实验报告

2022 年 1 月 1 日　　成绩：_____

姓名	蔡振	学号	19035312	班级	19052320
专业	计算机科学与技术		课程名称	嵌入式系统课程设计	
实验序号	无	实验名称	期末大作业		
实验时间	2022 年 1 月 1 日	实验地点	1 教 604	实验设备号	8

一、实验目的和实验要求

实验目的：

掌握 STM32 下的 LCD、触控板、串口等知识，并能实现一个人机交互的下棋项目。

实验要求：

实现一个人机交互的下棋项目。

游戏规则介绍：

1、双方各自拥有五个棋子。

2、每个棋子都是相同的（行为方式都是一样），每个棋子都可以在交点之间来回移动，每次一方只能移动一个棋子。

3、每次每个棋子只能移动一个单位。

4、当两个棋子和一个敌方棋子相对（在一条横线上连续的三点）时，敌方棋子去掉，游戏继续进行。

5、当一条线上只有一方两个棋子并相邻，此时敌方的棋子移动到这条线上，游戏继续进行，没有棋子被吃（主动走到含有 2 个对方棋子的线上）。

6、一条线上某个时刻有四个棋子，游戏继续进行，没有棋子被吃。

7、当其中一方棋子只有一个的时候，该方输掉游戏，游戏结束。

8、 当其中一方无法移动任何一个棋子的时候也算输掉游戏，游戏结束。

二、程序流程图

在本程序中，实现了人机对战和双人对战两种模式，由于两种方式的程序执行过程略有不同，故下面分开讨论他们的程序流程图。

双人对战模式的程序流程图：

1、双人对战模式的板子程序流程图：

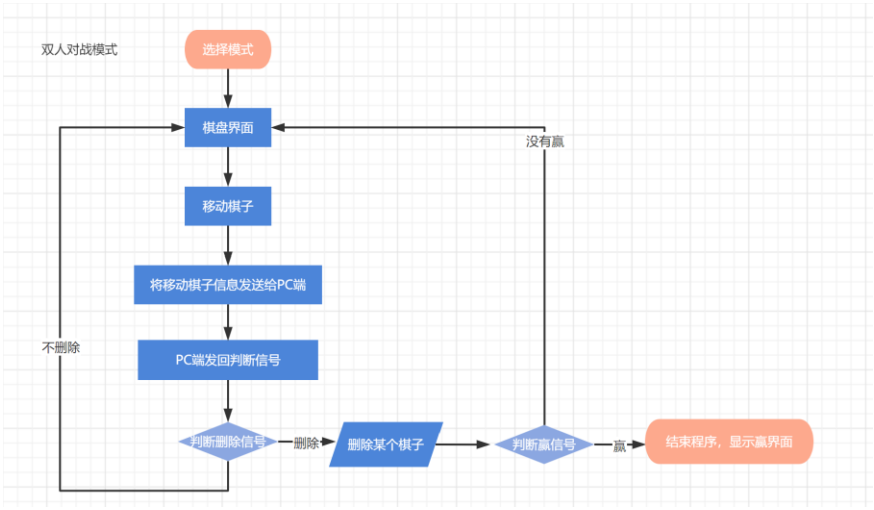


图 1：双人对战模式的板子程序流程图

2、双人对战模式的算法层面的程序流程图：

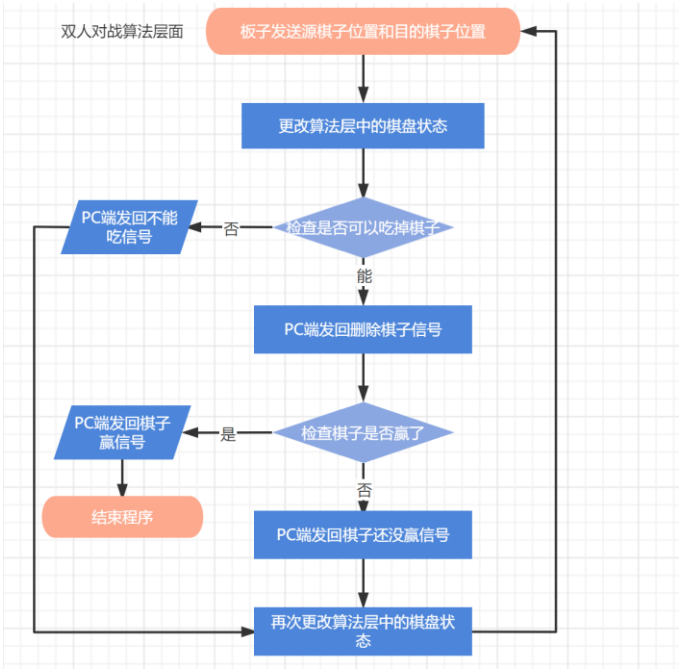


图 2：双人对战模式的算法层面的程序流程图

人机对战模式的程序流程图：

1、人机对战模式的板子程序流程图：

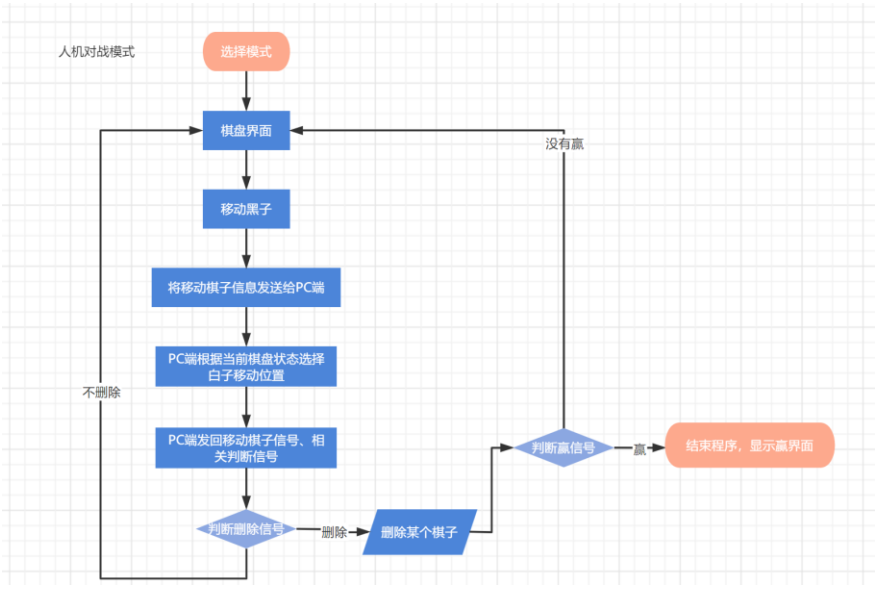


图 3：人机对战模式的板子程序流程图

2、人机对战模式的算法层面的程序流程图：

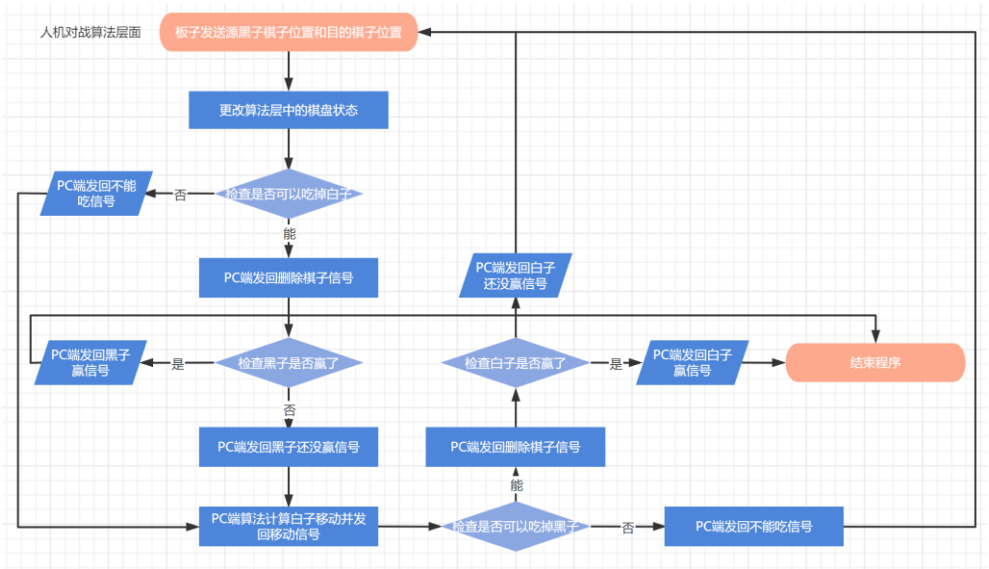


图 4：人机对战模式的算法层面的程序流程图

三、主要函数说明

Keil 程序实现：

LCD 初始化函数：

在本函数中，通过查阅相关手册进行配置 LCD 的 GPIO 相关信息。

```
void LCD_Configuration(void)
{

  GPIO_InitTypeDef GPIO_InitStructure;
  /* Enable the GPIO_LED Clock */
  RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC| RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO , ENABLE);
//  GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE); //Disable jtag  ,Enable SWD

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5 |GPIO_Pin_7 |GPIO_Pin_11|GPIO_Pin_12 ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  GPIO_SetBits(GPIOD, GPIO_Pin_12);  //reset pin

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);


}
```

图 5：LCD 初始化函数

UART 串口初始化函数：

在本函数中，通过查阅相关手册配置 RS232 串口，具体函数省略，可在上传的程序中查看具体函数实现过程。

```
void uart_test()
{
    NVIC_Configuration();
    RS232_Configuration();
    //LCD_to_USART();
    //RS232SendByte(sDataX_all_BLACK * 10 + sDataY_all_BLACK);
    //RS232SendByte(dDataX_all_BLACK * 10 + dDataY_all_BLACK);
  /*
  while(1) {

      if(recvDealFlag == 1) {
        if(cnt % 2 == 1) {
            sDataX_all_WHITE = recvData / 10;
            sDataY_all_WHITE = recvData % 10;
        }
        else if(cnt % 2 == 0) {
            dDataX_all_WHITE = recvData / 10;
            dDataY_all_WHITE = recvData % 10;
            USART_to_LCD();
        }
        recvDealFlag = 0;
      }
    }
    */
}
```

图 6：UART 串口初始化函数

触控板初始化函数：

在本函数中，通过查阅相关手册配置了触控板的 GPIO、中断配置、触控配置，具体函数实现

可见图 7、图 8。在图 7 中的 TOUCH_SCREEN_INIT 函数以及图 8 中的 TOUCH_INT_config 函数中用于实现 GPIO 的配置，在图 8 中的 TOUCH_INY_EXIT_Init 函数用于实现触摸屏的中断配置。

```
void  TOUCH_SCREEN_INIT() {

    GPIO_InitTypeDef GPIO_InitStructure;
    SPI_InitTypeDef  SPI_InitStructure;
  /* Enable GPIOB, GPIOC and AFIO clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO , ENABLE);
    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE); //Disable jtag ,Enable SWD

  /* SPI pins configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15| GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_5;
    GPIO_Init(GPIOB, &GPIO_InitStructure);// MOSI

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 ;
    GPIO_Init(GPIOA, &GPIO_InitStructure);// MISO

}
```

图 7：触控板初始化函数（第一部分）

```
static void TOUCH_INT_config(void) {
     GPIO_InitTypeDef GPIO_InitStructure;
     /* Enable GPIOB, GPIOC and AFIO clock */
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);
     /* LEDs pins configuration */
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
     //GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
     GPIO_Init(GPIOC, &GPIO_InitStructure);
}

static void TOUCH_INT_EXIT_Init(void) {
     EXTI_InitTypeDef EXTI_InitStructure;
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE); //AFIO??
    /* Connect Button EXTI Line to Button GPIO Pin */
     GPIO_EXTILineConfig(GPIO_PortSourceGPIOC,GPIO_PinSource2);
    /* Configure Button EXTI line */
     EXTI_InitStructure.EXTI_Line = EXTI_Line2;
     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
     EXTI_Init(&EXTI_InitStructure);
     EXTI_ClearITPendingBit(EXTI_Line2);


  }
```

图 8：触控板 IO 端口配置和中断配置（第二部分）

棋盘初始界面绘制函数：

在本函数中，主要根据 LCD 所定义的参数进行配置，并在 LCD 屏幕上进行画图、着色，绘制初始化界面，初始化界面可以查看“实验结果展示”中的图片，用来供使用者选择游戏模式。当使用者选择完游戏模式后，LCD 屏幕会绘制棋盘界面，棋盘开始界面可以查看“实验结果展示”中的图片。

```
void LCD_test()
{
   u8 x=0;
  u8 lcd_id[12];      //存放LCD ID字符串
  //初始化与LED连接的硬件接口
  LCD_Init();
  POINT_COLOR=BLACK;
  LCD_Clear(RED);
  startFlag = 0;
  LCD_ShowString(30,40,200,24,24,"  vs computer");
  LCD_DrawLine(10,150,225,150);
  LCD_ShowString(30,200,200,24,24,"  vs human");
  while(startFlag == 0) {}
    //times = 0;
    if(sDataY <= 150) {
      AI_VS_Flag = 0;
    }
    else {
      AI_VS_Flag = 1;
    }
  RS232SendByte(AI_VS_Flag);
    //LCD_ShowString(30,40,200,24,24,"Mini STM32 ^_^");
  LCD_Clear(YELLOW);

    LCD_DrawLine(25,75,25,275);
    LCD_DrawLine(75,75,75,275);
    LCD_DrawLine(125,75,125,275);
    LCD_DrawLine(175,75,175,275);
    LCD_DrawLine(225,75,225,275);

    LCD_DrawLine(25,75,225,75);
    LCD_DrawLine(25,125,225,125);
    LCD_DrawLine(25,175,225,175);
    LCD_DrawLine(25,225,225,225);
    LCD_DrawLine(25,275,225,275);
```

图 9：棋盘初始界面绘制（第一部分）

```
    LCD_Draw_Circle_Empty(25,75,15);
    LCD_Draw_Circle_Empty(25,125,15);
    LCD_Draw_Circle_Empty(25,175,15);
    LCD_Draw_Circle_Empty(25,225,15);
    LCD_Draw_Circle_Empty(25,275,15);

    LCD_Draw_Circle_Full(225,75,15);
    LCD_Draw_Circle_Full(225,125,15);
    LCD_Draw_Circle_Full(225,175,15);
    LCD_Draw_Circle_Full(225,225,15);
    LCD_Draw_Circle_Full(225,275,15);

}
```

图 10：棋盘初始界面绘制（第二部分）

串口中断处理程序：

在本函数中，由于每发送或接受一次数据都会触发一次中断，用于对 RS232 串口进行收发的处理。

```
char RS232InData;
char recvData;
int recvDealFlag = 0;
int cnt = 0;

void USART1_IRQHandler(void)
{
   if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
   {
     //RS232InData = USART1->DR;//
     recvData = USART1 -> DR;
     //recvData -= 0x30;
     recvDealFlag ++;
     USART_ClearITPendingBit(USART1, USART_IT_RXNE);
   }
   if (USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
   {
         USART_ClearITPendingBit(USART1, USART_IT_TXE);          /* Clear the USART transmit interrupt
   }
 }
```

图 11：串口中断处理程序

触摸屏中断处理程序：

在本程序中，使用者每次点击屏幕都会产生一次中断，在本程序中可以区分使用者的点击是源棋子还是将要移动的棋子位置。在本程序中的 TouchScreen 用于实际的获取当前点击的位置并赋值给 xScreen 和 ySceen。sDataX 用于接受源棋子的 X 轴，sDataY 用于接受源棋子的 Y 轴，dDataX 用于接受目的棋子的 X 轴，dDataY 用于接受目的棋子的 Y 轴。times 用来区分当前的点击事件是源棋子还是目的棋子。startFlag 用来传递用户在初始化界面中选择棋盘模式的开始信息。moveFlag 用来告知功能实现函数当前可以开始移动了。

```
void EXTI2_IRQHandler (void) {
  delay_ms(20);
  if(EXTI_GetITStatus(EXTI_Line2) != RESET)
  {
     TouchScreen();
     if(times == 0 && xScreen != 240) {
        sDataX = xScreen;
        sDataY = yScreen;
       //startFlag = 1;
     }
     else if(times == 2 && xScreen != 240) {
       dDataX = xScreen;
       dDataY = yScreen;
        moveFlag = 1;
     }
     times ++;
     if(times == 4) {
       times = 0;
     }

     startFlag = 1;

  }
  EXTI_ClearITPendingBit(EXTI_Line2);
}
```

图 12：触摸屏中断处理程序

功能实现函数（人机对战模式和双人对战两种）：

在功能实现函数中，调用了上述的各个函数，并且初始化了棋盘的当前状况，用 chess[5][5]

二维数组表示。AI_VS_Flag 参数用于区分使用者选择的模式，该参数的传递由串口实现函数定义，当 AI_VS_Flag=1 时，表示该模式为双人对战模式；当 AI_VS_Flag=0 时，表示该模式为人机对战模式。winFlag 用于表示赢状态，当 winFlag=0 时，表示当前还没有赢；当 winFlag=1 时，表示当前黑子赢；当 winFlag=2 时，表示当前白子赢。其中还有部分参数没有描述到，具体可以参见图片或者上传的程序。

在本函数中还调用的 jiaozhun 函数，该函数用于实现使用者在板子点按的位置转换为棋盘交点的位置。LCD_to_USART_X 和 LCD_to_USART_Y 函数用于实现 LCD 屏幕上的位置转换为 0-4 的串口传输位置。USART_to_LCD_X 和 USART_to_LCD_Y 函数用于实现 0-4 的串口传输位置转换为 LCD 屏幕上显示的位置。LCD_ClearFullColor_Circle 函数用于清除某个黑子。LCD_Draw_Circle_Full 函数用于绘制某个黑子。LCD_ClearColor_Circle 函数用于清除某个白子。LCD_Draw_Circle_Empty 函数用于绘制某个白子。RS232SendByte 函数用于实现串口信息传输。其中还有部分函数没有描述到，具体可以参见图片或者上传的程序。

```
int chess[5][5] = {0};

void Touch_test()
{
  int j;
  TOUCH_SCREEN_INIT();
  TOUCH_INT_config();
  TOUCH_INT_EXIT_Init();
  TOUCH_InterruptConfig();
    LCD_test();
  times = 0;

  for(j = 0; j < 5; j ++) {
      chess[0][j] = 1;
      chess[4][j] = 2;
  }
  if(AI_VS_Flag == 0) {
    while(!winFlag) {       //后续应该删除while
          jiaozhun(sDataX, sDataY);
          sDataX_all_BLACK = dataX_jiaozhun;
          sDataY_all_BLACK = dataY_jiaozhun;
          sDataX_all_BLACK = LCD_to_USART_X(sDataX_all_BLACK);
          sDataY_all_BLACK = LCD_to_USART_Y(sDataY_all_BLACK);
          if(sDataX_all_BLACK == 5 || sDataY_all_BLACK == 5 || chess[sDataX_all_BLACK][sDataY_all_BLACK] != 1) {
            times = 0;
            moveFlag = 0;
            recvDealFlag = 0;
            continue;
          }
        times = 2;
      if(moveFlag == 1) {
        /*
        jiaozhun(sDataX, sDataY);
        //if(jishu % 2 == 0) {
          //LCD_ClearFullColor_Circle(250 - dataX_jiaozhun, dataY_jiaozhun, 15);
          sDataX_all_BLACK = dataX_jiaozhun;
          sDataY_all_BLACK = dataY_jiaozhun;
```

图 13：功能实现函数（第一部分）

```
    sDataY_all_BLACK = dataY_jiaozhun;
    sDataX_all_BLACK = LCD_to_USART_X(sDataX_all_BLACK);
    sDataY_all_BLACK = LCD_to_USART_Y(sDataY_all_BLACK);

    if(sDataX_all_BLACK == 5 || sDataY_all_BLACK == 5 || chess[sDataX_all_BLACK][sDataY_all_BLACK] != 1) {
      times = 0;
      moveFlag = 0;
      recvDealFlag = 0;
      continue;
    }
  */
    jiaozhun(dDataX, dDataY);
    //LCD_Draw_Circle_Full(250 - dataX_jiaozhun, dataY_jiaozhun,15);
    dDataX_all_BLACK = dataX_jiaozhun;
    dDataY_all_BLACK = dataY_jiaozhun;

    dDataX_all_BLACK = LCD_to_USART_X(dDataX_all_BLACK);
    dDataY_all_BLACK = LCD_to_USART_Y(dDataY_all_BLACK);


    if(sDataX_all_BLACK == 5 || sDataY_all_BLACK == 5 || dDataX_all_BLACK == 5 || dDataY_all_BLACK == 5 || chess[sDataX_all_BLACK][sDataY_all
      moveFlag = 0;
      recvDealFlag = 0;
      continue;
    }
  chess[sDataX_all_BLACK][sDataY_all_BLACK] = 0;
  chess[dDataX_all_BLACK][dDataY_all_BLACK] = 1;
  sDataX_all_BLACK = USART_to_LCD_X(sDataX_all_BLACK);
    sDataY_all_BLACK = USART_to_LCD_Y(sDataY_all_BLACK);
    dDataX_all_BLACK = USART_to_LCD_X(dDataX_all_BLACK);
    dDataY_all_BLACK = USART_to_LCD_Y(dDataY_all_BLACK);
    LCD_ClearFullColor_Circle(250 - sDataX_all_BLACK, sDataY_all_BLACK, 15);
    LCD_Draw_Circle_Full(250 - dDataX_all_BLACK, dDataY_all_BLACK, 15);
    //LCD_to_USART();
    sDataX_all_BLACK = LCD_to_USART_X(sDataX_all_BLACK);
    sDataY_all_BLACK = LCD_to_USART_Y(sDataY_all_BLACK);
    dDataX_all_BLACK = LCD_to_USART_X(dDataX_all_BLACK);
```

图 14：功能实现函数（第二部分）

```
    dDataY_all_BLACK = LCD_to_USART_Y(dDataY_all_BLACK);


  //改成发4次
    RS232SendByte(sDataX_all_BLACK);
  while(USART_GetITStatus(USART1, USART_IT_TXE) != RESET) {};
    delay_ms(20);
    RS232SendByte(sDataY_all_BLACK);
    delay_ms(20);
    RS232SendByte(dDataX_all_BLACK);
    delay_ms(20);
    RS232SendByte(dDataY_all_BLACK);
    while(recvDealFlag != 1) {}
     clearFlag = recvData;
    //clear white
    if(clearFlag == 2) {
       while(recvDealFlag != 2) {}
       deleteX = recvData;
       while(recvDealFlag != 3) {}
       deleteY = recvData;
      chess[deleteX][deleteY] = 0;
       deleteX = USART_to_LCD_X(deleteX);
       deleteY = USART_to_LCD_Y(deleteY);
       LCD_ClearColor_Circle(250 - deleteX, deleteY, 15);
       while(recvDealFlag != 4) {}
       winFlag = recvData;
       if(winFlag == 2 || winFlag == 1) {
        break;
       }
     }
    recvDealFlag = 0;

    while(recvDealFlag != 1) {}
        sDataX_all_WHITE = recvData;
      while(recvDealFlag != 2) {}
        sDataY_all_WHITE = recvData;
```

图 15：功能实现函数（第三部分）

```
            sDataY_all_WHITE = recvData;
          while(recvDealFlag != 3) {}
            dDataX_all_WHITE = recvData;
          while(recvDealFlag != 4) {}
            dDataY_all_WHITE = recvData;
            recvDealFlag = 0;
          chess[sDataX_all_WHITE][sDataY_all_WHITE] = 0;
            chess[dDataX_all_WHITE][dDataY_all_WHITE] = 2;

          sDataX_all_WHITE = USART_to_LCD_X(sDataX_all_WHITE);
          sDataY_all_WHITE = USART_to_LCD_Y(sDataY_all_WHITE);
          dDataX_all_WHITE = USART_to_LCD_X(dDataX_all_WHITE);
          dDataY_all_WHITE = USART_to_LCD_Y(dDataY_all_WHITE);
        //}
        LCD_ClearColor_Circle(250 - sDataX_all_WHITE, sDataY_all_WHITE, 15);
        LCD_Draw_Circle_Empty(250 - dDataX_all_WHITE, dDataY_all_WHITE,15);
        while(recvDealFlag != 1) {}
         clearFlag = recvData;
        //clear black
        if(clearFlag == 1) {
           while(recvDealFlag != 2) {}
           deleteX = recvData;
           while(recvDealFlag != 3) {}
           deleteY = recvData;
            chess[deleteX][deleteY] = 0;
           deleteX = USART_to_LCD_X(deleteX);
           deleteY = USART_to_LCD_Y(deleteY);
           LCD_ClearFullColor_Circle(250 - deleteX, deleteY, 15);
           while(recvDealFlag != 4) {}
           winFlag = recvData;
          if(winFlag == 1 || winFlag == 2) {
            break;
          }
         }
        recvDealFlag = 0;
        moveFlag = 0;
      }
```

图 16：功能实现函数（第四部分）

```
else if(AI_VS_Flag == 1) {
  int count = 0;
  int sDataX_all;
  int sDataY_all;
  int dDataX_all;
  int dDataY_all;
  while(!winFlag) {       //后续应该删除while
    if(times != 0) {
        jiaozhun(sDataX, sDataY);
        sDataX_all = dataX_jiaozhun;
        sDataY_all = dataY_jiaozhun;
        sDataX_all = LCD_to_USART_X(sDataX_all);
        sDataY_all = LCD_to_USART_Y(sDataY_all);
        if(chess[sDataX_all][sDataY_all] != (count % 2 + 1)) {
          times = 0;
          moveFlag = 0;
          continue;
        }
      }
    if(moveFlag == 1) {               //黑棋先走
        //jiaozhun(sDataX, sDataY);
        //sDataX_all = dataX_jiaozhun;
        //sDataY_all = dataY_jiaozhun;
        jiaozhun(dDataX, dDataY);
        dDataX_all = dataX_jiaozhun;
        dDataY_all = dataY_jiaozhun;

        //sDataX_all = LCD_to_USART_X(sDataX_all);
        //sDataY_all = LCD_to_USART_Y(sDataY_all);
        dDataX_all = LCD_to_USART_X(dDataX_all);
        dDataY_all = LCD_to_USART_Y(dDataY_all);

        if(chess[sDataX_all][sDataY_all] != (count% 2 + 1) || chess[dDataX_all][dDataY_all] || abs(sDataX_all + sDataY_all - dDataX_all - dDataY
        moveFlag = 0;
        recvDealFlag = 0;
        continue;
      }
```

图 17：功能实现函数（第五部分）

```
        chess[sDataX_all][sDataY_all] = 0;
        chess[dDataX_all][dDataY_all] = count % 2 + 1;

        sDataX_all = USART_to_LCD_X(sDataX_all);
        sDataY_all = USART_to_LCD_Y(sDataY_all);
        dDataX_all = USART_to_LCD_X(dDataX_all);
        dDataY_all = USART_to_LCD_Y(dDataY_all);

      if(count % 2 == 0) {
        LCD_ClearFullColor_Circle(250 - sDataX_all, sDataY_all, 15);
        LCD_Draw_Circle_Full(250 - dDataX_all, dDataY_all,15);
      }
      else {
        LCD_ClearColor_Circle(250 - sDataX_all, sDataY_all, 15);
        LCD_Draw_Circle_Empty(250 - dDataX_all, dDataY_all, 15);
      }



        sDataX_all = LCD_to_USART_X(sDataX_all);
        sDataY_all = LCD_to_USART_Y(sDataY_all);
        dDataX_all = LCD_to_USART_X(dDataX_all);
        dDataY_all = LCD_to_USART_Y(dDataY_all);

      //改成发4次
        RS232SendByte(sDataX_all);
        delay_ms(20);
        RS232SendByte(sDataY_all);
        delay_ms(20);
        RS232SendByte(dDataX_all);
        delay_ms(20);
        RS232SendByte(dDataY_all);
        while(recvDealFlag != 1) {}
         clearFlag = recvData;
        //clear white
        if(clearFlag) {
           while(recvDealFlag != 2) {}
```

图 18：功能实现函数（第六部分）

```
           deleteX = recvData;
           while(recvDealFlag != 3) {}
           deleteY = recvData;

           if(clearFlag == 2) {
             chess[deleteX][deleteY] = 0;
             deleteX = USART_to_LCD_X(deleteX);
           deleteY = USART_to_LCD_Y(deleteY);
             LCD_ClearColor_Circle(250 - deleteX, deleteY, 15);
           }
          else {
            chess[deleteX][deleteY] = 0;
            deleteX = USART_to_LCD_X(deleteX);
           deleteY = USART_to_LCD_Y(deleteY);
            LCD_ClearFullColor_Circle(250 - deleteX, deleteY, 15);
          }
            while(recvDealFlag != 4) {}
           winFlag = recvData;
           if(winFlag) {
            break;
           }
         }
        recvDealFlag = 0;
        moveFlag = 0;
        count ++;
      }
    }
  }

  LCD_Clear(RED);
  if(winFlag == 1) {
    LCD_ShowString(50,80,200,24,24,"black win");
  }
  else if(winFlag == 2) {
    LCD_ShowString(50,80,200,24,24,"white win");
  }
}
```

图 19：功能实现函数（第七部分）

PC 端的程序实现：

发送串口函数：

PC 端的串口程序主要是调用了 windows.h 头文件下的内置库，在本项目中我使用了 github 上的一个开源实现函数，并对其进行了改进，具体的串口实现函数可在上传的代码中查看。

```
15    WzSerialPort w;
16  void sendDemo(int buf)
17  {
18      WzSerialPort w;
19      char a[1];
20      if (w.open("COM3", 9600, 0, 8, 1))
21      {
22          a[0] = buf;
23              w.send(a , 1);
24              w.close();
25      }
26      else
27      {
28          cout << "open serial port failed..." << endl;
29      }
30  }
31
```

图 20：发送串口函数

接受串口函数：

接受串口函数分为两种：接受单字节和四字节。由于在本项目中多次使用了四字节的传送，所以在这里对其进行了封装。

```
32  char* receive_Bytes() {
33      WzSerialPort w;
34      char buf[4];
35      if (w.open("COM3", 9600, 0, 8, 1)) {
36          memset(buf, 0, 4);
37          //w.receive(buf, 4);
38          while (!w.receive(buf, 4)) { cout << "no data, wait" << endl;  Sleep(1000); }
39          w.close();
40      }
41      return buf;
42  }
43  int receive_oneByte()
44  {
45      WzSerialPort w;
46      char buf[1];
47      if (w.open("COM3", 9600, 0, 8, 1))
48      {
49          memset(buf, 0, 1);
50          //w.receive(buf, 1);
51          while (!w.receive(buf, 1)) { cout << "no data, wait" << endl; Sleep(1000); };
52              //cout << buf << endl;
53          w.close();
54      }
55      return buf[0];
56  }
```

图 21：接受串口函数

判断当前棋子是否能吃函数：

在本函数中，调用了 check_EquX 和 checkEquY 函数用于检查 X 轴和 Y 轴上的棋子是否能吃。

```
147   /* 判断是否能吃、是否赢了 */
148  void check(int i, int j) {
149      int tempX, tempY;
150      for (int m = 0; m < 4; m++) {
151          tempX = i + dir[m][0];
152          tempY = j + dir[m][1];
153          if (tempX >= 0 && tempX < 5 && tempY >= 0 && tempY < 5 && chess[tempX][tempY] == chess[i][j]) {
154              if (tempX == i) {
155                  check_EquX(tempY, j, i);
156              }
157              else if (tempY == j) {
158                  check_EquY(tempX, i, j);
159              }
160          }
161      }
162  }
163
```

图 22：判断当前棋子是否能吃函数

更改棋盘状态和检查函数：

本函数主要实现了 PC 端的检查功能，实现了向板子发送删除信息、赢信息等功能。

```
164  int input_and_check(int sx, int sy, int dx, int dy, int w_or_b) {
165      int re = 0;
166      chess[sx][sy] = 0;
167      chess[dx][dy] = w_or_b;
168      check(dx, dy);
169      if (clearFlag) {
170          /* delete chess */
171          //cout << "clearFlag = " << clearFlag << ",deleteX = " << deleteX << ",deleteY = " << deleteY << endl;
172          sendDemo(clearFlag);
173          sendDemo(deleteX);
174          sendDemo(deleteY);
175
176          chess[deleteX][deleteY] = 0;
177
178          //cntW--;
179          if (chess[dx][dy] == 1) {
180              cntW--;
181          }
182          else if (chess[dx][dy] == 2) {
183              cntB--;
184          }
185          if (cntW == 1) {
186              winFlag = 1;
187              re = 1;
188          }
```

图 23：更改棋盘状态和检查函数（第一部分）

```
192              //cout << "winFlag = " << winFlag << endl;
193              sendDemo(winFlag);
194              clearFlag = 0;
195          }
196          else {
197
198              //cout << "clearFlag = " << clearFlag << endl;
199              sendDemo(clearFlag);
200          }
201          return re;
202      }
```

图 24：更改棋盘状态和检查函数（第二部分）

人机对战中的算法实现函数：

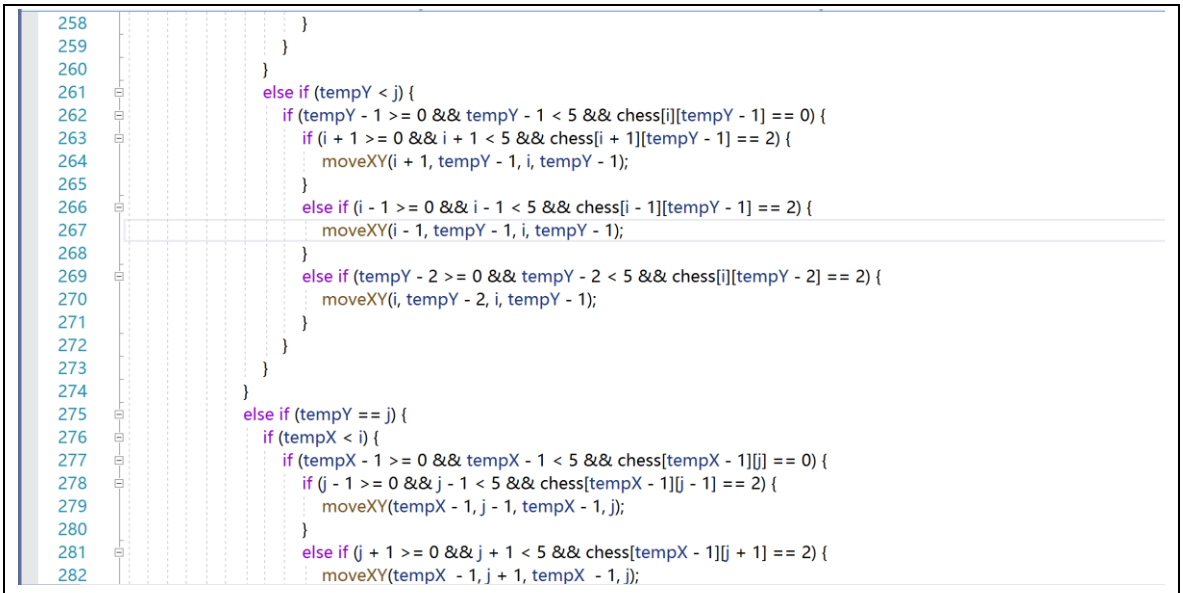
人机对战中的算法可以参考图 25，这里不再累述。

```
/*
*
* 如果出现以下情况则依靠算法移进，否则先随机选择一个白子，用BFS方法四周寻找可以落脚点。
* 1.
*    O          O *     *       *
*     O *      O       O       O          ...类似情况
*                   O         O
* 2.
*     O         * O      O    O *
*    * O         O      O *    O          ...类似情况
*
*/
```

图 25：人机对战中的算法解释

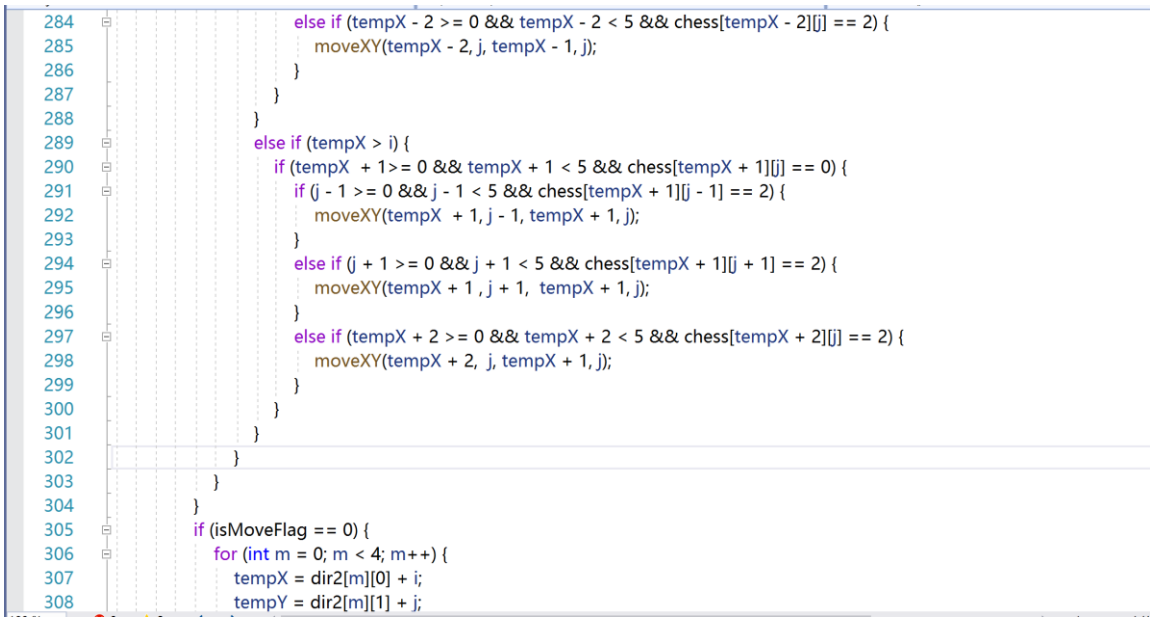
```
void moveWhite() {
    int tempX;
    int tempY;

    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (chess[i][j] == 1) {     /* find black chess */
                for (int m = 0; m < 4; m++) {
                    if (isMoveFlag == 1) {
                        return;
                    }
                    tempX = i + dir[m][0];
                    tempY = j + dir[m][1];
                    if (tempX >= 0 && tempX < 5 && tempY >= 0 && tempY < 5 && chess[tempX][tempY] == 2) {
                        if (tempX == i) {
                            if (tempY > j) {
                                if (tempY + 1 >= 0 && tempY + 1 < 5 && chess[i][tempY + 1] == 0) {
                                    if (i + 1 >= 0 && i + 1 < 5 && chess[i + 1][tempY + 1] == 2 ) {
                                        moveXY(i + 1, tempY + 1, i, tempY + 1);
                                    }
                                    else if (i - 1 >= 0 && i - 1 < 5 && chess[i - 1][tempY + 1] == 2) {
                                        moveXY(i - 1, tempY + 1, i, tempY + 1);
                                    }
                                    else if (tempY + 2 >= 0 && tempY + 2 < 5 && chess[i][tempY + 2] == 2) {
                                        moveXY(i, tempY + 2, i, tempY + 1);
```

图 25：人机对战中的算法（第一部分）



```
258                         }
259                     }
260                 }
261                 else if (tempY < j) {
262                     if (tempY - 1 >= 0 && tempY - 1 < 5 && chess[i][tempY - 1] == 0) {
263                         if (i + 1 >= 0 && i + 1 < 5 && chess[i + 1][tempY - 1] == 2) {
264                             moveXY(i + 1, tempY - 1, i, tempY - 1);
265                         }
266                         else if (i - 1 >= 0 && i - 1 < 5 && chess[i - 1][tempY - 1] == 2) {
267                             moveXY(i - 1, tempY - 1, i, tempY - 1);
268                         }
269                         else if (tempY - 2 >= 0 && tempY - 2 < 5 && chess[i][tempY - 2] == 2) {
270                             moveXY(i, tempY - 2, i, tempY - 1);
271                         }
272                     }
273                 }
274             }
275             else if (tempY == j) {
276                 if (tempX < i) {
277                     if (tempX - 1 >= 0 && tempX - 1 < 5 && chess[tempX - 1][j] == 0) {
278                         if (j - 1 >= 0 && j - 1 < 5 && chess[tempX - 1][j - 1] == 2) {
279                             moveXY(tempX - 1, j - 1, tempX - 1, j);
280                         }
281                         else if (j + 1 >= 0 && j + 1 < 5 && chess[tempX - 1][j + 1] == 2) {
282                             moveXY(tempX  - 1, j + 1, tempX  - 1, j);
```

图 26：人机对战中的算法（第二部分）



```
284                         else if (tempX - 2 >= 0 && tempX - 2 < 5 && chess[tempX - 2][j] == 2) {
285                             moveXY(tempX - 2, j, tempX - 1, j);
286                         }
287                     }
288                 }
289                 else if (tempX > i) {
290                     if (tempX  + 1>= 0 && tempX + 1 < 5 && chess[tempX + 1][j] == 0) {
291                         if (j - 1 >= 0 && j - 1 < 5 && chess[tempX + 1][j - 1] == 2) {
292                             moveXY(tempX  + 1, j - 1, tempX + 1, j);
293                         }
294                         else if (j + 1 >= 0 && j + 1 < 5 && chess[tempX + 1][j + 1] == 2) {
295                             moveXY(tempX + 1 , j + 1,  tempX + 1, j);
296                         }
297                         else if (tempX + 2 >= 0 && tempX + 2 < 5 && chess[tempX + 2][j] == 2) {
298                             moveXY(tempX + 2,  j, tempX + 1, j);
299                         }
300                     }
301                 }
302             }
303         }
304     }
305     if (isMoveFlag == 0) {
306         for (int m = 0; m < 4; m++) {
307             tempX = dir2[m][0] + i;
308             tempY = dir2[m][1] + j;
```

图 27：人机对战中的算法（第三部分）

```
309                 if (tempX >= 0 && tempX < 5 && tempY >= 0 && tempY < 5 && chess[tempX][tempY] == 2) {
310                     if (tempX == i) {
311                         if (!chess[i][(tempY + j) / 2]) {    //no chess
312                             if (i + 1 >= 0 && i + 1 < 5 && chess[i + 1][(tempY + j) / 2] == 2) {
313                                 moveXY(i + 1, (tempY + j) / 2, i, (tempY + j) / 2);
314                             }
315                             else if (i - 1 >= 0 && i - 1 < 5 && chess[i - 1][tempY + j / 2] == 2) {
316                                 moveXY(i - 1, (tempY + j) / 2, i, (tempY + j) / 2);
317                             }
318                         }
319                     }
320                     else if (tempY == j) {
321                         if (!chess[(i + tempX) / 2][j]) {
322                             if (j - 1 >= 0 && j - 1 < 5 && chess[(tempX + i) / 2][j - 1] == 2) {
323                                 moveXY((tempX + i) / 2, j - 1, (tempX + i) / 2, j);
324                             }
325                             else if (j + 1 >= 0 && j + 1 < 5 && chess[(tempX + i) / 2][j + 1] == 2) {
326                                 moveXY((tempX + i) / 2, j + 1, (tempX + i) / 2, j);
327                             }
328                         }
329                     }
330                 }
331             }
332         }
333     }
```

图 28：人机对战中的算法（第四部分）

在图 29 中，描述了人机对战中的算法（第五部分），这里是用于实现当上述的白子移动算法都无法匹配时，系统将使用随机判断函数，随机的选择一个白子进行移动，并且判断本次移动是否合法。

```
336     /* have not moved */
337     if (isMoveFlag == 0) {
338         int i;
339         int j;
340         /* due to find the first is  to find black,so i don't change that */
341         int isCheck[5][5] = { 0 };
342         while(1) {
343             /* begin: add random */
344             i = rand() % 5;
345             j = rand() % 5;
346             while (isCheck[i][j] || chess[i][j] != 2) {
347                 i = rand() % 5;
348                 j = rand() % 5;
349             }
350             for (int m = 0; m < 4; m++) {
351                 tempX = dir[m][0] + i;
352                 tempY = dir[m][1] + j;
353                 /* move to a blank position */
354                 if (tempX >= 0 && tempX < 5 && tempY >= 0 && tempY < 5 && !chess[tempX][tempY]) {
355                     moveXY(i, j, tempX, tempY);
356                     return;
357                 }
358             }
359             isCheck[i][j] = 1;
360
```

图 29：人机对战中的算法（第五部分）

PC 端主函数实现：

在 PC 端的主函数中，用于进行棋盘状态的初始化，接受使用者的选择对战模式。在双人对战

中，PC 端会接受两种不同的棋子信息并进行判断，并将相关信息发回给板子。在人机对战中，PC 端只会接受黑子的信息，并且调用白子移动算法计算白子的移动位置，并且将判断信息以及棋子信息发回给板子。

```
369 int main() {
370     int re = 0;
371     chess_Init();
372     //cin >> AI_VS_Flag;
373     //sendDemo(1);
374     AI_VS_Flag = receive_oneByte();
375     //AI_VS_Flag = 0;
376     if (AI_VS_Flag == 0) {          //AI
377         srand(int(time(0)));
378         while (!re) {
379             //cin >> sDataX_all_BLACK >> sDataY_all_BLACK >> dDataX_all_BLACK >> dDataY_all_BLACK;
380             char* buf = receive_Bytes();
381             sDataX_all_BLACK = buf[0];
382             sDataY_all_BLACK = buf[1];
383             dDataX_all_BLACK = buf[2];
384             dDataY_all_BLACK = buf[3];
385             re = input_and_check(sDataX_all_BLACK, sDataY_all_BLACK, dDataX_all_BLACK, dDataY_all_BLACK, 1);
386             if (re == 1) {
387                 break;
388             }
389             /* move white */
390             isMoveFlag = 0;
391             moveWhite();
392             /* check */
393             re = input_and_check(sDataX_all_WHITE, sDataY_all_WHITE, dDataX_all_WHITE, dDataY_all_WHITE, 2);
```

图 30：PC 端主函数实现（第一部分）

```
394         }
395     }
396     else if (AI_VS_Flag == 1) {         //human
397         while (!re) {
398             //cin >> sDataX_all_BLACK >> sDataY_all_BLACK >> dDataX_all_BLACK >> dDataY_all_BLACK;
399             char* buf = receive_Bytes();
400             sDataX_all_BLACK = buf[0];
401             sDataY_all_BLACK = buf[1];
402             dDataX_all_BLACK = buf[2];
403             dDataY_all_BLACK = buf[3];
404             re = input_and_check(sDataX_all_BLACK, sDataY_all_BLACK, dDataX_all_BLACK, dDataY_all_BLACK, 1);
405             if (re == 1) {
406                 break;
407             }
408             //cin >> sDataX_all_WHITE >> sDataY_all_WHITE >> dDataX_all_WHITE >> dDataY_all_WHITE;
409             buf = receive_Bytes();
410             sDataX_all_WHITE = buf[0];
411             sDataY_all_WHITE = buf[1];
412             dDataX_all_WHITE = buf[2];
413             dDataY_all_WHITE = buf[3];
414             re = input_and_check(sDataX_all_WHITE, sDataY_all_WHITE, dDataX_all_WHITE, dDataY_all_WHITE, 2);
415         }
416     }
417 }
```

图 31：PC 端主函数实现（第二部分）

四、实验结果展示

由于在压缩包中已经上传了本项目的视频，所以这里只展示部分的截图，具体的实验结果可以查看压缩包中的“双人对战模式视频”或“人机对战模式视频”。

用户选择模式（双人对战模式或人机对战模式）界面：

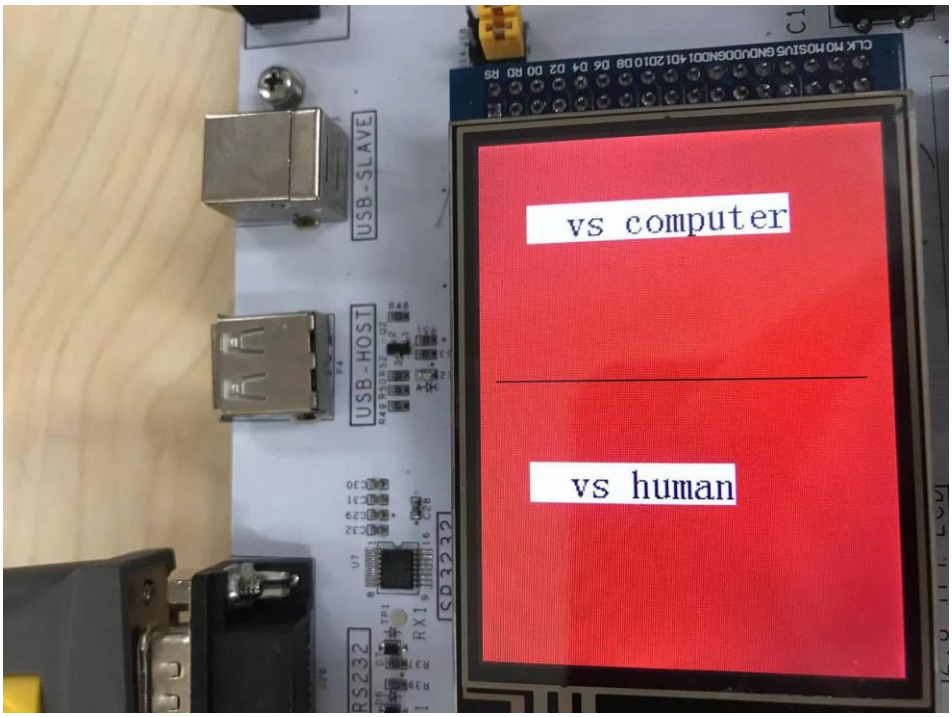


图 32：用户选择模式

棋盘初始化界面：

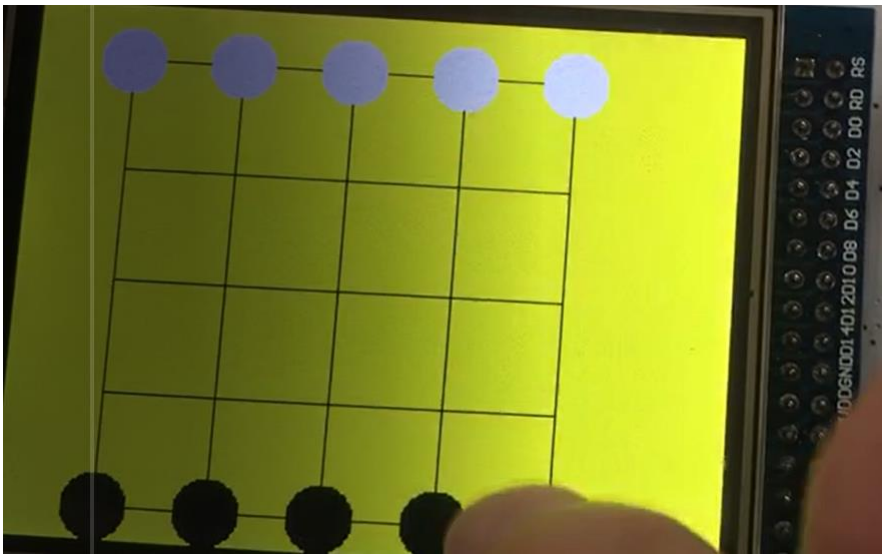


图 33：棋盘初始化界面

移动棋盘界面：

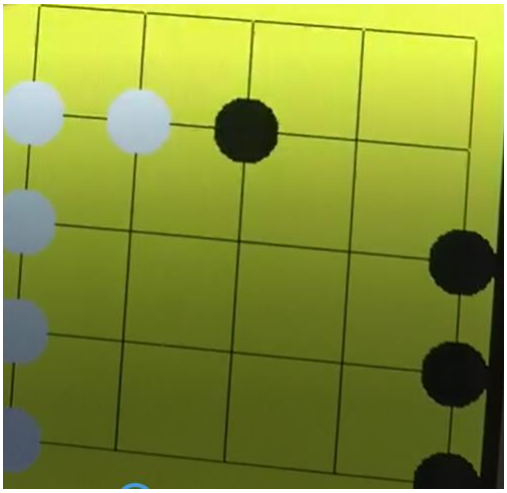


图 34：移动棋盘界面

游戏胜利界面：

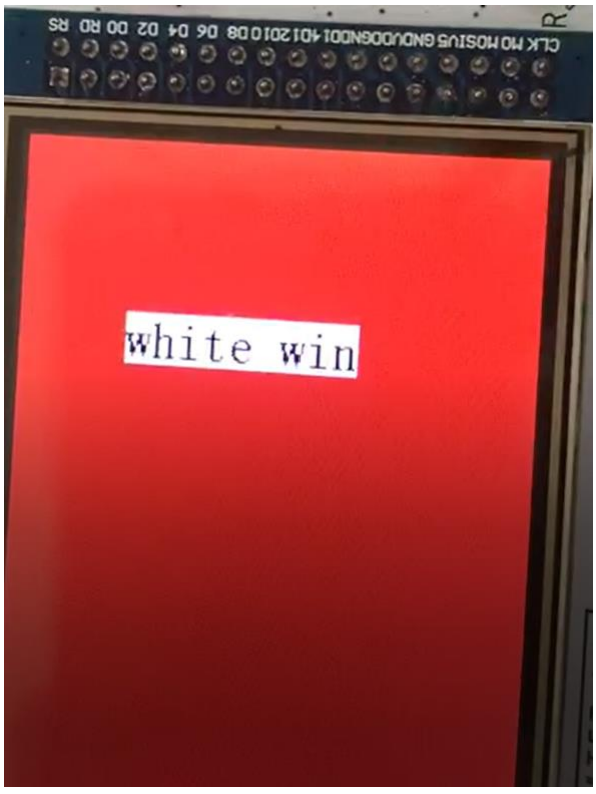


图 35：游戏胜利界面

五、实验收获（心得体会等）

通过本次实验，我独立完成了在 keil 环境下的板子程序开发以及在 PC 端的算法实现。在实验中，我通过查阅相关资料更加熟练的掌握并巩固了包括基于 STM32 的 LCD 配置、触摸屏配置、实验板串口配置、windows 端串口配置、SPI 通信等等知识点。为了实现 PC 端和实验板的通信，我查询了 RS232 的串口配置资料以及 windows 的串口配置资料，在经过很多次调试后最终实现了他们之间的通信。在 keil 环境下绘制棋子和棋盘的函数中，我融合的线性规划的思想。在一些初始化函数的配置中，我查阅了 STM32 的手册以及 CM3 的手册，极大的在增强了我对相关知识点的掌握以及对手册的阅读能力。在实现 PC 端的移动白子算法中，我想出了八种可以有选择的移动白子策略，对我的编程能力大有提高。在本次实验的过程中，增加了我对嵌入式系统的兴趣，希望在日后能继续嵌入式系统的学习。