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](mailto: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

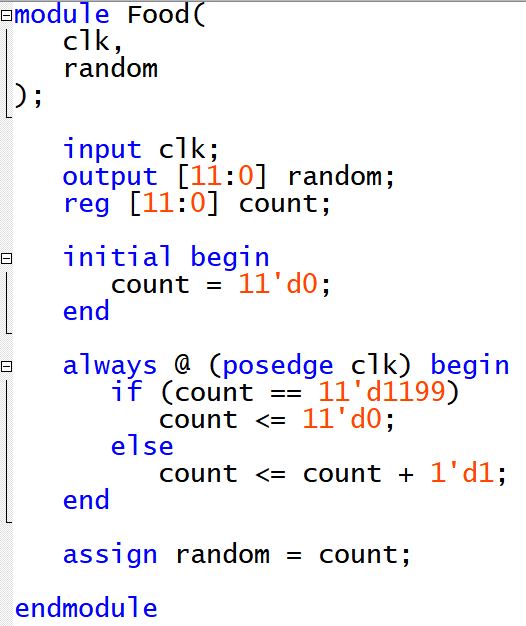
clk\_1s //生成游戏控制时钟（经调整其实最后周期并不是1s

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：

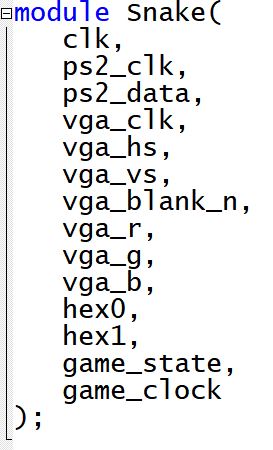
Food //随机生成食物（0~1199内的伪随机数



//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



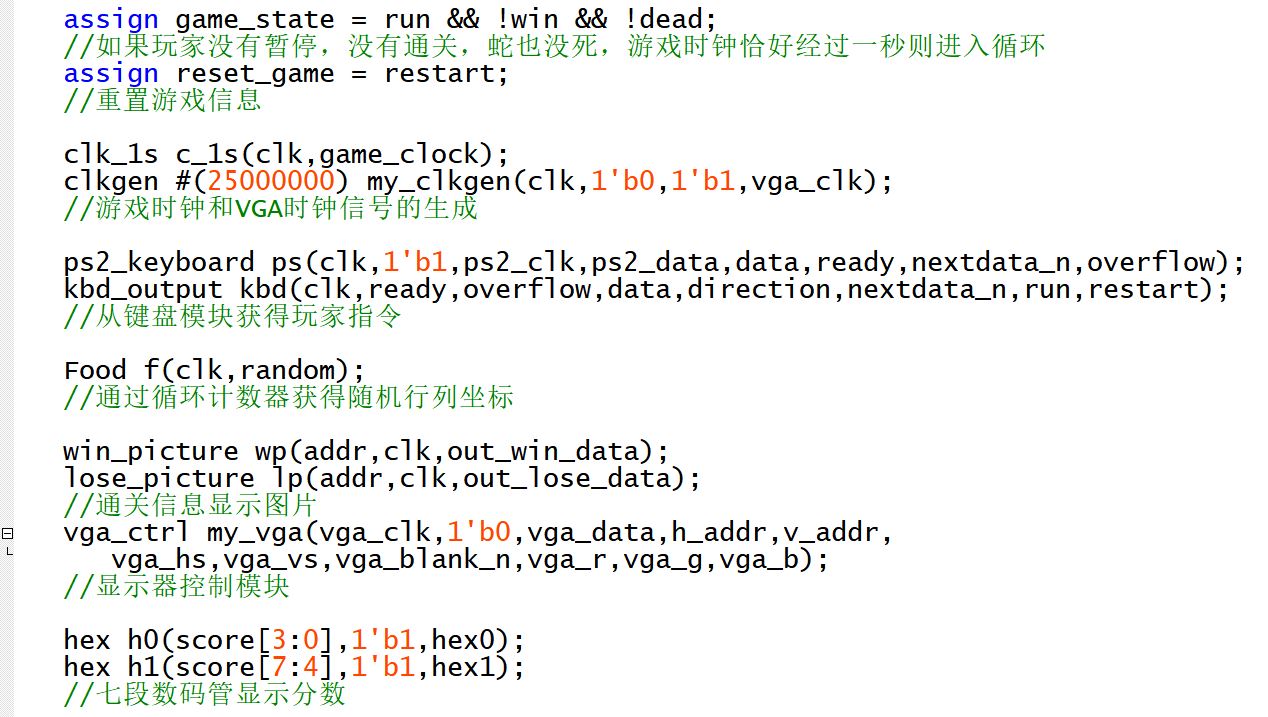
//模块提供系统时钟、键盘、显示器和七段数码管的接口

//game\_state和game\_clock接入发光二极管用于调试

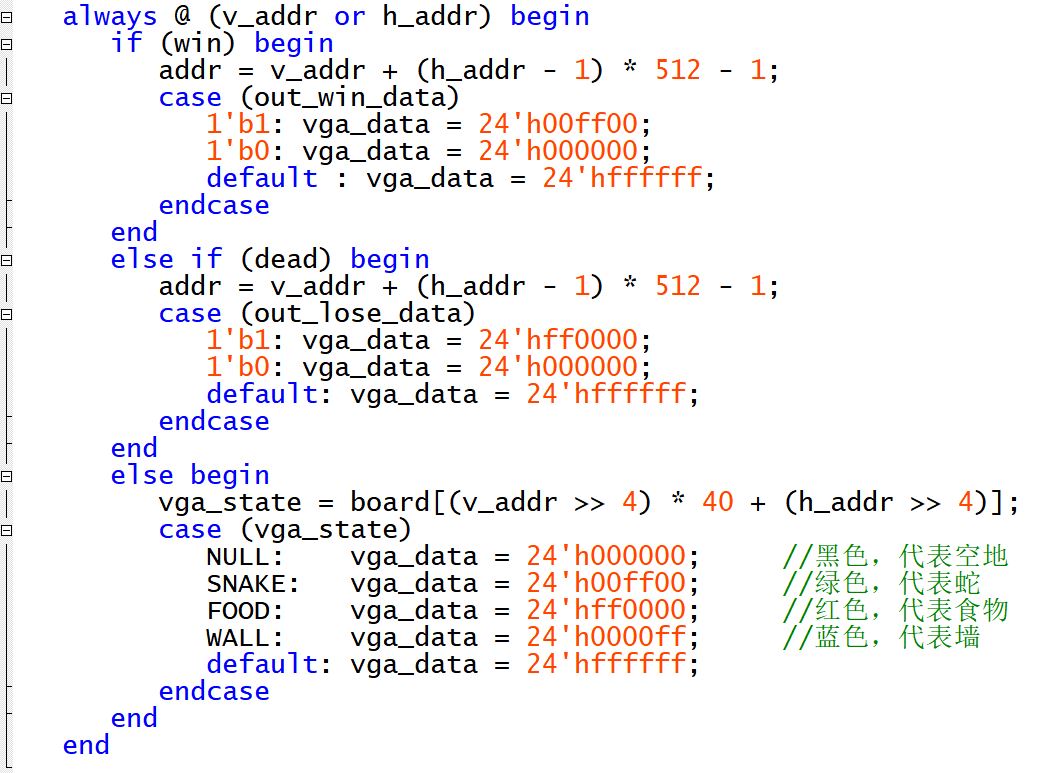




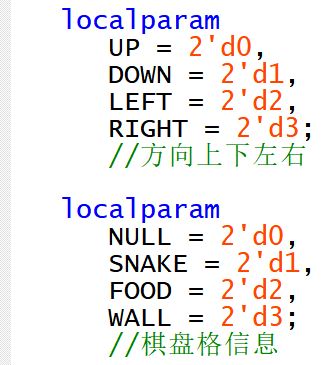
//模块定义的对外接口和内部使用的临时变量



//在主模块中调用的其他控制模块



//若通关或死亡则显示相应图片，否则显示当前地图信息



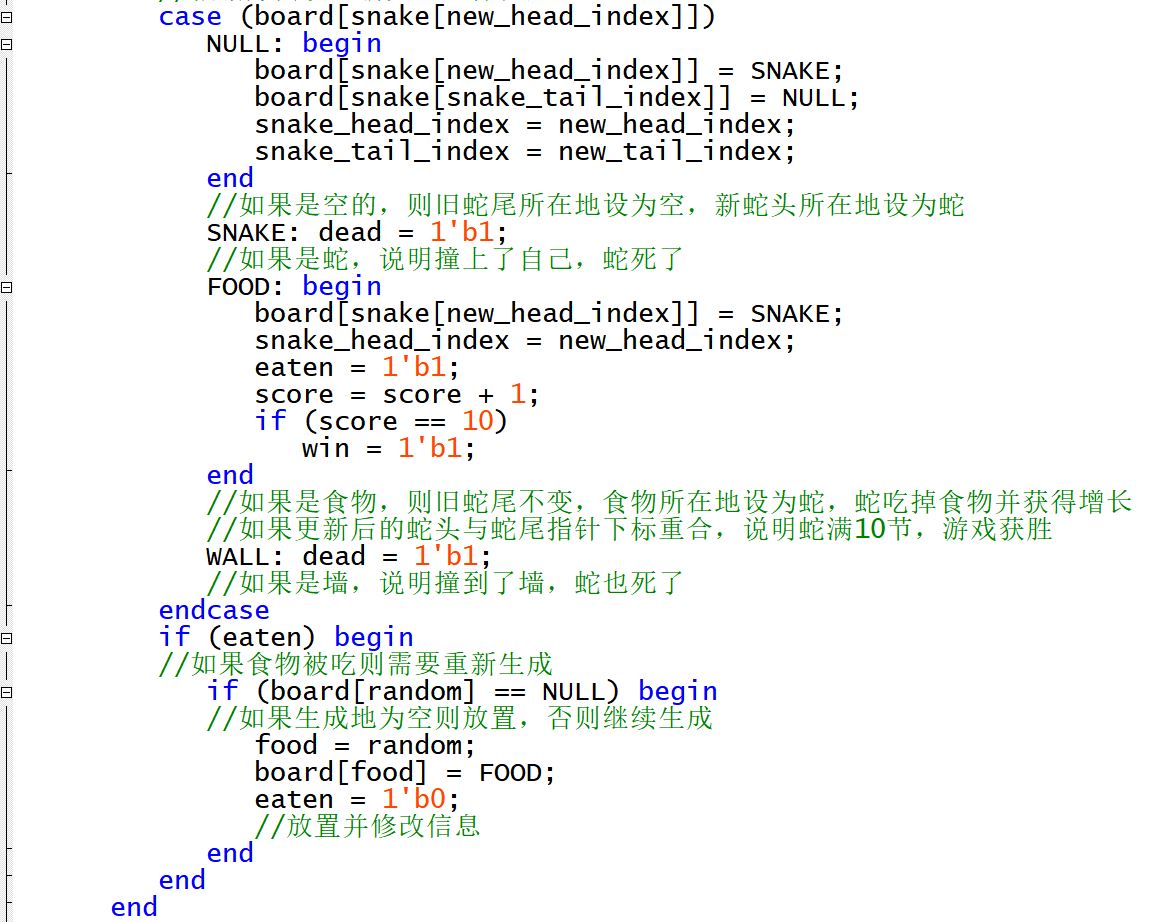
//定义一些有名称的变量，类似C语言中的#define

//当且仅当game\_state有效时更新游戏状态信息



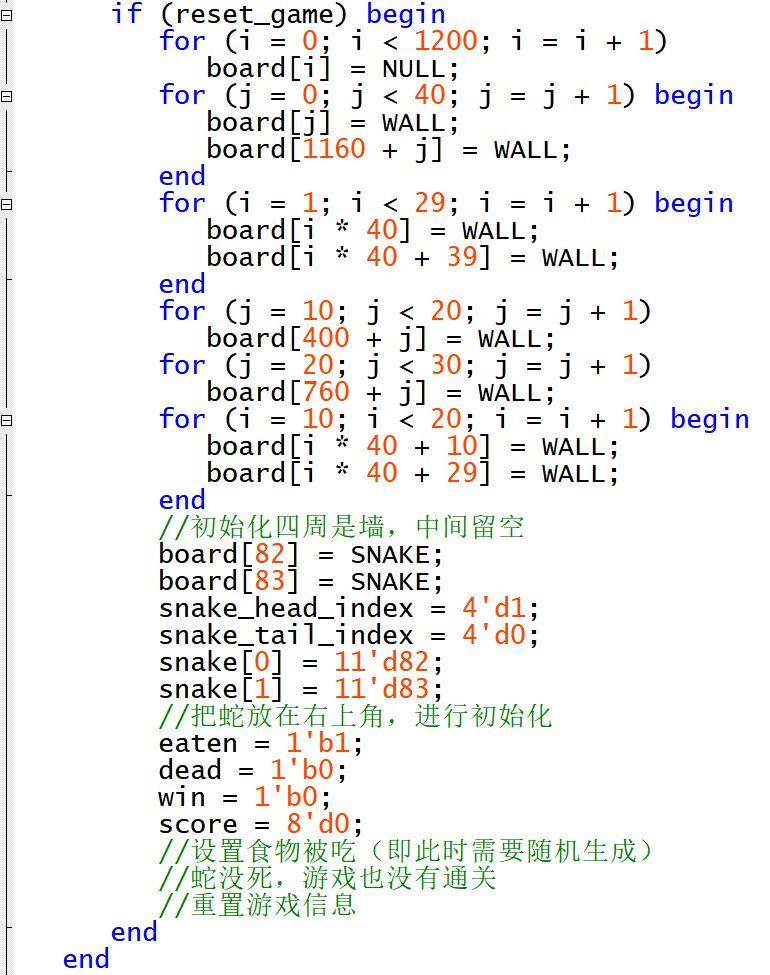
//更新头尾下标指针（蛇的坐标是长度为16的循环数组

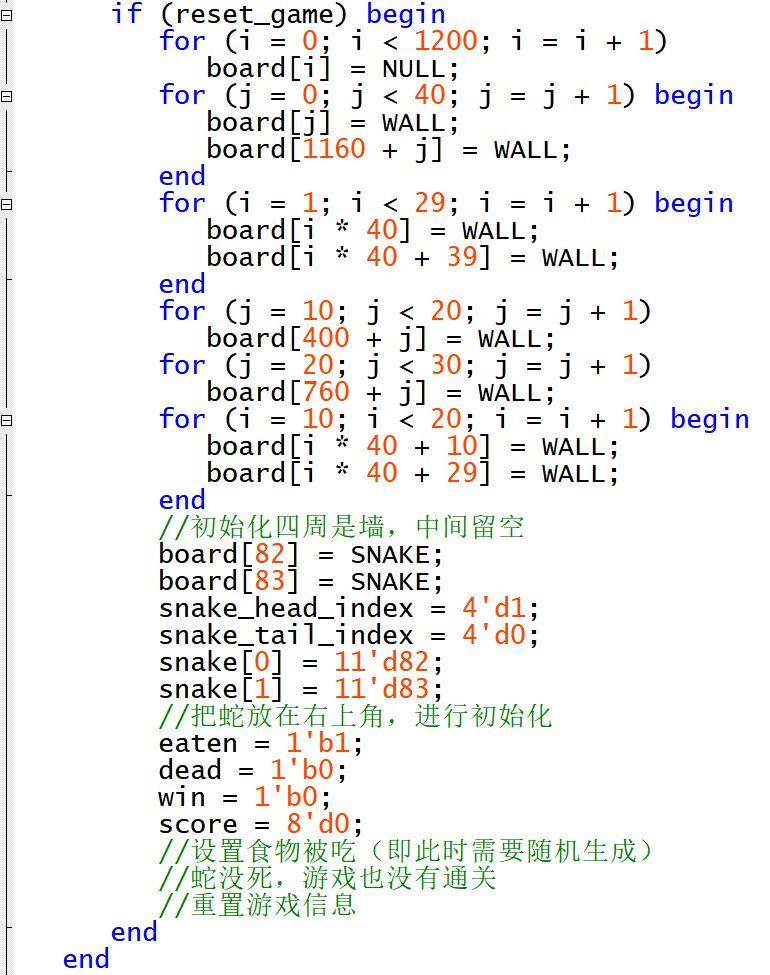
//根据当前的蛇头位置和运动方向设置蛇头新坐标



//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





//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。

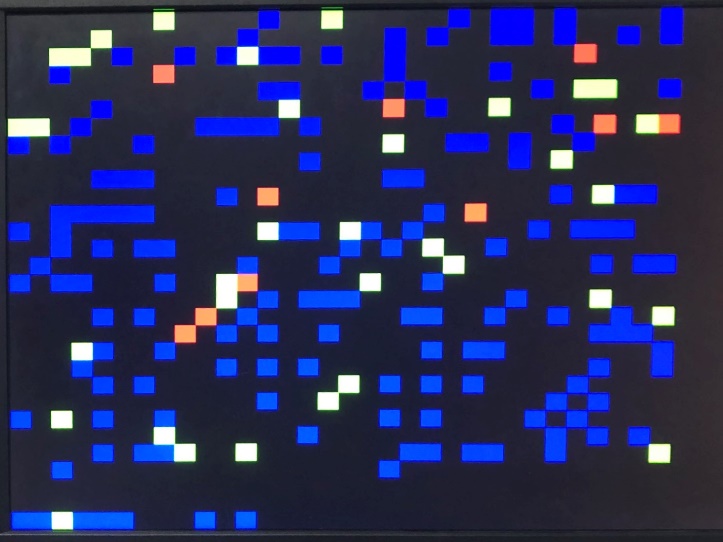


图1 现代抽象装置艺术 (Zhang, 2020)

//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：

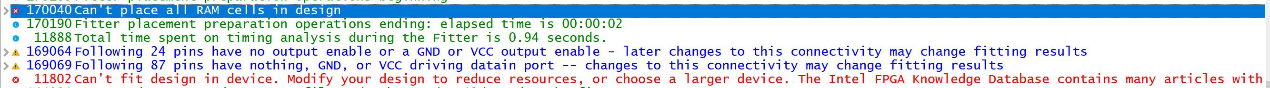


图2 令人精神崩溃的报错

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。

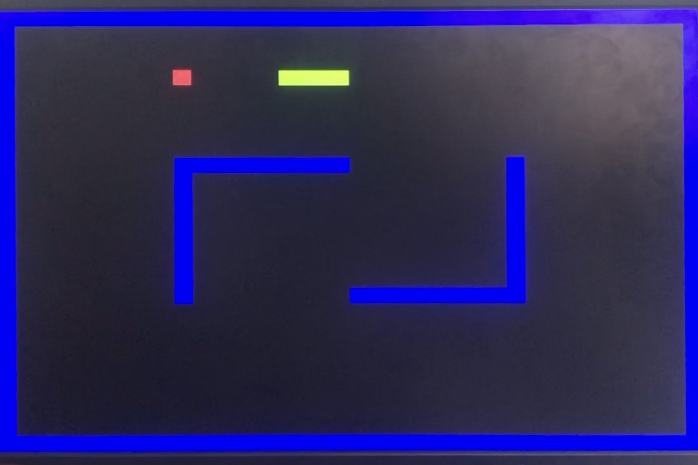


图4 最终实现的游戏效果

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。