技术文件

完成时间：2015 年 12 月 25 日

智能系统设计
设计报告

项目名称： 智能黑白棋游戏设计

设计小组编号：28

设计小组名单：刘聪（组长）

黄闻光

蒋芊珉

王尧

姓名	班级	学号	具体负责的工作	联系方式
刘聪	F1203001	5117129048	逻辑部分主程序的编写，以及人人对战模式逻辑的编写	18817559630
黄闻光	F1203013	5120309389	整体项目设计与监管，硬件部分的 HDL 编写，以及 C 部分的图形接口编写设计	18818272319
蒋芊珉	F1203007	5120309224	AI 算法的研究，以及相应模式程序编写	18817874081
王尧	F1203013	5120309375	项目测试除错，以及算法调优	18604820180

上海交通大学 电子信息与电气工程学院

地　址：东川路 800 号

邮　编：200240

摘 要：

本文主要介绍了一个基于 Nexys-3 FPGA 实验板与外接 VGA 显示器所开发的黑白棋游戏。在本个项目中，我们主要使用了 Nexys-3 FPGA 内嵌的 Microblaze 软核 CPU，借助其实现了对游戏的控制。具体而言，我们使用 Verilog 语言编写了 FPGA 中对于 VGA 显示器的底层驱动控制，并采用 C 语言来控制绘制出相应的游戏界面，和人人对战、人机对战等多种模式的游戏逻辑。最后，我们组还列出了一个关于黑白棋的开发范例，供老师参考。

关键词：

人工智能、黑白棋、FPGA、Microblaze、博弈树

ABSTRACT:

The report introduces the development of Othello, a strategy board game for two players, implemented by an Nexys-3 FPGA board and a monitor with VGA connectivity. In the project, we programmed the FPGA board with Verilog to control image displayed in the monitor and programmed the Microblaze, a soft microprocessor core designed for Xilinx FPGAs, with C to achieve the logical computation. Also, we provide a demo adopted from our own project for possible use by future students in the course.

KEYWORDS:

Artificial Intelligence, Othello, FPGA, Microblaze, Game Tree

---

上海交通大学 电子信息与电气工程学院

地 址： 东川路 800 号

邮 编： 200240

目　　录

1. 概述 1

1.1 编写说明 1

1.2 开发环境 1

1.3 缩略语 1

2. 系统总述 2

2.1 设计要求 2

2.2 功能介绍 2

3. 系统的硬件体系 4

3.1 硬件介绍 4

3.2 VGA IP CORE 5

3.2.1 功能描述 5

3.2.2 实现方法 5

4. 系统的算法设计 9

4.1 主程序设计 9

4.2 基本双人对战实现 9

4.3 AI 对战算法实现 11

4.3.1 AI-1 算法实现（静态估值函数） 11

4.3.2 AI-2 算法实现 12

4.3.3 AI-3 算法实现（博弈树- 极小化极大算法） 12

5. 测试结果 14

6. 范例与建议 16

6.1 范例提供 16

6.2 课程改进建议 16

7. 致谢 18

8. 参考文献 19

9. 附录 A：检测结果 20

10. 附录 B：软件程序清单 21

1. 概述

1.1 编写说明

本文介绍了基于 Xilinx Nexys3 FPGA 板卡的智能黑白棋系统设计，是上海交通大学电子工程系 2012 级大四上学期的智能系统课程的设计报告。适合电子工程专业学生或对此方面感兴趣及有一定基础阅读。

主要内容为硬件体系介绍、硬件部分编程即 VGA IP 核驱动编程介绍、上层程序设计、游戏 AI 算法实现，测试与分析结果。特别地，我们组在 GitHub 上为后续课程学习者提供工程框架，方便学弟学妹们更好地学习和用 XPS 和 SDK 等工具。

1.2 开发环境

硬件设备	Xilinx FPGA/SOPC 多媒体实验板卡- Nexys3 USB 键盘 VGA 显示器
软件环境	Microsoft Windows 7 / Windows 8, Xilinx ISE Design Suite 14.2

1.3 缩略语

XPS：Xilinx Platform Studio

SDK：Xilinx Software Development Kit

VGA：Video Graphics Array

2. 系统总述

2.1 设计要求

实现 4 种可切换的功能模式：人人对弈、3 种难度的人机对弈。

在显示界面方面：能稳定清晰显示棋盘和棋子、可以提示可以落子的地方以及轮到哪一方落子、有功能齐全操作方便的菜单、能完整地下完一盘棋，可以显示输赢、可以悔棋、可以重新对弈。

2.2 功能介绍

程序共有 5 中状态，其中 0 状态为初始状态即游戏模式选择状态，1-4 位 4 种不同的游戏模式 C1,C2,C3 代表三种难度的人机对战，P 代表人人对战模式，详细如表 2-1 所示：

表 2-1 游戏模式表

状态	*游戏模式*	*右下角显示图案*
0	新开/模式选择	0
1	人机对战（简单）	C1
2	人机对战（中等）	C2
3	人机对战（困难）	C3
4	人人对战	4

对应的按键如表 2-2 所示：

（具体的游戏模式跳转逻辑可参阅 4.1 章节的主程序设计框图，在此不加以赘述）

表 2-2 按键说明表

键盘按键	**作用**
1, 2, 3, 4	开始界面下模式选择
W, A, S, D	控制光标移动
ENTER	确认下棋
R	悔棋
ESC	复位，回到模式选择界面

黑白棋棋子每颗由黑白两色组成，一面白，一面黑，共 64 个（包括棋盘中央的 4 个）。棋子呈圆饼形。两个玩家各下一面棋子。黑白棋棋盘由 64（8*8）格的正方格组成，游戏进行时棋子要

下在格内。开局时在棋盘正中有摆好的四枚棋子，黑白各 2 枚，交叉放置（如图 2-1 所示），由执黑棋的一方先落子，双方交替下子，棋子落在方格内，一局游戏结束后双方更换执子颜色。

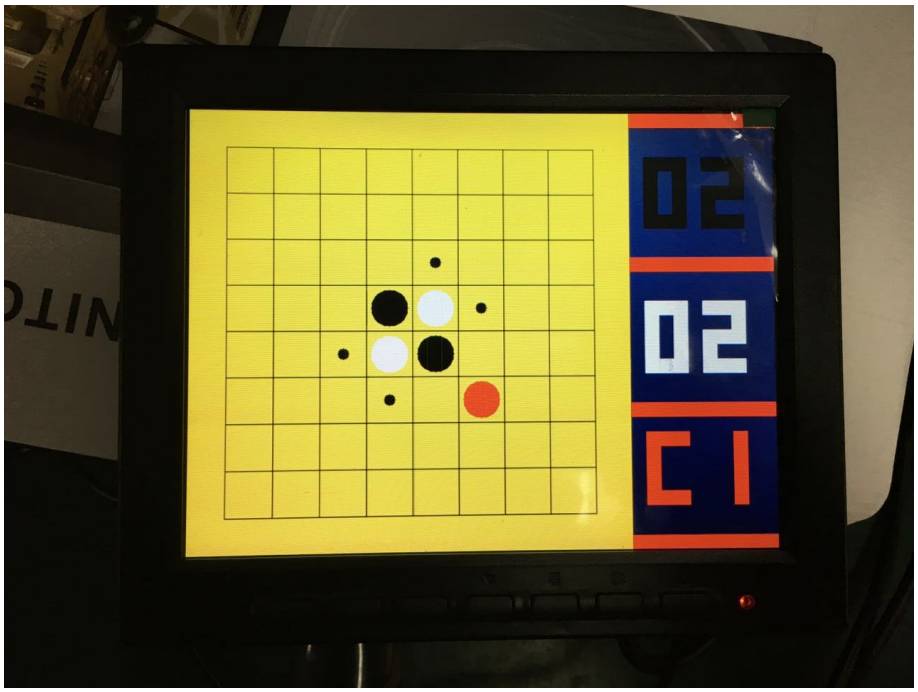


图 2-1 棋局开始时的显示界面（C1 模式下）

把自己颜色的棋子放在棋盘的空格上，而当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子，则被夹在中间的对方棋子全部翻转会成为自己的棋子。夹住的位置上必须全部是对手的棋子，不能有空格。并且，只有在可以翻转棋子的地方才可以下子。棋盘上会用黑白两色的小圆点显示出可以落子的位置，当小圆点为白色时表示轮到持白子一方落子，当小圆点为黑色时表示轮到持黑子一方落子。一步棋可以在数个方向上翻棋，任何被夹住的棋子都必须被翻转过来，必须是刚下的子夹对方才能够翻对方的子，因翻转对方的棋子而夹住的子是不能被翻的。每次下子最少必须翻转对方一个棋子，若棋局中有一方没有可落子处，自动轮到另一方下，若二个玩家都不能下子翻转对方棋子，游戏结束。

总结，游戏结束条件即为：

（1）、双方都无子可下游戏结束；

（2）、一方的子被翻完游戏结束；

（3）、棋格被全部下满；

（4）、强退。

游戏结束后（1-3 方式结束），按棋盘上谁的棋子多，谁胜利，若棋数一样，则为和局。

3. 系统的硬件体系

3.1 硬件介绍

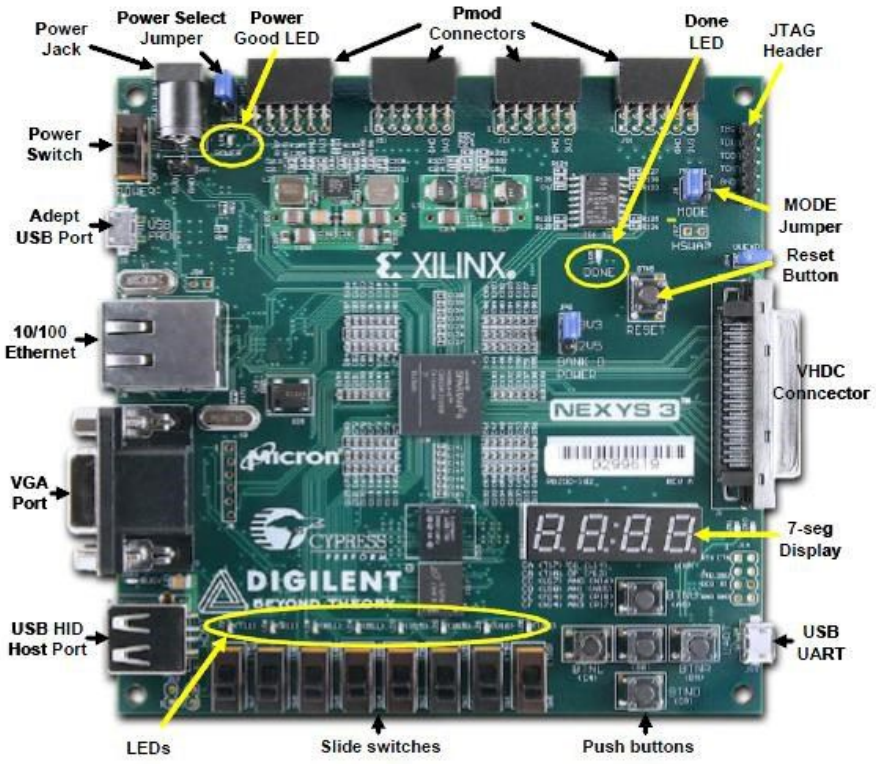


图 3-1 Nexys-3 开发板

如图 3-1 所示，Nexys 开发板是基于最新技术 Spartan-6 FPGA 的数字系统开发平台。它拥有 48M 字节的外部存储器（包括 2 个非易失性的相变存储器），以及丰富的 I/O 器件和接口，可以适用于各式各样的数字系统。 板上自带 AdeptTM 高速 USB2 接口可以为开发板提供电源，也可以烧录程序到 FPGA，用户数据的传输速率可以达到 38M 字节/秒。[1]

相关配置如下：

(1), Xilinx Spartan6 XC6LX16-CS324

(2), 16M 字节 Micron 公司的 Cellular RAM

(3), Digilent Adept USB 接口提供电源、程序烧录和数据传输

(4), USB-UART

(5), A 型 USB 接口，可以接鼠标、键盘和记忆棒

(6), 8 位 VGA

(7), 100MHz 晶振

(8), 8 个拨码开关, 4 个按钮, 4 个 7 段数码管, 8 个 LED

3.2 VGA IP CORE

3.2.1 功能描述

在 Xilinx 的 EDK 下使用 VGA 端口输出，最简单的方法是使用 Xilinx 官方提供的 TFT 的 IPCore，该 IPcore 包含了现成的硬件模块和 API，使用简单方便。但 TFT 需要 2MB 的 RAM 作为缓冲区，本次实验所使用的 Nexys3 开发板的片内 RAM 容量不够，片外 Celluar RAM 速度又不能满足要求，因此这里无法使用 TFT 的 IPcore。这里我们使用了老师给出的 VGA IP CORE 参考资料[2]，实现了黑白棋输出的 VGA 输出功能。

我们最终实现的黑白棋界面如图 3-2 所示：

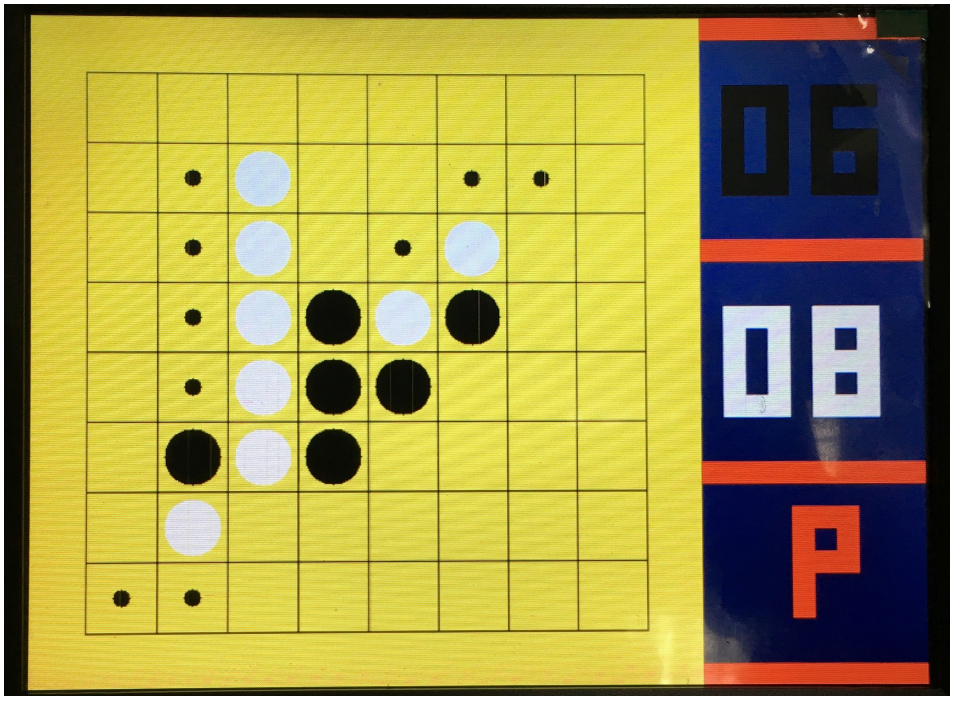


图 3-2 黑白棋显示界面

具体而言，我们达到的效果是：

（1）、能根据黑白棋的规则生成一个 8*8 的棋盘，能自定义棋盘颜色和棋子颜色。

（2）、能绘制出圆形的旗子，移动的光标和预测落点，并将其落在棋盘方框之中。

（3）、能在棋盘右侧绘制出数字，并根据局势实时更新双方得分。

（4）、能在右侧界面绘制出当前游戏模式。

3.2.2 实现方法

因为我们整个 Verilog 工程皆是参考老师所提供的 VGA 显示范例改编而来，所以端口定义，端口映射以及管脚绑定等和老师所提供的 VGA 显示范例基本一致，为防赘余，此处就不再叙述相关

生成流程和定义内容，如有疑问者可自行查找原范例的 vga_ip.vhd， vga_ip 的 MPD 文件，以及 vga_ip 的 UCF 文件进行查询。[2]

此处我们主要阐述 vga_ip 源文件中的 user_logic.v 文件的程序编写，即如何编写适用于黑白棋棋游戏的 VGA 输出。VGA 输出原理较简单，只需理清同步信号 VGA_HSYNC 和 VGA_VSYNC 的时序关系即可，不再赘述。我们首要解决的，是如何使用较少的资源和较简单的软硬件接口实现棋盘输出的问题。

因为 Microblaze 的数据总线宽度为 32 位，为了充分合理的利用总线宽度，定义[0 : 7]位表示要修改的色块的横坐标，[8 : 15]位表示要修改的色块的纵坐标，两者的实际取值范围均为 0~15。[24 : 31]位为该单位的类型。

首先，我们为了实时传递黑白棋那 8*8 棋盘的信息，我们通过下式定义了一个数组：

reg [0 : 1] board[0 : 63];

同时，为了实现屏幕右侧的时间显示和菜单界面，我们又定义了一个数组：

reg [0 : 1] menu[0 : 299];

在 C 程序中，我们使用如下语句对 VGA 数据总线进行传输：

vga_input = (x_pos << 24) + (y_pos << 16) + (1 << 8) + turn;

VGA_IP_mWriteReg(XPAR_VGA_IP_0_BASEADDR, 0, vga_input);

当 verilog 程序端接收到总线传来的信息之后，便会调用如图 3-3 所示的程序端进行处理，将相应的值传递给 board[] 或者 menu[] 数组。

```
always @( posedge Bus2IP_Clk )
begin
  if ( Bus2IP_Reset == 1 )
    begin
      for (i = 0; i < 64; i=i+1)
        board[i] <= 3'b010;
      for (i = 0; i < 300; i=i+1)
        menu[i] <= 2'b10;
    end
  else
    begin
      if ( Bus2IP_WrCE == 1'b1)
        begin
          if (Bus2IP_Data[16 : 23]== 0)
            board[Bus2IP_Data[0 : 7] + Bus2IP_Data[8 : 15] * 8] <= Bus2IP_Data[24 : 31];
          else if (Bus2IP_Data[16 : 23]== 1)
            menu[Bus2IP_Data[0 : 7] + Bus2IP_Data[8 : 15] * 10] <= Bus2IP_Data[24 : 31];
        end
    end
end
```

图 3-3 总线信息传递部分程序

接下来，我们开始对左侧的棋盘部分进行处理。

```
// ----------------------棋盘绘制和旗子绘制部分：开始--------------------------------------------------------------------
always@(*)
begin
  if (vga_ena == 1 && x_cor < 480 && y_cor < 480)
  // 开始绘制屏幕左边480*480的board部分=============================================================
    begin
      if ((x_cor >= 40 && x_cor <= 440 && y_cor >= 40 && y_cor <= 440))
        begin
        if (x_cor == 40 || x_cor == 90 || x_cor == 140 || x_cor == 190 || x_cor == 240 || x_cor == 290 || x_cor == 340 || x_cor == 390 || x_cor == 440
         || y_cor == 40 || y_cor == 90 || y_cor == 140 || y_cor == 190 || y_cor == 240 || y_cor == 290 || y_cor == 340 || y_cor == 390 || y_cor == 440)
          vga_data <= linecolor;                    // 第一个if，来绘制出棋盘的线
        else
          begin
            x_pos <= (x_cor - 40) / 50;               // 旗子在棋盘上的x轴位置
            y_pos <= (y_cor - 40) / 50;               // 旗子在棋盘上的y轴位置
            board_pos <= x_pos + y_pos * 8;           // 旗子位置在board数组上的位置
            x_center <= x_pos * 50 + 40 + 25;    // 绘制特定旗子的圆心x坐标
            y_center <= y_pos * 50 + 40 + 25;    // 绘制特定旗子的圆心y坐标
            if (board[board_pos] == 3'b100 ) // 第三个if，来判断现在是否是绘制白色小旗子（位置判断）
              begin
                if ((x_cor - x_center) * (x_cor - x_center) + (y_cor - y_center) * (y_cor - y_center) <= 36) // 给白色小旗子绘制圆
                  vga_data <= chesscolor1;
                else
                  vga_data <= boardcolor;
              end
            else if (board[board_pos] == 3'b101 ) // 第三个if，来判断现在是否是绘制黑色小旗子（位置判断）
              begin
                if ((x_cor - x_center) * (x_cor - x_center) + (y_cor - y_center) * (y_cor - y_center) <= 36) // 给黑色小旗子绘制圆
                  vga_data <= chesscolor2;
                else
                  vga_data <= boardcolor;
              end
            else if ((x_cor - x_center) * (x_cor - x_center) + (y_cor - y_center) * (y_cor - y_center) <= 400) // 给大旗子绘制圆
              vga_data <= colormap[board[board_pos]];
            else
              vga_data <= boardcolor;
          end
        end
      else
        vga_data <= boardcolor;
    end
    // board部分的绘制结束=============================================================================
```

图 3-4 棋盘部分 verilog 程序截图

正如图 3-4 代码中所示，x_cor 和 y_cor 为扫描像素的横纵坐标，vga_ena 为 VGA 输出的使能端，vga_data 为输出的八位 VGA 数据，寄存器组 colormap 负责将棋盘状态 board[]转换为 VGA 数据。通过控制 x_cor 以及 y_cor 的取值，我们可以画出最基本的棋盘的线。

之后，通过一定算法，我们可以算出特定 x_cor 和 y_cor 取值之时所对应的 x_center 以及 y_center，并通过 (x_cor - x_center) * (x_cor - x_center) + (y_cor - y_center) * (y_cor - y_center) <= r * r 的语句，来绘制出半径为 r 像素大小的圆。

最后，我们对右侧的彩蛋部分进行处理。

```
    // board部分的绘制结束=============================================================================
  else if (vga_ena == 1 && x_cor >= 480 && x_cor <= 640 && y_cor < 480)
  // 开始绘制右边160*480的menu部分=============================================================================
    begin
      menu_pos <= (x_cor - 480) / 16 + y_cor / 16 * 10;
      if (menu_pos < 300)
        begin
          if (menu[menu_pos] == 2'b10)
            vga_data <= menucolor;
          else
            vga_data <= colormap[menu[menu_pos]];
        end
      else
        vga_data <= 0;
    end
    // menu部分的绘制结束=============================================================================
  else
    vga_data <= 0;
end

// ----------------------棋盘绘制和旗子绘制部分：结束--------------------------------------------------------------------
```

图 3-5 菜单部分 verilog 程序截图

代码如图 3-5 所示，和棋盘处同理，寄存器组 colormap 负责将菜单状态 menu[]转换为 VGA 数

据。只不过在菜单部分的绘制中，我们组借鉴老师提供的 VGA 接口程序中的采用了降低分辨率的方法，将棋盘划分为 30*10 个颜色块，每个色块的分辨率为 16*16。这样我们在 C 代码中就不需要给每一个像素点赋值，只需要给每一个色块进行值传递就好了，减轻了工作量。

4. 系统的算法设计

4.1 主程序设计

算法流程图 4-1 所示：

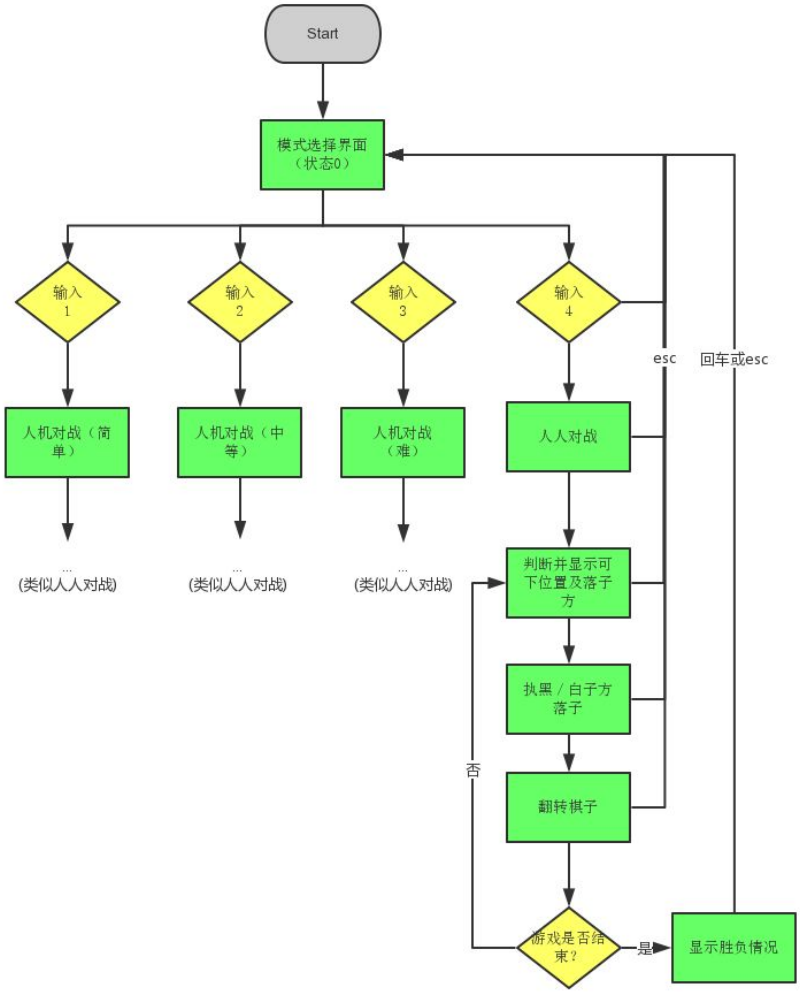


图 4-1 基本主程序的算法流程图

4.2 基本双人对战实现

主要变量介绍：

Black_num 记录棋盘上黑色棋子的个数；

White_num 记录棋盘上白色棋子的个数；

board_state[8][8] 记录棋盘上每个位置的状态，0 为有白色棋子，1 为有黑色棋子，2 为空闲，3 表示此处为光标所在处，4 表示此处可以落白子，5 表示此处可以落黑子；

win_status 记录输赢的状态，-1 表示没有玩家赢，游戏还未结束，0 表示持白色棋子方赢，1 表示持黑色棋子方赢，2 表示双方平手。

首先选择模式 4 进入棋局后，画出棋盘初始状态，即中间有四颗棋子并提示轮到黑方落子（如图 1），此时可以移动通过 A（左）W（上）S（下）D（右）四个键移动光标（光标所在位置显示为一个红色的棋子），当按下回车并且光标所在位置为可以落子处视为一次有效的落子，此时记录(x_cur, y_cur) 为新落子的坐标，用 DrawChess(x_cur, y_cur, turn) 画上新下的棋子并且用 EraseLittleChess()擦去上一轮可落子提示，累加黑／白子个数，通过 flip(x_cur, y_cur, turn, 1)函数翻转因此次落子引起应该翻转的棋子，通过 check_win()函数判断输赢状态，如果游戏结束，画出胜利的界面（如果白方胜为一个白色棋子在棋盘上组成的 W 图案，如果黑方胜为一个黑色棋子在棋盘上组成的 W 图案，如果为平手，为一个黑白棋子交错在棋盘上组成的 W 图案；如果游戏没有结束，轮到另一方落子（此处判断若该方无处可以落子，则 pass，轮到对方连下），通过函数 FindAvailable(turn, 1)找出并显示该方可以下子的地方用相应颜色的小圆点提示，继续等待光标移动和有效落子，重复以上过程，直到游戏结束。

接下来，悔棋部分利用两级堆栈实现。

对于第一级堆栈，我们定义了一个大小为 60 的整数指针数组为 p1 作为堆栈空间，又定义了一个整数 depth 用于记录堆栈顶部的位置。堆栈中的每一个指针指向一个保存了某一次落子时棋盘变化信息的堆栈。落子早的先入堆栈，落子晚的先出堆栈。

对于第二级堆栈，我们在每一次落子的时候，新建一个可以容纳大小为 50 个 int 的整数动态数组作为堆栈空间，用来记录落子时棋盘变化信息。数组的第一位用来记录堆栈顶部的位置。记录时，依次将这次落子之后翻转棋子的位置以一个整数的形式压入到堆栈中。之后将这次落子的位置以相同形式保存在堆栈中。最后，将这个数组压入 p1 中。

我们使用的在 C 语言中动态定义数组的方式是：

int *a=(int *)malloc(n*sizeof(int));

其中，a 为数组名称，n 为数组的大小，int 代表了数组元素的类型为整数。

每一步悔棋的时候。从第一级堆栈中弹出一个第二级堆栈。依次从第二级堆栈中弹出记录的落子位置和翻转位置，根据悔棋者执黑还是执白，改变棋盘上该位置的状态。

调用悔棋函数的方法是

void undo(int turn);

其中 turn = 1 代表悔棋者执黑，turn = 0 代表悔棋者执白。在人人对战模式下，每接到一次悔棋的命令，调用依次 undo 函数，turn 的值由当前的执黑执白情况决定。

在人机对战模式下，每接到一次悔棋的命令，调用两次 undo 函数，第一次调用时 turn 的值为机器的执黑执白情况决定，第二次调用时 turn 的值由人的执黑执白情况决定。

4.3 AI 对战算法实现

4.3.1 AI-1 算法实现（静态估值函数）

根据黑白棋的规则，黑白棋棋盘上的位置，有优劣之分。通常情况下，棋盘四个角的位置最好，四条边上的位置其次，角的周围的三个位置最差，其余的位置一般。这是因为根据黑白棋的规则，只有棋子周围八个方向当中，相对的两个方向上都有对方的棋子才会被翻转，在棋盘四角的格子，由于不存在相对的两个方向上都有对方棋子的可能，因此该处的棋子不会被翻转，是棋盘上最安全的位置。四条边上的位置，只有左右一对方向上都有对方的棋子才可能被翻转，也较为安全。而另一方面，根据规则，只有对手棋子的周围位置可以落子，当四角的位置没有被对方棋子占领的情况下，如果在四角周围落子，会给对手留下占领四角的机会；如果四角已经被对方棋子占领，在该处落子很有可能会被翻转。因此，棋盘角落周围的三个位置最不安全。如图 4-2 所示，红色是最优位置，橙色是次优位置，黑色是最差位置，绿色是普通位置。

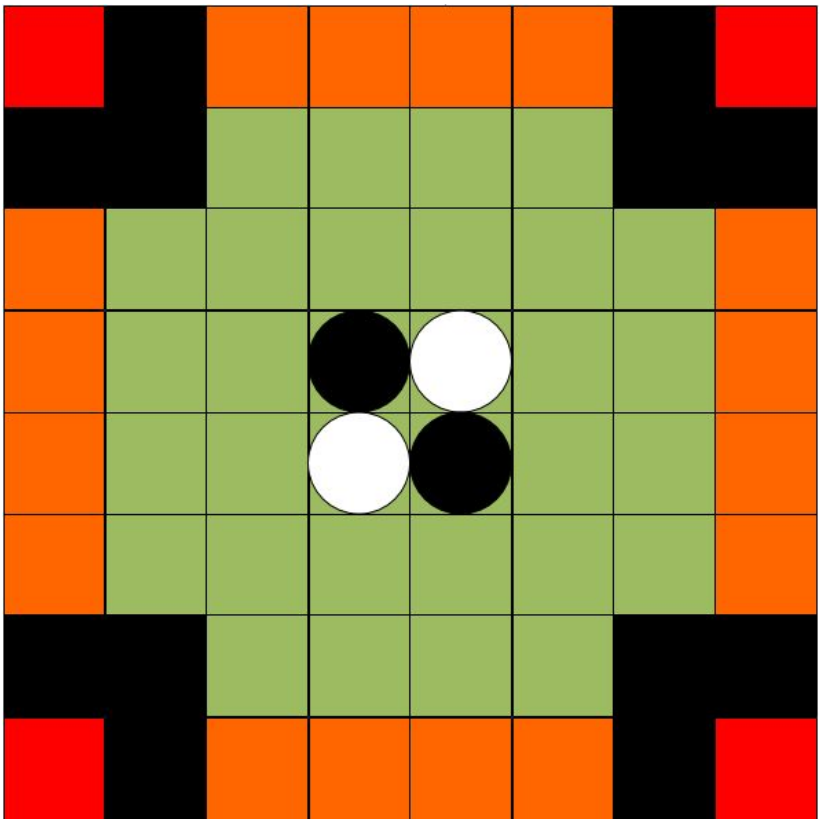


图 4-2 黑白棋棋盘位置的优劣程度

根据优劣程度，给棋盘上的每一个位置打分。在选择落子位子的时候，依次搜索可以落子的位置，选取打分最高的最为落子点。

这种算法是针对黑白棋的一种容易实现，效果比较好的一种算法。电子词典上的黑白棋游戏大多使用这种算法，可以战胜绝大多数的新手。

4.3.2 AI-2 算法实现

估值函数，又叫启发式估值函数或静态估值函数，是一种在博弈类游戏中用于分析某个位置的价值与优劣的函数。这种估值函数仅仅就目前局势作出估计，与可能的落子无关。

构建估值函数的最常见的方法是对影响一个位置的优劣的多种因素加权求和。

在这次的黑白棋游戏中，我们选取两种因素，一个是 AI-1 算法中用的打分，第二种是行动力。我们的评分函数可以表示为：

$$V(S) = \sum_{i=1}^{8}\sum_{j=1}^{8} \omega[i,\ j] \cdot s[i,\ j] + \text{Mobility}(\ my\_color)$$

其中，

$$s[i,\ j] = \begin{cases} 1 & color[i,\ j] = my\_color \\ -1 & color[i,\ j] = opp\_color \\ 0 & \text{otherwise} \end{cases}$$

Color[i,j]表示棋盘上第 i 行第 j 列处的颜色，my_color 是己方颜色，opp_color 是对方颜色。w[i,j]代表第 i 行，第 j 列这个位置的打分。Mobility 代表己方的行动力大小。我们的程序中用己方已落子处周围的空白位置数来衡量行动力，已落子位置周围的空白位置越多，行动力越强，位置越少，行动力越弱。[3]

在选择落子位子的时候，我们依次搜索可以落子的位置，选取估值最高的最为落子点。

4.3.3 AI-3 算法实现（博弈树- 极小化极大算法）

Minimax 算法常用于棋类等由两方较量的游戏和程序。该算法是一个零总和算法，即一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的方法。而开始的时候总和为 0。我们假设参与博弈的两个人执黑和执白，在执黑者的决策过程中，执黑者每一步行动总是争取自己的最大收益，称为 Max 过程，由于执黑者对执白者的棋力并不清楚，所以只能认为 执白者的每一步总是使自己的收益最大，也就是使执黑者的收益最小，称为 Min 过程于是下棋其实就是 Max 和 Min 交替进行的过程。

如图 4-3，我们假设某一时刻轮到执黑者下， Black 发现自己所有可能的行动派生的子节点中评估值最大的节点为，那么最大收益就为 V()−V(S)，对应的行动 ai 就是单步搜索所能找到的最优解。而在连续搜索两步之后，他也许会发现所派生的子节点中的最小值为 V () 小于某个其他节点所派生的子节点中的最小值 V()，不妨设为同层节点中所派生的子节点的最小值中最大的节点，那么他转而应该选择对应的行动 aj 作为自己的最佳行动，因为选择 aj 的收益最差不会少于 V ()−V(S)，而这已经是最好的情况了。如果搜索继续加深，那么选择会越来越“明智”。

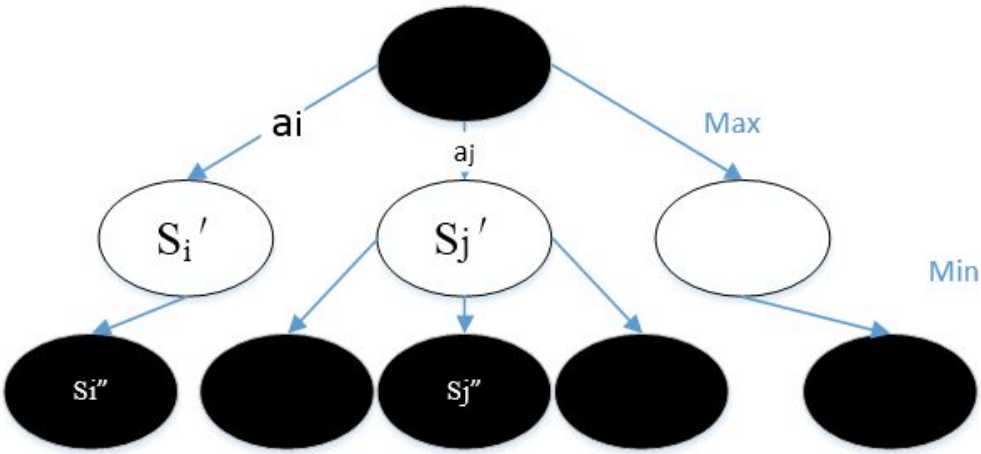


图 4-3 使用最小化最大算法的博弈树

对于一般的极小化极大搜索，即使每一步只有很少的下法，搜索位置的数目也会随着搜索深度呈指数级增长，算法的效率很不理想。若使用 AlphaBeta 算法，则没有必要每个节点都必须搜索完毕，优化了搜索效率。

AlphaBeta 算法根据根据以下三个原则执行：[4]

1. 对于一个 MIN 节点，若能估计出其倒推值的上确界 Beta，并且这个 Beta 值不大于 MIN 的父节点(MAX 节点)的估计倒推值的下确界 Alpha，即 Alpha≥Beta，则就不必再扩展该 MIN 节点的其余子节点了，因为这些节点的估值对 MIN 父节点的倒推值已无任何影响了，这一过程称为 Alpha 剪枝。

2. 对于一个 MAX 节点，若能估计出其倒推值的下确界 Alpha，并且这个 Alpha 值不小于 MAX 的父节点(MIN 节点)的估计倒推值的上确界 Beta，即 Alpha≥Beta，则就不必再扩展该 MAX 节点的其余子节点了，因为这些节点的估值对 MAX 父节点的倒推值已无任何影响了。这一过程称为 Beta 剪枝。

3. 一个 MAX 节点的 Alpha 值等于其后继节点当前最大的最终倒推值，一个 MIN 节点的 Beta 值等于其后继节点当前最小的最终倒推值

每次函数的调用，系统都会为该函数的变量开辟新的内存空间，递归调用的层次越多，同名变量的占用的存储单元也就越多。由于单片机的内存有限，在我们尽最大可能调整堆栈大小，设定栈的大小为 16KB，堆的大小为 4KB 之后，调整搜索深度为两层，勉强能够完成编译。因此，设定博弈树的搜索深度是两层。在这个搜索深度下，省略了不必要的 alpha-beta 剪枝。

5. 测试结果

游戏开始时，我们将会进入模式选择界面，如图 5-1 所示：

图 5-1 模式选择界面

在模式选择界面下，点击键盘下的 1,2,3 键将进入不同难度的 AI 人机对战模式，点击 4 键将进入人人对战模式，如图 5-2,5-3,5-4,5-5 所示：

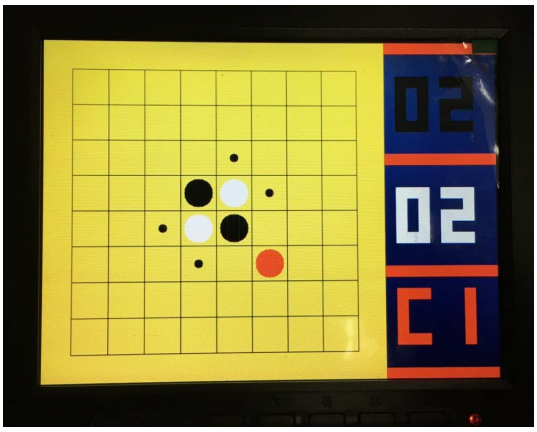


图 5-2 AI-1 人机对战模式

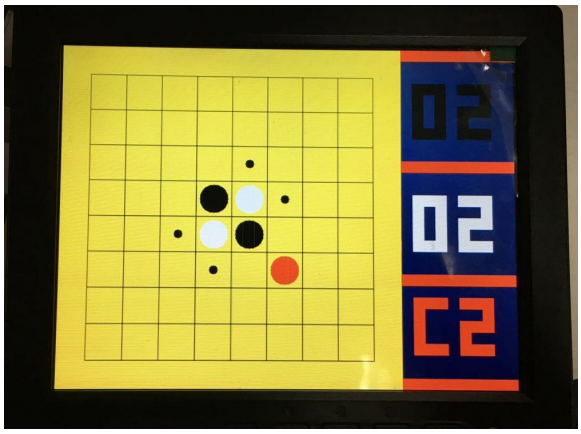


图 5-3 AI-1 人机对战模式

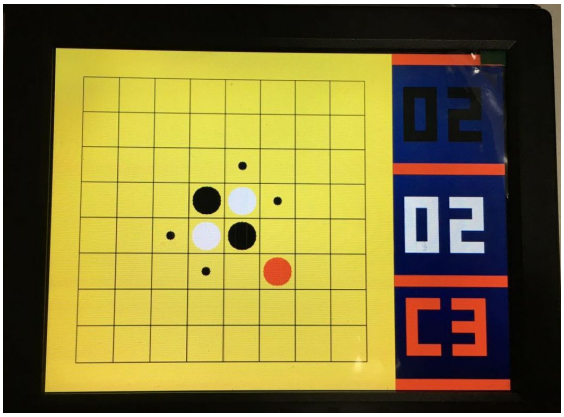


图 5-4AI-3 人机对战模式

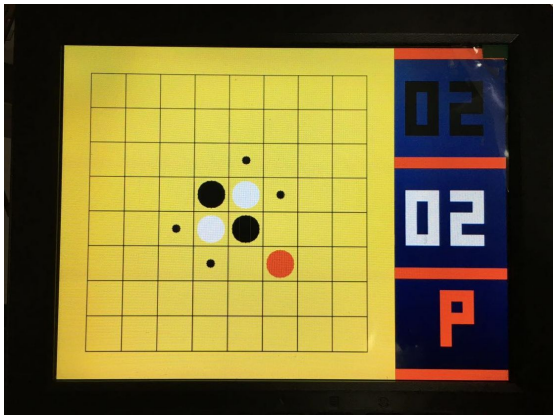


图 5-5 人人对战模式

无论在何种模式下，最终胜利后，将会进入胜利界面，如图 5-6 所示：

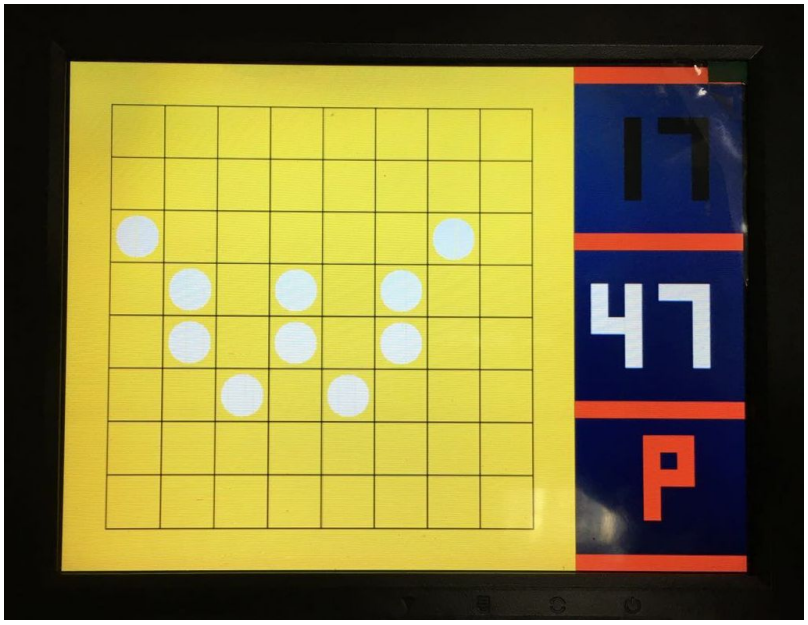


图 5-6 胜利界面

经过多次测试，本工程在各种模式下程序均能正常运行。

6. 范例与建议

6.1 范例提供

在我们这一届进行实验的问题中，发生了一个较为尴尬的问题，就是：以黑白棋为主题进行游戏设计的组实在是太少了，以至于在课程最后想要寻找能够和我们组进行机机对战的小组时，竟一时都找不到对手。

另外，我们组也确实认为，黑白棋（Othello）虽不如五子棋受众来的广，却也是一种在设计上颇具美感的棋类游戏。我们组真切的希望，这门棋类艺术不是仅仅能停留在一种小众的状态下，而是能推广开来，供大家游玩，欣赏。

借助于 GitHub 优秀的版本控制系统，我们保留下来了我们在开发过程中的几个版本，在此无偿提供给智能系统设计课程组，希望能将我们的成果传承下去，希望后人能在我们的肩膀上创造出更为优秀，更为令人眼前一亮的产品。

在此之前，先行说明，我们下面列出的两个 release 版本，均做到了：

（1）、C 代码和 HDL 源码中均列出了详细的注释，基本上做到了每一个函数都有注释解释用途，方便学弟学妹理解。（由于时间仓促，极少数注释可能有误，望谅解）

（2）、已经将 Nexys-3 所需的支持包放入项目之中，学弟学妹无需再自行从网上下载 Nexys-3 支持包，节约了开发时间。

以下为 release 版本：

版本 1: 0.1-release 版　https://github.com/SmartSys-28/Othello/archive/0.1-release.zip

版本描述：具有最基本的黑白棋双人对战逻辑，以及标准的 8*8 黑白棋棋盘，可显示黑白棋的预测落子，但是无右侧菜单，不可悔棋（无悔棋逻辑）。

版本 2: 0.2-release 版　https://github.com/SmartSys-28/Othello/archive/0.2-release.zip

版本描述：具有最基本的黑白棋双人对战逻辑，以及标准的 8*8 黑白棋棋盘，可显示黑白棋的预测落子，可在右侧的菜单中绘制出黑白双方旗子的实时得分，但不可悔棋（无悔棋逻辑）。

整体工程基于 Xilinx 产品套件的 14.2 版本，老师们可以下载下来进行尝试，并挑选合适的版本提供给下一届的学弟学妹。

6.2 课程改进建议

首先，是关于在 Windows 8 系统下使用 Xilinx 系列产品的问题。由于 Xilinx 系列产品一直对 Windows 8 系统的支持不太完善，像我这样使用 Windows 8 系统电脑的同学可能会遇到软件使用闪退的问题，从而不能工作。就算在 Google 搜索相关解决方案，也无甚收获。最后，在我自己手动尝试之下，算是用另一种“方法”（手动替换 DDL 文件）来解决了这个问题，具体过程我整理后放在了这个网址上，如果需要的话老师可以列出，供接下来选修这门课的同学作参考：

https://github.com/SmartSys-28/Othello/blob/master/tutorial.md

其次，是关于老师您之前在 FTP 上所给的 VGA 显示范例的问题。诚然，那是一个非常好的范例，然而却有一个小问题：这个范例里缺少了一些必要的包，导致学生们需要自行下载这些支持包。然而如果没有前人指引的话，学生们想在网上找到这个支持包其实是要花费一番精力的，在这里我们也列出了 Digilent 官方给出的 Nexys-3 支持包下载地址，供下一届的学弟学妹们使用：

http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_BSB_Support_v_2_8.zip

7. 致谢

终于，四年的大学本科生活，也来到了最终收尾的时刻。此次的实验报告，也是在本科毕业设计之前，我们所书写的最后一份课程实验报告了。

回溯本门课程这一学期的经历，感触颇多。从最初的全然无知，到之后的生嚼硬啃，再到最后的收官，一路下来，虽然并不容易，但又颇有成就感。我知道，我们最后释出的成品并不完美，但是我们能在这门课程中感到乐趣，能在这门课程中感到充实，能尽情地制作出一款能让我们满意的产品，我已经知足了。

同时，在此列出感谢名单，次序不分先后：

感谢诸多学长学姐对我们这门课程提供的建议以及帮助，有了前人的经验我们才能更好地踏过每一个坑坑洼洼。

感谢电院信工大四的孙建辉同学，以及周毓同学在这门课程中的大力支持，多谢你们在我们一开始的迷茫期中能够伸出援手。

感谢杨宇红老师为这门课程所付出的精力，您在 FTP 上所提供的资料让我们获益良多。

感谢交大电子系开设这门课程，让我们能在本科生活的最后，还能感受到“美好的事物在我们手中产生”的成就感。

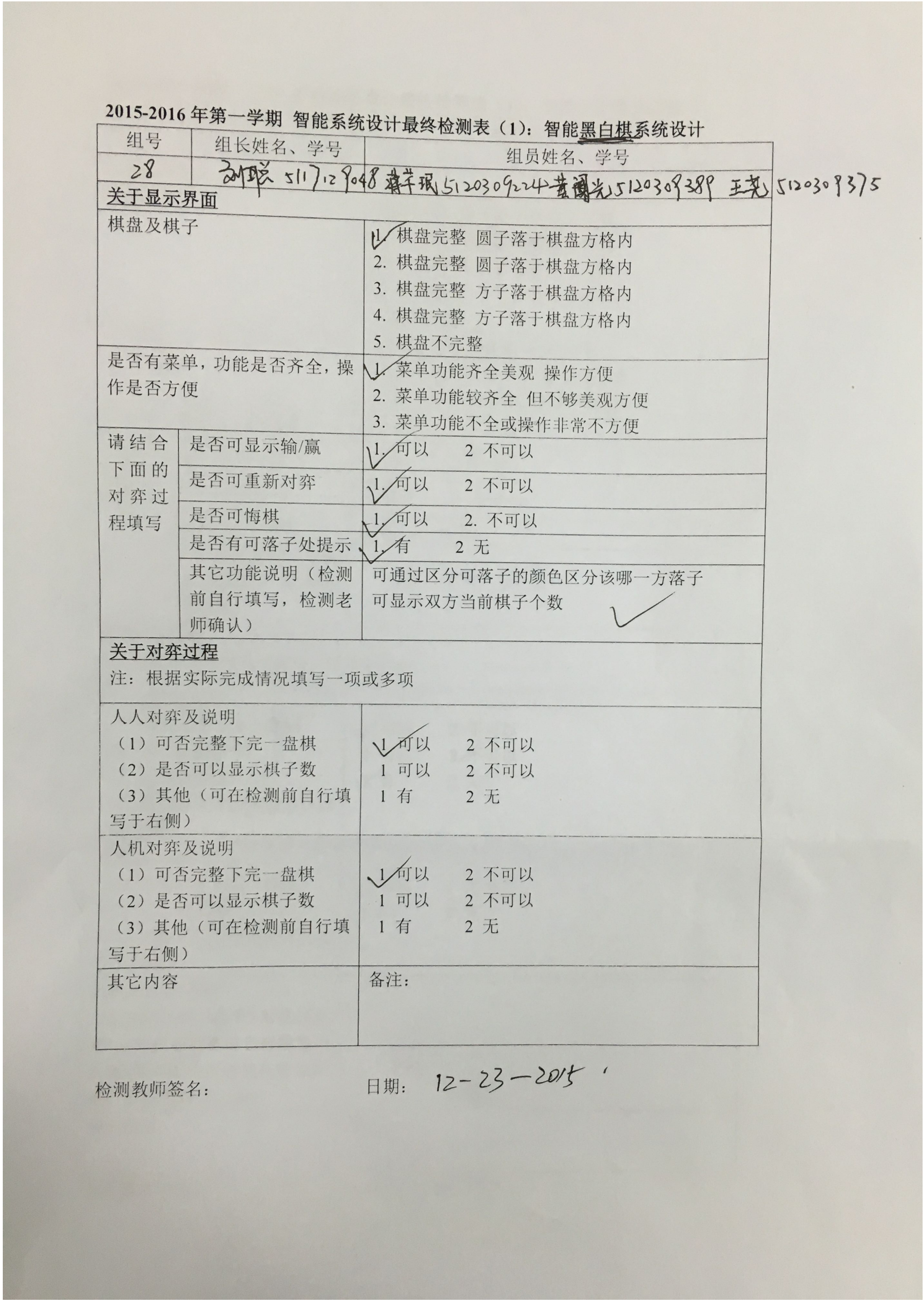
最后，感谢我的组员们在整个实验课程中对我的包容与合作，谢谢你们能忍耐那个直来直往，烦人话唠，脾气暴躁的我，你们辛苦了。

饮水思源，谢谢大家。

8. 参考文献

[1] 爱板网. Nexys3 FPGA 开发板[EB/OL].
http://www.eeboard.com/ziliao/nexys3-fpga%E5%BC%80%E5%8F%91%E6%9D%BF/.
[2] 上海交大电子工程系. VGA 接口的 FPGA 代码_Nexys3[EB/OL].ftp://202.120.39.248.
[3] 柏爱俊.几种智能算法在黑白棋程序中的应用[C].安徽：中国科技大学, 2007: 中国科学技术大学大学生研究计划结题报告
[4] 衣锦夜行.AlphaBeta 剪枝算法[EB/OL].http://blog.sina.com.cn/s/blog_5d9ee55e0100uuy2.html

9. 附录 A：检测结果



2015-2016 年第一学期 智能系统设计最终检测表（1）：智能黑白棋系统设计

组号	组长姓名、学号	组员姓名、学号
28	[illegible] 5117129048	蒋芊琨 5120309224 黄国光 5120309389 王尧 5120309375

关于显示界面		
棋盘及棋子		✓1. 棋盘完整 圆子落于棋盘方格内 2. 棋盘完整 圆子落于棋盘方格内 3. 棋盘完整 方子落于棋盘方格内 4. 棋盘完整 方子落于棋盘方格内 5. 棋盘不完整
是否有菜单，功能是否齐全，操作是否方便		✓1. 菜单功能齐全美观 操作方便 2. 菜单功能较齐全 但不够美观方便 3. 菜单功能不全或操作非常不方便
请结合下面的对弈过程填写	是否可显示输/赢	✓1. 可以 2 不可以
	是否可重新对弈	✓1. 可以 2 不可以
	是否可悔棋	✓1. 可以 2. 不可以
	是否有可落子处提示	✓1. 有 2 无
	其它功能说明（检测前自行填写，检测老师确认）	可通过区分可落子的颜色区分该哪一方落子 可显示双方当前棋子个数 ✓
关于对弈过程 注：根据实际完成情况填写一项或多项		
人人对弈及说明 （1）可否完整下完一盘棋 （2）是否可以显示棋子数 （3）其他（可在检测前自行填写于右侧）		 ✓1 可以 2 不可以 1 可以 2 不可以 1 有 2 无
人机对弈及说明 （1）可否完整下完一盘棋 （2）是否可以显示棋子数 （3）其他（可在检测前自行填写于右侧）		 ✓1 可以 2 不可以 1 可以 2 不可以 1 有 2 无
其它内容		备注：

检测教师签名： 日期： 12-23-2015

10. 附录 B：软件程序清单

因为我们全组的整个项目流程再是在 GitHub 上进行管理的，所以为了老师们的浏览方便起见，我们在此处便不再生硬地将代码复制粘贴在报告之中，而是将我们小组的 GitHub 项目地址放上，方便老师浏览与查阅。

最终完成版项目地址：

https://github.com/SmartSys-28/Othello

最终完成版项目 zip 包下载：

https://github.com/SmartSys-28/Othello/archive/1.0-release.zip