实验名称：实验十二 自选大实验-贪吃蛇

姓名：张涵之

学号：191220154

班级：周一 5-6

邮箱：191220154@smail.nju.edu.cn

实验时间：2020/12/21-2020/12/27

实验内容：像素风格贪吃蛇小游戏。

实验目的：实现一个键盘控制、显示器显示的贪吃蛇小游戏。

实验原理：
1）游戏逻辑：显示器屏幕划分为 16x16 像素的小方格，共 30 行，每行 40 列，长度 1200 每位 2 比特的一维数组储存地图信息，不同的数字分别表示蛇、食物、围墙和空白区域。
蛇的实现：循环数组，每位记录当前身体节点在地图中的坐标，通过头尾指针 index 的移动实现蛇在地图中的移动，以及蛇每次吃下食物时的身体长度的增加。
食物的实现：随机生成坐标，读入地图，如果当前位置非空则继续生成。
通关和死亡：蛇每吃下一次食物身体增长一节，同时分数加一，获得十分则通关成功，在这之前如果蛇头撞上围墙或者撞上自己的身体，都会导致蛇的死亡，即通关失败。

2）玩家交互：玩家通过键盘上的 WASD 按键控制蛇的运动方向，只允许直行、左转和右转而不允许直接掉头。按空格键可以开始和暂停游戏，按 enter 键可以重置游戏。
游戏数据的刷新由固定时钟控制，是否更新、如何更新取决于当前玩家传入的键盘状态。

2）扫描显示：VGA 控制模块可以输出当前扫描到的行和列的位置信息，稍加改动即可让其输出当前扫描的位置对应 30×40 的方格阵列。利用坐标查询地图数组，根据该位置对应的选择输出颜色，蛇输出绿色，食物输出红色，围墙输出蓝色，空白区域输出黑色。

实验分工：
张涵之（报告作者）：游戏逻辑模块、顶层综合模块、调试和 DEBUG。
罗思明（队友）：键盘、显示器和时钟控制模块，协助调试 DEBUG，
游戏通关界面制作（使用 PS 和 MATLAB 处理图片并提供最终生成的 mif 文件）。

程序代码或流程图：
(其中红色为本报告作者完成，蓝色为队友完成

键盘控制模块：
ps2_keyboard　　//键盘控制器获取键码（复用 exp08 中代码
kbd_output　　//输入键码输出游戏状态和蛇的走向

显示器控制模块：
vga_ctrl　　//显示器控制器提供接口（复用 exp09 中代码
win_picture　　//游戏通关（获得 10 分）时显示的图片
lose_picture　　//蛇死亡（通关失败）时显示的图片

时钟控制模块
clkgen　　//生成特定频率的时钟（复用 exp09 中代码
clk_1s　　//生成游戏控制时钟（经调整其实最后周期并不是 1s

七段数码管控制模块：
hex　　//将分数在七段数码管进行显示

游戏控制模块：

Food　　//随机生成食物（0~1199 内的伪随机数

```
module Food(
    clk,
    random
);

    input clk;
    output [11:0] random;
    reg [11:0] count;

    initial begin
        count = 11'd0;
    end

    always @ (posedge clk) begin
        if (count == 11'd1199)
            count <= 11'd0;
        else
            count <= count + 1'd1;
    end

    assign random = count;

endmodule
```

//模块的实际功能其实就是一个 0~1199 的循环计数器

//通过时钟控制每次取的时候获得不同数字，从而达到随机的效果

Snake　　//游戏主逻辑（相当于顶层模块

```
module Snake(
    clk,
    ps2_clk,
    ps2_data,
    vga_clk,
    vga_hs,
    vga_vs,
    vga_blank_n,
    vga_r,
    vga_g,
    vga_b,
    hex0,
    hex1,
    game_state,
    game_clock
);
```

//模块提供系统时钟、键盘、显示器和七段数码管的接口

//game_state 和 game_clock 接入发光二极管用于调试

```
    input clk;                              //系统时钟
    input ps2_clk,ps2_data;                 //键盘相关
    output vga_clk,vga_hs,vga_vs,vga_blank_n;
    output [7:0] vga_r;
    output [7:0] vga_g;
    output [7:0] vga_b;                     //显示器相关
    output [6:0] hex0;
    output [6:0] hex1;                      //分数显示

    wire run;                               //开始/暂停
    wire restart;                           //重玩
    output game_clock;                      //游戏时钟
    wire [1:0] direction;                   //方向
    reg [1:0] board [1199:0];               //棋盘格
    reg [11:0] snake [15:0];                //蛇行坐标
    reg [3:0] snake_head_index;             //蛇头指针
    reg [3:0] snake_tail_index;             //蛇尾指针
    reg [11:0] food;                        //食物坐标
    wire [11:0] random;                     //随机生成坐标
    reg [7:0] score;                        //得分
    integer i;
    integer j;                              //循环用计数器
    integer new_head_index;
    integer new_tail_index;                 //蛇头尾临时指针
    reg eaten;                              //食物是否被吃
    reg dead;                               //蛇是否死亡
    reg win;                                //游戏是否通关
    output game_state;                      //游戏状态（是否进入循环）
    wire reset_game;                        //重置游戏信息
```

```
    wire nextdata_n;
    wire [7:0] data;
    wire ready;
    wire overflow;                          //键盘相关

    wire [9:0] h_addr;
    wire [9:0] v_addr;
    reg [18:0] addr = 19'h0;
    wire out_win_data;
    wire out_lose_data;
    reg [1:0] vga_state;
    reg [23:0] vga_data = 24'hffffff;    //显示器相关
```

//模块定义的对外接口和内部使用的临时变量

```
    assign game_state = run && !win && !dead;
    //如果玩家没有暂停，没有通关，蛇也没死，游戏时钟恰好经过一秒则进入循环
    assign reset_game = restart;
    //重置游戏信息

    clk_1s c_1s(clk,game_clock);
    clkgen #(25000000) my_clkgen(clk,1'b0,1'b1,vga_clk);
    //游戏时钟和VGA时钟信号的生成

    ps2_keyboard ps(clk,1'b1,ps2_clk,ps2_data,data,ready,nextdata_n,overflow);
    kbd_output kbd(clk,ready,overflow,data,direction,nextdata_n,run,restart);
    //从键盘模块获得玩家指令

    Food f(clk,random);
    //通过循环计数器获得随机行列坐标

    win_picture wp(addr,clk,out_win_data);
    lose_picture lp(addr,clk,out_lose_data);
    //通关信息显示图片
    vga_ctrl my_vga(vga_clk,1'b0,vga_data,h_addr,v_addr,
       vga_hs,vga_vs,vga_blank_n,vga_r,vga_g,vga_b);
    //显示器控制模块

    hex h0(score[3:0],1'b1,hex0);
    hex h1(score[7:4],1'b1,hex1);
    //七段数码管显示分数
```

//在主模块中调用的其他控制模块

```
    always @ (v_addr or h_addr) begin
       if (win) begin
          addr = v_addr + (h_addr - 1) * 512 - 1;
          case (out_win_data)
             1'b1: vga_data = 24'h00ff00;
             1'b0: vga_data = 24'h000000;
             default : vga_data = 24'hffffff;
          endcase
       end
       else if (dead) begin
          addr = v_addr + (h_addr - 1) * 512 - 1;
          case (out_lose_data)
             1'b1: vga_data = 24'hff0000;
             1'b0: vga_data = 24'h000000;
             default: vga_data = 24'hffffff;
          endcase
       end
       else begin
          vga_state = board[(v_addr >> 4) * 40 + (h_addr >> 4)];
          case (vga_state)
             NULL:    vga_data = 24'h000000;      //黑色，代表空地
             SNAKE:   vga_data = 24'h00ff00;      //绿色，代表蛇
             FOOD:    vga_data = 24'hff0000;      //红色，代表食物
             WALL:    vga_data = 24'h0000ff;      //蓝色，代表墙
             default: vga_data = 24'hffffff;
          endcase
       end
    end
```

//若通关或死亡则显示相应图片，否则显示当前地图信息

```
    localparam
       UP = 2'd0,
       DOWN = 2'd1,
       LEFT = 2'd2,
       RIGHT = 2'd3;
       //方向上下左右

    localparam
       NULL = 2'd0,
       SNAKE = 2'd1,
       FOOD = 2'd2,
       WALL = 2'd3;
       //棋盘格信息
```

//定义一些有名称的变量，类似 C 语言中的#define

//当且仅当 game_state 有效时更新游戏状态信息

```
always @ (posedge game_clock) begin
    if (game_state) begin
        new_head_index = (snake_head_index + 1) % 15;
        new_tail_index = (snake_tail_index + 1) % 15;
        //蛇为循环数组
        case (direction)
            UP: begin
                snake[new_head_index] = snake[snake_head_index] - 40;
            end
            DOWN: begin
                snake[new_head_index] = snake[snake_head_index] + 40;
            end
            LEFT: begin
                snake[new_head_index] = snake[snake_head_index] - 1;
            end
            RIGHT: begin
                snake[new_head_index] = snake[snake_head_index] + 1;
            end
        endcase
        //根据方向设置新头的坐标位置
```

//更新头尾下标指针（蛇的坐标是长度为 16 的循环数组

//根据当前的蛇头位置和运动方向设置蛇头新坐标

```
        case (board[snake[new_head_index]])
            NULL: begin
                board[snake[new_head_index]] = SNAKE;
                board[snake[snake_tail_index]] = NULL;
                snake_head_index = new_head_index;
                snake_tail_index = new_tail_index;
            end
            //如果是空的，则旧蛇尾所在地设为空，新蛇头所在地设为蛇
            SNAKE: dead = 1'b1;
            //如果是蛇，说明撞上了自己，蛇死了
            FOOD: begin
                board[snake[new_head_index]] = SNAKE;
                snake_head_index = new_head_index;
                eaten = 1'b1;
                score = score + 1;
                if (score == 10)
                    win = 1'b1;
            end
            //如果是食物，则旧蛇尾不变，食物所在地设为蛇，蛇吃掉食物并获得增长
            //如果更新后的蛇头与蛇尾指针下标重合，说明蛇满10节，游戏获胜
            WALL: dead = 1'b1;
            //如果是墙，说明撞到了墙，蛇也死了
        endcase
        if (eaten) begin
        //如果食物被吃则需要重新生成
            if (board[random] == NULL) begin
            //如果生成地为空则放置，否则继续生成
                food = random;
                board[food] = FOOD;
                eaten = 1'b0;
                //放置并修改信息
            end
        end
    end
```

//根据新头在棋盘格上位置的内容判断状态

//是正常爬行、撞死还是吃食物，若吃则食物重新生成

```
    if (reset_game) begin
        for (i = 0; i < 1200; i = i + 1)
            board[i] = NULL;
        for (j = 0; j < 40; j = j + 1) begin
            board[j] = WALL;
            board[1160 + j] = WALL;
        end
        for (i = 1; i < 29; i = i + 1) begin
            board[i * 40] = WALL;
            board[i * 40 + 39] = WALL;
        end
        for (j = 10; j < 20; j = j + 1)
            board[400 + j] = WALL;
        for (j = 20; j < 30; j = j + 1)
            board[760 + j] = WALL;
        for (i = 10; i < 20; i = i + 1) begin
            board[i * 40 + 10] = WALL;
            board[i * 40 + 29] = WALL;
        end
        //初始化四周是墙，中间留空
        board[82] = SNAKE;
        board[83] = SNAKE;
        snake_head_index = 4'd1;
        snake_tail_index = 4'd0;
```

```
            snake[0] = 11'd82;
            snake[1] = 11'd83;
            //把蛇放在右上角，进行初始化
            eaten = 1'b1;
            dead = 1'b0;
            win = 1'b0;
            score = 8'd0;
            //设置食物被吃（即此时需要随机生成）
            //蛇没死，游戏也没有通关
            //重置游戏信息
        end
    end
//重置游戏：清空地图，放置墙和蛇，恢复状态，分数清零
//也用于开机启动时初始化（即能玩之前需要先 reset 一次
```

实验环境/器材：实验箱一个，笔记本电脑一台，键盘一个，显示器一个。

实验步骤/过程：

1）初步构思：与队友商议后共同敲定选题，游戏逻辑根据我上学期在“高级程序设计”课程设计中用 C++写的控制台贪吃蛇小游戏改编，受到 Verilog 语言特性的限制，将蛇由链表改成循环数组实现，每次移动时链表头尾节点的插入和删除操作改成数组下标的移动，起初仍然打算使用二维数组表示棋盘格，定义各种本地变量表示方向和地图内容。

2）模块分工：由于我之前写过 C++贪吃蛇小游戏，对游戏逻辑的设计有一定初步思考，因此分工定为我来写游戏逻辑和顶层模块，为键盘、显示器和时钟模块提供接口，队友编写键盘、显示器、时钟控制和通关提示图片的显存读取模块，为顶层模块提供由玩家按下键盘控制的开始/暂停、重置和方向键，并在屏幕上显示游戏地图或通关/失败提示信息。

3）模块综合：队友提供的键盘、时钟模块都可通关输出接入发光二极管、七段数码管显示键码等方式调试，经检验功能全面正常，但我在综合这些模块时发现显示器显示非常混乱的抽象艺术图案（如图 1），游戏逻辑也不能正常运行，推测是对 Verilog 的多维数组实现逻辑理解有误，经过多次修改，仍然没有解决问题。这时我想到：既然多维数组定义和使用都容易出错，那就改成一维数组，计算下标去读取吧。然而，下标的计算非常繁琐，如果每次读取和写入都根据 i 和 j 计算一次，会造成资源的极大浪费，编译非常缓慢，既然这样，那为什么还要用 i 和 j 计算下标？已知数组的宽度，每次行+1 时下标+40 即可，这样较为顺利地解决了乱码显示问题，并且缩短了编译耗费的时间，减少了开发板上浪费的资源。

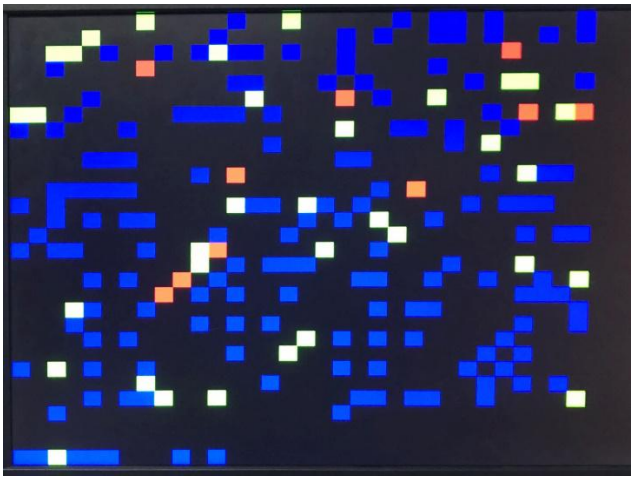

图 1 现代抽象装置艺术（Zhang, 2020）

//以下出现语气巨变，因为“现代抽象装置艺术”让我乐起来了，皮这一下很开心!
解决了地图数组的问题，游戏基本能玩了，第一次看见那个荧光绿的蛇（其实更像毛虫）在屏幕上爬起来的时候真的很激动，思明小朋友开始做游戏通关和失败界面，仿照实验 9 的实现用 PS 和 MATLAB 处理图片得到 mif 文件。此时我提出问题，假如通关和失败有两种不同的界面，那就需要读两张不同的图片，即两个超级巨大的 mif 文件。在实验 9 中，即使是单张图片的显示，都存在开发板内存资源不够，需要采用低比特颜色显示的问题，那么何况是两张图片呢！思明小朋友不信邪，我就让他试一试，果然不负众望地报错了：

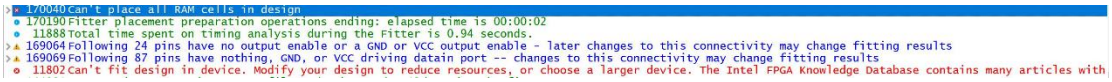

图 2 令人精神崩溃的报错

那么怎么办呢，既然通关和失败图片主要是显示艺术字，那么字的轮廓清晰，能看出是字就行了，颜色少点也无所谓。我让思明用查找-替换把 mif 文件中出现次数最多的数字（即背景底色）全都换成 0，其他全都换成 1，在 VGA 模块读 ram 设置显示的颜色时，读到 0 则设置黑色，读到 1 则设置其他，这样通关和失败图片就分别显示出黑底绿字和黑底红字。

图 3 用来生成 mif 文件的图片

//此处应有最终实现的效果图，但是我们都过于激动忘记了拍照

测试方法：各控制模块分别调试，游戏主模块综合调试，多玩几次看有什么 BUG。
实验结果：实现了类似诺基亚手机自带的复古像素风格小游戏，非常脑瘫，非常好玩，界面颜色非常鲜艳美丽，蛇的运动非常流畅自如，隔壁座位上的小朋友都馋哭了。

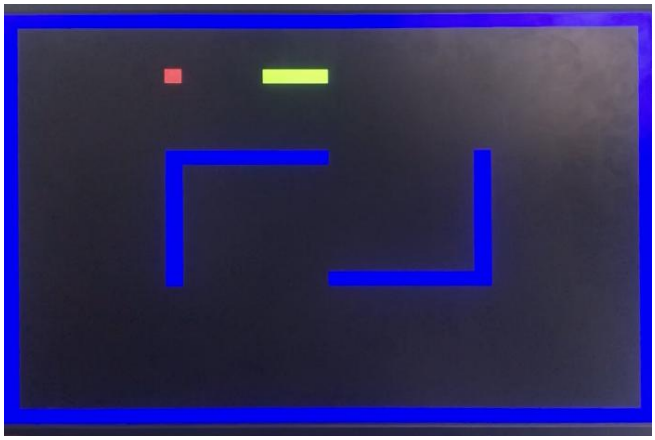

图 4 最终实现的游戏效果

实验中遇到的问题及解决办法：
1）随机数的生成：贪吃蛇小游戏中，必须在 30x40=1200 的界面内随机放置食物，且放置位置不能与已有的墙壁、蛇的身体重合，实验 6 中用移位寄存器实现随机数发生器。然而本实验中的随机数 n=11，有 1200 种状态，LFSR 反馈方程将十分复杂，对本来就使用大量资源、编译非常缓慢的工程项目雪上加霜，且生成的伪随机数序列仍然有一定的规律。
解决方法：采用 0~1199 递增的循环计数器，用系统时钟驱动，每次需要“随机数”时从循环计数器中读取当前计数，由于系统时钟的周期非常短，相比蛇的运动速度、人的反应时间几乎可以忽略不计，因此这种随机数生成方法类似高级编程语言中用系统时间作为“种子”，是可以保证一定的随机性的。如果食物生成位置恰好非空，则继续递增计数，在很短的时间里必定能找到一个空白位置能够生成的，对于玩家来说，这部分时间很难感知到。

2）多维数组的使用：因为对 Verilog 多维数组的定义和读取方式不确定，网上搜索也没找到令人信服的介绍，导致地图数组使用不当，游戏逻辑不能正常运行，屏幕显示乱码。
解决方法：将 30x40 的二维数组改成长度为 1200 的一维数组。

3）下标的计算：将 index = i * 40 + j 用于全部初始化和改动数组内容的操作，造成占用开发板大量资源、编译时间长达近 20 分钟，漫长的计算过程使时序配合也出现了问题。
解决方法：尽量减少二维转一维的下标计算方式，直接在一维模式下进行加减，如蛇爬行时若为左右移动，直接对下标加减 1，若为上下移动则直接加减 40，避免过多的下标转换。

4）通关图片的显示：通关和失败图片有两张，若仍用实验 9 中 640×512 的 12bit 图片，必然会造成 RAM 内存不足无法编译的现象，需要考虑如何节约内存的问题。
解决方法：考虑到通关图片实际上只有背景颜色和艺术字，屏幕上每个像素点只需要一个比特来判断是背景还是字，背景和字各自设成一种单色即可，用查找替换修改 mif 文件。

5）游戏的可玩性：起初设置游戏时钟周期为 1s，后来发现太慢了，不好玩。
解决方法：经过多次调整，请不同的小朋友来玩游戏并提出建议，确定了最后的速度。

实验得到的启示：
1）不会用的东西不要随便乱用（指在 Verilog 里面开多维数组）。
2）分模块调试，确认各个模块都功能正常以后再综合，有利于排查和寻找错误，如一段时间调试时发现 game_state 保持不变，但由于队友编写的键盘模块已经确认无误，不可能是从键盘获取玩家指令的问题，最后很容易找出是游戏时钟的调用和配合不妥。
3）优化算法、减少资源使用不仅影响到编译和生成可执行文件的速度，如果计算过于复杂且在运行过程中耗时过长，还有可能出现时序配合的问题，因此优化和精简非常重要。
4）在合作实验中，沟通和交流非常重要，队友误解了我的意思，没有及时求证，导致浪费许多时间编写功能错误的代码，与此同时我对“这么简单的功能写这么久”感到奇怪。如果在团队合作中不能确保理解了对方的需求，或者对方理解了自己的需求，一定要及时询问。

意见和建议：无。