite line
348         shift_reg_shift<=shift_reg_shift-12'h8; //minus 8
349         ra_g<=ra_g+1; //increment gen table address by one to read upcoming pixels
350         tile_pixel_counter<=tile_pixel_counter+8'b1;
351     end
352     READ_TILE_NEXT: begin
353         ra_n<=ra_n+1; //increment name table address
354         tile_pixel_counter<=8'b0;
355         tile_total_counter<=tile_total_counter+8'b1;
356     end
357
358     PREPARE_PIXELS: begin
359         if (VGA_BLANK_n && !hcount[0]) begin
360             display_pixel<=(shift_reg>>(shift_pos-12'h4)); //Only 4 LSB of display_pixel
matter
361             shift_pos<=shift_pos-12'h4; //minus 4
362         end
363     end
364 endcase
365
366 end
367
368 always_comb begin
369     case (state)
370         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
371         READ_TILE_ADDR_BASE: state_next = READ_TILE_ADDR_BASE_WAIT; //extra
cycle for reading vertical position in attr table
372         READ_TILE_ADDR_BASE_WAIT: state_next = READ_PATT_PIXELS_BASE; //check if true:
ra_a update needs 2 cycles for some reason
373         READ_PATT_PIXELS_BASE: state_next = READ_PATT_PIXELS_BASE_WAIT;
374         READ_PATT_PIXELS_BASE_WAIT: state_next= LOAD_SHIFT_REG;
375         LOAD_SHIFT_REG: state_next= (tile_pixel_counter==8'h3) ? READ_TILE_NEXT:
LOAD_SHIFT_REG;
376         READ_TILE_NEXT: state_next=READ_TILE_NEXT_WAIT;
377         READ_TILE_NEXT_WAIT: state_next=(tile_total_counter==tiles_per_row)? PATT_LOADED
: READ_PATT_PIXELS_BASE;
378
379         //if new vertical line started, begin down counting
380         PATT_LOADED: state_next= (hcount==11'd127) ? PREPARE_PIXELS : PATT_LOADED;
381         PREPARE_PIXELS: state_next= (shift_pos==12'b0 || vcount>10'd480) ? IDLE :
PREPARE_PIXELS;
382         default: state_next = IDLE;
383     endcase
384 end
385 endmodule
386
387 module sprite_attr_table( //stores sprite information (x, y, name, color)
388     //x and y position has to be a multiple (2x) of hcount/vcount since only 8 bits
389     input logic clk,
390     input logic [4:0] ra, wa, //change later

```

```

391     input logic we,
392     input logic [7:0] din,
393     output logic [7:0] dout);
394
395     logic [7:0] mem [31:0];
396
397     always_ff @(posedge clk) begin
398         if (we) mem[wa] <= din;
399         dout <= mem[ra];
400     end
401 endmodule
402
403 module sprite_gen_table( //stores 16x 16x16 sprites (only 10 needed)
404     input logic clk,
405     input logic [10:0] ra, wa, //change later
406     input logic we,
407     input logic [7:0] din,
408     output logic [7:0] dout);
409
410     logic [7:0] mem [2047:0]; //128 8 bit words need per sprite: 64 bits per pixel row
411
412     always_ff @(posedge clk) begin
413         if (we) mem[wa] <= din;
414         dout <= mem[ra];
415     end
416 endmodule
417
418 module patt_name_table( //stores 8 bit address of tiles on each row
419     input logic clk,
420     input logic [11:0] ra, wa, //12 bit addr
421     input logic we,
422     input logic [7:0] din,
423     output logic [7:0] dout);
424
425     logic [7:0] mem [4095:0];
426
427     always_ff @(posedge clk) begin
428         if (we) mem[wa] <= din;
429         dout <= mem[ra];
430     end
431 endmodule
432
433 module patt_gen_table( //stores 8x8 patterns
434     input logic clk,
435     input logic [10:0] ra, wa,
436     input logic we,
437     input logic [7:0] din,
438     output logic [7:0] dout);
439
440     logic [7:0] mem [2047:0]; //32 8 bit words need per pattern: 4 table rows (32 bits)
441         per pixel row
442
443     always_ff @(posedge clk) begin
444         if (we) mem[wa] <= din;
445         dout <= mem[ra];
446     end
447 endmodule
448
449 module color_lut(input logic [3:0] color_code,
450     output logic [23:0] rgb_val);
451     always_comb
452         case(color_code)
453             4'h1: rgb_val=24'hfef104; //yellow pac-man
454             4'h2: rgb_val=24'hfe0e03; //red ghost
455             4'h3: rgb_val=24'hfeb846; //orange ghost
456             4'h4: rgb_val=24'h00ecfe; //cyan ghost
457             4'h5: rgb_val=24'hfdbff9; //pink ghost
458             4'h6: rgb_val=24'he5dfef; //ghost eyes (whites)
459             4'h7: rgb_val=24'h1e26b8; //blue: ghost eye iris, maze walls, blue ghosts
460             4'h8: rgb_val=24'hffc0b7; //salmon food pellets
461             4'h9: rgb_val=24'hffffff; //white text

```

```

461     default: rgb_val=24'hffffff; //if something goes wrong, use white to make it
         obvious
462     endcase
463
464 endmodule
465
466
467
468 module vga_counters(
469     input logic      clk50, reset,
470     output logic [10:0] hcount, // hcount[10:1] is pixel column
471     output logic [9:0] vcount, // vcount[9:0] is pixel row
472     output logic     VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);
473
474 /*
475 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
476 *
477 * HCOUNT 1599 0          1279          1599 0
478 *
479 * -----|-----|-----|-----
480 * |-----|-----|-----|-----
481 *
482 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
483 *
484 * |-----|-----|-----|-----
485 * |____|          VGA_HS          |____|
486 */
487 // Parameters for hcount
488 parameter HACTIVE      = 11'd 1280,
489           HFRONT_PORCH = 11'd 32,
490           HSYNC        = 11'd 192,
491           HBACK_PORCH  = 11'd 96,
492           HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
493                       HBACK_PORCH; // 1600
494
495 // Parameters for vcount
496 parameter VACTIVE      = 10'd 480,
497           VFRONT_PORCH = 10'd 10,
498           VSYNC        = 10'd 2,
499           VBACK_PORCH  = 10'd 33,
500           VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
501                       VBACK_PORCH; // 525
502
503 logic endOfLine;
504
505 always_ff @(posedge clk50 or posedge reset)
506     if (reset) hcount <= 0;
507     else if (endOfLine) hcount <= 0;
508     else hcount <= hcount + 11'd 1;
509
510 assign endOfLine = hcount == HTOTAL - 1;
511
512 logic endOfField;
513
514 always_ff @(posedge clk50 or posedge reset)
515     if (reset) vcount <= 0;
516     else if (endOfLine)
517         if (endOfField) vcount <= 0;
518         else vcount <= vcount + 10'd 1;
519
520 assign endOfField = vcount == VTOTAL - 1;
521
522 // Horizontal sync: from 0x520 to 0x5DF (0x57F)
523 // 101 0010 0000 to 101 1101 1111 (active LOW during 1312-1503) (192 cycles)
524 assign VGA_HS = !( hcount[10:8] == 3'b101 & !(hcount[7:5] == 3'b111));
525 assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
526
527 assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused
528
529 // Horizontal active: 0 to 1279          Vertical active: 0 to 479
530 // 101 0000 0000 1280          01 1110 0000 480
531 // 110 0011 1111 1599          10 0000 1100 524

```

```

531 assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
532      !( vcount[9] | (vcount[8:5] == 4'b1111) );
533
534 /* VGA_CLK is 25 MHz
535  *
536  * clk50    __|__|__|__|__|__
537  *
538  * hcount[0]__|_____|_____|__
539  */
540
541 assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive
542
543 endmodule

```

Listing 1: vga-ball.sv

4.2 Hardware Testbench

```

1  `timescale 1ns/1ps
2  `define HALF_CLOCK_PERIOD #20 //design uses 50MHz clock
3
4  module testbench();
5      reg clk;
6      reg rst;
7      reg [31:0] writedata;
8      reg write;
9      reg chipselect;
10     reg [3:0] address;
11     wire [7:0] VGA_R, VGA_G, VGA_B;
12     reg [15:0] writedata_address;
13     reg [15:0] name_writedata_address;
14     reg name_value;
15     reg [3:0] color_counter;
16     reg [7:0] pg_table_data;
17
18     reg [1:0] table_select;
19     reg [7:0] gen_table_base_addr;
20     reg [7:0] v_pos;
21     reg [7:0] h_pos;
22     reg [15:0] j_counter1;
23     reg [15:0] i_counter1;
24     reg [7:0] table_data;
25
26
27
28     integer i, j, k;
29
30     vga_ball vga_ball0 ( .clk(clk), .reset(rst), .writedata(writedata), .write(write),
31     .chipselect(chipselect), .address(address),
32     .VGA_R(VGA_R), .VGA_G(VGA_G), .VGA_B(VGA_B), .VGA_CLK(), .VGA_HS(), .VGA_VS(), .
33     VGA_BLANK_n(), .VGA_SYNC_n());
34
35     always begin
36         `HALF_CLOCK_PERIOD;
37         clk = ~clk;
38     end
39
40     initial begin
41         // register setup
42         clk = 0;
43         rst = 1;
44         chipselect=0;
45         write=0;
46         writedata_address=0;
47         name_writedata_address=0;
48         color_counter=0;
49         name_value=1;
50         pg_table_data=0;
51
52         table_select=0;

```

```

51 gen_table_base_addr=0;
52 v_pos=0;
53 h_pos=0;
54 j_counter1=0;
55 i_counter1=0;
56 table_data=0;
57 @(posedge clk);
58
59 @(negedge clk); // release rst
60 rst = 0;
61
62 @(posedge clk); // start the first cycle
63
64 //start TB
65 //Pattern TB
66 //write pattern name table
67 chipselect=1;
68 write=1;
69 table_select=2'b0;
70
71 //Load pattern name table
72 for (j=0 ; j<60; j=j+1) begin //# of cols
73     for (i=0 ; i<64; i=i+1) begin
74         writedata={name_value , 6'b0, name_writedata_address , table_select};
75         name_value=~name_value;
76         @(posedge clk);
77         name_writedata_address=name_writedata_address+1;
78     end
79     name_value=~name_value; //extra toggle so each adjacent row has different
patterns
80 end
81
82 table_select=2'b1;
83 //write pattern generator table values
84 for (j=0 ; j<2; j=j+1) begin //one of two patterns
85     for (i=0 ; i<32; i=i+1) begin
86         if (j==0) begin
87             if (i<7) pg_table_data=8'b0011_0011; //orange sprite pixel data
88             else if (i>15) pg_table_data=8'b0101_0101; //pink sprite pixel data
89             else pg_table_data=8'b0000_0000; //transparent
90         end else
91             if (i%3) pg_table_data=8'b0000_0000; //transparent
92             else pg_table_data= 8'b0001_0001; //yellow
93         @(posedge clk);
94         writedata={pg_table_data, 6'b0, writedata_address , table_select};
95         writedata_address=writedata_address+1;
96     end
97 end
98
99 //full 5 sprites tb
100 @(posedge clk);
101
102 chipselect=1;
103 write=1;
104
105 //write to sprite attribute table
106 table_select=2'b10;
107 for (j=0 ; j<5; j=j+1) begin
108     i_counter1=0;
109     for (i=0 ; i<4; i=i+1) begin
110         case (i)
111             0 : begin //sprite vertical position
112                 writedata={v_pos , 6'b0, (j_counter1<<2) + i_counter1 , table_select};
113             end
114             1 : begin //sprite horizontal position
115                 writedata={h_pos , 6'b0, (j_counter1<<2) + i_counter1 , table_select};
116             end
117             2 : begin //gen table base addr
118                 writedata={gen_table_base_addr , 6'b0, (j_counter1<<2) + i_counter1 ,
table_select};
119             end

```

```

120     3 : begin //unused
121         writedata={8'b0, 6'b0, (j_counter1<<2) + i_counter1, table_select};
122     end
123 endcase
124 @(posedge clk); //wait 1 cycle
125 i_counter1=i_counter1+1;
126 end
127 if (j==3) begin //one more increment to get to edge (630, 460)
128     v_pos=v_pos+8'd46;
129     h_pos=h_pos+8'd50;
130 end
131 gen_table_base_addr=gen_table_base_addr+1;
132 j_counter1=j_counter1+1;
133 v_pos=v_pos+8'd46;
134 h_pos=h_pos+8'd50;
135 end
136
137
138 //write to sprite generator table
139 writedata_address=0;
140 table_select=2'b11;
141 for (k=0 ; k<5; k=k+1) begin
142     for (j=0 ; j<16; j=j+1) begin
143         for (i=0 ; i<8; i=i+1) begin
144             @(posedge clk); //wait 1 cycle
145             if (k%2) begin
146                 if (i<4) table_data=8'b0011_0011; //orange sprite pixel data
147                 else if (i>3 && i <6) table_data=8'b0000_0000; //transparent
148                 else table_data=8'b0101_0101; //pink sprite pixel data
149             end else
150                 if (i<4) table_data=8'b0000_0000; //transparent
151                 else if (i>3 && i <6) table_data=8'b0001_0001; //yellow
152                 else table_data=8'b0010_0010; //red
153             writedata={table_data, 6'b0, writedata_address, table_select};
154             writedata_address=writedata_address+1;
155         end
156     end
157 end
158
159 for (i=0 ; i<850000; i=i+1) begin
160     @(posedge clk); // next cycle
161 end
162 $finish;
163 end
164 endmodule // testbench

```

Listing 2: vga-ball-tb.sv

```

1 run:
2     vsim -do "runsim.do"
3
4 clean:
5     rm -rf work modelsim.ini
6     rm -rf *.log *.syn *.rpt *.mr *.nl.v *.sdf *.svf *.vcd transcript *.wlf
7     rm -rf *.result

```

Listing 3: Testbench Makefile

```

1 #####
2 # Modelsim do file to run simulation
3 # MS 7/2015
4 #####
5
6 vlib work
7 vmap work work
8
9 # include netlist and testbench files
10 vlog +acc -incr ./vga_ball.sv
11 vlog +acc -incr ./vga_ball_tb.sv
12
13 # run simulation

```



```

14 vsim -t ps -lib work testbench
15 do waveform.do
16 run -all

```

Listing 4: runsim.do

```

1 onerror {resume}
2 quietly WaveActivateNextPane {} 0
3 add wave -noupdate /testbench/clock
4 add wave -noupdate /testbench/rst
5 add wave -noupdate /testbench/i
6 add wave -noupdate /testbench/writedata
7 add wave -noupdate /testbench/table_data
8 add wave -noupdate /testbench/vga_ball0/din_a
9 add wave -noupdate /testbench/vga_ball0/wa_a
10 add wave -noupdate /testbench/vga_ball0/din_g
11 add wave -noupdate /testbench/vga_ball0/wa_g
12 add wave -noupdate /testbench/vga_ball0/sp0/ra_a
13 add wave -noupdate /testbench/vga_ball0/sp0/dout_a
14 add wave -noupdate /testbench/vga_ball0/sp0/ra_g
15 add wave -noupdate /testbench/vga_ball0/sp0/dout_g
16 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/down_counter
17 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/shift_pos
18 add wave -noupdate /testbench/vga_ball0/sp0/sprite_offset
19 add wave -noupdate /testbench/vga_ball0/sp0/shift_reg
20 add wave -noupdate /testbench/vga_ball0/sp0/display_pixel
21 add wave -noupdate /testbench/vga_ball0/sp0/state
22 add wave -noupdate /testbench/vga_ball0/sp0/state_next
23
24
25 add wave -noupdate /testbench/vga_ball0/sp4/ra_a
26 add wave -noupdate /testbench/vga_ball0/sp4/dout_a
27 add wave -noupdate /testbench/vga_ball0/sp4/ra_g
28 add wave -noupdate /testbench/vga_ball0/sp4/dout_g
29 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp4/down_counter
30 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp4/shift_pos
31 add wave -noupdate /testbench/vga_ball0/sp4/sprite_offset
32 add wave -noupdate /testbench/vga_ball0/sp4/shift_reg
33 add wave -noupdate /testbench/vga_ball0/sp4/display_pixel
34 add wave -noupdate /testbench/vga_ball0/sp4/state
35 add wave -noupdate /testbench/vga_ball0/sp4/state_next
36
37 add wave -noupdate /testbench/vga_ball0/din_n
38 add wave -noupdate /testbench/vga_ball0/wa_n
39 add wave -noupdate /testbench/vga_ball0/din_pg
40 add wave -noupdate /testbench/vga_ball0/wa_pg
41 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/ra_n
42 add wave -noupdate /testbench/vga_ball0/pp0/dout_n
43 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/ra_g
44 add wave -noupdate /testbench/vga_ball0/pp0/dout_g
45 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/shift_pos
46 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/shift_reg_shift
47 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/tile_total_counter
48 add wave -noupdate -radix unsigned /testbench/vga_ball0/pp0/tile_pixel_counter
49 add wave -noupdate /testbench/vga_ball0/pp0/pattern_row_offset
50 add wave -noupdate /testbench/vga_ball0/pp0/shift_reg
51 add wave -noupdate /testbench/vga_ball0/pp0/out_pixel
52 add wave -noupdate /testbench/vga_ball0/pp0/state
53 add wave -noupdate /testbench/vga_ball0/pp0/state_next
54
55 add wave -noupdate -radix unsigned /testbench/write
56 add wave -noupdate -radix unsigned /testbench/chipselect
57 add wave -noupdate -radix unsigned /testbench/address
58 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/vcount
59 add wave -noupdate -radix unsigned /testbench/vga_ball0/sp0/hcount
60 add wave -noupdate /testbench/VGA_R
61 add wave -noupdate /testbench/VGA_G
62 add wave -noupdate /testbench/VGA_B
63 TreeUpdate [SetDefaultTree]
64 WaveRestoreCursors {{Cursor 1} {3 ns} 0}
65 quietly wave cursor active 1

```

```

66 configure wave -namecolwidth 223
67 configure wave -valuecolwidth 89
68 configure wave -justifyvalue left
69 configure wave -signalnamewidth 0
70 configure wave -snapdistance 10
71 configure wave -datasetprefix 0
72 configure wave -rowmargin 4
73 configure wave -childrowmargin 2
74 configure wave -gridoffset 0
75 configure wave -gridperiod 1
76 configure wave -griddelta 40
77 configure wave -timeline 0
78 configure wave -timelineunits ns
79 update
80 WaveRestoreZoom {0 ns} {12 ns}

```

Listing 5: waveformat.do

4.3 Software

```

1 ifneq (${KERNELRELEASE},)
2
3 # KERNELRELEASE defined: we are being compiled as part of the Kernel
4     obj-m := vga_ball.o
5
6 else
7
8 # We are being compiled as a module: use the Kernel build system
9
10 KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
11     PWD := $(shell pwd)
12 #
13 default: module pacman
14 # default: pacman
15
16 CFLAGS = -Wall
17
18 OBJECTS = pacman.o gamepad.o pattern.o sprite.o vga_ball_user.o map.o gameplay.o
19
20 pacman: $(OBJECTS)
21     cc $(CFLAGS) -o pacman $(OBJECTS) -lusb-1.0 -pthread
22
23 pacman.o: pacman.c gamepad.h pattern.h map.h vga_ball_user.h
24
25 gamepad.o: gamepad.h
26
27 pattern.o: pattern.c pattern.h color.h vga_ball_user.h
28
29 sprite.o: sprite.c sprite.h
30
31 vga_ball_user.o: vga_ball_user.c vga_ball_user.h
32
33 map.o: map.c map.h pattern.h
34
35 gameplay.o: gameplay.c gameplay.h map.h pattern.h
36
37
38 module:
39     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules
40
41 clean:
42     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
43     ${RM} -rf *.o pacman
44
45 # TARFILES = Makefile README vga_ball.h vga_ball.c hello.c
46 # TARFILE = lab3-sw.tar.gz
47 # .PHONY : tar
48 # tar : $(TARFILE)
49 #
50 # $(TARFILE) : $(TARFILES)

```

```

51 # tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)
52
53 endif

```

Listing 6: Makefile

```

1 #include "gamepad.h"
2 #include "gameplay.h"
3 #include "map.h"
4 #include "pattern.h"
5 #include "sprite.h"
6 #include "vga_ball_user.h"
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <time.h>
10 #include <unistd.h>
11
12 void gameplay_listener(gamepad_button_event_t e, gamepad_button_t b) {
13     switch (get_game_stage()) {
14         case STAGE_MENU:
15             if (e == GAMEPAD_KEY_UP && b == GAMEPAD_A) {
16                 press_start_game();
17             }
18             break;
19         case STAGE_IN_GAME:
20             if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_LEFT) {
21                 set_pacman_dir(DIR_LEFT);
22             } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_RIGHT) {
23                 set_pacman_dir(DIR_RIGHT);
24             } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_UP) {
25                 set_pacman_dir(DIR_UP);
26             } else if (e == GAMEPAD_KEY_DOWN && b == GAMEPAD_DOWN) {
27                 set_pacman_dir(DIR_DOWN);
28             }
29             break;
30         case STAGE_END_GAME:
31             break;
32     }
33 }
34
35 void show_ui() {
36     set_map_at(25, 1, PAT_W);
37     set_map_at(25, 2, PAT_E);
38     set_map_at(25, 3, PAT_L);
39     set_map_at(25, 4, PAT_C);
40     set_map_at(25, 5, PAT_O);
41     set_map_at(25, 6, PAT_M);
42     set_map_at(25, 7, PAT_E);
43
44     set_map_at(27, 1, PAT_T);
45     set_map_at(27, 2, PAT_O);
46
47     set_map_at(29, 1, PAT_D);
48     set_map_at(29, 2, PAT_E);
49     set_map_at(29, 3, PAT_1);
50     set_map_at(29, 5, PAT_S);
51     set_map_at(29, 6, PAT_O);
52     set_map_at(29, 7, PAT_C);
53
54     set_map_at(31, 1, PAT_P);
55     set_map_at(31, 2, PAT_A);
56     set_map_at(31, 3, PAT_C);
57     set_map_at(31, 4, PAT_M);
58     set_map_at(31, 5, PAT_A);
59     set_map_at(31, 6, PAT_N);
60 }
61
62 void hide_side() {
63     for (int r = 25; r <= 31; r++) {
64         for (int c = 50; c <= 55; c++) {
65             set_map_at(r, c, PAT_BACKGROUND);

```

```

66     }
67     }
68 }
69
70 void show_help() {
71     static int counter = 0;
72     static int flip = 1;
73     counter = (counter + 1) % 800;
74
75     if (counter == 0)
76         flip *= -1;
77
78     if (flip == 1) {
79         set_map_at(25, 51, PAT_P);
80         set_map_at(25, 52, PAT_R);
81         set_map_at(25, 53, PAT_E);
82         set_map_at(25, 54, PAT_S);
83         set_map_at(25, 55, PAT_S);
84
85         set_map_at(27, 51, PAT_A);
86
87         set_map_at(29, 51, PAT_T);
88         set_map_at(29, 52, PAT_O);
89
90         set_map_at(31, 51, PAT_S);
91         set_map_at(31, 52, PAT_T);
92         set_map_at(31, 53, PAT_A);
93         set_map_at(31, 54, PAT_R);
94         set_map_at(31, 55, PAT_T);
95     } else {
96         hide_side();
97     }
98 }
99
100 void show_game_over() {
101     static int counter = 0;
102     static int flip = 1;
103     counter = (counter + 1) % 800;
104
105     if (counter == 0)
106         flip *= -1;
107
108     if (flip == 1) {
109         set_map_at(27, 51, PAT_G);
110         set_map_at(27, 52, PAT_A);
111         set_map_at(27, 53, PAT_M);
112         set_map_at(27, 54, PAT_E);
113
114         set_map_at(29, 51, PAT_O);
115         set_map_at(29, 52, PAT_V);
116         set_map_at(29, 53, PAT_E);
117         set_map_at(29, 54, PAT_R);
118     } else {
119         hide_side();
120     }
121 }
122
123 void show_congrats() {
124     static int counter = 0;
125     static int flip = 1;
126     counter = (counter + 1) % 800;
127
128     if (counter == 0)
129         flip *= -1;
130
131     if (flip == 1) {
132         set_map_at(27, 51, PAT_N);
133         set_map_at(27, 52, PAT_E);
134         set_map_at(27, 53, PAT_W);
135
136         set_map_at(29, 51, PAT_H);

```

```

137     set_map_at(29, 52, PAT_I);
138     set_map_at(29, 53, PAT_G);
139     set_map_at(29, 54, PAT_H);
140
141     set_map_at(31, 51, PAT_S);
142     set_map_at(31, 52, PAT_C);
143     set_map_at(31, 53, PAT_O);
144     set_map_at(31, 54, PAT_R);
145     set_map_at(31, 55, PAT_E);
146 } else {
147     hide_side();
148 }
149 }
150
151 int main() {
152     srand(time(NULL));
153     vga_ball_init();
154     gamepad_init();
155
156     load_pattern_bitmaps();
157     load_sprite_bitmaps();
158
159     clear_screen();
160     show_ui();
161     setup_map();
162     setup_map_foods();
163     setup_map_score();
164
165     setup_game();
166
167     gamepad_set_listener(&gameplay_listener);
168
169     while (1) {
170         switch (get_game_stage()) {
171             case STAGE_MENU:
172                 show_help();
173                 if (ghost_trapped_move_timer()) {
174                     move_ghosts_trapped();
175                 }
176                 break;
177             case STAGE_IN_GAME:
178                 hide_side();
179
180                 if (pacman_move_timer()) {
181                     move_pacman();
182                 }
183                 if (blink_timer()) {
184                     animate_pacman();
185                 }
186                 if (ghost_trapped_move_timer()) {
187                     move_ghosts_trapped();
188                     move_ghosts_release();
189                 }
190                 if (ghost_move_timer()) {
191                     move_ghosts();
192                 }
193
194                 if (ghost_release_timer()) {
195                     release_next_ghost();
196                 }
197
198                 eat_food();
199                 ghosts_catch_pacman();
200
201                 scatter_timer();
202                 break;
203             case STAGE_END_GAME: {
204                 // show game-over
205                 uint32_t counter = 0;
206                 while (counter < 3000) {
207                     counter++;

```

```

208     show_game_over();
209     usleep(1000);
210 }
211 }
212
213     hide_side();
214
215     if (beat_best_score()) {
216         // show congrats
217         uint32_t counter = 0;
218         while (counter < 3000) {
219             counter++;
220             show_congrats();
221             usleep(1000);
222         }
223     }
224     update_scores();
225     reset_game();
226     break;
227 }
228
229     usleep(1000);
230 }
231 return 1;
232 }

```

Listing 7: pacman.c

```

1 #ifndef _PACMAN_GAMEPAD_H
2 #define _PACMAN_GAMEPAD_H
3
4 #include <stdint.h>
5
6 struct gamepad_packet {
7     uint8_t reserved0;
8     uint8_t reserved1;
9     uint8_t reserved2;
10    uint8_t dir_x;
11    uint8_t dir_y;
12    uint8_t primary;
13    uint8_t secondary;
14 };
15
16 typedef uint16_t gamepad_button_t;
17 #define GAMEPAD_LEFT (((gamepad_button_t)1) << 0)
18 #define GAMEPAD_RIGHT (((gamepad_button_t)1) << 1)
19 #define GAMEPAD_UP (((gamepad_button_t)1) << 2)
20 #define GAMEPAD_DOWN (((gamepad_button_t)1) << 3)
21 #define GAMEPAD_X (((gamepad_button_t)1) << 4)
22 #define GAMEPAD_Y (((gamepad_button_t)1) << 5)
23 #define GAMEPAD_A (((gamepad_button_t)1) << 6)
24 #define GAMEPAD_B (((gamepad_button_t)1) << 7)
25 #define GAMEPAD_L (((gamepad_button_t)1) << 8)
26 #define GAMEPAD_R (((gamepad_button_t)1) << 9)
27 #define GAMEPAD_SELECT (((gamepad_button_t)1) << 10)
28 #define GAMEPAD_START (((gamepad_button_t)1) << 11)
29
30 #define GAMEPAD_DEFAULT ((gamepad_button_t)0)
31
32 typedef enum { GAMEPAD_KEY_DOWN, GAMEPAD_KEY_UP } gamepad_button_event_t;
33
34 void gamepad_init();
35 void gamepad_destroy();
36 void gamepad_set_listener(void (*listener)(gamepad_button_event_t,
37                                           gamepad_button_t));
38
39 #endif

```

Listing 8: gamepad.h

```

1 #include "gamepad.h"

```

```

2 #include <libusb-1.0/libusb.h>
3 #include <pthread.h>
4 #include <stdbool.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <unistd.h>
9
10 /* ----- Gamepad USB info ----- */
11 #define GAMEPAD_ID_VENDOR 0x79
12 #define GAMEPAD_ID_PRODUCT 0x11
13
14 /* Private function declarations */
15 void *gamepad_worker(void *arg);
16 void gamepad_generate_events(gamepad_button_t prev, gamepad_button_t next);
17 void gamepad_set_buttons(gamepad_button_t buttons);
18 struct libusb_device_handle *gamepad_open(uint8_t *endpoint_address);
19 gamepad_button_t gamepad_decode_packet(struct gamepad_packet packet);
20
21 /* ----- States ----- */
22 typedef struct {
23     /* control */
24     pthread_mutex_t mu;
25     pthread_t tid;
26     bool dead;
27
28     /* current button state */
29     gamepad_button_t buttons;
30
31     /* usb */
32     uint8_t endpoint;
33     struct libusb_device_handle *handle;
34
35     /* called by gamepad_worker() */
36     void (*listener)(gamepad_button_event_t e, gamepad_button_t bs);
37 } gamepad_state_t;
38
39 static gamepad_state_t gp;
40
41 /* ----- Implementations ----- */
42 void gamepad_init() {
43     int error;
44
45     pthread_mutex_init(&gp.mu, NULL);
46     pthread_mutex_lock(&gp.mu);
47
48     gp.dead = false;
49     gp.listener = NULL;
50     gp.buttons = GAMEPAD_DEFAULT;
51
52     if ((gp.handle = gamepad_open(&gp.endpoint)) == NULL) {
53         fprintf(stderr, "Did not find a gamepad\n");
54         exit(1);
55     }
56
57     if ((error = pthread_create(&gp.tid, NULL, &gamepad_worker, NULL)) != 0) {
58         fprintf(stderr, "Gamepad worker could not be created: %s\n",
59             strerror(error));
60         exit(1);
61     }
62
63     pthread_mutex_unlock(&gp.mu);
64
65     printf("Gamepad initialized\n");
66 }
67
68 void gamepad_destroy() {
69     pthread_mutex_lock(&gp.mu);
70     gp.dead = true;
71     pthread_mutex_unlock(&gp.mu);
72

```

```

73 pthread_join(gp.tid, NULL);
74 pthread_mutex_destroy(&gp.mu);
75
76 printf("Gamepad destroyed\n");
77 }
78
79 void gamepad_set_listener(void (*listener)(gamepad_button_event_t,
80                                         gamepad_button_t)) {
81     pthread_mutex_lock(&gp.mu);
82     gp.listener = listener;
83     pthread_mutex_unlock(&gp.mu);
84
85     printf("Set gamepad listener\n");
86 }
87
88 void *gamepad_worker(void *arg) {
89     struct gamepad_packet packet;
90     gamepad_button_t buttons;
91     int transferred;
92
93     /* Handle button data */
94     while (true) {
95         pthread_mutex_lock(&gp.mu);
96
97         /* exit worker if dead */
98         if (gp.dead) {
99             pthread_mutex_unlock(&gp.mu);
100            break;
101        }
102
103        /* retrieve */
104        libusb_interrupt_transfer(gp.handle, gp.endpoint, (unsigned char *)&packet,
105                                sizeof(packet), &transferred, 0);
106        buttons = gamepad_decode_packet(packet);
107
108        /* process */
109        gamepad_generate_events(gp.buttons, buttons);
110        gp.buttons = buttons;
111
112        pthread_mutex_unlock(&gp.mu);
113        usleep(100);
114    }
115
116    printf("Gamepad worker exited\n");
117    return NULL;
118 }
119
120 void gamepad_generate_events(gamepad_button_t prev, gamepad_button_t next) {
121     /* no need to generate event if no one cares */
122     if (gp.listener == NULL)
123         return;
124
125     /* 1. KEY_DOWN */
126     if (next & GAMEPAD_LEFT)
127         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_LEFT);
128     if (next & GAMEPAD_RIGHT)
129         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_RIGHT);
130     if (next & GAMEPAD_UP)
131         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_UP);
132     if (next & GAMEPAD_DOWN)
133         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_DOWN);
134     if (next & GAMEPAD_X)
135         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_X);
136     if (next & GAMEPAD_Y)
137         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_Y);
138     if (next & GAMEPAD_A)
139         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_A);
140     if (next & GAMEPAD_B)
141         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_B);
142     if (next & GAMEPAD_L)
143         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_L);

```



```

144     if (next & GAMEPAD_R)
145         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_R);
146     if (next & GAMEPAD_SELECT)
147         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_SELECT);
148     if (next & GAMEPAD_START)
149         gp.listener(GAMEPAD_KEY_DOWN, GAMEPAD_START);
150
151     /* 2. KEY_UP */
152     if (!(next & GAMEPAD_LEFT) && (prev & GAMEPAD_LEFT))
153         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_LEFT);
154     if (!(next & GAMEPAD_RIGHT) && (prev & GAMEPAD_RIGHT))
155         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_RIGHT);
156     if (!(next & GAMEPAD_UP) && (prev & GAMEPAD_UP))
157         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_UP);
158     if (!(next & GAMEPAD_DOWN) && (prev & GAMEPAD_DOWN))
159         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_DOWN);
160     if (!(next & GAMEPAD_X) && (prev & GAMEPAD_X))
161         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_X);
162     if (!(next & GAMEPAD_Y) && (prev & GAMEPAD_Y))
163         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_LEFT);
164     if (!(next & GAMEPAD_A) && (prev & GAMEPAD_A))
165         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_A);
166     if (!(next & GAMEPAD_B) && (prev & GAMEPAD_B))
167         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_B);
168     if (!(next & GAMEPAD_L) && (prev & GAMEPAD_L))
169         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_L);
170     if (!(next & GAMEPAD_R) && (prev & GAMEPAD_R))
171         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_R);
172     if (!(next & GAMEPAD_SELECT) && (prev & GAMEPAD_SELECT))
173         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_SELECT);
174     if (!(next & GAMEPAD_START) && (prev & GAMEPAD_START))
175         gp.listener(GAMEPAD_KEY_UP, GAMEPAD_START);
176 }
177
178 struct libusb_device_handle *gamepad_open(uint8_t *endpoint_address) {
179     libusb_device **devs;
180     struct libusb_device_handle *handle = NULL;
181     struct libusb_device_descriptor desc;
182     ssize_t num_devs, d;
183     uint8_t i, k;
184
185     /* Start the library */
186     if (libusb_init(NULL) < 0) {
187         fprintf(stderr, "Error: libusb_init failed\n");
188         exit(1);
189     }
190
191     /* Enumerate all the attached USB devices */
192     if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
193         fprintf(stderr, "Error: libusb_get_device_list failed\n");
194         exit(1);
195     }
196
197     /* Look at each device, remembering the first HID device that speaks
198        the keyboard protocol */
199
200     for (d = 0; d < num_devs; d++) {
201         libusb_device *dev = devs[d];
202         if (libusb_get_device_descriptor(dev, &desc) < 0) {
203             fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
204             exit(1);
205         }
206
207         if (desc.idVendor == GAMEPAD_ID_VENDOR &&
208             desc.idProduct == GAMEPAD_ID_PRODUCT) {
209             struct libusb_config_descriptor *config;
210             libusb_get_config_descriptor(dev, 0, &config);
211
212             for (i = 0; i < config->bNumInterfaces; i++) {
213                 for (k = 0; k < config->interface[i].num_altsetting; k++) {
214                     int r;

```

```

215     const struct libusb_interface_descriptor *inter =
216         config->interface[i].altsetting + k;
217     if ((r = libusb_open(dev, &handle)) != 0) {
218         fprintf(stderr, "Error: libusb_open failed: %d\n", r);
219         exit(1);
220     }
221     if (libusb_kernel_driver_active(handle, i)) {
222         libusb_detach_kernel_driver(handle, i);
223     }
224     libusb_set_auto_detach_kernel_driver(handle, i);
225     if ((r = libusb_claim_interface(handle, i)) != 0) {
226         fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
227         exit(1);
228     }
229     *endpoint_address = inter->endpoint[0].bEndpointAddress;
230     goto found;
231 }
232 }
233 }
234 }
235
236 found:
237     libusb_free_device_list(devs, 1);
238
239     return handle;
240 }
241
242 gamepad_button_t gamepad_decode_packet(struct gamepad_packet packet) {
243     gamepad_button_t buttons = 0;
244
245     if (packet.dir_x == 0x00)
246         buttons |= GAMEPAD_LEFT;
247     if (packet.dir_x == 0xff)
248         buttons |= GAMEPAD_RIGHT;
249
250     if (packet.dir_y == 0x00)
251         buttons |= GAMEPAD_UP;
252     if (packet.dir_y == 0xff)
253         buttons |= GAMEPAD_DOWN;
254
255     if (packet.primary & (1 << 7))
256         buttons |= GAMEPAD_Y;
257     if (packet.primary & (1 << 6))
258         buttons |= GAMEPAD_B;
259     if (packet.primary & (1 << 5))
260         buttons |= GAMEPAD_A;
261     if (packet.primary & (1 << 4))
262         buttons |= GAMEPAD_X;
263
264     if (packet.secondary & (1 << 5))
265         buttons |= GAMEPAD_START;
266     if (packet.secondary & (1 << 4))
267         buttons |= GAMEPAD_SELECT;
268     if (packet.secondary & (1 << 1))
269         buttons |= GAMEPAD_R;
270     if (packet.secondary & (1 << 0))
271         buttons |= GAMEPAD_L;
272
273     return buttons;
274 }

```

Listing 9: gamepad.c

```

1 #ifndef _VGA BALL_H
2 #define _VGA BALL_H
3
4 #include <linux/ioctl.h>
5 #include <linux/types.h>
6
7 typedef struct {
8     u8 table;

```

```

9     u16 addr;
10    u8 data;
11 } vga_ball_arg_t;
12
13 #define VGA BALL_MAGIC 'q'
14
15 /* ioctls and their arguments */
16 #define VGA BALL_WRITE _IOW(VGA BALL_MAGIC, 1, vga_ball_arg_t *)
17
18 #endif

```

Listing 10: vga_ball.h

```

1 /* * Device driver for the VGA video generator
2 *
3 * A Platform device implemented using the misc subsystem
4 *
5 * Stephen A. Edwards
6 * Columbia University
7 *
8 * References:
9 * Linux source: Documentation/driver-model/platform.txt
10 *                drivers/misc/arm-charlcd.c
11 * http://www.linuxforu.com/tag/linux-device-drivers/
12 * http://free-electrons.com/docs/
13 *
14 * "make" to build
15 * insmod vga_ball.ko
16 *
17 * Check code style with
18 * checkpatch.pl --file --no-tree vga_ball.c
19 */
20
21 #include "vga_ball.h"
22 #include <linux/errno.h>
23 #include <linux/fs.h>
24 #include <linux/init.h>
25 #include <linux/io.h>
26 #include <linux/kernel.h>
27 #include <linux/miscdevice.h>
28 #include <linux/module.h>
29 #include <linux/of.h>
30 #include <linux/of_address.h>
31 #include <linux/platform_device.h>
32 #include <linux/slab.h>
33 #include <linux/uaccess.h>
34 #include <linux/version.h>
35
36 #define DRIVER_NAME "vga_ball"
37
38 /*
39 * Information about our device
40 */
41 struct vga_ball_dev {
42     struct resource res; /* Resource: our registers */
43     void __iomem *virtbase; /* Where registers can be accessed in memory */
44 } dev;
45
46 /*
47 * Write segments of a single digit
48 * Assumes digit is in range and the device information has been set up
49 */
50 static void vga_ball_write(vga_ball_arg_t *arg) {
51     u32 command = (((u32)(arg->data)) << 24) | (((u32)(arg->addr)) << 2) |
52                 (((u32)(arg->table)) & 0x3);
53     iowrite32(command, dev.virtbase);
54 }
55
56 /*
57 * Handle ioctl() calls from userspace:
58 * Read or write the segments on single digits.

```

```

59 * Note extensive error checking of arguments
60 */
61 static long vga_ball_ioctl(struct file *f, unsigned int cmd,
62                          unsigned long arg) {
63     vga_ball_arg_t vla;
64
65     switch (cmd) {
66     case VGA BALL_WRITE:
67         if (copy_from_user(&vla, (vga_ball_arg_t *)arg, sizeof(vga_ball_arg_t)))
68             return -EACCES;
69         vga_ball_write(&vla);
70         break;
71
72     default:
73         return -EINVAL;
74     }
75
76     return 0;
77 }
78
79 /* The operations our device knows how to do */
80 static const struct file_operations vga_ball_fops = {
81     .owner = THIS_MODULE,
82     .unlocked_ioctl = vga_ball_ioctl,
83 };
84
85 /* Information about our device for the "misc" framework -- like a char dev */
86 static struct miscdevice vga_ball_misc_device = {
87     .minor = MISC_DYNAMIC_MINOR,
88     .name = DRIVER_NAME,
89     .fops = &vga_ball_fops,
90 };
91
92 /*
93  * Initialization code: get resources (registers) and display
94  * a welcome message
95  */
96 static int __init vga_ball_probe(struct platform_device *pdev) {
97     int ret;
98
99     /* Register ourselves as a misc device: creates /dev/vga_ball */
100    ret = misc_register(&vga_ball_misc_device);
101
102    /* Get the address of our registers from the device tree */
103    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
104    if (ret) {
105        ret = -ENOENT;
106        goto out_deregister;
107    }
108
109    /* Make sure we can use these registers */
110    if (request_mem_region(dev.res.start, resource_size(&dev.res), DRIVER_NAME) ==
111        NULL) {
112        ret = -EBUSY;
113        goto out_deregister;
114    }
115
116    /* Arrange access to our registers */
117    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
118    if (dev.virtbase == NULL) {
119        ret = -ENOMEM;
120        goto out_release_mem_region;
121    }
122
123    return 0;
124
125    out_release_mem_region:
126        release_mem_region(dev.res.start, resource_size(&dev.res));
127    out_deregister:
128        misc_deregister(&vga_ball_misc_device);
129    return ret;

```

```

130 }
131
132 /* Clean-up code: release resources */
133 static int vga_ball_remove(struct platform_device *pdev) {
134     iounmap(dev.virtbase);
135     release_mem_region(dev.res.start, resource_size(&dev.res));
136     misc_deregister(&vga_ball_misc_device);
137     return 0;
138 }
139
140 /* Which "compatible" string(s) to search for in the Device Tree */
141 #ifdef CONFIG_OF
142 static const struct of_device_id vga_ball_of_match[] = {
143     {.compatible = "csee4840,vga_ball-1.0"},
144     {}},
145 };
146 MODULE_DEVICE_TABLE(of, vga_ball_of_match);
147 #endif
148
149 /* Information for registering ourselves as a "platform" driver */
150 static struct platform_driver vga_ball_driver = {
151     .driver =
152     {
153         .name = DRIVER_NAME,
154         .owner = THIS_MODULE,
155         .of_match_table = of_match_ptr(vga_ball_of_match),
156     },
157     .remove = __exit_p(vga_ball_remove),
158 };
159
160 /* Called when the module is loaded: set things up */
161 static int __init vga_ball_init(void) {
162     pr_info(DRIVER_NAME ": init\n");
163     return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
164 }
165
166 /* Calball when the module is unloaded: release resources */
167 static void __exit vga_ball_exit(void) {
168     platform_driver_unregister(&vga_ball_driver);
169     pr_info(DRIVER_NAME ": exit\n");
170 }
171
172 module_init(vga_ball_init);
173 module_exit(vga_ball_exit);
174
175 MODULE_LICENSE("GPL");
176 MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
177 MODULE_DESCRIPTION("VGA ball driver");

```

Listing 11: vga_ball.c

```

1 #ifndef _VGA BALL_USER_H
2 #define _VGA BALL_USER_H
3
4 #include <sys/ioctl.h>
5 #include <stdint.h>
6
7 #define PATTERN_NAME_TABLE 0
8 #define PATTERN_GENERATOR_TABLE 1
9 #define SPRITE_ATTRIBUTE_TABLE 2
10 #define SPRITE_GENERATOR_TABLE 3
11
12 typedef struct {
13     uint8_t table;
14     uint16_t addr;
15     uint8_t data;
16 } vga_ball_arg_t;
17
18 #define VGA BALL_MAGIC 'q'
19
20 #define VGA BALL_WRITE _IOW(VGA BALL_MAGIC, 1, vga_ball_arg_t *)

```

```

21
22 void vga_ball_init();
23
24 void vga_ball_write(vga_ball_arg_t *arg);
25
26 #endif

```

Listing 12: vga_ball_user.h

```

1 #include "vga_ball_user.h"
2 #include <fcntl.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 static const char filename[] = "/dev/vga_ball";
7 static int vga_ball_fd;
8
9 void vga_ball_init() {
10     if ((vga_ball_fd = open(filename, O_RDWR)) == -1) {
11         fprintf(stderr, "could not open %s\n", filename);
12         exit(-1);
13     }
14 }
15
16 void vga_ball_write(vga_ball_arg_t *arg) {
17     if (ioctl(vga_ball_fd, VGA BALL_WRITE, arg)) {
18         perror("ioctl(VGA BALL_SET_BACKGROUND) failed");
19         return;
20     }
21 }

```

Listing 13: vga_ball_user.c

```

1 #ifndef _PATTERN_H
2 #define _PATTERN_H
3
4 #include <stdint.h>
5
6 #define PATTERN_BITMAP_SIZE 32
7 #define PATTERN_BITMAP_NROW 8
8 #define PATTERN_BITMAP_NCOL 8
9
10 #define PATTERN_NROW 60
11 #define PATTERN_NCOL 64
12
13 #define pattern_pixel(x) ((x)&0xf)
14
15 void load_pattern_bitmaps();
16 void set_pattern_bitmap(int i, const uint8_t *pat);
17 void set_pattern_at(uint8_t r, uint8_t c, uint8_t name);
18
19 typedef enum {
20     PAT_BACKGROUND = 0,
21     PAT_0,
22     PAT_1,
23     PAT_2,
24     PAT_3,
25     PAT_4,
26     PAT_5,
27     PAT_6,
28     PAT_7,
29     PAT_8,
30     PAT_9,
31     PAT_A,
32     PAT_B,
33     PAT_C,
34     PAT_D,
35     PAT_E,
36     PAT_F,
37     PAT_G,
38     PAT_H,

```

```

39 PAT_I,
40 PAT_J,
41 PAT_K,
42 PAT_L,
43 PAT_M,
44 PAT_N,
45 PAT_O,
46 PAT_P,
47 PAT_Q,
48 PAT_R,
49 PAT_S,
50 PAT_T,
51 PAT_U,
52 PAT_V,
53 PAT_W,
54 PAT_X,
55 PAT_Y,
56 PAT_Z,
57 PAT_FOOD_SM,
58 PAT_FOOD_LG,
59 PAT_WALL_0,
60 PAT_WALL_1,
61 PAT_WALL_2,
62 PAT_WALL_3,
63 PAT_WALL_4,
64 PAT_WALL_5,
65 PAT_WALL_6,
66 PAT_WALL_7,
67 PAT_WALL_8,
68 PAT_WALL_9,
69 PAT_WALL_10,
70 PAT_WALL_11,
71 PAT_WALL_12,
72 PAT_WALL_13,
73 PAT_WALL_14,
74 PAT_WALL_15,
75 PAT_WALL_16,
76 PAT_WALL_17,
77 PAT_WALL_18,
78 PAT_WALL_19,
79 PAT_WALL_20,
80 PAT_WALL_21,
81 PAT_WALL_22,
82 PAT_WALL_23,
83 PAT_GATE,
84 } pattern_name_t;
85
86 #endif

```

Listing 14: pattern.h

```

1 #include "pattern.h"
2 #include "color.h"
3 #include "vga_ball_user.h"
4 #include <stdint.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 const uint8_t pat_background[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
9     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
10    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
11    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
12    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
13    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
14    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
15    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
16    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
17 };
18
19 const uint8_t pat_0[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
20     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},

```

```

21     {Transp, Transp, Transp, White, White, White, Transp, Transp},
22     {Transp, Transp, White, Transp, Transp, White, White, Transp},
23     {Transp, White, White, Transp, Transp, Transp, White, White},
24     {Transp, White, White, Transp, Transp, Transp, White, White},
25     {Transp, White, White, Transp, Transp, Transp, White, White},
26     {Transp, Transp, White, White, Transp, Transp, White, Transp},
27     {Transp, Transp, Transp, White, White, White, Transp, Transp},
28 };
29
30 const uint8_t pat_1[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
31     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
32     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
33     {Transp, Transp, Transp, White, White, White, Transp, Transp},
34     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
35     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
36     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
37     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
38     {Transp, Transp, White, White, White, White, White, White},
39 };
40
41 const uint8_t pat_2[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
42     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
43     {Transp, Transp, White, White, White, White, White, Transp},
44     {Transp, White, White, Transp, Transp, Transp, White, White},
45     {Transp, Transp, Transp, Transp, Transp, White, White, White},
46     {Transp, Transp, Transp, White, White, White, White, Transp},
47     {Transp, Transp, White, White, White, White, Transp, Transp},
48     {Transp, White, White, White, Transp, Transp, Transp, Transp},
49     {Transp, White, White, White, White, White, White, White},
50 };
51
52 const uint8_t pat_3[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
53     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
54     {Transp, Transp, White, White, White, White, White, White},
55     {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
56     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
57     {Transp, Transp, Transp, White, White, White, White, Transp},
58     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
59     {Transp, White, White, Transp, Transp, Transp, White, White},
60     {Transp, Transp, White, White, White, White, White, Transp},
61 };
62
63 const uint8_t pat_4[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
64     {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
65     {Transp, Transp, Transp, Transp, White, White, White, Transp},
66     {Transp, Transp, Transp, Transp, White, White, White, White, Transp},
67     {Transp, Transp, White, White, Transp, White, White, Transp},
68     {Transp, White, White, Transp, Transp, White, White, Transp},
69     {Transp, White, White, White, White, White, White, White},
70     {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
71     {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
72 };
73
74 const uint8_t pat_5[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
75     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
76     {Transp, White, White, White, White, White, White, Transp},
77     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
78     {Transp, White, White, White, White, White, White, Transp},
79     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
80     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
81     {Transp, White, White, Transp, Transp, Transp, White, White},
82     {Transp, Transp, White, White, White, White, White, Transp},
83 };
84
85 const uint8_t pat_6[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
86     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
87     {Transp, Transp, Transp, White, White, White, White, Transp},
88     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
89     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
90     {Transp, White, White, Transp, White, White, White, Transp},
91     {Transp, White, White, Transp, Transp, Transp, White, White},

```



```

92     {Transp, White, White, Transp, Transp, Transp, White, White},
93     {Transp, Transp, White, White, White, White, White, Transp},
94 };
95
96 const uint8_t pat_7[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
97     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
98     {Transp, White, White, White, White, White, White, White},
99     {Transp, White, White, Transp, Transp, Transp, White, White},
100    {Transp, Transp, Transp, Transp, Transp, Transp, White, White, Transp},
101    {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
102    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
103    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
104    {Transp, Transp, Transp, White, White, Transp, Transp, Transp},
105 };
106
107 const uint8_t pat_8[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
108     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
109     {Transp, Transp, White, White, White, White, White, Transp},
110     {Transp, White, White, Transp, Transp, Transp, White, Transp},
111     {Transp, White, White, White, Transp, Transp, White, Transp},
112     {Transp, Transp, White, White, White, White, Transp, Transp},
113     {Transp, White, Transp, Transp, White, White, White, White},
114     {Transp, White, Transp, Transp, Transp, Transp, White, White},
115     {Transp, Transp, White, White, White, White, White, Transp},
116 };
117
118 const uint8_t pat_9[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
119     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
120     {Transp, Transp, White, White, White, White, White, Transp},
121     {Transp, White, White, Transp, Transp, Transp, White, White},
122     {Transp, White, White, Transp, Transp, Transp, White, White},
123     {Transp, Transp, White, White, White, White, White, White},
124     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
125     {Transp, Transp, Transp, Transp, Transp, White, White, Transp},
126     {Transp, Transp, White, White, White, White, Transp, Transp},
127 };
128
129 const uint8_t pat_A[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
130     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
131     {Transp, Transp, Transp, White, White, White, Transp, Transp},
132     {Transp, Transp, White, White, Transp, White, White, Transp},
133     {Transp, White, White, Transp, Transp, Transp, White, White},
134     {Transp, White, White, Transp, Transp, Transp, White, White},
135     {Transp, White, White, White, White, White, White, White},
136     {Transp, White, White, Transp, Transp, Transp, White, White},
137     {Transp, White, White, Transp, Transp, Transp, White, White},
138 };
139
140 const uint8_t pat_B[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
141     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
142     {Transp, White, White, White, White, White, White, White, Transp},
143     {Transp, White, White, Transp, Transp, Transp, White, White},
144     {Transp, White, White, Transp, Transp, Transp, White, White},
145     {Transp, White, White, White, White, White, White, Transp},
146     {Transp, White, White, Transp, Transp, Transp, White, White},
147     {Transp, White, White, Transp, Transp, Transp, White, White},
148     {Transp, White, White, White, White, White, White, Transp},
149 };
150
151 const uint8_t pat_C[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
152     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
153     {Transp, Transp, Transp, White, White, White, White, Transp},
154     {Transp, Transp, White, White, Transp, Transp, White, White},
155     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
156     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
157     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
158     {Transp, Transp, White, White, Transp, Transp, White, White},
159     {Transp, Transp, Transp, White, White, White, White, Transp},
160 };
161
162 const uint8_t pat_D[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {

```

```

163     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
164     {Transp, White, White, White, White, White, Transp, Transp},
165     {Transp, White, White, Transp, Transp, White, White, Transp},
166     {Transp, White, White, Transp, Transp, Transp, White, White},
167     {Transp, White, White, Transp, Transp, Transp, White, White},
168     {Transp, White, White, Transp, Transp, Transp, White, White},
169     {Transp, White, White, Transp, Transp, White, White, Transp},
170     {Transp, White, White, White, White, White, Transp, Transp},
171 };
172
173 const uint8_t pat_E[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
174     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
175     {Transp, Transp, White, White, White, White, White, White},
176     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
177     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
178     {Transp, Transp, White, White, White, White, White, Transp},
179     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
180     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
181     {Transp, Transp, White, White, White, White, White, White},
182 };
183
184 const uint8_t pat_F[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
185     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
186     {Transp, White, White, White, White, White, White, White},
187     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
188     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
189     {Transp, White, White, White, White, White, White, Transp},
190     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
191     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
192     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
193 };
194
195 const uint8_t pat_G[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
196     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
197     {Transp, Transp, Transp, White, White, White, White, White},
198     {Transp, Transp, White, White, Transp, Transp, Transp, Transp},
199     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
200     {Transp, White, White, Transp, Transp, White, White, White},
201     {Transp, White, White, Transp, Transp, Transp, White, White},
202     {Transp, Transp, White, White, Transp, Transp, White, White},
203     {Transp, Transp, Transp, White, White, White, White, White},
204 };
205
206 const uint8_t pat_H[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
207     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
208     {Transp, White, White, Transp, Transp, Transp, White, White},
209     {Transp, White, White, Transp, Transp, Transp, White, White},
210     {Transp, White, White, Transp, Transp, Transp, White, White},
211     {Transp, White, White, White, White, White, White, White},
212     {Transp, White, White, Transp, Transp, Transp, White, White},
213     {Transp, White, White, Transp, Transp, Transp, White, White},
214     {Transp, White, White, Transp, Transp, Transp, White, White},
215 };
216
217 const uint8_t pat_I[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
218     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
219     {Transp, Transp, White, White, White, White, White, White},
220     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
221     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
222     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
223     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
224     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
225     {Transp, Transp, White, White, White, White, White, White},
226 };
227
228 const uint8_t pat_J[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
229     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
230     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
231     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
232     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
233     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},

```

```

234     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
235     {Transp, White, White, Transp, Transp, Transp, White, White},
236     {Transp, Transp, White, White, White, White, White, Transp},
237 };
238
239 const uint8_t pat_K[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
240     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
241     {Transp, White, White, Transp, Transp, Transp, White, White},
242     {Transp, White, White, Transp, Transp, Transp, White, White, Transp},
243     {Transp, White, White, Transp, White, White, Transp, Transp},
244     {Transp, White, White, White, White, Transp, Transp, Transp},
245     {Transp, White, White, White, White, White, Transp, Transp},
246     {Transp, White, White, Transp, Transp, White, White, Transp},
247     {Transp, White, White, Transp, Transp, Transp, White, White},
248 };
249
250 const uint8_t pat_L[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
251     {Transp, Transp, Transp, Transp, Transp, Transp, Transp},
252     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
253     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
254     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
255     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
256     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
257     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
258     {Transp, White, White, White, White, White, White, White},
259 };
260
261 const uint8_t pat_M[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
262     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
263     {Transp, White, White, Transp, Transp, Transp, White, White},
264     {Transp, White, White, White, Transp, White, White, White},
265     {Transp, White, White, White, White, White, White, White},
266     {Transp, White, White, White, White, White, White, White},
267     {Transp, White, White, Transp, White, Transp, White, White},
268     {Transp, White, White, Transp, Transp, Transp, White, White},
269     {Transp, White, White, Transp, Transp, Transp, White, White},
270 };
271
272 const uint8_t pat_N[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
273     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
274     {Transp, White, White, Transp, Transp, Transp, White, White},
275     {Transp, White, White, White, Transp, Transp, White, White},
276     {Transp, White, White, White, White, Transp, White, White},
277     {Transp, White, White, White, White, White, White, White},
278     {Transp, White, White, Transp, White, White, White, White},
279     {Transp, White, White, Transp, Transp, White, White, White},
280     {Transp, White, White, Transp, Transp, Transp, White, White},
281 };
282
283 const uint8_t pat_O[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
284     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
285     {Transp, Transp, White, White, White, White, White, Transp},
286     {Transp, White, White, Transp, Transp, Transp, White, White},
287     {Transp, White, White, Transp, Transp, Transp, White, White},
288     {Transp, White, White, Transp, Transp, Transp, White, White},
289     {Transp, White, White, Transp, Transp, Transp, White, White},
290     {Transp, White, White, Transp, Transp, Transp, White, White},
291     {Transp, Transp, White, White, White, White, White, Transp},
292 };
293
294 const uint8_t pat_P[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
295     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
296     {Transp, White, White, White, White, White, White, Transp},
297     {Transp, White, White, Transp, Transp, Transp, White, White},
298     {Transp, White, White, Transp, Transp, Transp, White, White},
299     {Transp, White, White, Transp, Transp, Transp, White, White},
300     {Transp, White, White, White, White, White, White, Transp},
301     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
302     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
303 };
304

```

```

305 const uint8_t pat_Q[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
306     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
307     {Transp, Transp, White, White, White, White, White, Transp},
308     {Transp, White, White, Transp, Transp, Transp, White, White},
309     {Transp, White, White, Transp, Transp, Transp, White, White},
310     {Transp, White, White, Transp, Transp, Transp, White, White},
311     {Transp, White, White, Transp, White, White, White, White},
312     {Transp, White, White, Transp, Transp, White, White, Transp},
313     {Transp, Transp, White, White, White, White, Transp, White},
314 };
315
316 const uint8_t pat_R[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
317     {Transp, Transp, Transp, Transp, Transp, Transp, Transp},
318     {Transp, White, White, White, White, White, White, Transp},
319     {Transp, White, White, Transp, Transp, Transp, White, White},
320     {Transp, White, White, Transp, Transp, Transp, White, White},
321     {Transp, White, White, Transp, Transp, White, White, White},
322     {Transp, White, White, White, White, White, Transp, Transp},
323     {Transp, White, White, Transp, White, White, White, Transp},
324     {Transp, White, White, Transp, Transp, White, White, White},
325 };
326
327 const uint8_t pat_S[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
328     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
329     {Transp, Transp, White, White, White, White, Transp, Transp},
330     {Transp, White, White, Transp, Transp, White, White, Transp},
331     {Transp, White, White, Transp, Transp, Transp, Transp, Transp},
332     {Transp, Transp, White, White, White, White, White, Transp},
333     {Transp, Transp, Transp, Transp, Transp, Transp, White, White},
334     {Transp, White, White, Transp, Transp, Transp, White, White},
335     {Transp, Transp, White, White, White, White, White, Transp},
336 };
337
338 const uint8_t pat_T[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
339     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
340     {Transp, Transp, White, White, White, White, White, White},
341     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
342     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
343     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
344     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
345     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
346     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
347 };
348
349 const uint8_t pat_U[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
350     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
351     {Transp, White, White, Transp, Transp, Transp, White, White},
352     {Transp, White, White, Transp, Transp, Transp, White, White},
353     {Transp, White, White, Transp, Transp, Transp, White, White},
354     {Transp, White, White, Transp, Transp, Transp, White, White},
355     {Transp, White, White, Transp, Transp, Transp, White, White},
356     {Transp, White, White, Transp, Transp, Transp, White, White},
357     {Transp, Transp, White, White, White, White, White, Transp},
358 };
359
360 const uint8_t pat_V[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
361     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
362     {Transp, White, White, Transp, Transp, Transp, White, White},
363     {Transp, White, White, Transp, Transp, Transp, White, White},
364     {Transp, White, White, Transp, Transp, Transp, White, White},
365     {Transp, White, White, White, Transp, White, White, White},
366     {Transp, Transp, White, White, White, White, White, Transp},
367     {Transp, Transp, Transp, White, White, White, Transp, Transp},
368     {Transp, Transp, Transp, Transp, White, Transp, Transp, Transp},
369 };
370
371 const uint8_t pat_W[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
372     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
373     {Transp, White, White, Transp, Transp, Transp, White, White},
374     {Transp, White, White, Transp, Transp, Transp, White, White},
375     {Transp, White, White, Transp, White, Transp, White, White},

```

```

376     {Transp, White, White, White, White, White, White, White},
377     {Transp, White, White, White, White, White, White, White},
378     {Transp, White, White, White, Transp, White, White, White},
379     {Transp, White, White, Transp, Transp, Transp, White, White},
380 };
381
382 const uint8_t pat_X[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
383     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
384     {Transp, White, White, Transp, Transp, Transp, Transp, White},
385     {Transp, White, White, White, Transp, White, White, White},
386     {Transp, Transp, White, White, White, White, White, Transp},
387     {Transp, Transp, Transp, White, White, White, Transp, Transp},
388     {Transp, Transp, White, White, White, White, White, Transp},
389     {Transp, White, White, White, Transp, White, White, White},
390     {Transp, White, White, Transp, Transp, Transp, White, White},
391 };
392
393 const uint8_t pat_Y[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
394     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
395     {Transp, Transp, White, White, Transp, Transp, White, White},
396     {Transp, Transp, White, White, Transp, Transp, White, White},
397     {Transp, Transp, White, White, Transp, Transp, White, White},
398     {Transp, Transp, Transp, White, White, White, White, Transp},
399     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
400     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
401     {Transp, Transp, Transp, Transp, White, White, Transp, Transp},
402 };
403
404 const uint8_t pat_Z[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
405     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
406     {Transp, White, White, White, White, White, White, White},
407     {Transp, Transp, Transp, Transp, Transp, White, White, White},
408     {Transp, Transp, Transp, Transp, White, White, White, Transp},
409     {Transp, Transp, Transp, White, White, White, Transp, Transp},
410     {Transp, Transp, White, White, White, Transp, Transp, Transp},
411     {Transp, White, White, White, Transp, Transp, Transp, Transp},
412     {Transp, White, White, White, White, White, White, White},
413 };
414
415 const uint8_t pat_food_sm[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
416     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
417     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
418     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
419     {Transp, Transp, Transp, Salmon, Salmon, Transp, Transp, Transp},
420     {Transp, Transp, Transp, Salmon, Salmon, Transp, Transp, Transp},
421     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
422     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
423     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
424 };
425
426 const uint8_t pat_food_lg[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
427     {Transp, Transp, Salmon, Salmon, Salmon, Salmon, Transp, Transp},
428     {Transp, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Transp},
429     {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
430     {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
431     {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
432     {Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon},
433     {Transp, Salmon, Salmon, Salmon, Salmon, Salmon, Salmon, Transp},
434     {Transp, Transp, Salmon, Salmon, Salmon, Salmon, Transp, Transp},
435 };
436
437 const uint8_t pat_wall_o[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
438     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
439     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
440     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
441     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
442     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
443     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
444     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
445     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
446 };

```

```

447
448 const uint8_t pat_wall_1[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
449     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
450     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
451     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
452     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
453     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
454     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
455     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
456     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
457 };
458
459 const uint8_t pat_wall_2[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
460     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
461     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
462     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
463     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
464     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
465     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
466     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
467     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
468 };
469
470 const uint8_t pat_wall_3[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
471     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
472     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
473     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
474     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
475     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
476     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
477     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
478     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
479 };
480
481 const uint8_t pat_wall_4[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
482     {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue},
483     {Transp, Transp, Blue, Blue, Transp, Transp, Transp, Transp},
484     {Transp, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
485     {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Blue},
486     {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
487     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
488     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
489     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
490 };
491
492 const uint8_t pat_wall_5[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
493     {Blue, Blue, Blue, Blue, Transp, Transp, Transp, Transp},
494     {Transp, Transp, Transp, Transp, Blue, Blue, Transp, Transp},
495     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
496     {Blue, Blue, Blue, Transp, Transp, Transp, Blue, Transp},
497     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
498     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
499     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
500     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
501 };
502
503 const uint8_t pat_wall_6[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
504     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
505     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
506     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
507     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
508     {Blue, Blue, Blue, Transp, Transp, Transp, Blue, Transp},
509     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
510     {Transp, Transp, Transp, Transp, Blue, Blue, Transp, Transp},
511     {Blue, Blue, Blue, Blue, Transp, Transp, Transp, Transp},
512 };
513
514 const uint8_t pat_wall_7[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
515     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
516     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
517     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},

```

```

518     {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
519     {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Blue},
520     {Transp, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
521     {Transp, Transp, Blue, Blue, Transp, Transp, Transp, Transp},
522     {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue},
523 };
524
525 const uint8_t pat_wall_8[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
526     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
527     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
528     {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
529     {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
530     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
531     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
532     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
533     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
534 };
535
536 const uint8_t pat_wall_9[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
537     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
538     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
539     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
540     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
541     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
542     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
543     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
544     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
545 };
546
547 const uint8_t pat_wall_10[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
548     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
549     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
550     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
551     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
552     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue},
553     {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Transp},
554     {Transp, Transp, Transp, Transp, Transp, Blue, Transp, Transp},
555     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
556 };
557
558 const uint8_t pat_wall_11[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
559     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
560     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
561     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
562     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
563     {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
564     {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
565     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
566     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
567 };
568
569 const uint8_t pat_wall_12[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
570     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
571     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
572     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
573     {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Transp},
574     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
575     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
576     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
577     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
578 };
579
580 const uint8_t pat_wall_13[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
581     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
582     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
583     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
584     {Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
585     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
586     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
587     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
588     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},

```

```

589 };
590
591 const uint8_t pat_wall_14[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
592     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
593     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
594     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
595     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
596     {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Blue},
597     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
598     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
599     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
600 };
601
602 const uint8_t pat_wall_15[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
603     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
604     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
605     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Blue},
606     {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Blue},
607     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Blue},
608     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
609     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
610     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Blue},
611 };
612
613 const uint8_t pat_wall_16[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
614     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
615     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
616     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
617     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
618     {Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
619     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
620     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
621     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
622 };
623
624 const uint8_t pat_wall_17[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
625     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
626     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
627     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
628     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
629     {Blue, Blue, Blue, Transp, Transp, Transp, Transp, Transp},
630     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
631     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
632     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
633 };
634
635 const uint8_t pat_wall_18[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
636     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
637     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
638     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
639     {Blue, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
640     {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
641     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
642     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
643     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
644 };
645
646 const uint8_t pat_wall_19[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
647     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
648     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
649     {Blue, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
650     {Blue, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
651     {Blue, Transp, Transp, Transp, Transp, Blue, Blue, Blue},
652     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
653     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
654     {Blue, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
655 };
656
657 const uint8_t pat_wall_20[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
658     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
659     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},

```



```

731 //     {Transp, Transp, Transp, Transp, Blue, Transp, Transp, Transp},
732 // };
733 //
734 // const uint8_t pat_wall_27[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
735 //     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
736 //     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
737 //     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
738 //     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
739 //     {Blue, Blue, Transp, Transp, Transp, Transp, Transp, Transp},
740 //     {Transp, Transp, Blue, Transp, Transp, Transp, Transp, Transp},
741 //     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
742 //     {Transp, Transp, Transp, Blue, Transp, Transp, Transp, Transp},
743 // };
744 //
745 const uint8_t pat_gate[PATTERN_BITMAP_NROW][PATTERN_BITMAP_NCOL] = {
746     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
747     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
748     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
749     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
750     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
751     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
752     {Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue},
753     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp},
754 };
755
756 const uint8_t *patterns[] = {
757     (uint8_t *)pat_background, (uint8_t *)pat_0, (uint8_t *)pat_1,
758     (uint8_t *)pat_2, (uint8_t *)pat_3, (uint8_t *)pat_4,
759     (uint8_t *)pat_5, (uint8_t *)pat_6, (uint8_t *)pat_7,
760     (uint8_t *)pat_8, (uint8_t *)pat_9, (uint8_t *)pat_A,
761     (uint8_t *)pat_B, (uint8_t *)pat_C, (uint8_t *)pat_D,
762     (uint8_t *)pat_E, (uint8_t *)pat_F, (uint8_t *)pat_G,
763     (uint8_t *)pat_H, (uint8_t *)pat_I, (uint8_t *)pat_J,
764     (uint8_t *)pat_K, (uint8_t *)pat_L, (uint8_t *)pat_M,
765     (uint8_t *)pat_N, (uint8_t *)pat_O, (uint8_t *)pat_P,
766     (uint8_t *)pat_Q, (uint8_t *)pat_R, (uint8_t *)pat_S,
767     (uint8_t *)pat_T, (uint8_t *)pat_U, (uint8_t *)pat_V,
768     (uint8_t *)pat_W, (uint8_t *)pat_X, (uint8_t *)pat_Y,
769     (uint8_t *)pat_Z, (uint8_t *)pat_food_sm, (uint8_t *)pat_food_lg,
770     (uint8_t *)pat_wall_0, (uint8_t *)pat_wall_1, (uint8_t *)pat_wall_2,
771     (uint8_t *)pat_wall_3, (uint8_t *)pat_wall_4, (uint8_t *)pat_wall_5,
772     (uint8_t *)pat_wall_6, (uint8_t *)pat_wall_7, (uint8_t *)pat_wall_8,
773     (uint8_t *)pat_wall_9, (uint8_t *)pat_wall_10, (uint8_t *)pat_wall_11,
774     (uint8_t *)pat_wall_12, (uint8_t *)pat_wall_13, (uint8_t *)pat_wall_14,
775     (uint8_t *)pat_wall_15, (uint8_t *)pat_wall_16, (uint8_t *)pat_wall_17,
776     (uint8_t *)pat_wall_18, (uint8_t *)pat_wall_19, (uint8_t *)pat_wall_20,
777     (uint8_t *)pat_wall_21, (uint8_t *)pat_wall_22, (uint8_t *)pat_wall_23,
778     (uint8_t *)pat_gate,
779 };
780
781 void load_pattern_bitmaps() {
782     for (int i = 0; i < sizeof(patterns) / sizeof(const uint8_t *); i++) {
783         const uint8_t *pat = patterns[i];
784         set_pattern_bitmap(i, pat);
785     }
786 }
787
788 void set_pattern_bitmap(int pati, const uint8_t *pat) {
789     vga_ball_arg_t arg;
790     int start;
791
792     arg.table = PATTERN_GENERATOR_TABLE;
793     start = pati * PATTERN_BITMAP_SIZE;
794     for (int i = 0; i < PATTERN_BITMAP_SIZE; i++) {
795         arg.addr = start + i;
796         arg.data = pattern_pixel(pat[2 * i]) << 4 | pattern_pixel(pat[2 * i + 1]);
797         vga_ball_write(&arg);
798     }
799 }
800
801 void set_pattern_at(uint8_t r, uint8_t c, uint8_t name) {

```

```

802     if (r >= PATTERN_NROW) {
803         fprintf(stderr, "Row %d is too large\n", r);
804         exit(-1);
805     }
806
807     if (c >= PATTERN_NCOL) {
808         fprintf(stderr, "Column %d is too large\n", r);
809         exit(-1);
810     }
811
812     vga_ball_arg_t arg;
813
814     arg.table = PATTERN_NAME_TABLE;
815     arg.addr = r * PATTERN_NCOL + c;
816     arg.data = name;
817
818     vga_ball_write(&arg);
819 }

```

Listing 15: pattern.c

```

1  #ifndef _SPRITE_H
2  #define _SPRITE_H
3
4  #include <stdint.h>
5
6  typedef struct {
7      uint8_t i;
8      uint16_t y;
9      uint16_t x;
10     uint8_t name;
11 } sprite_attr_t;
12
13 #define SPRITE_BITMAP_SIZE 128
14 #define SPRITE_BITMAP_NROW 16
15 #define SPRITE_BITMAP_NCOL 16
16 #define sprite_pixel(x) ((x)&0xf)
17
18 void load_sprite_bitmaps();
19 void set_sprite_bitmap(int spriti, const uint8_t *sprite);
20 void set_sprite(sprite_attr_t attr);
21
22 typedef enum {
23     SPRITE_PACMAN_CLOSED = 0,
24     SPRITE_PACMAN_LEFT,
25     SPRITE_PACMAN_RIGHT,
26     SPRITE_PACMAN_UP,
27     SPRITE_PACMAN_DOWN,
28     SPRITE_GHOST_RED,
29     SPRITE_GHOST_CYAN,
30     SPRITE_GHOST_PINK,
31     SPRITE_GHOST_ORANGE,
32     SPRITE_GHOST_SCATTER,
33 } sprite_name_t;
34
35 #endif

```

Listing 16: sprite.h

```

1  #include "sprite.h"
2  #include "color.h"
3  #include "vga_ball_user.h"
4  #include <stdint.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  const uint8_t sprite_pacman_closed[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
9      {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
10       Transp, Transp, Transp, Transp, Transp, Transp, Transp},
11      {Transp, Transp, Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow,
12       Yellow, Transp, Transp, Transp, Transp, Transp, Transp},

```



```

155 {Transp, Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
156     Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp},
157 {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
158     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
159 {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow,
160     Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp},
161 {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
162     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
163 {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
164     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
165 {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Yellow,
166     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
167 {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
168     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
169 {Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
170     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
171 {Transp, Transp, Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp, Transp,
172     Yellow, Yellow, Yellow, Yellow, Yellow, Transp, Transp},
173 {Transp, Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp, Transp, Transp,
174     Transp, Yellow, Yellow, Yellow, Transp, Transp, Transp},
175 {Transp, Transp, Transp, Yellow, Yellow, Transp, Transp, Transp, Transp, Transp,
176     Transp, Yellow, Yellow, Transp, Transp, Transp, Transp},
177 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
178     Transp, Transp, Transp, Transp, Transp, Transp},
179 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
180     Transp, Transp, Transp, Transp, Transp, Transp},
181 };
182
183 const uint8_t sprite_ghost_red[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
184     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
185      Transp, Transp, Transp, Transp, Transp, Transp, Transp},
186     {Transp, Transp, Transp, Transp, Transp, Transp, Red, Red, Red, Red, Transp,
187      Transp, Transp, Transp, Transp, Transp},
188     {Transp, Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red,
189      Transp, Transp, Transp, Transp},
190     {Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red,
191      Transp, Transp},
192     {Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
193      Transp, Transp},
194     {Transp, Transp, Red, Red, White, White, Red, Red, Red, Red, White, White,
195      Red, Red, Transp, Transp},
196     {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
197      White, White, Red, Red, Transp},
198     {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
199      White, White, Red, Red, Transp},
200     {Transp, Red, Red, White, Blue, Blue, White, Red, Red, White, Blue, Blue,
201      White, Red, Red, Transp},
202     {Transp, Red, Red, Red, Blue, Blue, Red, Red, Red, Red, Blue, Blue, Red,
203      Red, Red, Transp},
204     {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
205      Red, Transp},
206     {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
207      Red, Transp},
208     {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
209      Red, Transp},
210     {Transp, Red, Red, Transp, Red, Red, Red, Transp, Transp, Red, Red, Red,
211      Transp, Red, Red, Transp},
212     {Transp, Red, Transp, Transp, Transp, Red, Red, Transp, Transp, Red, Red,
213      Transp, Transp, Transp, Red, Transp},
214     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
215      Transp, Transp, Transp, Transp, Transp},
216 };
217
218 const uint8_t sprite_ghost_orange[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
219     {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
220      Transp, Transp, Transp, Transp, Transp, Transp, Transp},
221     {Transp, Transp, Transp, Transp, Transp, Transp, Orange, Orange, Orange,
222      Orange, Transp, Transp, Transp, Transp, Transp, Transp},
223     {Transp, Transp, Transp, Transp, Orange, Orange, Orange, Orange, Orange,
224      Orange, Orange, Transp, Transp},
225     {Transp, Transp, Transp, Orange, Orange, Orange, Orange, Orange, Orange,

```

```

226     Orange, Orange, Orange, Orange, Transp, Transp, Transp},
227 {Transp, Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
228  Orange, Orange, Orange, Orange, Orange, Transp, Transp},
229 {Transp, Transp, Orange, Orange, Orange, White, White, Orange, Orange, Orange,
230  Orange, White, White, Orange, Orange, Transp, Transp},
231 {Transp, Orange, Orange, White, White, White, White, Orange, Orange, White,
232  White, White, White, Orange, Orange, Transp},
233 {Transp, Orange, Orange, Orange, White, White, White, White, Orange, Orange, White,
234  White, White, White, Orange, Orange, Transp},
235 {Transp, Orange, Orange, White, Blue, Blue, White, Orange, Orange, White,
236  Blue, Blue, White, Orange, Orange, Transp},
237 {Transp, Orange, Orange, Orange, Orange, Blue, Blue, Orange, Orange, Orange, Orange,
238  Blue, Blue, Orange, Orange, Orange, Transp},
239 {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
240  Orange, Orange, Orange, Orange, Orange, Transp},
241 {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
242  Orange, Orange, Orange, Orange, Orange, Transp},
243 {Transp, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange, Orange,
244  Orange, Orange, Orange, Orange, Orange, Transp},
245 {Transp, Orange, Orange, Transp, Orange, Orange, Orange, Transp, Transp,
246  Orange, Orange, Orange, Transp, Orange, Orange, Transp},
247 {Transp, Orange, Transp, Transp, Transp, Transp, Orange, Orange, Transp, Transp,
248  Orange, Orange, Transp, Transp, Transp, Transp, Orange, Transp},
249 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
250  Transp, Transp, Transp, Transp, Transp, Transp, Transp},
251 };
252
253 const uint8_t sprite_ghost_cyan[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
254 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
255  Transp, Transp, Transp, Transp, Transp, Transp, Transp},
256 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan,
257  Transp, Transp, Transp, Transp, Transp},
258 {Transp, Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
259  Cyan, Transp, Transp, Transp, Transp},
260 {Transp, Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
261  Cyan, Cyan, Transp, Transp},
262 {Transp, Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
263  Cyan, Cyan, Transp, Transp},
264 {Transp, Transp, Cyan, Cyan, White, White, Cyan, Cyan, Cyan, Cyan, White,
265  White, Cyan, Cyan, Transp, Transp},
266 {Transp, Cyan, Cyan, White, White, White, White, Cyan, Cyan, White, White,
267  White, White, Cyan, Cyan, Transp},
268 {Transp, Cyan, Cyan, White, White, White, White, Cyan, Cyan, White, White,
269  White, White, Cyan, Cyan, Transp},
270 {Transp, Cyan, Cyan, White, Blue, Blue, White, Cyan, Cyan, White, Blue,
271  Blue, White, Cyan, Cyan, Transp},
272 {Transp, Cyan, Cyan, Cyan, Blue, Blue, Cyan, Cyan, Cyan, Cyan, Blue, Blue,
273  Cyan, Cyan, Cyan, Transp},
274 {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
275  Cyan, Cyan, Cyan, Transp},
276 {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
277  Cyan, Cyan, Cyan, Transp},
278 {Transp, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan, Cyan,
279  Cyan, Cyan, Cyan, Transp},
280 {Transp, Cyan, Cyan, Transp, Cyan, Cyan, Cyan, Transp, Transp, Cyan, Cyan,
281  Cyan, Transp, Cyan, Cyan, Transp},
282 {Transp, Cyan, Transp, Transp, Transp, Cyan, Cyan, Transp, Transp, Cyan,
283  Cyan, Transp, Transp, Transp, Cyan, Transp},
284 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
285  Transp, Transp, Transp, Transp, Transp, Transp, Transp},
286 };
287
288 const uint8_t sprite_ghost_pink[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
289 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
290  Transp, Transp, Transp, Transp, Transp},
291 {Transp, Transp, Transp, Transp, Transp, Transp, Pink, Pink, Pink, Pink,
292  Transp, Transp, Transp, Transp, Transp},
293 {Transp, Transp, Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
294  Pink, Transp, Transp, Transp, Transp},
295 {Transp, Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
296  Pink, Pink, Transp, Transp, Transp},

```

```

297 {Transp, Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
298   Pink, Pink, Transp, Transp},
299 {Transp, Transp, Pink, Pink, White, White, Pink, Pink, Pink, Pink, White,
300   White, Pink, Pink, Transp, Transp},
301 {Transp, Pink, Pink, White, White, White, White, Pink, Pink, White, White,
302   White, White, Pink, Pink, Transp},
303 {Transp, Pink, Pink, White, White, White, White, Pink, Pink, White, White,
304   White, White, Pink, Pink, Transp},
305 {Transp, Pink, Pink, White, Blue, Blue, White, Pink, Pink, White, Blue,
306   Blue, White, Pink, Pink, Transp},
307 {Transp, Pink, Pink, Pink, Blue, Blue, Pink, Pink, Pink, Pink, Blue, Blue,
308   Pink, Pink, Pink, Transp},
309 {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
310   Pink, Pink, Pink, Transp},
311 {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
312   Pink, Pink, Pink, Transp},
313 {Transp, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink,
314   Pink, Pink, Pink, Transp},
315 {Transp, Pink, Pink, Transp, Pink, Pink, Pink, Transp, Transp, Pink, Pink,
316   Pink, Transp, Pink, Pink, Transp},
317 {Transp, Pink, Transp, Transp, Transp, Pink, Pink, Transp, Transp, Pink,
318   Pink, Transp, Transp, Transp, Pink, Transp},
319 {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
320   Transp, Transp, Transp, Transp, Transp, Transp},
321 };
322
323 const uint8_t sprite_ghost_scatter[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
324   {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
325     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
326   {Transp, Transp, Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue,
327     Transp, Transp, Transp, Transp, Transp, Transp},
328   {Transp, Transp, Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue,
329     Blue, Transp, Transp, Transp, Transp},
330   {Transp, Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
331     Blue, Blue, Transp, Transp, Transp},
332   {Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
333     Blue, Blue, Transp, Transp},
334   {Transp, Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
335     Blue, Blue, Transp, Transp},
336   {Transp, Blue, Blue, Blue, Blue, White, White, Blue, Blue, White, White,
337     Blue, Blue, Blue, Blue, Transp},
338   {Transp, Blue, Blue, Blue, Blue, White, White, Blue, Blue, White, White,
339     Blue, Blue, Blue, Blue, Transp},
340   {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
341     Blue, Blue, Blue, Transp},
342   {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
343     Blue, Blue, Blue, Transp},
344   {Transp, Blue, Blue, White, White, Blue, Blue, White, White, Blue, Blue,
345     White, White, Blue, Blue, Transp},
346   {Transp, Blue, White, Blue, Blue, White, White, Blue, Blue, White, White,
347     Blue, Blue, White, Blue, Transp},
348   {Transp, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue, Blue,
349     Blue, Blue, Blue, Transp},
350   {Transp, Blue, Blue, Transp, Blue, Blue, Blue, Transp, Transp, Blue, Blue,
351     Blue, Transp, Blue, Blue, Transp},
352   {Transp, Blue, Transp, Transp, Transp, Blue, Blue, Transp, Transp, Blue,
353     Blue, Transp, Transp, Transp, Blue, Transp},
354   {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
355     Transp, Transp, Transp, Transp, Transp, Transp},
356 };
357
358 const uint8_t *sprites[] = {
359   (uint8_t *)sprite_pacman_closed, (uint8_t *)sprite_pacman_left,
360   (uint8_t *)sprite_pacman_right, (uint8_t *)sprite_pacman_up,
361   (uint8_t *)sprite_pacman_down, (uint8_t *)sprite_ghost_red,
362   (uint8_t *)sprite_ghost_cyan, (uint8_t *)sprite_ghost_pink,
363   (uint8_t *)sprite_ghost_orange, (uint8_t *)sprite_ghost_scatter,
364 };
365
366 void load_sprite_bitmaps() {
367   for (int i = 0; i < sizeof(sprites) / sizeof(const uint8_t *); i++) {

```



```

368     const uint8_t *pat = sprites[i];
369     set_sprite_bitmap(i, pat);
370 }
371 }
372
373 void set_sprite_bitmap(int spritei, const uint8_t *pat) {
374     vga_ball_arg_t arg;
375     int start;
376
377     arg.table = SPRITE_GENERATOR_TABLE;
378     start = spritei * SPRITE_BITMAP_SIZE;
379     for (int i = 0; i < SPRITE_BITMAP_SIZE; i++) {
380         arg.addr = start + i;
381         arg.data = sprite_pixel(pat[2 * i]) << 4 | sprite_pixel(pat[2 * i + 1]);
382         vga_ball_write(&arg);
383     }
384 }
385
386 void set_sprite(sprite_attr_t attr) {
387     vga_ball_arg_t arg;
388     int start;
389
390     start = 4 * attr.i;
391     arg.table = SPRITE_ATTRIBUTE_TABLE;
392
393     arg.addr = start;
394     arg.data = (uint8_t)(attr.y / 2);
395     vga_ball_write(&arg);
396
397     arg.addr = start + 1;
398     arg.data = (uint8_t)(attr.x / 2);
399     vga_ball_write(&arg);
400
401     arg.addr = start + 2;
402     arg.data = attr.name;
403     vga_ball_write(&arg);
404 }

```

Listing 17: sprite.c

```

1 #ifndef _MAP_H
2 #define _MAP_H
3
4 #include <stdint.h>
5
6 #define MAP_NROW 36
7 #define MAP_NCOL 28
8
9 #define MAP_ROW_OFFSET 12
10 #define MAP_COL_OFFSET 16
11
12 void clear_screen();
13 void set_map_at(int r, int c, uint8_t name);
14 uint8_t get_map_at(int r, int c);
15 void set_local_map_at(int r, int c, uint8_t name);
16 void setup_map();
17 void setup_map_foods();
18 void setup_map_score();
19 void set_map_best_score(uint32_t score);
20 void set_map_player_score(uint32_t score);
21 void set_map_lives(uint8_t lives);
22
23 #endif

```

Listing 18: map.h

```

1 #include "map.h"
2 #include "pattern.h"
3 #include "sprite.h"
4
5 // static uint8_t map[31][28];

```

```

6
7 static uint8_t map[PATTERN_NROW][PATTERN_NCOL];
8
9 void clear_screen() {
10     int r, c;
11     for (r = 0; r < PATTERN_NROW; r++) {
12         for (c = 0; c < PATTERN_NCOL; c++) {
13             set_map_at(r, c, PAT_BACKGROUND);
14         }
15     }
16 }
17
18 void set_map_best_score(uint32_t score) {
19     // clear
20     int i;
21     for (i = 22; i <= 27; i++) {
22         set_local_map_at(1, i, PAT_BACKGROUND);
23     }
24
25     // digits
26     for (i = 27; i >= 22; i--) {
27         set_local_map_at(1, i, PAT_0 + score % 10);
28         score /= 10;
29         if (score == 0)
30             return;
31     }
32 }
33
34 void set_map_player_score(uint32_t score) {
35     // clear
36     int i;
37     for (i = 0; i <= 5; i++) {
38         set_local_map_at(1, i, PAT_BACKGROUND);
39     }
40
41     // digits
42     for (i = 5; i >= 0; i--) {
43         set_local_map_at(1, i, PAT_0 + score % 10);
44         score /= 10;
45         if (score == 0)
46             return;
47     }
48 }
49
50 void set_map_lives(uint8_t lives) {
51     set_local_map_at(35, 0, PAT_L);
52     set_local_map_at(35, 1, PAT_I);
53     set_local_map_at(35, 2, PAT_V);
54     set_local_map_at(35, 3, PAT_E);
55     set_local_map_at(35, 4, PAT_S);
56
57     set_local_map_at(35, 6, PAT_0 + lives);
58 }
59
60 void set_map_at(int r, int c, uint8_t name) {
61     map[r][c] = name;
62     set_pattern_at(r, c, name);
63 }
64
65 uint8_t get_map_at(int r, int c) { return map[r][c]; }
66
67 void set_local_map_at(int r, int c, uint8_t name) {
68     set_map_at(MAP_ROW_OFFSET + r, MAP_COL_OFFSET + c, name);
69 }
70
71 void setup_map_score() {
72     set_local_map_at(0, 0, PAT_P);
73     set_local_map_at(0, 1, PAT_L);
74     set_local_map_at(0, 2, PAT_A);
75     set_local_map_at(0, 3, PAT_Y);
76     set_local_map_at(0, 4, PAT_E);

```

```

77 set_local_map_at(0, 5, PAT_R);
78
79 set_local_map_at(0, 18, PAT_H);
80 set_local_map_at(0, 19, PAT_I);
81 set_local_map_at(0, 20, PAT_G);
82 set_local_map_at(0, 21, PAT_H);
83
84 set_local_map_at(0, 23, PAT_S);
85 set_local_map_at(0, 24, PAT_C);
86 set_local_map_at(0, 25, PAT_O);
87 set_local_map_at(0, 26, PAT_R);
88 set_local_map_at(0, 27, PAT_E);
89 }
90
91 void setup_map() {
92     int r;
93     int i;
94
95     // row 3
96     r = 3;
97     set_local_map_at(r, 0, PAT_WALL_4);
98     for (i = 1; i < 13; i++)
99         set_local_map_at(3, i, PAT_WALL_0);
100    set_local_map_at(r, 13, PAT_WALL_12);
101    set_local_map_at(r, 14, PAT_WALL_13);
102    for (i = 15; i < 27; i++)
103        set_local_map_at(3, i, PAT_WALL_0);
104    set_local_map_at(r, 27, PAT_WALL_5);
105
106    // row 4
107    r = 4;
108    set_local_map_at(r, 0, PAT_WALL_3);
109
110    set_local_map_at(r, 13, PAT_WALL_20);
111    set_local_map_at(r, 14, PAT_WALL_22);
112
113    set_local_map_at(r, 27, PAT_WALL_1);
114
115    // row 5
116    r = 5;
117    set_local_map_at(r, 0, PAT_WALL_3);
118
119    set_local_map_at(r, 2, PAT_WALL_10);
120    set_local_map_at(r, 3, PAT_WALL_21);
121    set_local_map_at(r, 4, PAT_WALL_21);
122    set_local_map_at(r, 5, PAT_WALL_11);
123
124    set_local_map_at(r, 7, PAT_WALL_10);
125    set_local_map_at(r, 8, PAT_WALL_21);
126    set_local_map_at(r, 9, PAT_WALL_21);
127    set_local_map_at(r, 10, PAT_WALL_21);
128    set_local_map_at(r, 11, PAT_WALL_11);
129
130    set_local_map_at(r, 13, PAT_WALL_20);
131    set_local_map_at(r, 14, PAT_WALL_22);
132
133    set_local_map_at(r, 16, PAT_WALL_10);
134    set_local_map_at(r, 17, PAT_WALL_21);
135    set_local_map_at(r, 18, PAT_WALL_21);
136    set_local_map_at(r, 19, PAT_WALL_21);
137    set_local_map_at(r, 20, PAT_WALL_11);
138
139    set_local_map_at(r, 22, PAT_WALL_10);
140    set_local_map_at(r, 23, PAT_WALL_21);
141    set_local_map_at(r, 24, PAT_WALL_21);
142    set_local_map_at(r, 25, PAT_WALL_11);
143
144    set_local_map_at(r, 27, PAT_WALL_1);
145
146    // row 6
147    r = 6;

```

```

148 set_local_map_at(r, 0, PAT_WALL_3);
149
150 set_local_map_at(r, 2, PAT_WALL_20);
151 set_local_map_at(r, 5, PAT_WALL_22);
152
153 set_local_map_at(r, 7, PAT_WALL_20);
154 set_local_map_at(r, 11, PAT_WALL_22);
155
156 set_local_map_at(r, 13, PAT_WALL_20);
157 set_local_map_at(r, 14, PAT_WALL_22);
158
159 set_local_map_at(r, 16, PAT_WALL_20);
160 set_local_map_at(r, 20, PAT_WALL_22);
161
162 set_local_map_at(r, 22, PAT_WALL_20);
163 set_local_map_at(r, 25, PAT_WALL_22);
164
165 set_local_map_at(r, 27, PAT_WALL_1);
166
167 // row 7
168 r = 7;
169 set_local_map_at(r, 0, PAT_WALL_3);
170
171 set_local_map_at(r, 2, PAT_WALL_9);
172 set_local_map_at(r, 3, PAT_WALL_23);
173 set_local_map_at(r, 4, PAT_WALL_23);
174 set_local_map_at(r, 5, PAT_WALL_8);
175
176 set_local_map_at(r, 7, PAT_WALL_9);
177 set_local_map_at(r, 8, PAT_WALL_23);
178 set_local_map_at(r, 9, PAT_WALL_23);
179 set_local_map_at(r, 10, PAT_WALL_23);
180 set_local_map_at(r, 11, PAT_WALL_8);
181
182 set_local_map_at(r, 13, PAT_WALL_9);
183 set_local_map_at(r, 14, PAT_WALL_8);
184
185 set_local_map_at(r, 16, PAT_WALL_9);
186 set_local_map_at(r, 17, PAT_WALL_23);
187 set_local_map_at(r, 18, PAT_WALL_23);
188 set_local_map_at(r, 19, PAT_WALL_23);
189 set_local_map_at(r, 20, PAT_WALL_8);
190
191 set_local_map_at(r, 22, PAT_WALL_9);
192 set_local_map_at(r, 23, PAT_WALL_23);
193 set_local_map_at(r, 24, PAT_WALL_23);
194 set_local_map_at(r, 25, PAT_WALL_8);
195
196 set_local_map_at(r, 27, PAT_WALL_1);
197
198 // row 8
199 r = 8;
200 set_local_map_at(r, 0, PAT_WALL_3);
201
202 set_local_map_at(r, 27, PAT_WALL_1);
203
204 // row 9
205 r = 9;
206 set_local_map_at(r, 0, PAT_WALL_3);
207
208 set_local_map_at(r, 2, PAT_WALL_10);
209 set_local_map_at(r, 3, PAT_WALL_21);
210 set_local_map_at(r, 4, PAT_WALL_21);
211 set_local_map_at(r, 5, PAT_WALL_11);
212
213 set_local_map_at(r, 7, PAT_WALL_10);
214 set_local_map_at(r, 8, PAT_WALL_11);
215
216 set_local_map_at(r, 10, PAT_WALL_10);
217 set_local_map_at(r, 11, PAT_WALL_21);
218 set_local_map_at(r, 12, PAT_WALL_21);

```

```

219 set_local_map_at(r, 13, PAT_WALL_21);
220 set_local_map_at(r, 14, PAT_WALL_21);
221 set_local_map_at(r, 15, PAT_WALL_21);
222 set_local_map_at(r, 16, PAT_WALL_21);
223 set_local_map_at(r, 17, PAT_WALL_11);
224
225 set_local_map_at(r, 19, PAT_WALL_10);
226 set_local_map_at(r, 20, PAT_WALL_11);
227
228 set_local_map_at(r, 22, PAT_WALL_10);
229 set_local_map_at(r, 23, PAT_WALL_21);
230 set_local_map_at(r, 24, PAT_WALL_21);
231 set_local_map_at(r, 25, PAT_WALL_11);
232
233 set_local_map_at(r, 27, PAT_WALL_1);
234
235 // row 10
236 r = 10;
237 set_local_map_at(r, 0, PAT_WALL_3);
238
239 set_local_map_at(r, 2, PAT_WALL_9);
240 set_local_map_at(r, 3, PAT_WALL_23);
241 set_local_map_at(r, 4, PAT_WALL_23);
242 set_local_map_at(r, 5, PAT_WALL_8);
243
244 set_local_map_at(r, 7, PAT_WALL_20);
245 set_local_map_at(r, 8, PAT_WALL_22);
246
247 set_local_map_at(r, 10, PAT_WALL_9);
248 set_local_map_at(r, 11, PAT_WALL_23);
249 set_local_map_at(r, 12, PAT_WALL_23);
250 set_local_map_at(r, 13, PAT_WALL_11);
251 set_local_map_at(r, 14, PAT_WALL_10);
252 set_local_map_at(r, 15, PAT_WALL_23);
253 set_local_map_at(r, 16, PAT_WALL_23);
254 set_local_map_at(r, 17, PAT_WALL_8);
255
256 set_local_map_at(r, 19, PAT_WALL_20);
257 set_local_map_at(r, 20, PAT_WALL_22);
258
259 set_local_map_at(r, 22, PAT_WALL_9);
260 set_local_map_at(r, 23, PAT_WALL_23);
261 set_local_map_at(r, 24, PAT_WALL_23);
262 set_local_map_at(r, 25, PAT_WALL_8);
263
264 set_local_map_at(r, 27, PAT_WALL_1);
265
266 // row 11
267 r = 11;
268 set_local_map_at(r, 0, PAT_WALL_3);
269
270 set_local_map_at(r, 7, PAT_WALL_20);
271 set_local_map_at(r, 8, PAT_WALL_22);
272
273 set_local_map_at(r, 13, PAT_WALL_20);
274 set_local_map_at(r, 14, PAT_WALL_22);
275
276 set_local_map_at(r, 19, PAT_WALL_20);
277 set_local_map_at(r, 20, PAT_WALL_22);
278
279 set_local_map_at(r, 27, PAT_WALL_1);
280
281 // row 12
282 r = 12;
283 set_local_map_at(r, 0, PAT_WALL_7);
284 set_local_map_at(r, 1, PAT_WALL_2);
285 set_local_map_at(r, 2, PAT_WALL_2);
286 set_local_map_at(r, 3, PAT_WALL_2);
287 set_local_map_at(r, 4, PAT_WALL_2);
288 set_local_map_at(r, 5, PAT_WALL_11);
289

```

```

290 set_local_map_at(r, 7, PAT_WALL_20);
291 set_local_map_at(r, 8, PAT_WALL_9);
292 set_local_map_at(r, 9, PAT_WALL_21);
293 set_local_map_at(r, 10, PAT_WALL_21);
294 set_local_map_at(r, 11, PAT_WALL_11);
295
296 set_local_map_at(r, 13, PAT_WALL_20);
297 set_local_map_at(r, 14, PAT_WALL_22);
298
299 set_local_map_at(r, 16, PAT_WALL_10);
300 set_local_map_at(r, 17, PAT_WALL_21);
301 set_local_map_at(r, 18, PAT_WALL_21);
302 set_local_map_at(r, 19, PAT_WALL_8);
303 set_local_map_at(r, 20, PAT_WALL_22);
304
305 set_local_map_at(r, 22, PAT_WALL_10);
306 set_local_map_at(r, 23, PAT_WALL_2);
307 set_local_map_at(r, 24, PAT_WALL_2);
308 set_local_map_at(r, 25, PAT_WALL_2);
309 set_local_map_at(r, 26, PAT_WALL_2);
310 set_local_map_at(r, 27, PAT_WALL_6);
311
312 // row 13
313 r = 13;
314
315 set_local_map_at(r, 5, PAT_WALL_3);
316
317 set_local_map_at(r, 7, PAT_WALL_20);
318 set_local_map_at(r, 8, PAT_WALL_10);
319 set_local_map_at(r, 9, PAT_WALL_23);
320 set_local_map_at(r, 10, PAT_WALL_23);
321 set_local_map_at(r, 11, PAT_WALL_8);
322
323 set_local_map_at(r, 13, PAT_WALL_9);
324 set_local_map_at(r, 14, PAT_WALL_8);
325
326 set_local_map_at(r, 16, PAT_WALL_9);
327 set_local_map_at(r, 17, PAT_WALL_23);
328 set_local_map_at(r, 18, PAT_WALL_23);
329 set_local_map_at(r, 19, PAT_WALL_11);
330 set_local_map_at(r, 20, PAT_WALL_22);
331
332 set_local_map_at(r, 22, PAT_WALL_1);
333
334 // row 14
335 r = 14;
336 set_local_map_at(r, 5, PAT_WALL_3);
337
338 set_local_map_at(r, 7, PAT_WALL_20);
339 set_local_map_at(r, 8, PAT_WALL_22);
340
341 set_local_map_at(r, 19, PAT_WALL_20);
342 set_local_map_at(r, 20, PAT_WALL_22);
343
344 set_local_map_at(r, 22, PAT_WALL_1);
345
346 // row 15
347 r = 15;
348 set_local_map_at(r, 5, PAT_WALL_3);
349
350 set_local_map_at(r, 7, PAT_WALL_20);
351 set_local_map_at(r, 8, PAT_WALL_22);
352
353 set_local_map_at(r, 10, PAT_WALL_10);
354 set_local_map_at(r, 11, PAT_WALL_2);
355 set_local_map_at(r, 12, PAT_WALL_2);
356 set_local_map_at(r, 13, PAT_GATE);
357 set_local_map_at(r, 14, PAT_GATE);
358 set_local_map_at(r, 15, PAT_WALL_2);
359 set_local_map_at(r, 16, PAT_WALL_2);
360 set_local_map_at(r, 17, PAT_WALL_11);

```

```

361
362 set_local_map_at(r, 19, PAT_WALL_20);
363 set_local_map_at(r, 20, PAT_WALL_22);
364
365 set_local_map_at(r, 22, PAT_WALL_1);
366
367 // row 16
368 r = 16;
369 set_local_map_at(r, 5, PAT_WALL_3);
370
371 set_local_map_at(r, 7, PAT_WALL_9);
372 set_local_map_at(r, 8, PAT_WALL_8);
373
374 set_local_map_at(r, 10, PAT_WALL_1);
375
376 set_local_map_at(r, 17, PAT_WALL_3);
377
378 set_local_map_at(r, 19, PAT_WALL_9);
379 set_local_map_at(r, 20, PAT_WALL_8);
380
381 set_local_map_at(r, 22, PAT_WALL_1);
382
383 // row 17
384 r = 17;
385 set_local_map_at(r, 5, PAT_WALL_3);
386
387 set_local_map_at(r, 10, PAT_WALL_1);
388
389 set_local_map_at(r, 17, PAT_WALL_3);
390
391 set_local_map_at(r, 22, PAT_WALL_1);
392
393 // row 18
394 r = 18;
395 set_local_map_at(r, 5, PAT_WALL_3);
396
397 set_local_map_at(r, 7, PAT_WALL_10);
398 set_local_map_at(r, 8, PAT_WALL_11);
399
400 set_local_map_at(r, 10, PAT_WALL_1);
401
402 set_local_map_at(r, 17, PAT_WALL_3);
403
404 set_local_map_at(r, 19, PAT_WALL_10);
405 set_local_map_at(r, 20, PAT_WALL_11);
406
407 set_local_map_at(r, 22, PAT_WALL_1);
408
409 // row 19
410 r = 19;
411 set_local_map_at(r, 5, PAT_WALL_3);
412
413 set_local_map_at(r, 7, PAT_WALL_20);
414 set_local_map_at(r, 8, PAT_WALL_22);
415
416 set_local_map_at(r, 10, PAT_WALL_9);
417 set_local_map_at(r, 11, PAT_WALL_0);
418 set_local_map_at(r, 12, PAT_WALL_0);
419 set_local_map_at(r, 13, PAT_WALL_0);
420 set_local_map_at(r, 14, PAT_WALL_0);
421 set_local_map_at(r, 15, PAT_WALL_0);
422 set_local_map_at(r, 16, PAT_WALL_0);
423 set_local_map_at(r, 17, PAT_WALL_8);
424
425 set_local_map_at(r, 19, PAT_WALL_20);
426 set_local_map_at(r, 20, PAT_WALL_22);
427
428 set_local_map_at(r, 22, PAT_WALL_1);
429
430 // row 20
431 r = 20;

```

```

432 set_local_map_at(r, 5, PAT_WALL_3);
433
434 set_local_map_at(r, 7, PAT_WALL_20);
435 set_local_map_at(r, 8, PAT_WALL_22);
436
437 set_local_map_at(r, 19, PAT_WALL_20);
438 set_local_map_at(r, 20, PAT_WALL_22);
439
440 set_local_map_at(r, 22, PAT_WALL_1);
441
442 // row 21
443 r = 21;
444 set_local_map_at(r, 5, PAT_WALL_3);
445
446 set_local_map_at(r, 7, PAT_WALL_20);
447 set_local_map_at(r, 8, PAT_WALL_22);
448
449 set_local_map_at(r, 10, PAT_WALL_10);
450 set_local_map_at(r, 11, PAT_WALL_21);
451 set_local_map_at(r, 12, PAT_WALL_21);
452 set_local_map_at(r, 13, PAT_WALL_21);
453 set_local_map_at(r, 14, PAT_WALL_21);
454 set_local_map_at(r, 15, PAT_WALL_21);
455 set_local_map_at(r, 16, PAT_WALL_21);
456 set_local_map_at(r, 17, PAT_WALL_11);
457
458 set_local_map_at(r, 19, PAT_WALL_20);
459 set_local_map_at(r, 20, PAT_WALL_22);
460
461 set_local_map_at(r, 22, PAT_WALL_1);
462
463 // row 22
464 r = 22;
465 set_local_map_at(r, 0, PAT_WALL_4);
466 set_local_map_at(r, 1, PAT_WALL_0);
467 set_local_map_at(r, 2, PAT_WALL_0);
468 set_local_map_at(r, 3, PAT_WALL_0);
469 set_local_map_at(r, 4, PAT_WALL_0);
470 set_local_map_at(r, 5, PAT_WALL_8);
471
472 set_local_map_at(r, 7, PAT_WALL_9);
473 set_local_map_at(r, 8, PAT_WALL_8);
474
475 set_local_map_at(r, 10, PAT_WALL_9);
476 set_local_map_at(r, 11, PAT_WALL_23);
477 set_local_map_at(r, 12, PAT_WALL_23);
478 set_local_map_at(r, 13, PAT_WALL_11);
479 set_local_map_at(r, 14, PAT_WALL_10);
480 set_local_map_at(r, 15, PAT_WALL_23);
481 set_local_map_at(r, 16, PAT_WALL_23);
482 set_local_map_at(r, 17, PAT_WALL_8);
483
484 set_local_map_at(r, 19, PAT_WALL_9);
485 set_local_map_at(r, 20, PAT_WALL_8);
486
487 set_local_map_at(r, 22, PAT_WALL_9);
488 set_local_map_at(r, 23, PAT_WALL_0);
489 set_local_map_at(r, 24, PAT_WALL_0);
490 set_local_map_at(r, 25, PAT_WALL_0);
491 set_local_map_at(r, 26, PAT_WALL_0);
492 set_local_map_at(r, 27, PAT_WALL_5);
493
494 // row 23
495 r = 23;
496 set_local_map_at(r, 0, PAT_WALL_3);
497
498 set_local_map_at(r, 13, PAT_WALL_20);
499 set_local_map_at(r, 14, PAT_WALL_22);
500
501 set_local_map_at(r, 27, PAT_WALL_1);
502

```



```

503 // row 24
504 r = 24;
505 set_local_map_at(r, 0, PAT_WALL_3);
506
507 set_local_map_at(r, 2, PAT_WALL_10);
508 set_local_map_at(r, 3, PAT_WALL_21);
509 set_local_map_at(r, 4, PAT_WALL_21);
510 set_local_map_at(r, 5, PAT_WALL_11);
511
512 set_local_map_at(r, 7, PAT_WALL_10);
513 set_local_map_at(r, 8, PAT_WALL_21);
514 set_local_map_at(r, 9, PAT_WALL_21);
515 set_local_map_at(r, 10, PAT_WALL_21);
516 set_local_map_at(r, 11, PAT_WALL_11);
517
518 set_local_map_at(r, 13, PAT_WALL_20);
519 set_local_map_at(r, 14, PAT_WALL_22);
520
521 set_local_map_at(r, 16, PAT_WALL_10);
522 set_local_map_at(r, 17, PAT_WALL_21);
523 set_local_map_at(r, 18, PAT_WALL_21);
524 set_local_map_at(r, 19, PAT_WALL_21);
525 set_local_map_at(r, 20, PAT_WALL_11);
526
527 set_local_map_at(r, 22, PAT_WALL_10);
528 set_local_map_at(r, 23, PAT_WALL_21);
529 set_local_map_at(r, 24, PAT_WALL_21);
530 set_local_map_at(r, 25, PAT_WALL_11);
531
532 set_local_map_at(r, 27, PAT_WALL_1);
533
534 // row 25
535 r = 25;
536
537 set_local_map_at(r, 0, PAT_WALL_3);
538
539 set_local_map_at(r, 2, PAT_WALL_9);
540 set_local_map_at(r, 3, PAT_WALL_23);
541 set_local_map_at(r, 4, PAT_WALL_11);
542 set_local_map_at(r, 5, PAT_WALL_22);
543
544 set_local_map_at(r, 7, PAT_WALL_9);
545 set_local_map_at(r, 8, PAT_WALL_23);
546 set_local_map_at(r, 9, PAT_WALL_23);
547 set_local_map_at(r, 10, PAT_WALL_23);
548 set_local_map_at(r, 11, PAT_WALL_8);
549
550 set_local_map_at(r, 13, PAT_WALL_9);
551 set_local_map_at(r, 14, PAT_WALL_8);
552
553 set_local_map_at(r, 16, PAT_WALL_9);
554 set_local_map_at(r, 17, PAT_WALL_23);
555 set_local_map_at(r, 18, PAT_WALL_23);
556 set_local_map_at(r, 19, PAT_WALL_23);
557 set_local_map_at(r, 20, PAT_WALL_8);
558
559 set_local_map_at(r, 22, PAT_WALL_20);
560 set_local_map_at(r, 23, PAT_WALL_10);
561 set_local_map_at(r, 24, PAT_WALL_23);
562 set_local_map_at(r, 25, PAT_WALL_8);
563
564 set_local_map_at(r, 27, PAT_WALL_1);
565
566 // row 26
567 r = 26;
568
569 set_local_map_at(r, 0, PAT_WALL_3);
570
571 set_local_map_at(r, 4, PAT_WALL_20);
572 set_local_map_at(r, 5, PAT_WALL_22);
573

```

```

574 set_local_map_at(r, 22, PAT_WALL_20);
575 set_local_map_at(r, 23, PAT_WALL_22);
576
577 set_local_map_at(r, 27, PAT_WALL_1);
578
579 // row 27
580 r = 27;
581
582 set_local_map_at(r, 0, PAT_WALL_19);
583 set_local_map_at(r, 1, PAT_WALL_21);
584 set_local_map_at(r, 2, PAT_WALL_11);
585
586 set_local_map_at(r, 4, PAT_WALL_20);
587 set_local_map_at(r, 5, PAT_WALL_22);
588
589 set_local_map_at(r, 7, PAT_WALL_10);
590 set_local_map_at(r, 8, PAT_WALL_11);
591
592 set_local_map_at(r, 10, PAT_WALL_10);
593 set_local_map_at(r, 11, PAT_WALL_21);
594 set_local_map_at(r, 12, PAT_WALL_21);
595 set_local_map_at(r, 13, PAT_WALL_21);
596 set_local_map_at(r, 14, PAT_WALL_21);
597 set_local_map_at(r, 15, PAT_WALL_21);
598 set_local_map_at(r, 16, PAT_WALL_21);
599 set_local_map_at(r, 17, PAT_WALL_11);
600
601 set_local_map_at(r, 19, PAT_WALL_10);
602 set_local_map_at(r, 20, PAT_WALL_11);
603
604 set_local_map_at(r, 22, PAT_WALL_20);
605 set_local_map_at(r, 23, PAT_WALL_22);
606
607 set_local_map_at(r, 25, PAT_WALL_10);
608 set_local_map_at(r, 26, PAT_WALL_21);
609 set_local_map_at(r, 27, PAT_WALL_14);
610
611 // row 28
612 r = 28;
613
614 set_local_map_at(r, 0, PAT_WALL_18);
615 set_local_map_at(r, 1, PAT_WALL_23);
616 set_local_map_at(r, 2, PAT_WALL_8);
617
618 set_local_map_at(r, 4, PAT_WALL_9);
619 set_local_map_at(r, 5, PAT_WALL_8);
620
621 set_local_map_at(r, 7, PAT_WALL_20);
622 set_local_map_at(r, 8, PAT_WALL_22);
623
624 set_local_map_at(r, 10, PAT_WALL_9);
625 set_local_map_at(r, 11, PAT_WALL_23);
626 set_local_map_at(r, 12, PAT_WALL_23);
627 set_local_map_at(r, 13, PAT_WALL_11);
628 set_local_map_at(r, 14, PAT_WALL_10);
629 set_local_map_at(r, 15, PAT_WALL_23);
630 set_local_map_at(r, 16, PAT_WALL_23);
631 set_local_map_at(r, 17, PAT_WALL_8);
632
633 set_local_map_at(r, 19, PAT_WALL_20);
634 set_local_map_at(r, 20, PAT_WALL_22);
635
636 set_local_map_at(r, 22, PAT_WALL_9);
637 set_local_map_at(r, 23, PAT_WALL_8);
638
639 set_local_map_at(r, 25, PAT_WALL_9);
640 set_local_map_at(r, 26, PAT_WALL_23);
641 set_local_map_at(r, 27, PAT_WALL_15);
642
643 // row 29
644 r = 29;

```

```

645
646 set_local_map_at(r, 0, PAT_WALL_3);
647
648 set_local_map_at(r, 7, PAT_WALL_20);
649 set_local_map_at(r, 8, PAT_WALL_22);
650
651 set_local_map_at(r, 13, PAT_WALL_20);
652 set_local_map_at(r, 14, PAT_WALL_22);
653
654 set_local_map_at(r, 19, PAT_WALL_20);
655 set_local_map_at(r, 20, PAT_WALL_22);
656
657 set_local_map_at(r, 27, PAT_WALL_1);
658
659 // row 30
660 r = 30;
661
662 set_local_map_at(r, 0, PAT_WALL_3);
663
664 set_local_map_at(r, 2, PAT_WALL_10);
665 set_local_map_at(r, 3, PAT_WALL_21);
666 set_local_map_at(r, 4, PAT_WALL_21);
667 set_local_map_at(r, 5, PAT_WALL_21);
668 set_local_map_at(r, 6, PAT_WALL_21);
669 set_local_map_at(r, 7, PAT_WALL_8);
670 set_local_map_at(r, 8, PAT_WALL_9);
671 set_local_map_at(r, 9, PAT_WALL_21);
672 set_local_map_at(r, 10, PAT_WALL_21);
673 set_local_map_at(r, 11, PAT_WALL_11);
674
675 set_local_map_at(r, 13, PAT_WALL_20);
676 set_local_map_at(r, 14, PAT_WALL_22);
677
678 set_local_map_at(r, 16, PAT_WALL_10);
679 set_local_map_at(r, 17, PAT_WALL_21);
680 set_local_map_at(r, 18, PAT_WALL_21);
681 set_local_map_at(r, 19, PAT_WALL_8);
682 set_local_map_at(r, 20, PAT_WALL_9);
683 set_local_map_at(r, 21, PAT_WALL_21);
684 set_local_map_at(r, 22, PAT_WALL_21);
685 set_local_map_at(r, 23, PAT_WALL_21);
686 set_local_map_at(r, 24, PAT_WALL_21);
687 set_local_map_at(r, 25, PAT_WALL_11);
688
689 set_local_map_at(r, 27, PAT_WALL_1);
690
691 // row 31
692 r = 31;
693
694 set_local_map_at(r, 0, PAT_WALL_3);
695
696 set_local_map_at(r, 2, PAT_WALL_9);
697 set_local_map_at(r, 3, PAT_WALL_23);
698 set_local_map_at(r, 4, PAT_WALL_23);
699 set_local_map_at(r, 5, PAT_WALL_23);
700 set_local_map_at(r, 6, PAT_WALL_23);
701 set_local_map_at(r, 7, PAT_WALL_23);
702 set_local_map_at(r, 8, PAT_WALL_23);
703 set_local_map_at(r, 9, PAT_WALL_23);
704 set_local_map_at(r, 10, PAT_WALL_23);
705 set_local_map_at(r, 11, PAT_WALL_8);
706
707 set_local_map_at(r, 13, PAT_WALL_9);
708 set_local_map_at(r, 14, PAT_WALL_8);
709
710 set_local_map_at(r, 16, PAT_WALL_9);
711 set_local_map_at(r, 17, PAT_WALL_23);
712 set_local_map_at(r, 18, PAT_WALL_23);
713 set_local_map_at(r, 19, PAT_WALL_23);
714 set_local_map_at(r, 20, PAT_WALL_23);
715 set_local_map_at(r, 21, PAT_WALL_23);

```

```

716 set_local_map_at(r, 22, PAT_WALL_23);
717 set_local_map_at(r, 23, PAT_WALL_23);
718 set_local_map_at(r, 24, PAT_WALL_23);
719 set_local_map_at(r, 25, PAT_WALL_8);
720
721 set_local_map_at(r, 27, PAT_WALL_1);
722
723 // row 32
724 r = 32;
725 set_local_map_at(r, 0, PAT_WALL_3);
726
727 set_local_map_at(r, 27, PAT_WALL_1);
728
729 // row 33
730 r = 33;
731 set_local_map_at(r, 0, PAT_WALL_7);
732 for (i = 1; i < 27; i++)
733     set_local_map_at(r, i, PAT_WALL_2);
734 set_local_map_at(r, 27, PAT_WALL_6);
735 }
736
737 void setup_map_foods() {
738     int i, r;
739
740     // row 4
741     r = 4;
742     for (i = 1; i < 13; i++)
743         set_local_map_at(r, i, PAT_FOOD_SM);
744     for (i = 15; i < 27; i++)
745         set_local_map_at(r, i, PAT_FOOD_SM);
746
747     // row 5
748     r = 5;
749     set_local_map_at(r, 1, PAT_FOOD_SM);
750     set_local_map_at(r, 6, PAT_FOOD_SM);
751     set_local_map_at(r, 12, PAT_FOOD_SM);
752     set_local_map_at(r, 15, PAT_FOOD_SM);
753     set_local_map_at(r, 21, PAT_FOOD_SM);
754     set_local_map_at(r, 26, PAT_FOOD_SM);
755
756     // row 6
757     r = 6;
758     set_local_map_at(r, 1, PAT_FOOD_LG);
759     set_local_map_at(r, 6, PAT_FOOD_SM);
760     set_local_map_at(r, 12, PAT_FOOD_SM);
761     set_local_map_at(r, 15, PAT_FOOD_SM);
762     set_local_map_at(r, 21, PAT_FOOD_SM);
763     set_local_map_at(r, 26, PAT_FOOD_LG);
764
765     // row 7
766     r = 7;
767     set_local_map_at(r, 1, PAT_FOOD_SM);
768     set_local_map_at(r, 6, PAT_FOOD_SM);
769     set_local_map_at(r, 12, PAT_FOOD_SM);
770     set_local_map_at(r, 15, PAT_FOOD_SM);
771     set_local_map_at(r, 21, PAT_FOOD_SM);
772     set_local_map_at(r, 26, PAT_FOOD_SM);
773
774     // row 8
775     r = 8;
776     for (i = 1; i < 27; i++)
777         set_local_map_at(r, i, PAT_FOOD_SM);
778
779     // row 9
780     r = 9;
781     set_local_map_at(r, 1, PAT_FOOD_SM);
782     set_local_map_at(r, 6, PAT_FOOD_SM);
783     set_local_map_at(r, 9, PAT_FOOD_SM);
784     set_local_map_at(r, 18, PAT_FOOD_SM);
785     set_local_map_at(r, 21, PAT_FOOD_SM);
786     set_local_map_at(r, 26, PAT_FOOD_SM);

```

```

787
788 // row 10
789 r = 10;
790 set_local_map_at(r, 1, PAT_FOOD_SM);
791 set_local_map_at(r, 6, PAT_FOOD_SM);
792 set_local_map_at(r, 9, PAT_FOOD_SM);
793 set_local_map_at(r, 18, PAT_FOOD_SM);
794 set_local_map_at(r, 21, PAT_FOOD_SM);
795 set_local_map_at(r, 26, PAT_FOOD_SM);
796
797 // row 11
798 r = 11;
799 for (i = 1; i < 7; i++)
800     set_local_map_at(r, i, PAT_FOOD_SM);
801 for (i = 9; i < 13; i++)
802     set_local_map_at(r, i, PAT_FOOD_SM);
803 for (i = 15; i < 19; i++)
804     set_local_map_at(r, i, PAT_FOOD_SM);
805 for (i = 21; i < 27; i++)
806     set_local_map_at(r, i, PAT_FOOD_SM);
807
808 // row 12
809 r = 12;
810 set_local_map_at(r, 6, PAT_FOOD_SM);
811 set_local_map_at(r, 21, PAT_FOOD_SM);
812
813 // row 13
814 r = 13;
815 set_local_map_at(r, 6, PAT_FOOD_SM);
816 set_local_map_at(r, 21, PAT_FOOD_SM);
817
818 // row 14
819 r = 14;
820 set_local_map_at(r, 6, PAT_FOOD_SM);
821 set_local_map_at(r, 21, PAT_FOOD_SM);
822
823 // row 15
824 r = 15;
825 set_local_map_at(r, 6, PAT_FOOD_SM);
826 set_local_map_at(r, 21, PAT_FOOD_SM);
827
828 // row 16
829 r = 16;
830 set_local_map_at(r, 6, PAT_FOOD_SM);
831 set_local_map_at(r, 21, PAT_FOOD_SM);
832
833 // row 17
834 r = 17;
835 set_local_map_at(r, 6, PAT_FOOD_SM);
836 set_local_map_at(r, 21, PAT_FOOD_SM);
837
838 // row 18
839 r = 18;
840 set_local_map_at(r, 6, PAT_FOOD_SM);
841 set_local_map_at(r, 21, PAT_FOOD_SM);
842
843 // row 19
844 r = 19;
845 set_local_map_at(r, 6, PAT_FOOD_SM);
846 set_local_map_at(r, 21, PAT_FOOD_SM);
847
848 // row 20
849 r = 20;
850 set_local_map_at(r, 6, PAT_FOOD_SM);
851 set_local_map_at(r, 21, PAT_FOOD_SM);
852
853 // row 21
854 r = 21;
855 set_local_map_at(r, 6, PAT_FOOD_SM);
856 set_local_map_at(r, 21, PAT_FOOD_SM);
857

```

```

858 // row 22
859 r = 22;
860 set_local_map_at(r, 6, PAT_FOOD_SM);
861 set_local_map_at(r, 21, PAT_FOOD_SM);
862
863 // row 23
864 r = 23;
865 for (i = 1; i < 13; i++)
866     set_local_map_at(r, i, PAT_FOOD_SM);
867 for (i = 15; i < 27; i++)
868     set_local_map_at(r, i, PAT_FOOD_SM);
869
870 // row 24
871 r = 24;
872 set_local_map_at(r, 1, PAT_FOOD_SM);
873 set_local_map_at(r, 6, PAT_FOOD_SM);
874 set_local_map_at(r, 12, PAT_FOOD_SM);
875 set_local_map_at(r, 15, PAT_FOOD_SM);
876 set_local_map_at(r, 21, PAT_FOOD_SM);
877 set_local_map_at(r, 26, PAT_FOOD_SM);
878
879 // row 25
880 r = 25;
881 set_local_map_at(r, 1, PAT_FOOD_SM);
882 set_local_map_at(r, 6, PAT_FOOD_SM);
883 set_local_map_at(r, 12, PAT_FOOD_SM);
884 set_local_map_at(r, 15, PAT_FOOD_SM);
885 set_local_map_at(r, 21, PAT_FOOD_SM);
886 set_local_map_at(r, 26, PAT_FOOD_SM);
887
888 // row 26
889 r = 26;
890 set_local_map_at(r, 1, PAT_FOOD_LG);
891 set_local_map_at(r, 2, PAT_FOOD_SM);
892 set_local_map_at(r, 3, PAT_FOOD_SM);
893 for (i = 6; i < 13; i++)
894     set_local_map_at(r, i, PAT_FOOD_SM);
895 for (i = 15; i < 22; i++)
896     set_local_map_at(r, i, PAT_FOOD_SM);
897 set_local_map_at(r, 24, PAT_FOOD_SM);
898 set_local_map_at(r, 25, PAT_FOOD_SM);
899 set_local_map_at(r, 26, PAT_FOOD_LG);
900
901 // row 27
902 r = 27;
903 set_local_map_at(r, 3, PAT_FOOD_SM);
904 set_local_map_at(r, 6, PAT_FOOD_SM);
905 set_local_map_at(r, 9, PAT_FOOD_SM);
906 set_local_map_at(r, 18, PAT_FOOD_SM);
907 set_local_map_at(r, 21, PAT_FOOD_SM);
908 set_local_map_at(r, 24, PAT_FOOD_SM);
909
910 // row 28
911 r = 28;
912 set_local_map_at(r, 3, PAT_FOOD_SM);
913 set_local_map_at(r, 6, PAT_FOOD_SM);
914 set_local_map_at(r, 9, PAT_FOOD_SM);
915 set_local_map_at(r, 18, PAT_FOOD_SM);
916 set_local_map_at(r, 21, PAT_FOOD_SM);
917 set_local_map_at(r, 24, PAT_FOOD_SM);
918
919 // row 29
920 r = 29;
921 for (i = 1; i < 7; i++)
922     set_local_map_at(r, i, PAT_FOOD_SM);
923 for (i = 9; i < 13; i++)
924     set_local_map_at(r, i, PAT_FOOD_SM);
925 for (i = 15; i < 19; i++)
926     set_local_map_at(r, i, PAT_FOOD_SM);
927 for (i = 21; i < 27; i++)
928     set_local_map_at(r, i, PAT_FOOD_SM);

```

```

929
930 // row 30
931 r = 30;
932 set_local_map_at(r, 1, PAT_FOOD_SM);
933 set_local_map_at(r, 12, PAT_FOOD_SM);
934 set_local_map_at(r, 15, PAT_FOOD_SM);
935 set_local_map_at(r, 26, PAT_FOOD_SM);
936
937 // row 31
938 r = 31;
939 set_local_map_at(r, 1, PAT_FOOD_SM);
940 set_local_map_at(r, 12, PAT_FOOD_SM);
941 set_local_map_at(r, 15, PAT_FOOD_SM);
942 set_local_map_at(r, 26, PAT_FOOD_SM);
943
944 // row 32
945 r = 32;
946 for (i = 1; i < 27; i++)
947     set_local_map_at(r, i, PAT_FOOD_SM);
948 }

```

Listing 19: map.c

```

1 #ifndef _GAMEPLAY_H
2 #define _GAMEPLAY_H
3
4 #include "sprite.h"
5 #include <stdbool.h>
6
7 typedef enum {
8     STAGE_MENU,
9     STAGE_IN_GAME,
10    STAGE_END_GAME,
11 } game_stage_t;
12
13 typedef enum {
14     DIR_NONE,
15     DIR_LEFT,
16     DIR_RIGHT,
17     DIR_UP,
18     DIR_DOWN,
19 } dir_t;
20
21 typedef struct {
22     dir_t dir0;
23     dir_t dir1;
24     sprite_attr_t attr;
25 } pacman_t;
26
27 typedef struct {
28     dir_t dir;
29     sprite_attr_t attr;
30     int release;
31     int trapped_dir;
32     int scatter;
33 } ghost_t;
34
35 void setup_game();
36
37 bool can_turn(int r, int c, dir_t);
38
39 bool will_collide(int r, int c, dir_t);
40
41 bool is_wall(uint8_t pat);
42
43 void reset_characters();
44
45 void reset_lives();
46
47 void reset_scores();
48

```

```

49 void set_pacman_dir(dir_t dir);
50
51 bool blink_timer();
52
53 bool pacman_move_timer();
54
55 bool ghost_move_timer();
56
57 bool ghost_release_timer();
58
59 bool ghost_trapped_move_timer();
60
61 void move_pacman();
62
63 void eat_food();
64
65 void ghosts_catch_pacman();
66
67 void update_player_score(int s);
68
69 void animate_pacman();
70
71 void release_ghost();
72
73 void move_ghosts();
74
75 void move_ghosts_trapped();
76
77 void move_ghosts_release();
78
79 void move_ghost_random();
80
81 void release_next_ghost();
82
83 bool need_turn(int r, int c, dir_t dir);
84
85 void reset_game();
86
87 game_stage_t get_game_stage();
88
89 void press_start_game();
90
91 void end_game();
92
93 void update_scores();
94
95 bool beat_best_score();
96
97 void scatter_timer();
98
99 void next_life();
100
101 #endif

```

Listing 20: gameplay.h

```

1 #include "gameplay.h"
2 #include "map.h"
3 #include "pattern.h"
4 #include <limits.h>
5 #include <pthread.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <sys/queue.h>
9 #include <unistd.h>
10
11 typedef struct {
12     pthread_mutex_t mu;
13     game_stage_t stage;
14     int scatter_time;
15     bool scatter_mode;

```



```

16  uint32_t player_score;
17  uint32_t best_score;
18  uint8_t lives;
19  uint8_t nreleased;
20  uint32_t release_timer;
21  pacman_t pacman;
22  ghost_t ghost_red;
23  ghost_t ghost_cyan;
24  ghost_t ghost_pink;
25  ghost_t ghost_orange;
26 } game_state_t;
27
28 static game_state_t game;
29
30 void setup_game() {
31     pthread_mutex_init(&game.mu, NULL);
32     pthread_mutex_lock(&game.mu);
33
34     game.stage = STAGE_MENU;
35     game.release_timer = 0;
36     game.nreleased = 1;
37
38     game.scatter_time = 0;
39     game.scatter_mode = false;
40     game.ghost_red.scatter = 0;
41     game.ghost_cyan.scatter = 0;
42     game.ghost_pink.scatter = 0;
43     game.ghost_orange.scatter = 0;
44
45     setup_map_foods();
46
47     reset_scores();
48     reset_lives();
49     reset_characters();
50
51     printf("Game is ready\n");
52     pthread_mutex_unlock(&game.mu);
53 }
54
55 void reset_game() {
56     pthread_mutex_lock(&game.mu);
57
58     game.stage = STAGE_MENU;
59     game.release_timer = 0;
60     game.nreleased = 1;
61
62     game.scatter_time = 0;
63     game.scatter_mode = false;
64     game.ghost_red.scatter = 0;
65     game.ghost_cyan.scatter = 0;
66     game.ghost_pink.scatter = 0;
67     game.ghost_orange.scatter = 0;
68
69     setup_map_foods();
70     reset_lives();
71     reset_characters();
72
73     pthread_mutex_unlock(&game.mu);
74 }
75
76 void hide_lives() {
77     for (int c = 0; c <= 6; c++) {
78         set_local_map_at(35, c, PAT_BACKGROUND);
79     }
80 }
81
82 void show_lives() {
83     static int counter = 0;
84     static int flip = 1;
85     counter = (counter + 1) % 800;
86

```

```

87  if (counter == 0)
88      flip *= -1;
89
90  if (flip == 1) {
91      set_map_lives(game.lives);
92  } else {
93      hide_lives();
94  }
95 }
96
97 void next_life() {
98     pthread_mutex_lock(&game.mu);
99
100    game.lives--;
101    if (game.lives >= 1) {
102
103        // blink lives
104        {
105            uint32_t counter = 0;
106            while (counter < 2500) {
107                counter++;
108                show_lives();
109                usleep(1000);
110            }
111        }
112        set_map_lives(game.lives);
113
114        // reset positions
115        game.release_timer = 0;
116        game.nreleased = 1;
117
118        game.scatter_time = 0;
119        game.scatter_mode = false;
120        game.ghost_red.scatter = 0;
121        game.ghost_cyan.scatter = 0;
122        game.ghost_pink.scatter = 0;
123        game.ghost_orange.scatter = 0;
124
125        reset_characters();
126
127        {
128            uint32_t counter = 0;
129            while (counter < 2000) {
130                counter++;
131                usleep(1000);
132            }
133        }
134
135        pthread_mutex_unlock(&game.mu);
136    } else {
137
138        // blink lives
139        {
140            uint32_t counter = 0;
141            while (counter < 2500) {
142                counter++;
143                show_lives();
144                usleep(1000);
145            }
146        }
147        set_map_lives(game.lives);
148
149        pthread_mutex_unlock(&game.mu);
150        end_game();
151    }
152 }
153
154 game_stage_t get_game_stage() { return game.stage; }
155
156 void press_start_game() {
157     pthread_mutex_lock(&game.mu);

```

```

158     game.stage = STAGE_IN_GAME;
159     pthread_mutex_unlock(&game.mu);
160 }
161
162 void start_scatter_mode() {
163     game.scatter_mode = true;
164
165     game.ghost_red.attr.name = SPRITE_GHOST_SCATTER;
166     game.ghost_red.dir = DIR_NONE;
167     game.ghost_red.scatter = 1;
168     game.ghost_cyan.attr.name = SPRITE_GHOST_SCATTER;
169     game.ghost_cyan.dir = DIR_NONE;
170     game.ghost_cyan.scatter = 1;
171     game.ghost_pink.attr.name = SPRITE_GHOST_SCATTER;
172     game.ghost_pink.dir = DIR_NONE;
173     game.ghost_pink.scatter = 1;
174     game.ghost_orange.attr.name = SPRITE_GHOST_SCATTER;
175     game.ghost_orange.dir = DIR_NONE;
176     game.ghost_orange.scatter = 1;
177 }
178
179 void eat_food() {
180     uint8_t pat;
181     int r, c;
182     int tiler, tilec;
183
184     r = game.pacman.attr.y;
185     c = game.pacman.attr.x;
186
187     if (r % 8 == 4 && c % 8 == 4) {
188         tiler = 1 + r / 8;
189         tilec = 1 + c / 8;
190         pat = get_map_at(tiler, tilec);
191         if (pat == PAT_FOOD_SM) {
192             set_map_at(tiler, tilec, PAT_BACKGROUND);
193             update_player_score(10);
194         } else if (pat == PAT_FOOD_LG) {
195             set_map_at(tiler, tilec, PAT_BACKGROUND);
196             start_scatter_mode();
197             game.scatter_time = 0;
198             update_player_score(50);
199         }
200     }
201 }
202
203 void update_scores() {
204     if (game.player_score > game.best_score) {
205         game.best_score = game.player_score;
206     }
207     game.player_score = 0;
208     set_map_player_score(game.player_score);
209     set_map_best_score(game.best_score);
210 }
211
212 bool beat_best_score() { return game.player_score > game.best_score; }
213
214 void end_game() {
215     pthread_mutex_lock(&game.mu);
216     game.stage = STAGE_END_GAME;
217     pthread_mutex_unlock(&game.mu);
218 }
219
220 bool __ghost_catch_pacman(ghost_t *ghost) {
221     int r = ghost->attr.y;
222     int c = ghost->attr.x;
223
224     int aleft, aright, atop, abottom;
225     int bleft, bright, btop, bbottom;
226
227     aleft = c;
228     aright = c + 12;

```

```

229 atop = r + 12;
230 abottom = r;
231
232 bleft = game.pacman.attr.x;
233 bright = game.pacman.attr.x + 12;
234 btop = game.pacman.attr.y + 12;
235 bbottom = game.pacman.attr.y;
236
237 return aleft < bright && aright > bleft && atop > bbottom && abottom < btop;
238 }
239
240 void caught_ghost(int g, ghost_t *ghost) {
241     update_player_score(400);
242     ghost->scatter = 0;
243     ghost->release = 1;
244     ghost->attr.name = SPRITE_GHOST_RED + g;
245     ghost->attr.y = (MAP_ROW_OFFSET + 16) * 8;
246     ghost->attr.x = (MAP_COL_OFFSET + 13) * 8;
247
248     set_sprite(ghost->attr);
249 }
250
251 bool ghost_catch_pacman(int g, ghost_t *ghost) {
252     if (game.stage != STAGE_IN_GAME)
253         return false;
254
255     if (__ghost_catch_pacman(ghost)) {
256         if (ghost->scatter != 0) {
257             caught_ghost(g, ghost);
258             return true;
259         } else {
260             next_life();
261             return false;
262         }
263     }
264
265     return true;
266 }
267
268 void ghosts_catch_pacman() {
269     if (ghost_catch_pacman(0, &game.ghost_red)) {
270         if (ghost_catch_pacman(1, &game.ghost_cyan)) {
271             if (ghost_catch_pacman(2, &game.ghost_pink)) {
272                 ghost_catch_pacman(3, &game.ghost_orange);
273             }
274         }
275     }
276 }
277
278 void flip_ghost_scatter(int g, ghost_t *ghost) {
279     if (ghost->scatter == 0)
280         return;
281
282     if (ghost->attr.name == SPRITE_GHOST_SCATTER) {
283         ghost->attr.name = SPRITE_GHOST_RED + g;
284     } else {
285         ghost->attr.name = SPRITE_GHOST_SCATTER;
286     }
287
288     set_sprite(ghost->attr);
289 }
290
291 void end_ghost_scatter(int g, ghost_t *ghost) {
292     if (ghost->scatter == 0)
293         return;
294
295     ghost->scatter = 0;
296     ghost->attr.name = SPRITE_GHOST_RED + g;
297
298     set_sprite(ghost->attr);
299 }

```

```

300
301 void scatter_timer() {
302     if (game.scatter_mode) {
303         game.scatter_time++;
304         if (game.scatter_time >= 4000) {
305             if (game.scatter_time % 200 == 0) {
306                 flip_ghost_scatter(0, &game.ghost_red);
307                 flip_ghost_scatter(1, &game.ghost_cyan);
308                 flip_ghost_scatter(2, &game.ghost_pink);
309                 flip_ghost_scatter(3, &game.ghost_orange);
310             }
311
312             if (game.scatter_time >= 6000) {
313                 end_ghost_scatter(0, &game.ghost_red);
314                 end_ghost_scatter(1, &game.ghost_cyan);
315                 end_ghost_scatter(2, &game.ghost_pink);
316                 end_ghost_scatter(3, &game.ghost_orange);
317
318                 game.scatter_mode = false;
319             }
320         }
321     }
322 }
323
324 void update_player_score(int s) {
325     game.player_score += s;
326     set_map_player_score(game.player_score);
327 }
328
329 bool is_perpendicular(dir_t dir1, dir_t dir2) {
330     if (dir1 == DIR_LEFT || dir1 == DIR_RIGHT) {
331         return dir2 == DIR_UP || dir2 == DIR_DOWN;
332     }
333
334     if (dir1 == DIR_UP || dir1 == DIR_DOWN) {
335         return dir2 == DIR_LEFT || dir2 == DIR_RIGHT;
336     }
337
338     return false;
339 }
340
341 bool is_not_backward(dir_t dir1, dir_t dir2) {
342     return is_perpendicular(dir1, dir2) || dir1 == dir2;
343 }
344
345 void set_pacman_dir(dir_t dir) {
346     pthread_mutex_lock(&game.mu);
347
348     if (is_perpendicular(game.pacman.dir0, dir)) {
349         game.pacman.dir1 = dir;
350     } else {
351         game.pacman.dir0 = dir;
352         game.pacman.dir1 = DIR_NONE;
353     }
354
355     pthread_mutex_unlock(&game.mu);
356 }
357
358 void reset_lives() {
359     game.lives = 3;
360
361     set_map_lives(game.lives);
362 }
363
364 void reset_scores() {
365     game.player_score = 0;
366     game.best_score = 0;
367
368     set_map_player_score(game.player_score);
369     set_map_best_score(game.best_score);
370 }

```

```

371
372 void animate_pacman() {
373     if (game.pacman.attr.name == SPRITE_PACMAN_CLOSED) {
374         game.pacman.attr.name = SPRITE_PACMAN_CLOSED + game.pacman.dir0;
375     } else {
376         game.pacman.attr.name = SPRITE_PACMAN_CLOSED;
377     }
378 }
379
380 void reset_characters() {
381     game.pacman.dir0 = DIR_NONE;
382     game.pacman.dir1 = DIR_NONE;
383     game.pacman.attr.i = 0;
384     game.pacman.attr.y = (MAP_ROW_OFFSET + 25) * 8 + 4;
385     game.pacman.attr.x = (MAP_COL_OFFSET + 13) * 8;
386     game.pacman.attr.name = SPRITE_PACMAN_CLOSED;
387
388     game.ghost_red.release = 2;
389     game.ghost_red.trapped_dir = -1;
390     game.ghost_red.dir = DIR_NONE;
391     game.ghost_red.attr.i = 1;
392     game.ghost_red.attr.y = (MAP_ROW_OFFSET + 13) * 8 + 4;
393     game.ghost_red.attr.x = (MAP_COL_OFFSET + 13) * 8;
394     game.ghost_red.attr.name = SPRITE_GHOST_RED;
395
396     game.ghost_cyan.release = 0;
397     game.ghost_cyan.trapped_dir = -1;
398     game.ghost_cyan.dir = DIR_NONE;
399     game.ghost_cyan.attr.i = 2;
400     game.ghost_cyan.attr.y = (MAP_ROW_OFFSET + 17) * 8;
401     game.ghost_cyan.attr.x = (MAP_COL_OFFSET + 11) * 8;
402     game.ghost_cyan.attr.name = SPRITE_GHOST_CYAN;
403
404     game.ghost_pink.release = 0;
405     game.ghost_pink.trapped_dir = 1;
406     game.ghost_pink.dir = DIR_NONE;
407     game.ghost_pink.attr.i = 3;
408     game.ghost_pink.attr.y = (MAP_ROW_OFFSET + 16) * 8;
409     game.ghost_pink.attr.x = (MAP_COL_OFFSET + 13) * 8;
410     game.ghost_pink.attr.name = SPRITE_GHOST_PINK;
411
412     game.ghost_orange.release = 0;
413     game.ghost_orange.trapped_dir = -1;
414     game.ghost_orange.dir = DIR_NONE;
415     game.ghost_orange.attr.i = 4;
416     game.ghost_orange.attr.y = (MAP_ROW_OFFSET + 17) * 8;
417     game.ghost_orange.attr.x = (MAP_COL_OFFSET + 15) * 8;
418     game.ghost_orange.attr.name = SPRITE_GHOST_ORANGE;
419
420     set_sprite(game.pacman.attr);
421     set_sprite(game.ghost_red.attr);
422     set_sprite(game.ghost_cyan.attr);
423     set_sprite(game.ghost_pink.attr);
424     set_sprite(game.ghost_orange.attr);
425 }
426
427 bool blink_timer() {
428     static int counter = 0;
429     counter = (counter + 1) % 120;
430     return counter == 0;
431 }
432
433 bool pacman_move_timer() {
434     static int counter = 0;
435     counter = (counter + 1) % 15;
436     return counter == 0;
437 }
438
439 bool ghost_move_timer() {
440     static int counter = 0;
441     counter = (counter + 1) % 20;

```

```

442     return counter == 0;
443 }
444
445 bool ghost_trapped_move_timer() {
446     static int counter = 0;
447     counter = (counter + 1) % 50;
448     return counter == 0;
449 }
450
451 bool ghost_release_timer() {
452     game.release_timer = (game.release_timer + 1) % 2000;
453     return game.release_timer == 0;
454 }
455
456 void release_next_ghost() {
457     if (game.ghost_pink.release == 0) {
458         game.ghost_pink.release = 1;
459         return;
460     }
461     if (game.ghost_cyan.release == 0) {
462         game.ghost_cyan.release = 1;
463         return;
464     }
465     if (game.ghost_orange.release == 0) {
466         game.ghost_orange.release = 1;
467         return;
468     }
469 }
470
471 void move_ghost_with_dir(ghost_t *ghost) {
472     switch (ghost->dir) {
473     case DIR_NONE:
474         break;
475     case DIR_LEFT:
476         ghost->attr.x--;
477         break;
478     case DIR_RIGHT:
479         ghost->attr.x++;
480         break;
481     case DIR_UP:
482         ghost->attr.y--;
483         break;
484     case DIR_DOWN:
485         ghost->attr.y++;
486         break;
487     }
488
489     set_sprite(ghost->attr);
490 }
491
492 void release_ghost(ghost_t *ghost) {
493     if (ghost->release == 0) {
494         ghost->release = 1;
495     }
496 }
497
498 void move_ghost_trapped(ghost_t *ghost) {
499     if (ghost->release != 0)
500         return;
501
502     if (ghost->attr.y >= (MAP_ROW_OFFSET + 17) * 8) {
503         ghost->trapped_dir = -1;
504     }
505     if (ghost->attr.y <= (MAP_ROW_OFFSET + 16) * 8) {
506         ghost->trapped_dir = 1;
507     }
508
509     ghost->attr.y += ghost->trapped_dir;
510     set_sprite(ghost->attr);
511 }
512

```

```

513 void move_ghosts_trapped() {
514     move_ghost_trapped(&game.ghost_red);
515     move_ghost_trapped(&game.ghost_cyan);
516     move_ghost_trapped(&game.ghost_pink);
517     move_ghost_trapped(&game.ghost_orange);
518 }
519
520 void move_ghost_release(ghost_t *ghost) {
521     if (ghost->release != 1)
522         return;
523
524     int target_r = (MAP_ROW_OFFSET + 13) * 8 + 4;
525     int target_c = (MAP_COL_OFFSET + 13) * 8;
526
527     if (ghost->attr.x != target_c) {
528         int d = 1;
529         if (ghost->attr.x > target_c)
530             d = -1;
531
532         ghost->attr.x += d;
533         set_sprite(ghost->attr);
534         return;
535     }
536
537     if (ghost->attr.y != target_r) {
538         int d = 1;
539         if (ghost->attr.y > target_r) {
540             d = -1;
541         }
542         ghost->attr.y += d;
543         set_sprite(ghost->attr);
544         return;
545     }
546
547     ghost->release = 2;
548 }
549
550 void move_ghosts_release() {
551     move_ghost_release(&game.ghost_red);
552     move_ghost_release(&game.ghost_cyan);
553     move_ghost_release(&game.ghost_pink);
554     move_ghost_release(&game.ghost_orange);
555 }
556
557 typedef struct {
558     int r;
559     int c;
560 } coordinate_t;
561
562 int search_depth_bfs(coordinate_t pos0, coordinate_t pacman_pos, dir_t dir0,
563                     coordinate_t visited[], int nvisited) {
564     int found_depth;
565
566     if (pos0.r == pacman_pos.r && pos0.c == pacman_pos.c) {
567         return 0;
568     }
569
570     TAILQ_HEAD(tailhead, entry) head;
571     struct entry {
572         coordinate_t pos;
573         dir_t dir;
574         int depth;
575         TAILQ_ENTRY(entry) entries;
576     };
577     TAILQ_INIT(&head);
578
579     // add initial pos to visited
580     visited[nvisited++] = pos0;
581
582     // add initial pos to queue
583     struct entry *n0 = malloc(sizeof(struct entry));

```



```

584 n0->pos = pos0;
585 n0->dir = dir0;
586 n0->depth = 0;
587 TAILQ_INSERT_HEAD(&head, n0, entries);
588
589 while (!TAILQ_EMPTY(&head)) {
590     // dequeue
591     struct entry *e = TAILQ_FIRST(&head);
592
593     if (e->pos.r % 8 != 4 || e->pos.c % 8 != 4) {
594         fprintf(stderr, "WHAT\n");
595         exit(1);
596     }
597
598     // get neighbor directions
599     int n = 0;
600     dir_t candidates[4];
601     dir_t all_dirs[4] = {
602         DIR_LEFT,
603         DIR_RIGHT,
604         DIR_UP,
605         DIR_DOWN,
606     };
607     for (int i = 0; i < 4; i++) {
608         if (e->dir == DIR_NONE || is_not_backward(e->dir, all_dirs[i])) {
609             if (!will_collide(e->pos.r, e->pos.c, all_dirs[i])) {
610                 candidates[n] = all_dirs[i];
611                 n++;
612             }
613         }
614     }
615
616     if (n == 0) {
617         fprintf(stderr, "Can't have zero directions\n");
618         exit(1);
619     }
620
621     // loop through possible directions
622     for (int i = 0; i < n; i++) {
623         coordinate_t new_pos;
624         switch (candidates[i]) {
625             case DIR_LEFT:
626                 new_pos.r = e->pos.r;
627                 new_pos.c = e->pos.c - 8;
628                 break;
629             case DIR_RIGHT:
630                 new_pos.r = e->pos.r;
631                 new_pos.c = e->pos.c + 8;
632                 break;
633             case DIR_UP:
634                 new_pos.r = e->pos.r - 8;
635                 new_pos.c = e->pos.c;
636                 break;
637             case DIR_DOWN:
638                 new_pos.r = e->pos.r + 8;
639                 new_pos.c = e->pos.c;
640                 break;
641             case DIR_NONE:
642                 fprintf(stderr, "Search depth can't have a DIR_NONE.\n");
643                 exit(1);
644         }
645
646         // found pacman
647         if (new_pos.r == pacman_pos.r && new_pos.c == pacman_pos.c) {
648             found_depth = e->depth + 1;
649             goto found_target;
650         }
651
652         // test visited
653         bool has_visited = false;
654         for (int i = 0; i < nvisited; i++) {

```

```

655     coordinate_t v = visited[i];
656     if (v.r == new_pos.r && v.c == new_pos.c) {
657         has_visited = true;
658     }
659 }
660
661     if (!has_visited) {
662         struct entry *n = malloc(sizeof(struct entry));
663         n->pos = new_pos;
664         n->dir = candidates[i];
665         n->depth = e->depth + 1;
666         TAILQ_INSERT_TAIL(&head, n, entries);
667     }
668 }
669 TAILQ_REMOVE(&head, e, entries);
670 free(e);
671 }
672
673 found_target:
674 // cleanup
675 while (!TAILQ_EMPTY(&head)) {
676     struct entry *e = TAILQ_FIRST(&head);
677     TAILQ_REMOVE(&head, e, entries);
678     free(e);
679 }
680
681 return found_depth;
682 }
683
684 int search_depth_with_dir(coordinate_t pos, coordinate_t pacman_pos,
685                          dir_t dir) {
686     coordinate_t visited[2500];
687
688     // printf("search_depth_with_dir\n");
689     if (pos.r == pacman_pos.r && pos.c == pacman_pos.c)
690         return 0;
691
692     visited[0] = pos;
693
694     if (will_collide(pos.r, pos.c, dir)) {
695         // printf("search_depth_with_dir: will_collide\n");
696         return INT_MAX;
697     }
698
699     coordinate_t new_pos;
700
701     switch (dir) {
702     case DIR_LEFT:
703         new_pos.r = pos.r;
704         new_pos.c = pos.c - 8;
705         break;
706     case DIR_RIGHT:
707         new_pos.r = pos.r;
708         new_pos.c = pos.c + 8;
709         break;
710     case DIR_UP:
711         new_pos.r = pos.r - 8;
712         new_pos.c = pos.c;
713         break;
714     case DIR_DOWN:
715         new_pos.r = pos.r + 8;
716         new_pos.c = pos.c;
717         break;
718     case DIR_NONE:
719         fprintf(stderr, "Search depth can't have a DIR_NONE.");
720         exit(1);
721     }
722
723     return search_depth_bfs(new_pos, pacman_pos, dir, visited, 1);
724 }
725

```

```

726 void move_ghost_scatter(ghost_t *ghost) {
727     dir_t final_dir;
728     int results[4];
729
730     if (!need_turn(ghost->attr.y, ghost->attr.x, ghost->dir)) {
731         move_ghost_with_dir(ghost);
732         return;
733     }
734
735     coordinate_t pos, pacman_pos;
736
737     pos.r = (ghost->attr.y / 8) * 8 + 4;
738     pos.c = (ghost->attr.x / 8) * 8 + 4;
739
740     pacman_pos.r = (game.pacman.attr.y / 8) * 8 + 4;
741     pacman_pos.c = (game.pacman.attr.x / 8) * 8 + 4;
742
743     if (pos.r == pacman_pos.r && pos.c == pacman_pos.c) {
744         final_dir = DIR_NONE;
745     } else if (ghost->dir == DIR_NONE ||
746              (ghost->attr.x % 8 == 4 && ghost->attr.y % 8 == 4)) {
747         results[0] = search_depth_with_dir(pos, pacman_pos, DIR_LEFT);
748         results[1] = search_depth_with_dir(pos, pacman_pos, DIR_RIGHT);
749         results[2] = search_depth_with_dir(pos, pacman_pos, DIR_UP);
750         results[3] = search_depth_with_dir(pos, pacman_pos, DIR_DOWN);
751
752         int max_dir = -1;
753         int max_depth = -1;
754         for (int i = 0; i < 4; i++) {
755             if (results[i] > max_depth && results[i] != INT_MAX) {
756                 max_dir = i;
757                 max_depth = results[i];
758             }
759         }
760
761         if (max_dir == -1) {
762             final_dir = DIR_NONE;
763         } else {
764             final_dir = DIR_LEFT + max_dir;
765         }
766     } else {
767         final_dir = ghost->dir;
768     }
769
770     if (final_dir == DIR_NONE) {
771         move_ghost_random(ghost);
772     } else if (will_collide(ghost->attr.y, ghost->attr.x, final_dir)) {
773         final_dir = DIR_NONE;
774         move_ghost_random(ghost);
775     } else {
776         ghost->dir = final_dir;
777         move_ghost_with_dir(ghost);
778     }
779 }
780
781 void move_ghosts_scatter() {
782     move_ghost_scatter(&game.ghost_red);
783     move_ghost_scatter(&game.ghost_cyan);
784     move_ghost_scatter(&game.ghost_pink);
785     move_ghost_scatter(&game.ghost_orange);
786 }
787
788 void move_ghost_random(ghost_t *ghost) {
789     int n = 0;
790     dir_t candidates[4];
791     dir_t all_dirs[4] = {
792         DIR_LEFT,
793         DIR_RIGHT,
794         DIR_UP,
795         DIR_DOWN,
796     };

```

```

797
798 // get candidates
799 for (int i = 0; i < 4; i++) {
800     if (ghost->dir == DIR_NONE || is_not_backward(ghost->dir, all_dirs[i])) {
801         if (!will_collide(ghost->attr.y, ghost->attr.x, all_dirs[i])) {
802             candidates[n] = all_dirs[i];
803             n++;
804         }
805     }
806 }
807
808 // choose direction
809 if (n == 0) {
810     fprintf(stderr, "Ghost has nowhere to go. Can't happen\n");
811 }
812 int choice = rand() % n;
813 ghost->dir = candidates[choice];
814
815 move_ghost_with_dir(ghost);
816 }
817
818 void move_ghost_targeted(ghost_t *ghost) {
819     dir_t final_dir;
820     int results[4];
821
822     if (!need_turn(ghost->attr.y, ghost->attr.x, ghost->dir)) {
823         move_ghost_with_dir(ghost);
824         return;
825     }
826
827     coordinate_t pos, pacman_pos;
828
829     pos.r = (ghost->attr.y / 8) * 8 + 4;
830     pos.c = (ghost->attr.x / 8) * 8 + 4;
831
832     pacman_pos.r = (game.pacman.attr.y / 8) * 8 + 4;
833     pacman_pos.c = (game.pacman.attr.x / 8) * 8 + 4;
834
835     if (pos.r == pacman_pos.r && pos.c == pacman_pos.c) {
836         final_dir = DIR_NONE;
837     } else if (ghost->dir == DIR_NONE ||
838              (ghost->attr.x % 8 == 4 && ghost->attr.y % 8 == 4)) {
839         results[0] = search_depth_with_dir(pos, pacman_pos, DIR_LEFT);
840         results[1] = search_depth_with_dir(pos, pacman_pos, DIR_RIGHT);
841         results[2] = search_depth_with_dir(pos, pacman_pos, DIR_UP);
842         results[3] = search_depth_with_dir(pos, pacman_pos, DIR_DOWN);
843
844         int min_dir = -1;
845         int min_depth = INT_MAX;
846         for (int i = 0; i < 4; i++) {
847             if (results[i] < min_depth) {
848                 min_dir = i;
849                 min_depth = results[i];
850             }
851         }
852
853         if (min_dir == -1) {
854             final_dir = DIR_NONE;
855         } else {
856             final_dir = DIR_LEFT + min_dir;
857         }
858     } else {
859         final_dir = ghost->dir;
860     }
861
862     if (final_dir == DIR_NONE) {
863         move_ghost_random(ghost);
864     } else if (will_collide(ghost->attr.y, ghost->attr.x, final_dir)) {
865         final_dir = DIR_NONE;
866         move_ghost_random(ghost);
867     } else {

```

```

868     ghost->dir = final_dir;
869     move_ghost_with_dir(ghost);
870 }
871 }
872
873 void move_ghost(ghost_t *ghost) {
874     if (ghost->release != 2)
875         return;
876
877     int32_t deltar = ((int32_t)ghost->attr.y) - ((int32_t)game.pacman.attr.y);
878     int32_t deltac = ((int32_t)ghost->attr.x) - ((int32_t)game.pacman.attr.x);
879
880     // move_ghost_random(ghost);
881     if (ghost->scatter == 0) {
882         if (deltar * deltar + deltac * deltac <= (10 * 8) * (10 * 8)) {
883             move_ghost_targeted(ghost);
884         } else {
885             move_ghost_random(ghost);
886         }
887     } else if (ghost->scatter == 1) {
888         if (deltar * deltar + deltac * deltac <= (10 * 8) * (10 * 8)) {
889             move_ghost_scatter(ghost);
890         } else {
891             move_ghost_random(ghost);
892         }
893     }
894 }
895
896 void move_ghosts() {
897     move_ghost(&game.ghost_red);
898     move_ghost(&game.ghost_cyan);
899     move_ghost(&game.ghost_pink);
900     move_ghost(&game.ghost_orange);
901 }
902
903 void move_pacman() {
904     pthread_mutex_lock(&game.mu);
905
906     // try turn
907     if (can_turn(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir1)) {
908         game.pacman.dir0 = game.pacman.dir1;
909         game.pacman.dir1 = DIR_NONE;
910     }
911
912     // try dir0
913     if (will_collide(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir0)) {
914         game.pacman.dir0 = game.pacman.dir1;
915         game.pacman.dir1 = DIR_NONE;
916     }
917
918     // try dir1
919     if (will_collide(game.pacman.attr.y, game.pacman.attr.x, game.pacman.dir0)) {
920         game.pacman.dir0 = game.pacman.dir1;
921         game.pacman.dir1 = DIR_NONE;
922     }
923
924     switch (game.pacman.dir0) {
925     case DIR_NONE:
926         break;
927     case DIR_LEFT:
928         game.pacman.attr.x--;
929         break;
930     case DIR_RIGHT:
931         game.pacman.attr.x++;
932         break;
933     case DIR_UP:
934         game.pacman.attr.y--;
935         break;
936     case DIR_DOWN:
937         game.pacman.attr.y++;
938         break;

```

```

939     }
940
941     set_sprite(game.pacman.attr);
942
943     pthread_mutex_unlock(&game.mu);
944 }
945
946 bool can_turn(int r, int c, dir_t dir) {
947     uint8_t pat;
948
949     if (r % 8 != 4 || c % 8 != 4)
950         return false;
951
952     switch (dir) {
953     case DIR_NONE:
954         return false;
955     case DIR_LEFT:
956         pat = get_map_at(1 + r / 8, c / 8);
957         return !is_wall(pat);
958     case DIR_RIGHT:
959         pat = get_map_at(1 + r / 8, 2 + c / 8);
960         return !is_wall(pat);
961     case DIR_UP:
962         pat = get_map_at(r / 8, 1 + c / 8);
963         return !is_wall(pat);
964     case DIR_DOWN:
965         pat = get_map_at(2 + r / 8, 1 + c / 8);
966         return !is_wall(pat);
967     }
968
969     return false;
970 }
971
972 bool need_turn(int r, int c, dir_t dir) {
973     dir_t all_dirs[4] = {
974         DIR_LEFT,
975         DIR_RIGHT,
976         DIR_UP,
977         DIR_DOWN,
978     };
979
980     if (dir == DIR_NONE)
981         return true;
982
983     // get candidates
984     for (int i = 0; i < 4; i++) {
985         if (all_dirs[i] != dir && is_not_backward(dir, all_dirs[i])) {
986             if (!will_collide(r, c, all_dirs[i])) {
987                 return true;
988             }
989         }
990     }
991
992     return false;
993 }
994
995 bool will_collide(int r, int c, dir_t dir) {
996     uint8_t pat;
997     bool is_h, is_v;
998
999     is_h = (r % 8 == 4);
1000    is_v = (c % 8 == 4);
1001
1002    switch (dir) {
1003    case DIR_NONE:
1004        return false;
1005    case DIR_LEFT:
1006        if (!is_h)
1007            return true;
1008        if (!is_v)
1009            return false;

```

```

1010     pat = get_map_at(1 + r / 8, c / 8);
1011     return is_wall(pat);
1012 case DIR_RIGHT:
1013     if (!is_h)
1014         return true;
1015     if (!is_v)
1016         return false;
1017     pat = get_map_at(1 + r / 8, 2 + c / 8);
1018     return is_wall(pat);
1019 case DIR_UP:
1020     if (!is_v)
1021         return true;
1022     if (!is_h)
1023         return false;
1024     pat = get_map_at(r / 8, 1 + c / 8);
1025     return is_wall(pat);
1026 case DIR_DOWN:
1027     if (!is_v)
1028         return true;
1029     if (!is_h)
1030         return false;
1031     pat = get_map_at(2 + r / 8, 1 + c / 8);
1032     return is_wall(pat);
1033 }
1034
1035 return true; // shouldn't reach here
1036 }
1037
1038 bool is_eating_small(sprite_attr_t s) {
1039     uint8_t pat;
1040
1041     if (s.y % 8 == 4 && s.x % 8 == 4) {
1042         pat = get_map_at(1 + s.y / 8, 1 + s.x / 8);
1043         return pat == PAT_FOOD_SM;
1044     }
1045
1046     return false;
1047 }
1048
1049 bool is_eating_large(sprite_attr_t s) {
1050     uint8_t pat;
1051
1052     if (s.y % 8 == 4 && s.x % 8 == 4) {
1053         pat = get_map_at(1 + s.y / 8, 1 + s.x / 8);
1054         return pat == PAT_FOOD_LG;
1055     }
1056
1057     return false;
1058 }
1059
1060 bool is_wall(uint8_t pat) {
1061     if (PAT_WALL_0 <= pat && pat <= PAT_GATE) {
1062         return true;
1063     }
1064     return false;
1065 }

```

Listing 21: gameplay.c

```

1 #ifndef _COLOR_H
2 #define _COLOR_H
3
4 #define Transp 0x0
5 #define Yellow 0x1
6 #define Red 0x2
7 #define Orange 0x3
8 #define Cyan 0x4
9 #define Pink 0x5
10 #define Ivory 0x6
11 #define Blue 0x7
12 #define Salmon 0x8

```

```
13 #define White 0x9
14
15 #endif
```

Listing 22: color.h