

# 课程大作业：“黑白棋”实验报告

吴立文 23061925

## 1. 实验要求

- 1) 通过板上 LCD 实现黑白棋游戏；
- 2) 能实现双人对弈功能；
- 3) 能与 PC 机交互，实现“人——机中转——PC”对弈。

## 2. 实验设计

### 2.1. LCD 部分

本次实验基于 TFT LCD 显示屏与电阻式触摸屏实现棋盘的显示与交互。通过并行 GPIO 总线驱动 LCD 显示，以及 GPIO 模拟 SPI 协议读取触摸屏 AD 数据。大致可分为图像显示与触摸屏实现两个部分。

#### 2.1.1. 图像显示

LCD 采用 16 位并行数据总线与控制信号线相结合的方式实现通信。通过对 GPIO 端口的直接配置与操作，系统模拟 8080 时序协议，完成对 LCD 控制器寄存器的读写操作。该方式具有数据传输速度快、时序可控性强的优点，能够满足棋盘界面频繁刷新的需求。

在软件设计上，LCD 驱动层首先完成对 LCD 控制器的初始化，包括控制器型号识别、显示参数配置、扫描方向设置以及颜色格式选择。系统统一采用 RGB565 颜色格式，屏幕分辨率固定为 240\*320。

在此基础上，驱动层封装了一组基础图形显示接口，包括像素点绘制、直线绘制、矩形与圆形绘制、区域填充以及清屏操作。同时，系统还实现了字符和字符串显示功能，支持不同字号的 ASCII 字符显示，可用于输出调试信息、提示文本及游戏状态信息。

#### 2.1.2. 触摸屏实现

本系统采用电阻式触摸屏作为输入设备，用于获取用户在 LCD 屏幕上的触摸位置，实现黑白棋的交互式落子操作。触摸屏通过触摸控制芯片对触点位置进行模数转换，并将转换后的坐标数据传输给 STM32 微控制器进行处理。

在硬件接口方面，触摸屏控制芯片采用 **SPI** 通信方式与主控芯片连接。由于系统资源限制，程序中使用 **GPIO** 模拟 **SPI** 协议完成数据的发送与接收，通过软件控制时钟线、数据输出线和数据输入线，实现对触摸坐标的读取。该方式不依赖硬件 **SPI** 外设，具有较好的灵活性和可移植性。

为提高系统响应效率，触摸屏的按压检测采用外部中断方式实现。当用户触摸屏幕时，触摸控制芯片产生中断信号，触发微控制器进入中断服务程序，在中断中完成触摸数据的采集与处理，从而避免了轮询方式带来的资源浪费。

由于触摸屏采集到的坐标值与 **LCD** 实际显示像素坐标不一致，系统通过触摸校准机制建立触摸坐标与屏幕坐标之间的映射关系。程序通过在 **LCD** 屏上输出指定图案，引导用户点击采集屏幕指定位置并收集触摸数据，计算触摸坐标的最小值与最大值，并采用线性映射的方法将触摸 **AD** 值转换为对应的屏幕像素坐标，实现触摸点与显示内容的一致对应。在实际数据采集过程中，系统会对每个坐标进行多次采样。通过对采样结果取加权平均值的方式，有效降低了触摸抖动和噪声干扰，提高了触摸坐标的稳定性和可靠性。每

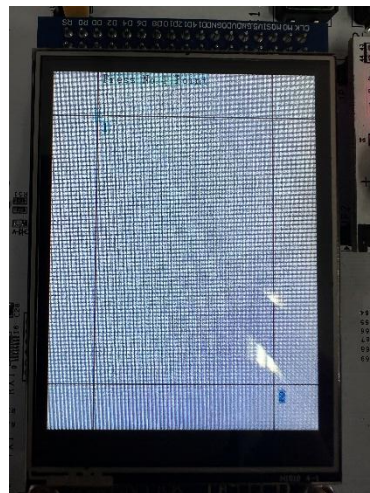


图 1 触摸校准显示内容，交替触摸左上角与右下角完成校准

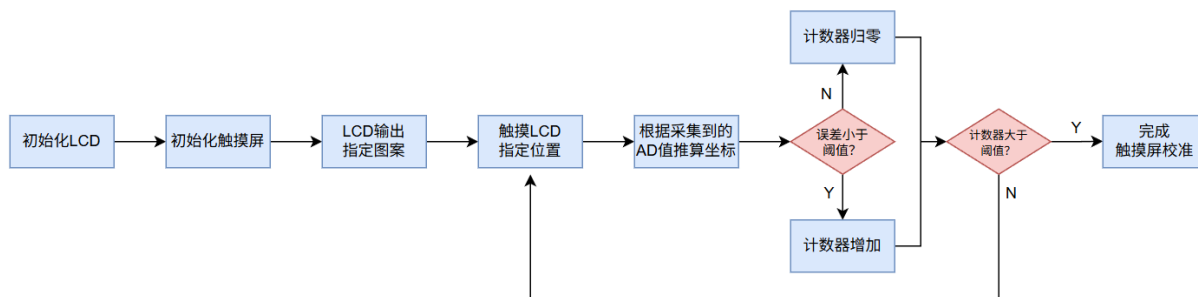


图 2 LCD 触摸屏校准流程图

次采集的触摸坐标会与 **LCD** 实际显示坐标进行比对，满足连续多次误差在一定范围内即通过触摸校准。实际阈值可根据需求调节。

## 2.2. STM32 黑白棋部分

本实验实现了一种基于  $5 \times 5$  棋盘的简化黑白棋游戏，系统以二维数组作为棋盘数据结构，通过 **LCD** 显示模块进行可视化呈现，并结合触摸屏输入完

成玩家交互。游戏支持棋盘初始化、棋子选择与移动、吃子判定、胜负判断以及回合切换等完整流程。

### 2.2.1. 棋盘绘制

再完成触摸屏校准后，通过实现好的 LCD 驱动将 LCD 屏填充为需要的棋盘底色，再在上面画上棋盘与棋子。

在棋盘设计方面，系统将 LCD 显示区域中的  $240 \times 240$  像素划分为规则的  $5 \times 5$  网格，每个交叉点对应一个棋盘位置。程序通过绘制横线和竖线生成棋盘网格，并使用黑白圆形图形表示棋子。棋盘状态由二维数组 **Table** 维护，其中不同数值分别表示空位、白棋和黑棋，实现了棋盘逻辑状态与显示状态的统一管理。代码如下：

```
// 初始化棋盘，包括了棋子位置的初始化
void drawChessboard() {
    LCD_Clear(CHESSBOARD_COL);

    // 240*320 选取 240*240 区域座位棋盘。
    // 五条线各占 48，上下左右各留 24
    for (int i = 0; i < 5; ++i) {
        LCD_DrawLine(24, 24 + i * 48, 240 - 24, 24 + i * 48); // 横线
        LCD_DrawLine(24 + i * 48, 24, 24 + i * 48, 240 - 24); // 竖线
    }

    // 初始化棋子位置，并绘制
    memset(Table, 0, sizeof(Table));
    Table[0][0] = Table[0][1] = Table[0][2] = Table[0][3] = Table[0][4]
= 2; // 黑子
    BCnt = 5;
    Table[4][0] = Table[4][1] = Table[4][2] = Table[4][3] = Table[4][4]
= 1; // 白子
    WCnt = 5;

    for (int i = 0; i < 5; ++i)
        for (int j = 0; j < 5; ++j)
            if (Table[i][j] != 0) {
                POINT_COLOR = (Table[i][j] == 1) ? WHITE : BLACK;
                LCD_Draw_Circle(CheessBoardPos[i], ChessBoardPos[j],
PIECE_RADIUS, 1);
            }
}
```

```
// 重置各种标志位
WinFlag = 0;
PieceX = PieceY = -1, PieceValid = 0;
LastPieceX = LastPieceY = -1, LastPieceValid = 0;
whichTurn = 2;
SelectPieceFlag = 0;
AIFlag = 0;
MovePieceFlag = 0;

sprintf(showstr, "Time for:");
POINT_COLOR = BLACK;
BACK_COLOR = CHESSBOARD_COL;
LCD_ShowString(65, 260, 240, 20, 16, showstr);
LCD_Draw_Circle(160, 265, PIECE_RADIUS, 1); // 黑子先行
}
```

### 2.2.2. 移子逻辑

在移动棋子的逻辑上，程序预先保存了棋盘交叉点（即可落子位置）的坐标，每当玩家触摸便将到的  $x, y$  坐标与交叉点坐标进行比对，误差在一定范围内便视为选择了这个交叉点，实现将  $x, y$  坐标转为棋盘坐标方便后续处理。

每次触摸，若能得到有效棋盘坐标，便需要绘制选中框（还需要擦除上次的选中框，方法下文会介绍）。选中框使用比棋子半径大 1 像素的黄色空心圆表示，代码如下：

```
void selectPiece() {
    if (PressFlag > 0) {
        SelectPieceFlag = 1;
        // 擦除上次选中
        // 如果本次触摸到棋盘外，相当于取消上次的选择
        LastPieceX = PieceX;
        LastPieceY = PieceY;
        LastPieceValid = PieceValid;
        if (LastPieceValid) {
            POINT_COLOR = CHESSBOARD_COL;
            LCD_Draw_Circle(ChessBoardPos[LastPieceX],
ChessBoardPos[LastPieceY], PIECE_RADIUS + 1, 0);
            reDrawChessboardLine(LastPieceX, LastPieceY);
        }

        PressFlag = 0;
        PieceX = PieceY = -1;
    }
}
```

```
for (int i = 0; i < 5; ++i)
    if (abs(xScreen - ChessBoardPos[i]) < PIECE_RADIUS) {
        PieceX = i;
    }
for (int i = 0; i < 5; ++i)
    if (abs(yScreen - ChessBoardPos[i]) < PIECE_RADIUS) {
        PieceY = i;
    }
if (PieceX != -1 && PieceY != -1) {
    PieceValid = 1;
    // 记录本次选中
    POINT_COLOR = CYAN;
    LCD_Draw_Circle(ChessBoardPos[PieceX],
ChessBoardPos[PieceY], PIECE_RADIUS + 1, 0);
} else {
    PieceValid = 0;
}
} else
    SelectPieceFlag = 0;
}
```

为了实现移动棋子功能，系统会记录上次选中的位置。每次触摸都会进行判断：

- 上次选中位置有棋子，且是该棋子的回合；
- 这次选中位置是空位；
- 能从上个位置合法走到当前位置。

同时满足这三条便能满足走棋要求。更新 LCD 显示内容与系统 Table 数组。

刷新 LCD 时，要显示走棋后的棋子是方便的，直接在原有棋盘基础上绘制棋子即可。而走棋前的棋子要变回棋盘需要同时考虑棋盘底色与棋盘网格线：为了同时覆盖选中框，填充棋盘底色时需要绘制“棋子半径+1”的圆。而棋盘网格线也不是简单的填充“十”字，棋盘外围的网格线并不是完整的“十”字，特判即可。代码如下：

```
void movePiece() {
    if (SelectPieceFlag || AIFlag) {
        int LastTouchValid, TouchValid, Distance;
        LastTouchValid = LastPieceValid && Table[LastPieceX][LastPieceY]
== whichTurn;
```

```
TouchValid = PieceValid && Table[PieceX][PieceY] == 0;
Distance = abs>LastPieceX - PieceX) + abs>LastPieceY - PieceY);

// 判断是否可以移动
if (LastTouchValid && TouchValid && (Distance == 1)) {
    // 移动棋子
    Table[PieceX][PieceY] = Table>LastPieceX][LastPieceY];
    Table>LastPieceX][LastPieceY] = 0;

    // 绘制移动后的棋子
    POINT_COLOR = (Table[PieceX][PieceY] == 1) ? WHITE : BLACK;
    LCD_Draw_Circle(ChessBoardPos[PieceX],
ChessBoardPos[PieceY], PIECE_RADIUS, 1);

    // 重新绘制原有位置
    POINT_COLOR = CHESSBOARD_COL;
    LCD_Draw_Circle(ChessBoardPos>LastPieceX],
ChessBoardPos>LastPieceY], PIECE_RADIUS, 1);
    reDrawChessboardLine>LastPieceX, LastPieceY);

    // 置位 AI 输入、移动标志
    AIFlag = 0;
    MovePieceFlag = 1;
    return;
}
}
MovePieceFlag = 0;
}
```

### 2.2.3. 维护棋盘状态

棋子移动完成后，需要更新棋盘的状态，检查是否有吃子，是否有玩家获胜，换边等等。这个部分只需要根据规则检查 **Table** 数组即可完成。为了提升效率有如下一些小剪枝：

- 只有有棋子移动了才回去检测是否需要更新棋盘；
- 吃子检测只对刚刚移动的棋子进行，没必要扫描整个棋盘；
- 只有在双方棋子数量差在 2 及以上时才进行无法移动检查。

同时通过 LCD 屏直接输出回合信息（轮谁下）以及获胜信息，更直观的反馈给玩家。

为了方便多次游玩，还通过中断实现了按下按键（PE6）重置游戏的功能。至此已经可以流畅的进行双人对弈。

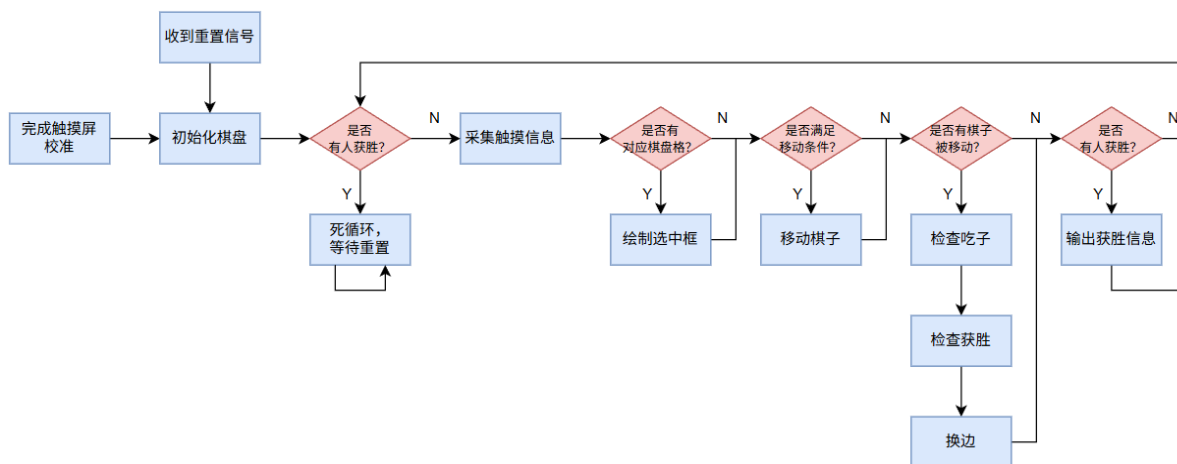


图 3 黑白棋游戏流程图

## PC 机黑白棋部分

与 PC 机对弈时，规定玩家执黑先行，PC 机执白。使用 python 编写对弈 AI。

### 2.2.4. 通信部分

使用 UART 与 PC 机进行通信，仿照 Modbus，以数据帧为单位发送信息。每个数据帧包括走棋信息（两对坐标）与两位 CRC 校验码共 6 个字节信息。特别的，重置信号为“RESET”与两位 CRC 校验码。

走棋信息的合法性由发送方保证，且不实现超时、重传等机制。数据帧的划分通过空闲 3.5 字节（类似 Modbus）为标志。

具体实现时，只需要在每次走棋后向 PC 机发送走棋信息，白棋走棋时使用 PC 机发送的信息即可，其他逻辑可复用双人对弈的实现。

### 2.2.5. AI 部分

下棋 AI 采用基于 Minimax 博弈搜索的 Alpha-Beta 剪枝算法。大致内容为：核心为一个打分函数，打分内容为子力差、机动性、是否能吃子。三者会乘上一个系数来表示重要性。每次 AI 会搜索所有可行走子方案，并调用这个打分函数，选择分数最高的执行。



为了增加 AI 的“考虑能力”，可以设置搜索层数：当 AI 枚举了一个走子方案，不再直接使用打分函数的结果，而是再次枚举对手所有走子方案并打分，选择打分结果最低的（对手总是充分聪明）作为这个走子方案的结果。再在所有结果中选择最佳的。这个过程可多次嵌套，搜索深度越深，AI 越智能。

同时可增加 Alpha-Beta 剪枝，即在搜索过程中直接过滤掉已经劣于当前得到的方案的分支，加快 AI 搜索速度。

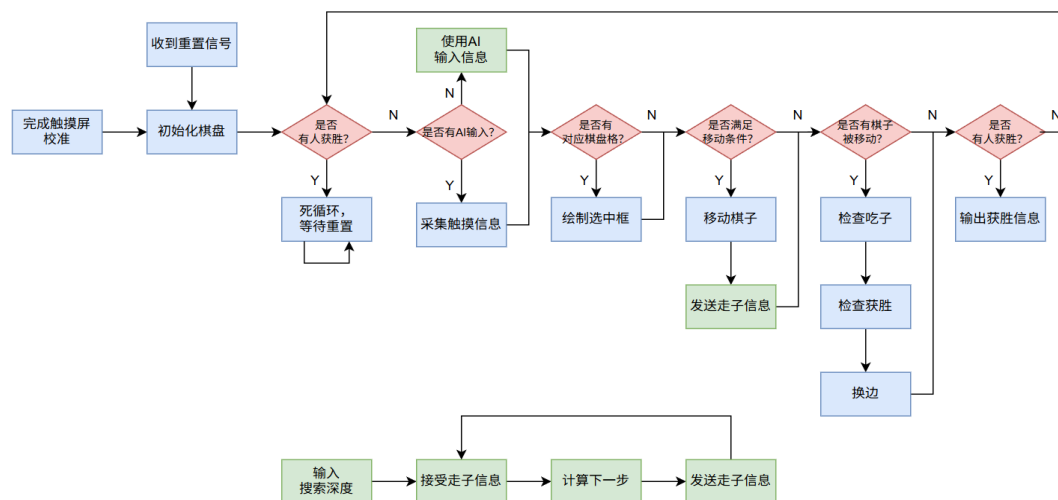
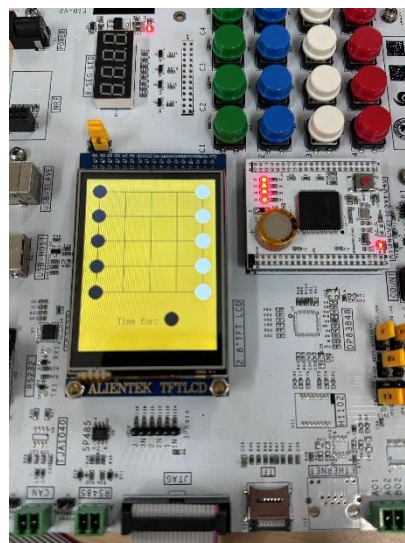


图 4 引入 AI 后的黑白棋游戏流程图

AI 的搜索深度作为输入数据，实现玩家可以动态调整对局难度。

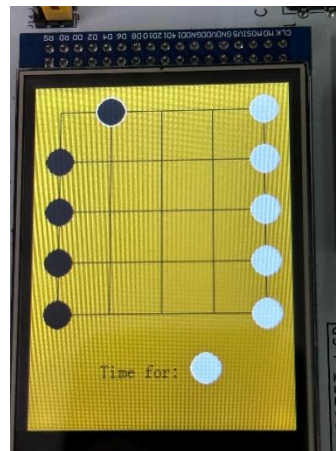
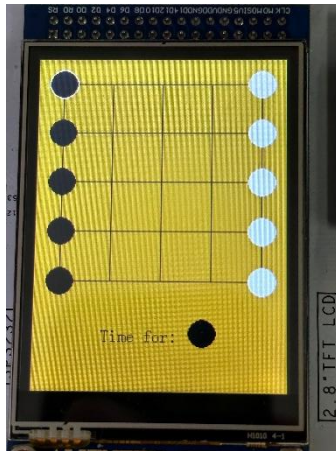
### 3. 实验结果

完成屏幕校准后，LCD 显示棋盘与棋子，准备开始游戏：

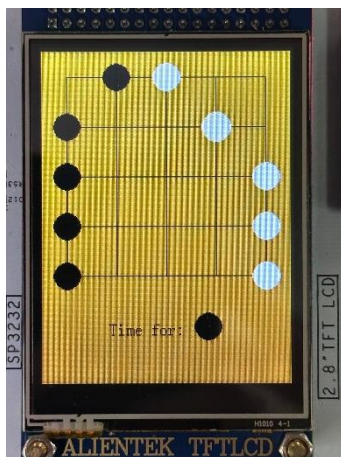




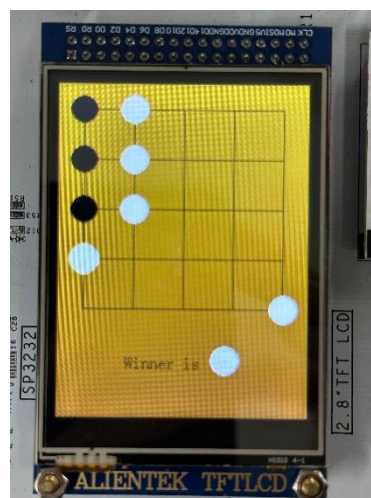
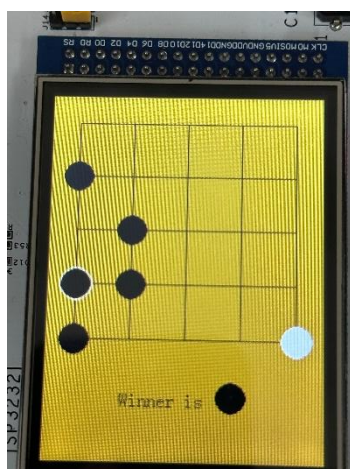
触摸左上方黑棋，可以看到黑棋出现了选中框效果。再次触摸相邻棋格，成功走子：



此时我们未连接 PC，可以移动白字完成人人对弈。简单测试吃子功能：



简单测试两种获胜情况：



启动对弈 AI 后，首先输入搜索层数，此时按下重置游戏按钮，PC 机能收到对应数据帧：

```

(venv) PS C:\Users\JustinWu\Desktop\Embedded-Systems-Course-Project\AI> python .\chessAI.py
请输入 AI 搜索深度 (例如 0 / 1 / 2 / 3 / 4) : 0
AI 搜索深度已设置为 0
等待走子...

=====

收到数据帧: bytearray(b' RESETK\xbe')
CRC 校验通过。数据: bytearray(b' RESET')

游戏重置

B..W

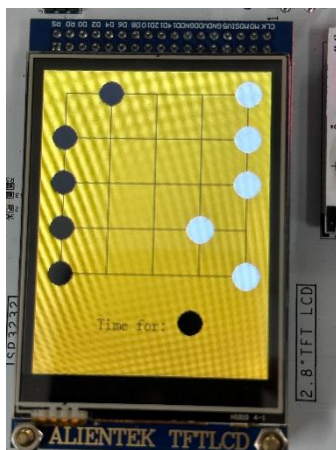
B..W

B..W

B..W

B..W
    
```

此时左上方黑子向右移动一步，可以看到对弈 AI 收到走子信息，计算应对方法后发送回 STM32，STM32 完成走子并显示到 LCD 上：



```

=====

收到数据帧: bytearray(b' \x00\x00\x01\x00\x01\xb4')
CRC 校验通过。数据: bytearray(b' \x00\x00\x01\x00')
Sent frame: b' \x04\x03\x03\x03\xb1\xe5'

BLACK> (0,0) -> (1,0)
WHITE> (4,3) -> (3,3)

. B. . W

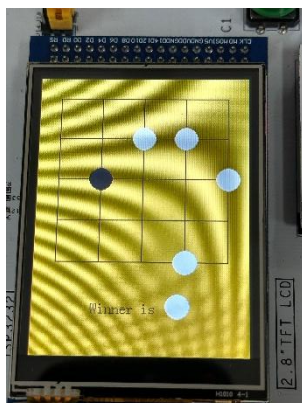
B. . W

B. . W

B. . W

B. . W
    
```

与 AI 对弈一会儿，遗憾告负：



```

=====

收到数据帧: bytearray(b' \x01\x00\x01\x01\x01\x88')
CRC 校验通过。数据: bytearray(b' \x01\x00\x01\x01')
Sent frame: b' \x02\x02\x02\x01\xfc'

BLACK> (1,0) -> (1,1)
WHITE> (2,2) -> (2,1)

.....

.. WW.

. B. . W

.....

... W.
    
```

通过多次尝试，发现即使搜索深度为 0，即只为落子后的局面进行打分，已经难以战胜 AI 了。一方面说明笔者编写的 AI 实力强劲，一方面说明游戏规则可能有修改空间（并非笔者棋力问题）。

按下重置按钮后又可以开始新的一局！

#### 4. 改进空间

虽然目前已经实现了基本的人人、人机对弈功能，但是系统还存在许多需要完善的细节：

每次重新上电都需要从 0 开始校准触摸屏，有时这一过程会显得有些烦琐，可以考虑使用实验板上的 EEPROM 资源，将本次采集的 AD 数据写入，重新上电时读出，从而达到快速校准或者直接跳过校准的效果。

目前 UART 通信过程中没有任何超时、重传机制，发现了 CRC 校验错误也只能手动重置游戏。实际使用时可能会因为落子过快导致错误 PC 机发送的数据帧导致 STM32 与 PC 机的棋盘失去同步。可以仿照 TCP 协议编写完整的收发检验协议，在一方检测到错误或者超时能自动重发数据帧。

LCD 的显示内容均为英文，可以考虑实现中文显示甚至中英文切换，让游戏变得更加用户友好。

#### 5. 心得与体会

通过本次《嵌入式原理》课程的大作业，我对“嵌入式系统”这一概念有了从抽象到具体、从模块到整体的深入认识。作为一名计算机科学与技术专业的大三学生，在以往的学习中，我更多接触的是偏软件方向的内容，例如数据结构、操作系统、编译原理等，而本次实验则让我第一次真正站在“软硬件结合”的视角，完整地设计并实现一个可交互、可扩展的嵌入式系统。

在技术层面，这次实验极大地加深了我对底层硬件工作方式的理解决。无论是通过 GPIO 模拟 8080 总线驱动 LCD，还是通过软件模拟 SPI 协议读取触摸屏数据，亦或是使用外部中断提升系统响应效率，这些内容都让我意识到：嵌入式开发并不仅仅是“把代码写到单片机里”，而是需要在时序、资源、实时性等多方面做权衡。很多在 PC 上“理所当然”的事情，在嵌入式环境下都需要仔细推敲，例如刷新策略、通信可靠性、状态同步等，这种工程层面的约束让我受益匪浅。

在系统设计与抽象能力方面，本次实验对我也是一次很好的锻炼。从 LCD 驱动层、触摸屏输入层，到黑白棋逻辑层、UART 通信层，再到 PC 端 AI 对弈程序，每一部分都需要相对清晰的分层和接口设计。尤其是在复用“人人对弈”逻辑实现“人机对弈”的过程中，我逐渐体会到良好模块划分的重要性。这种思路与我在操作系统课程中学习到的“分层设计”、在软件工程课程中接触到的“高内聚、低耦合”是高度一致的，也让我感受到不同课程之间并非割裂，而是在不同层次上相互呼应。

在与 PC 端 AI 对弈的实现中，将嵌入式系统与 Python 编写的博弈搜索算法结合起来，也让我对“嵌入式 + 上位机 + 算法”的协同模式有了直观认识。这种模式在现实中非常常见，例如智能设备、工业控制、物联网终端等，都往往需要在资源受限的设备端完成采集与控制，而将复杂计算放在性能更强的上位机或服务器上完成。近年来随着人工智能和边缘计算的发展，这类“端一边一云”协同架构也成为技术热点，本次实验在一个小型项目中让我提前体验了这种工程思想。

从个人发展的角度来看，这次课程设计让我更加明确了自身能力结构中的短板与方向。一方面，我具备一定的软件和算法基础，这在编写游戏逻辑和 AI 搜索算法时体现得较为明显；另一方面，我也清楚地认识到自己在硬件细节理解、通信协议健壮性设计等方面仍有提升空间。这种认识对我后续的学习规划具有积极意义，无论是继续深入嵌入式与系统方向，还是将其作为理解底层原理、反哺上层软件设计的基础，都具有长远价值。

总体而言，本次《嵌入式原理》大作业不仅是一项课程任务，更是一次将课堂知识、工程实践与个人兴趣相结合的完整训练过程。它让我真正体会到“系统”二字的含义，也让我对未来在计算机领域中的学习和发展方向有了更加清晰、务实的思考。这种从“能跑起来”到“想做得更好”的转变，是我认为本次实验带给我最重要的收获。