2023 程序设计 II 荣誉课程大作业报告

中国人民大学 李修羽

摘 要

本文主要探究了基于 Qt 图形界面应用设计的不围棋联网对战软件制作过程，对一些重要功能的实现进行了说明，同时对于团队分工过程有一定记录。在联网体系之外，额外探究了不围棋 AI 的算法设计与实现。

目 录

1 团队构成

1.1 闲话

理论上这个大作业一个人也可以做，但是既然都称之为团队大作业了，那就团队做。

1.2 小组分工

笔者负责了项目 UI 的主要设计、棋盘逻辑的维护、文件部分的编写和报告的撰写。
冯友和同学负责了项目框架的建构、初始 bot 的设计、联机部分的编写和项目的美化。
赵培宇同学负责了棋盘的生成、按钮的部分实现和状态的显示。

2 Qt 图形界面应用设计

2.1 UI 设计

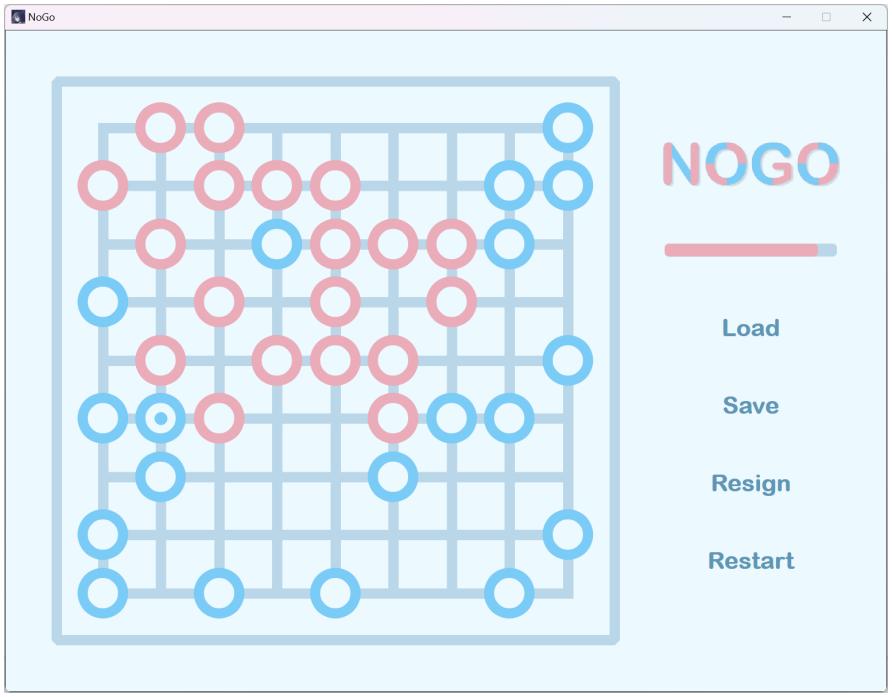

图 2.1.1: demo 示例

本应用 UI 设计[1]采用扁平化的设计风格，搭配上饱满、明亮、可爱且前卫的颜色设计，

[1]部分设计参考：https://github.com/epcm/QtNoGo

亮度统一且冷暖色调搭配，具有舒适的观感。字体使用 Sans-serif 字体 Arial Rounded MT Bold，平衡现代感和亲和感。棋子中心保留背景色，使棋盘整体更为协调。

由于棋盘大小可以自定义，棋盘采用程序绘图而非图片覆盖的形式，可以根据棋盘的大小自适应调整棋盘线条粗细和棋子大小，保持程序页面分辨率相对不变。

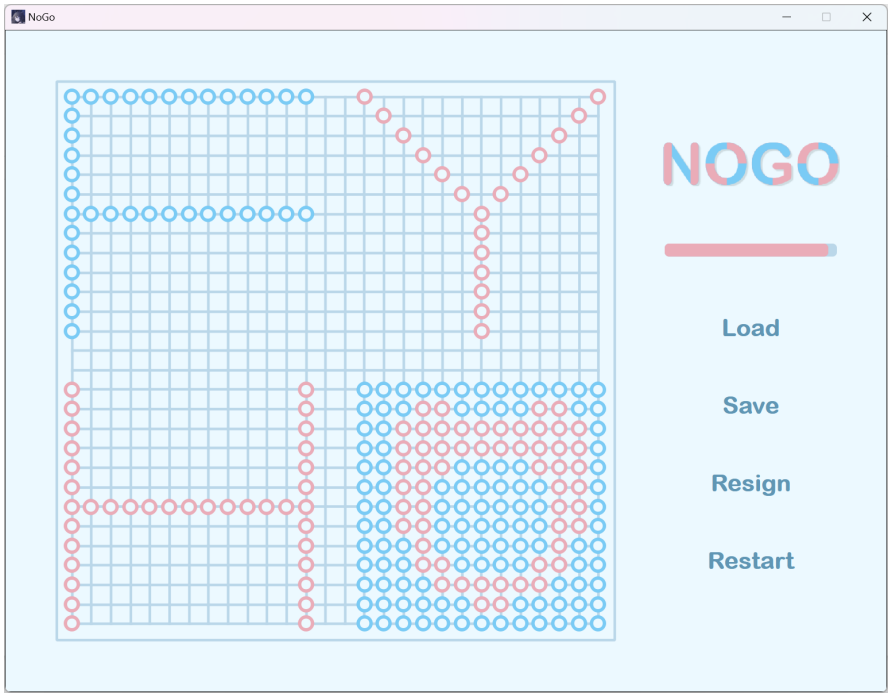

图 2.1.2: 自定义棋盘大小 (28×28)

Logo 以棋子底色为设计基础，增加了颜色块的分割，具有棋类游戏的风格，也维持了项目的主要设计基调。

2.2 基本逻辑

```
|-- Qt-NoGo
    |-- DialogBox
    |   |-- messagebox.cpp
    |   |-- messagebox.h        // 弹出提示或结算信息
    |   |-- optiondialog.cpp
    |   |-- optiondialog.h      // 在联机对局时弹出申请窗口
    |   |-- optiondialog.ui
    |   |-- settingdialog.cpp
    |   |-- settingdialog.h     // 实现图形化更改游戏选项
```

```
|   |-- settingdialog.ui
|-- Object
|   |-- bot.cpp
|   |-- bot.h              // 随机下棋 bot
|   |-- judge.cpp
|   |-- judge.h            // 控制下棋过程逻辑，进程调度，局面判断
|-- Resource.qrc           // 图片配置文件
|-- Widget
|   |-- gamewidget.cpp
|   |-- gamewidget.h       // 维护对局棋盘状态，控制信号槽及文件读写
|   |-- gamewidget.ui
|   |-- startwidget.cpp    // 实现欢迎界面
|   |-- startwidget.h
|   |-- startwidget.ui
|-- img                    // Logo 和 icon 图片
|-- main.cpp
|-- mynogo.pro             // QMake 项目管理文件
|-- mynogo.pro.user
|-- network                // 网络库相关
|   |-- networkdata.cpp
|   |-- networkdata.h
|   |-- networkserver.cpp
|   |-- networkserver.h
|   |-- networksocket.cpp
|   |-- networksocket.h
|-- report                 // 项目报告
```

基本逻辑为 main 连接信号槽后切换到 startwidget，通过 optiondialog 设置后进入 gamewidget，由 gamewidget 初始化 judge 开始游戏。游戏过程由 judge 判断，发现终局后传输到 gamewidget 然后输出 messagebox 信息，至此一局游戏结束。

2.3 棋盘逻辑和结局判断

由于最后 NoGo-Cup 给 AI 的限时是 1s，留给 AI 计算的时间并不充裕，所以本项目采用了高效率的棋盘底层逻辑，以期为 AI 的运行提供尽可能多的时间。

相较于传统的 dfs 判断棋子，笔者创新性地采用了启发式合并的方法来判断棋子的连通性与气数，在 judge.h 中的定义如下：

```
typedef std::pair<int, int> Item;
typedef std::vector<Item> ItemVector;
typedef std::set<Item> LibertySet;

int board[CHESSBOARD_SIZE + 2][CHESSBOARD_SIZE + 2]; // 当前棋盘状态
int chessBelong[CHESSBOARD_SIZE + 2][CHESSBOARD_SIZE + 2]; // 棋子属于的棋子块
int blockVis[(CHESSBOARD_SIZE + 2) * (CHESSBOARD_SIZE + 2)]; // 棋子块至多只能累加一次
int blockCnt; // 棋子块个数

LibertySet blockLiberty[(CHESSBOARD_SIZE + 2) * (CHESSBOARD_SIZE + 2)]; // 气的 Set
ItemVector chessBlock[(CHESSBOARD_SIZE + 2) * (CHESSBOARD_SIZE + 2)]; // 棋子块的编号
std::vector<int>mergedBlock;
```

本程序用 `vector` 存储每一个连通块棋子的位置，用 `set` 存储每一个连通块气的位置。对于一次合并，将两端连通块的 `vector` 和 `set` 分别启发式合并即可。此处采用 `set` 是因为其本身具有判重的特性，可以在合并时对气做出高效率的处理。

```
void Judge::MergeSet(LibertySet &x, LibertySet y)
{
    if(x.size() < y.size()) std::swap(x, y);
    for(Item u : y) x.insert(u);
}
void Judge::MergeBlock(int x, int y) // 启发式合并
{
    if(chessBlock[x].size() < chessBlock[y].size()) std::swap(x, y);
    MergeSet(blockLiberty[x], blockLiberty[y]);
    for(Item u : chessBlock[y])
    {
        chessBlock[x].push_back(u);
        chessBelong[u.first][u.second] = x;
    } // 合并
    chessBlock[y].clear(); // 清空
}
```

相较于 dfs 判断单次 $O(n^2)$ 的复杂度，启发式合并的复杂度可以做到 $O\left(\log^2 n\right)$，且此处的 n^2 仅为理论上界，对于大规模棋盘（例如 20×20）有极高的效率。

对于一次落子操作，如果落子后判负，会弹出此处无法落子的弹窗。同时在一次操作超时后，会直接超时判负。对于现有版本来说，页面实现了认输按钮，玩家在无法落子后可以等到超时判负，也可以认输判负。

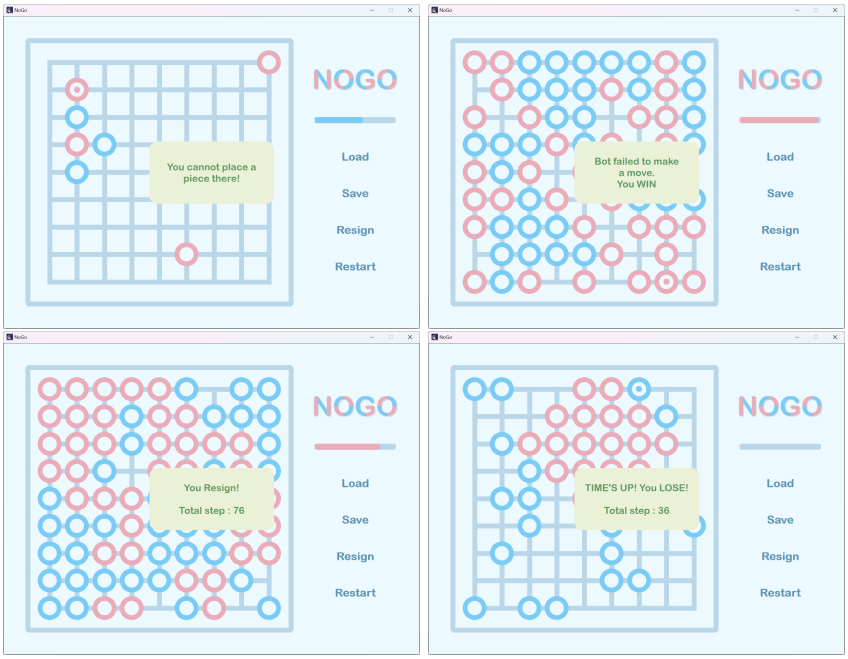

图 2.3.1: 状态判断

2.4 设置

设置部分基于 `optiondialog` 类，使用 Qt 自带的文本选择框和按钮对 `gamewidget` 传入棋盘大小、游戏模式和联机部分的信息，将在线与离线、PVP 与 PVE 利用有限的空间优秀地集成为一体，用按钮切换在线模式与离线模式。

在线模式下，参数采用统一的 9×9 棋盘大小。

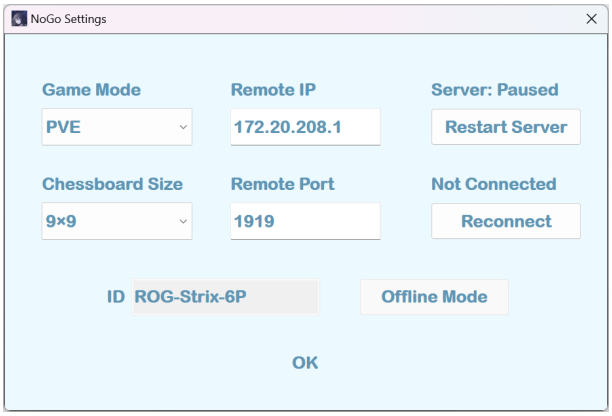

图 2.4.1: 设置页面

2.5 计时器

计时器基于 QProgressBar 自带的 TimeBar，用 CSS 设置美化后用于本项目的计时器设计。计时器采用从右向左线性读条的方法计时，进度条结束后判定为超时。

在终局（胜利、失败、认输）后，计时器会与棋盘本题一起冻结，读取终局存档依然保持冻结。

2.6 读档和存档

项目将棋盘的每一步存入 vector 中，并编码为形如 0/2 0/1 A1 B4 D3 G/W/L 的字符串形式存入 DATA(.dat) 文件中。第一个数字 0 代表执黑、2 代表执白，第二个数字 0 代表 PVP、1 代表 PVE，最后的字符 G/W/L 代表认输/胜利/失败。

文件操作基于 QFile 和 QFileDialog 实现，QFileDialog::getSaveFileName 可以实现打开文件框保存文件，QFileDialog::getOpenFileName 可以实现打开文件框读取文件。文件流使用 QFile::readAll 和 QFile::write，传输类型为 QByteArray 与 char*。

以下是存档部分的实现：

```
// 存档按钮信号
void GameWidget::on_saveButton_clicked()
{
    dataToString(); // 将数据编码到 char* dataStr 中
    QString fileName = QFileDialog::getSaveFileName(nullptr, tr("Save Data"), "", tr("
        DATA (*.dat)"));
    // QFileDialog 读取文件 fileName
    if (fileName.isEmpty())
        return;
    else
    {
        QFile file(fileName);
        if (!file.open(QIODevice::WriteOnly)) // 无法覆写时报错
        {
            QMessageBox::information(nullptr, tr("Unable to open file"),file.
                errorString());
            return;
        }
        file.write(dataStr); // QFile 直接读写文件
        file.close();
    }
}
```

读档后，程序重新初始化棋盘，强制传入存档内的设置参数，顺序模拟所有下棋操作。操作完成后得到的便是 Save 时的游戏状态，读档后 TimeBar 重置为初始时间，如果不是终局仍可以继续下棋，并随时可以执行新的 Save 和 Load。

对于终局情况的 DATA，目前只能复现出终局。

2.7 信息窗口

messagebox 类实现了页面内的弹出信息，下棋界面内的信息包括自杀的濒死检测和游戏终局的判断。

信息窗口会一直显示到检测鼠标单击，而对于终局显示的信息窗口，至少显示 3s 后才能关闭。

2.8 更多功能

有没有可能原来大作业的要求越来越低了，所以我们还没有写更多功能。

没有写更多功能是怎么回事呢？大作业相信大家很熟悉，但是没有写更多功能是怎么回事呢？笔者也不是很清楚，大家也可能感到很惊讶，没有写更多功能是怎么回事呢？但事实就是这样，可以前往 ce-amtic 的博客 看博客。

理论上我们实现了半个附加功能。

哦，应该是 duality314 比较摆烂没写，记得去 push。

~~这是什么？更多功能，写一下。这是什么？更多功能，写一下。这是什么？更多功能，写一下。~~

反正第三阶段都要写，不如再去看一下隔壁队伍写的前后端分离。

3 联机对战设计

3.1 通信协议

联机功能使用基于 QTcpServer，QTcpSocket 二次封装的 NetworkServer，NetworkSocket 类实现，可以通过 TCP 协议进行同一局域网内两个终端点对点的数据交换。

联机传输的数据通过 NetworkData 类进行封装，数据类型如下：

```
enum class OPCODE : int {
    READY_OP = 200000,  // 申请对局
    REJECT_OP,          // 拒绝对局
    MOVE_OP,            // 一方落子
    GIVEUP_OP,          // 一方认输
    TIMEOUT_END_OP,     // 结算，终局条件为一方超时未落子
    GIVEUP_END_OP,      // 结算，终局条件为一方认输
    LEAVE_OP,           // 一方断开连接
    CHAT_OP,            // 进行聊天数据交换
};
```

3.2 联机对战逻辑

进行一场联机对战可以抽象为一下三个步骤：1. 双方建立连接；2. 一方发起对局，另一方确认对局；3. 完成对局并确认胜负，返回第一步结束后的状态。

3.2.1 双方建立连接

首先在设置 UI 中加入 IP、端口、用户昵称的输入框，以获取必要的信息。然后在数据库 `Judge` 类中创建 `(NetworkServer*)server`, `(NetworkSocket*)socket` 对象，在构造函数中初始化，用于建立连接。

在设置 UI 中加入启动服务器 (Restart Server) 与连接服务器 (Reconnect) 的按钮，行为如下：当 Restart Server 按钮被点击时，调用 `QTcpServer::listen(port)` API 来监听用户所输入的端口；当 Reconnect 按钮被点击时，调用 `NetworkSocket::hello(host, port)` API 来尝试与地址为 `host:port` 的主机通信。

调用 `QTcpSocket::waitForConnected()` API 来判断是否连接超时，若超时则提示未连接并禁止用户发起对局，否则提示连接正常并等待下一步。

3.2.2 一方发起对局，另一方确认对局

当 Restart Server 和 Reconnect 按钮中有一者被点击时，视作应用程序进入了联机模式。在联机模式下，点击对局开始按钮并不会立刻跳转到棋盘界面，而是会弹出 `OptionDialog` 类所创建的提示窗口，提示等待连接。其中 `OptionDialog` 类基于 `QDialog` 类二次封装，并进行了一些美化。

此时，提出邀请发发送 `READY_OP` 型数据，接收方识别该数据，并弹出同样的 `OptionDialog` 来确认对局是否成立，返回 `READY_OP` 或 `REJECT_OP`。

当对局成立时，双方同时进入棋盘界面，否则返回第一步结束后的状态。

3.2.3 完成对局并确认胜负

对局成立后执黑先行。对于当前落子一方，每次落子成立时均会向对方发送 `MOVE_OP` 信号，并结束监听鼠标点击。对方接受 `MOVE_OP` 信号后会绘制该棋子，并开始监听己方鼠标点击。

在对局中，双方可以通过界面右下的聊天框进行实时沟通，通过互相发送 `CHAT_OP` 实现。

由于在判断落子成立时禁止了棋手自杀，因此对局结束的条件只有超时和认输两种。

当一方认输，即点击 Resign 按钮后，认输方发送 `GIVE_UP_OP`，胜方在收到该信息后回复 `GIVE_UO_END_OP` 表示请求以“一方认输”为理由结束对局。认输方收到 `GIVE_UO_END_OP`

图 3.2.1: 联机对局界面演示

回复同样的信息以确认。此时联机对战结束，棋盘不能被操作。

当一方超时，胜方的计时器会调用 `GameWidget::playerTimeout_OL()` 槽函数，发送 `TIMEOUT_END_OP` 表示请求以“一方超时”为理由结束对局，负方回复同样的信息以确认。此时联机对战结束，棋盘不能被操作。

当联机对战结束时，对局双方可以分别选择是否返回开始界面。当双方都返回开始界面后，此时程序视作重置到了第一步完成后的状态，可以选择重新开始联机或单人对局。

4 AI 算法设计

由于现在还是第二阶段，所以我们依然采用了非常粗犷的 bot 写法：随机下棋。

对于没有围棋基础的人来说，随机下棋的 bot 分布较好，bot 有获胜的可能。但是对于有基本围棋知识的人来说，只要斜着下棋、构造活眼就可以几乎 100% 战胜这个随机下棋 bot。

4.1 min-max 搜索和 $\alpha - \beta$ 剪枝

别卷了，还没写，你家程设就 2 分。

有没有可能程设半期已经寄了。

5 感谢

感谢孙亚辉老师、潘俊达助教在学习生活上的指导和关心。

感谢中国人民大学图灵实验班提供交流的平台与机会。

感谢冯友和、赵培宇同学组成的团队。

感谢彭文博、李知非同学于 2023/3/2 军理课后请本队吃的 KFC 疯狂星期四，并日常交流四人立直麻将。

感谢北京大学 ghastlcon 同学提供的帮助。