

第二阶段报告

一、分工情况

1. 祝凯：
 - 游戏逻辑：保存对局信息并且实现复现功能、围棋路数设置；
 - 网络：客户端向服务端发起连接、实现发起对局以及协议中的大部分指令；
2. 李雨城：
 - 游戏逻辑：游戏高亮上一步的棋子、优化AI功能；
 - 网络：实现聊天功能；
3. 马博靖：
 - 游戏逻辑：优化倒计时更加界面友好、对局结束显示结算信息；
 - 网络：实现联机再来一局、解决联机时连自己的问题、窗口逻辑；

二、代码框架设计

完善游戏逻辑

1. 结算信息呈现：在倒计时函数中根据此时的playerFlag，加入totalTime_black与totalTime_white的累加，并且在chessOneByPerson函数中根据playerFlag加入totalSteps,totalSteps_white,totalSteps_black的累加，最后在游戏结束时使用QMessageBox弹出游戏总时间、游戏总步数、黑子落棋平均时间、白子落棋平均时间，即为结算信息：

```

void MainWindow::TimerCount()
{
    game->totalTime++;
    if(game->playerFlag == true)
        game->totalTime_black++;
    if(game->playerFlag == false)
        game->totalTime_white++;

    ... ..
}
void MainWindow::chessOneByPerson()
{
    ... ..

    if (clickPosRow != -1 && clickPosCol != -1 && game->gameMapVec[clickPosRow][clickPosCol]
    {
        game->totalSteps++;
        if(game->playerFlag == true)
            game->totalSteps_black++;
        if(game->playerFlag == false)
            game->totalSteps_white++;

        ... ..
    }
}
void MainWindow::ask_keeplogs(QString str)
{
    int res = QMessageBox::question(this, tr("NoGo Result:"), str + " wins!" + "\n Total step

    ... ..
}

```

2. 对局信息的保存与复现：

- 使用vector数组在对局中顺次保存黑白两方落子位置编码，对局结束询问是否保存：

```

#define info pair<char,int>
MainWindow::vector<vector<info>> Logs;//记录对局的数组 0为白棋 1为黑棋
if (game_type != View)//非复现模式中记录行棋编码
    Logs[game->playerFlag].emplace_back(make_pair(clickPosRow - 1 + 'A',clickPosCol));
void MainWindow::ask_keeplogs(QString str) {...}

```

- 增加view复现模式，进入后选择本地存档文件并读取：

```

void MainWindow::choose_logs() {...}

```

- 进入复现模式后点击任意落子处即可触发鼠标释放函数，在view模式下根据记录数组落子，由于存档文件并未显式记录胜负，吃子胜负判断调用阶段一中的函数实现，体现了代码良好的可扩展性（可以偷懒），而超时，认输判断根据“无子可下”实现，例如：

```

if (Logs[game->playerFlag].empty()) { //认输编码'G', 超时编码'T'不会读入, 等效无子
    QString str;
    if (game->playerFlag)
        str = "The white"; //黑色无子白色赢!
    else
        str = "The black"; //白色无子黑色win!
    QMessageBox::StandardButton btnValue = QMessageBox::information (this, "NoGo Resu
    if (btnValue == QMessageBox::Ok) {
        ask_keeplogs(str); //询问是否保存对局记录
        view_lose = true;
    }
}
...

```

3. 高亮上一步:

- 就是在上一步的地方画一个比一般棋子大的绿色圆圈:

```

if(game->totalSteps>0&&i==lastx&&j==lasty){ //高亮上一点
    QLinearGradient gradient(MARGIN + BLOCK_SIZE * j - 1.5 * CHESS_RADIUS, M
    gradient.setColorAt(1, QColor(0, 255, 0));
    gradient.setColorAt(0, QColor(255,0,0));
    painter.setBrush(gradient);
    painter.drawEllipse(MARGIN + BLOCK_SIZE * j - CHESS_RADIUS*1.2,MARGIN + BLO
}

```

4. 关于AI的优化:

- 显示双方的优势。AI选择的依据是使自己能下的地方尽可能多, 对方能下的地方尽可能少, 二者相减即为优势 (如果有两个相关的空位, 它会认为优势加2, 但其实这两个空位都填上就没气了, 只能算一个优势, 尚未考虑):

```

int ai::get_possi(brd &board,int use,int size){ //判断优势
    return ai_calc(board,use,size)/100;
}

```

- 新增提示功能: 在pve模式中, 让AI推荐的位置在棋盘上得到显示, 同时显示不能落子的地方。online_pvp模式中大概算作弊, 就不显示了:

```

bool ai::ai_try(brd &board,int x,int y,int col,int size){ //判断是否可行
    if(board[x][y]!=ai_empty)return false;
    board[x][y]=col;
    bool ret=ai_check(board,size);
    board[x][y]=ai_empty;
    return ret;
}

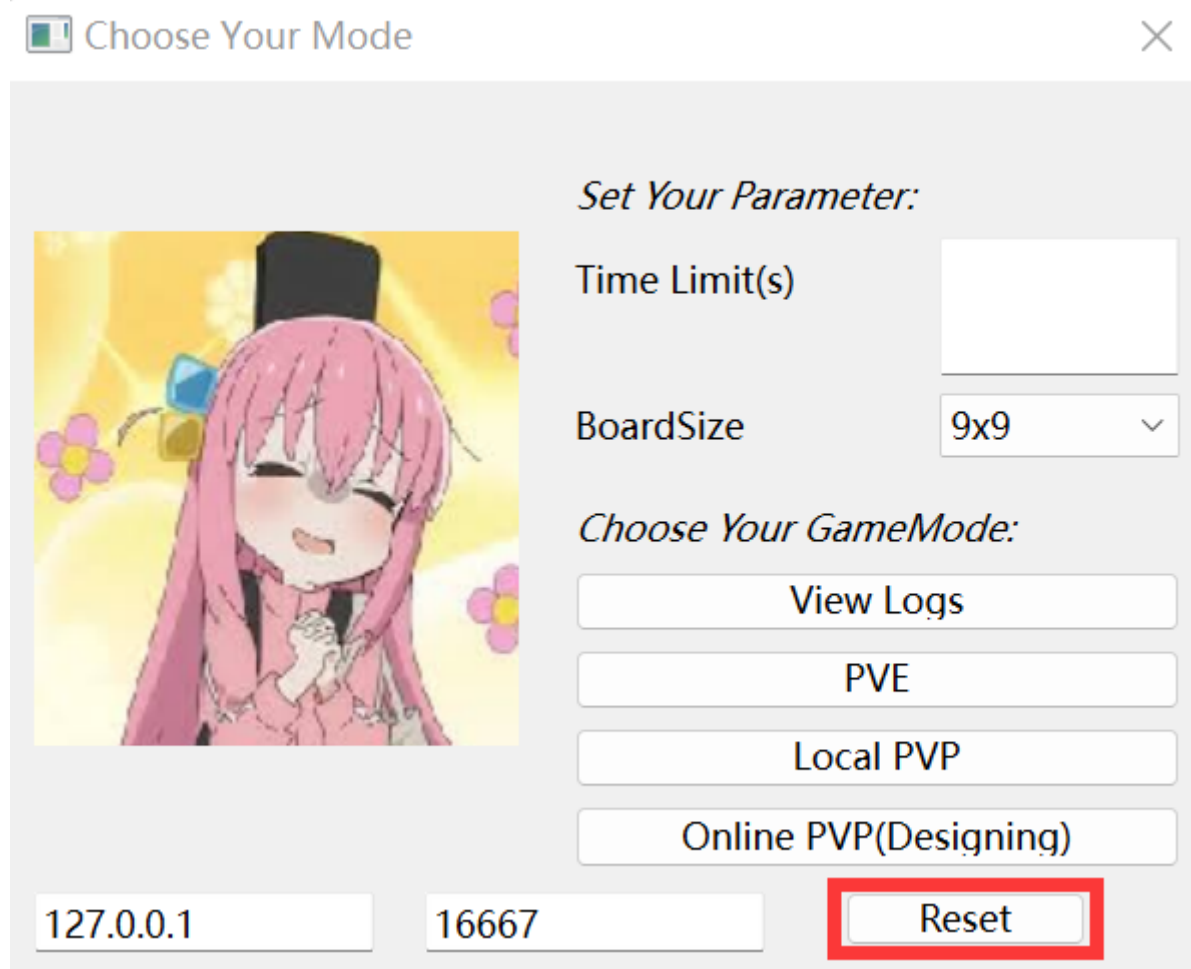
```

5. 默认密码:

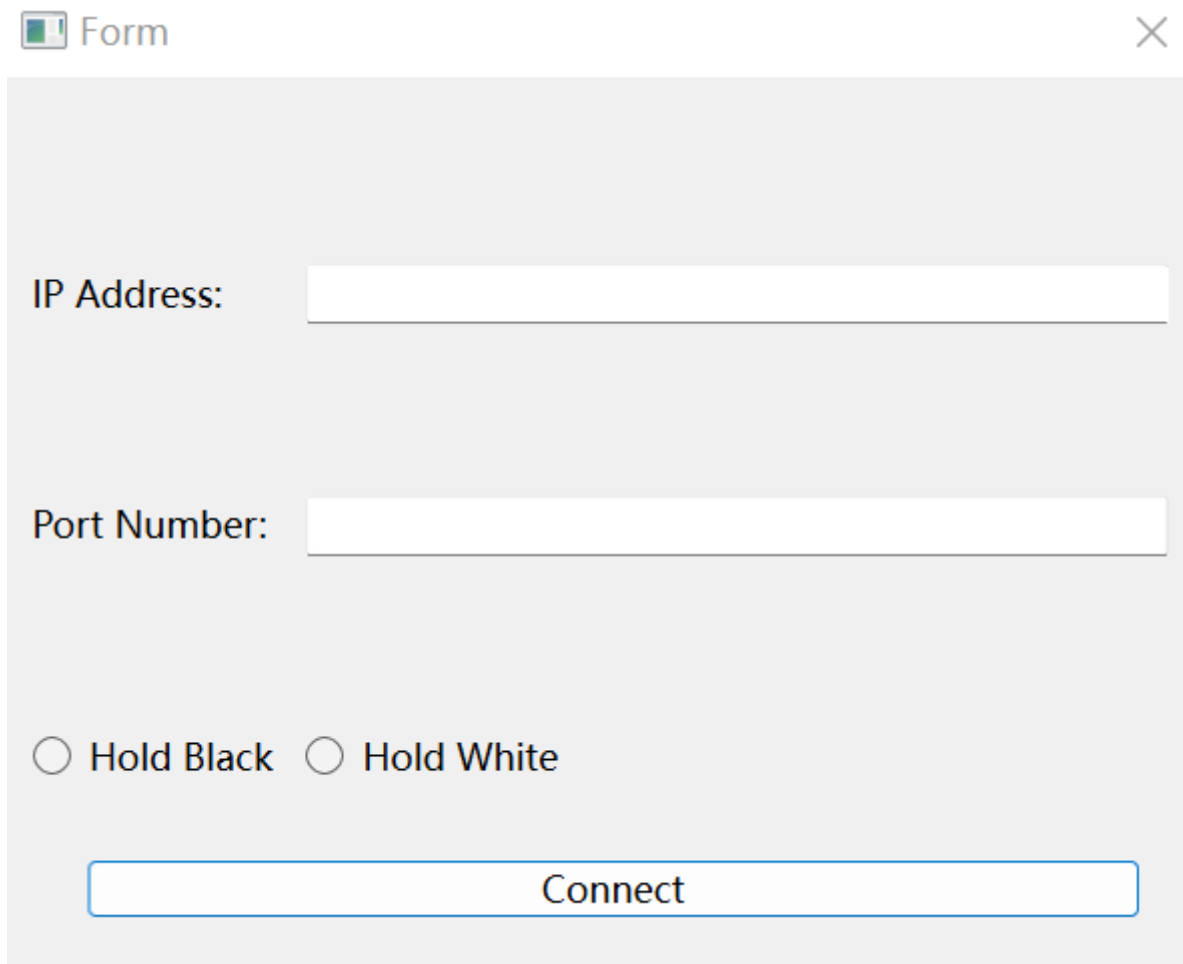
```
//即给账号密码设置初始文本。(最后可能会去掉, 只是方便调试)
ui->lineEdit_UserName->setText("test");
ui->lineEdit_Password->setText("test");//方便调试
```

联网逻辑

1. 在choosemode窗口中加入reset按钮用来打开本地服务端监听(此处127.0.0.1应该设置为硬编码, 但是PORT要设置成软编码, 因为本地回环地址不只一个, 后续或许有别的设置方式, 所以也用lineEdit来承接本地的IP设置了):



2. 增添了connetdialog类, 用来让用户填入客户端IP和PORT, 并且选择所执棋子的颜色, 用来发起对局:



Form

IP Address:

Port Number:

☐ Hold Black ☐ Hold White

3. 在mainwindow类中加入server和socket两个变量，利用一个变量online_agreed记录在onlinePVP模式中是否接受了对局，若接受对局则是服务端；反之是客户端，根据此变量分别处理客户端与服务端的情况：

```
if (game_type == Online) {
    NetworkData give_up(OPCODE::GIVEUP_OP,UserName,"QAQ");
    online_failure = true;
    if (!online_agreed) { //以此判断是服务端还是客户端 以下同理
        socket->send(give_up);
        qDebug() << QDateTime::currentMSecsSinceEpoch() << game->totalSteps << "Client sends {
    }
    else {
        server->send(opponent,give_up);
        qDebug() << QDateTime::currentMSecsSinceEpoch() << game->totalSteps << "Server sends {
    }
}
```

4. 使用connect函数将socket和server的receive信号分别与对应的槽函数连接，使得收到相关信号时能够自动调用处理函数，对接受到的data进行处理：

```

connect(this->server,&NetworkServer::receive,this,&MainWindow::receiveData);
connect(socket, &NetworkSocket::receive, this, &MainWindow::receive_fromServer);

void receiveData(QTcpSocket* client, NetworkData data);
void receive_fromServer(NetworkData data);

```

5. 利用online_player_flag记录联机对战时己方所执的棋子颜色(客户端和服务端有所区别, 通过online_agreed进行区别), 分别对客户端和服务端的online_player_flag进行初始化。服务端和客户端的game->playerFlag是一样的, 如果监测到game->playerFlag和本端的online_player_flag相同则下棋, 不同则不下棋:

```

if (game->gameType == Online && online_player_flag == game->playerFlag) {
    chessOneOnline();
    return;
}
if (game->gameType == Online && online_player_flag != game->playerFlag)//不是用户下棋的时机
    return;

```

6. 胜负判断: 落子胜负判断与第一阶段完全一致, 超时与认输判断与第一阶段基本一致, 只是通过上文提到的bool变量online_player_flag判断超时方与认输方
例如超时判断

```

if (game_type == Online && online_player_flag != game->playerFlag) {
    NetworkData tle(OPCODE::TIMEOUT_END_OP,UserName,"You have exceeded the time limit!");//发
    if (!online_agreed) {
        socket->send(tle);
        qDebug() << QDateTime::currentMSecsSinceEpoch() << game->totalSteps << "Client sends t
    }

    else {
        server->send(opponent,tle);
        qDebug() << QDateTime::currentMSecsSinceEpoch() << game->totalSteps << "Server sends t
    }
}
if (game_type == Online && online_player_flag == game->playerFlag)
    online_failure = true;//己方超时

```

7. 再来一局: 游戏结束时客户端自动给服务端重新发起对局邀请, 若服务端接受了, 则称为“再来一局”, 不接受则不能再来一局(此处还需要整理一下);
8. 聊天功能, 网络方面模仿了已实现的联机功能, 显示的内容chat模仿了dialogchoosemod:

```

void Chat::on_pushButton_btyes_clicked(){
    QString text = this->ui->chatEdit->text();
    NetworkData chat(OPCODE::CHAT_OP,text,"");
    if (!ol_agr)
        socket->send(chat);
    else
        server->send(opponent,chat);
    qDebug() << "Client sends CHAT_OP" << '\n';
}

```

三、遇到的问题

1. 再来一局的逻辑问题：

- 我们希望能够将再来一局作为一个按钮放在结算的窗口中，点击之后将自动向上一局对战过后的对手发起申请，并且能够选择黑白执子。但是由于我们将客户端与服务端区分，导致我们无法实现已经成为服务端的部分向客户端主动发起联机；
- 解决方案（待解决）：需要原客户端先reset一下服务器，然后获取原客户端的IP地址，再由原服务端发起联机，同时注意更换online_agreed的值。主要需要解决的问题是获取客户端的IP，我们将在下一阶段中进行对“再来一局”进行重写；

2. 窗口逻辑问题：

- 对于暂时没有使用的窗口没有进行隐藏，使得窗口频繁弹出，要关闭多次才能关闭整个程序；
- 解决方案：在弹出新一个窗口的时候对上一个窗口进行hide()，在关闭某个主要窗口时（如choosemode窗口），视作关闭整个程序，通过重写choosemode的closeEvent解决此问题；

```

void closeEvent(QCloseEvent *event) override
{
    event->accept();
    QApplication::closeAllWindows();
    QCoreApplication::quit();
}

```