北京大学
PEKING UNIVERSITY

计算概论A五子棋大作业报告

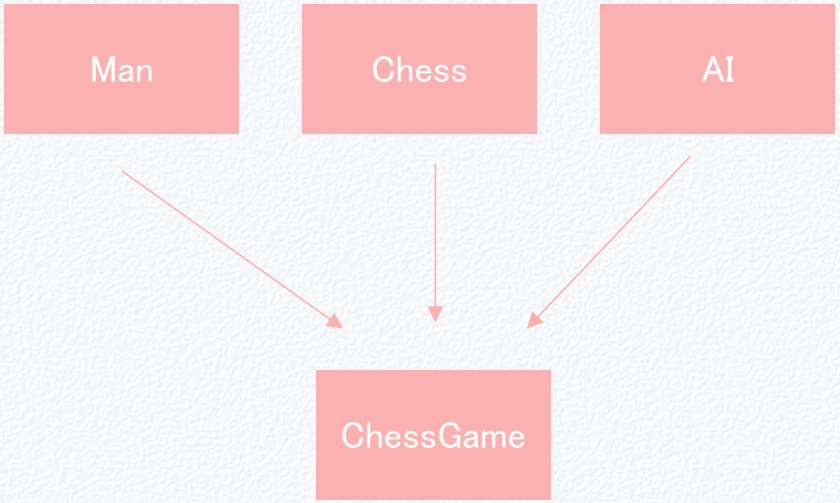
2025年1月14日

01 效果展示

02 代码展示

代码展示

大体想法：
借助c++面向对象设计四个类，分别表示玩家行动、棋盘信息、AI行动、游戏总控制



代码展示

各个类添加接口与数据成员：

```
class Man
{
public:
    void init(Chess *chess);
    void go();

private:
    Chess *chess;
};
```

```
class AI
{
public:
    void init(Chess* chess);
    void go();

private:
    Chess* chess;
    vector<vector<int>> scoreMap;

private:
    void calculateScore();//评分函数
    ChessPos think();//评分确定后，“思考”落子位置
};
```

```
class ChessGame
{
public:
    ChessGame(Man* man, AI* ai, Chess* chess);
    void play();//开始对局

private:
    Man* man;
    AI* ai;
    Chess* chess;
};
```

代码展示

各个类添加接口与数据成员：

```
private:
    IMAGE chessBlackImg;//黑旗棋子
    IMAGE chessWhiteImg;//白棋棋子

    int gradeSize;//棋盘大小（13,15,17,19）
    int margin_x;//棋盘左侧边界
    int margin_y;//棋盘顶部边界
    float chessSize;//棋子的大小

    //表示现在该谁下棋
    bool playerFlag;//true 该黑子走  false 该白字走

    void updateGameMap(ChessPos* pos);//更新棋盘状态

    bool checkWin();//如果胜负已分，返回true
    ChessPos lastPos;//最近落子位置
};
```

```
struct ChessPos {
    int row;
    int col;
    ChessPos(int r = 0, int c = 0) :row(r), col(c) {}
    //row被赋值为r，col被赋值为c
};

typedef enum {//白棋为-1，黑棋为1
    CHESS_WHITE = -1,
    CHESS_BLACK=1
}chess_kind_t;

class Chess
{
public:
    Chess(int gradeSize, int margin_x, int margin_y, float chessSize);
    void init();
    //棋盘的初始化如加载棋盘图片资源、初始化棋盘相关数据

    bool clickBoard(int x, int y, ChessPos* pos);
    // 判断(x,y)位置否是有效点击并把有效点击的位置的行列保存在参数pos中

    void chessDown(ChessPos* pos, chess_kind_t kind);//落子
    int getGradeSize();//获取棋盘大小

    int getChessData(ChessPos* pos);
    int getChessData(int row, int col);// 看指定位置是黑棋/白棋/空白

    bool checkOver();//是否对局结束

    vector<vector<int>> chessMap;
    //存储当期棋局的棋子分布数据 0-空白 1-黑子 -1-白子

    void saveGame(const std::string& filename); // 存档
    void loadGame(const std::string& filename); // 读档
```

代码展示

在ChessGame内
实现游戏控制：

```
//为了便于调用各个类的功能，在ChessGame中添加3个数据成员，并构造函数中初始化这三个数据成员
ChessGame::ChessGame(Man* man, AI* ai, Chess* chess)
{
    this->man = man;
    this->ai = ai;
    this->chess = chess;

    man->init(chess);
    ai->init(chess);
}
```

```
//对局：开始五子棋
void ChessGame::play()
{
    chess->init();

    while (1) {
        if (exitgame) {
            return;
            break;
        }
        //人走
        man->go();
        if (chess->checkOver()) {
            chess->init();
            continue;
        }

        if (playMode == 1) {
            //人机对战，接下来由ai走
            ai->go();
            if (chess->checkOver()) {
                chess->init();
                continue;
            }
        }
        else if (playMode == 2) {
            //双人对战，依然人走
            man->go();
            if (chess->checkOver()) {
                chess->init();
                continue;
            }
        }
    }
}
```

代码展示

在main函数中创建游戏：


```
extern bool exitgame;//在游戏页面内，点击直接退出时
int playMode = 0;//1表示人机对战，2表示双人对战

void drawButton(int x, int y, int w, int h){ ... }
void writeinstruction(int x, int y){ ... }
void writeexit(int x, int y){ ... }
void writeloadfile(int x, int y){ ... }
void writenewgame(int x, int y){ ... }
void writetitle (int x, int y){ ... }
void drawMenu(){ ... }
void drawInstructionPage(){ ... }
```

```
int main(void)
{
    drawMenu();
    bool gotogame = false;//进入游戏界面

    if (gotogame)
    {
        if (playMode == 1) {
            Man man;
            Chess chess(15, 30 + 25, 27 + 25, 31.71);

            AI ai;
            ChessGame game(&man, &ai, &chess);

            game.play();
            return 0;
        }
        else if (playMode == 2) {
            Man man;
            Chess chess(15, 30 + 25, 27 + 25, 31.71);
            AI ai;
            ChessGame game(&man, &ai, &chess);

            game.play();
            return 0;
        }
    }

    closegraph();
    return 0;
}

            gotogame = true;
            break;
        }
    }
}
```

代码展示

玩家操作部分


```
void Man::go()
{
    MOUSEMSG msg;
    ChessPos pos;

    while (1) {
        msg = GetMouseMsg();
        if (msg.uMsg == WM_LBUTTONDOWN ) {
            int mx = msg.x, my = msg.y;

            //点击棋盘落子
            if (chess->clickBoard(mx, my, &pos) ) { ... }
            //直接退出
            if (mx >= 557 && mx <= 718 && my <= 390 && my >= 340) { ... }
            //暂停
            if (mx >= 557 && mx <= 718 && my >= 40 && my <= 90) { ... }
            //继续
            if (mx >= 557 && mx <= 718 && my >= 115 && my <= 165) { ... }
            //存档
            if (mx >= 557 && mx <= 718 && my >= 190 && my <= 240) { ... }
            //读档
            if (mx >= 557 && mx <= 718 && my >= 265 && my <= 315) { ... }
        }
    }
    //未“暂停”，正常落子
    if (!ispaused) { ... }
}
```

代码展示

- 玩家操作——落子：

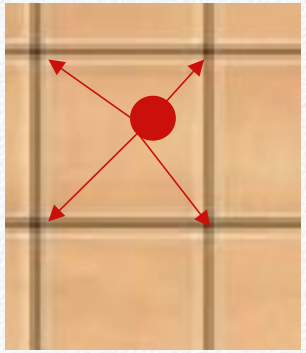
```
//点击棋盘落子
if (chess->clickBoard(mx, my, &pos) ) {
    if (ispaused) {
        mciSendString("play resource/无效点击.mp3", 0, 0, 0);
        cout << "游戏已暂停！请点击“继续”按钮继续游戏。" << endl;
    }
    break;
}
```

```
//未“暂停”，正常落子
if (!ispaused) {
    if(playMode==1)chess->chessDown(&pos, CHESS_BLACK);
    else if (playMode == 2) {
        chess->chessDown(&pos, chessflag);
        chessflag = (chessflag == CHESS_BLACK ? CHESS_WHITE : CHESS_BLACK);
    }
}
```

ClickBoard(mx,my,&pos):判断点击位置是否有效

原理：计算点击位置附近的4个点的位置，然后再计算鼠标点击位置分别到这四个点的距离，如果离某个点的距离小于给定的“阈值”，就认为该点是落子位置。

代码中选取“阈值”为棋子大小的0.4倍



//点击棋盘落子

```
ckBoard(mx, my, &pos) ) {
ed) {
dStr
< "
```

```
//点击棋盘
bool Chess::clickBoard(int x, int y, ChessPos* pos)
{
    if (x > 525 || y > 525)return false;

    int col = (x - margin_x) / chessSize;
    int row = (y - margin_y) / chessSize;
    int leftTopPosX = margin_x + chessSize * col;
    int leftTopPosY = margin_y + chessSize * row;
    int delta = chessSize * 0.4;//点击位置距离棋盘交点处的距离

    int len;
    bool ret = false;//表示是否为有效点击

    do {
        //左上角判断
        len = sqrt((x - leftTopPosX) * (x - leftTopPosX) + (y - leftTopPosY) * (y - leftTopPosY));
        if (len <= delta) {
            pos->row = row;
            pos->col = col;
            if (chessMap[pos->row][pos->col] == 0) {
                ret = true;
            }
            break;
        }

        //右上角判断
        int x2 = leftTopPosX + chessSize;
        int y2 = leftTopPosY;
        len = sqrt((x - x2) * (x - x2) + (y - y2) * (y - y2));
        if (len <= delta) {
            pos->row = row;
            pos->col = col+1;
            if (chessMap[pos->row][pos->col] == 0) {
                ret = true;
            }
            break;
        }

        //左下角判断
        int x3 = leftTopPosX;
        int y3 = leftTopPosY + chessSize;
        len = sqrt((x - x3) * (x - x3) + (y - y3)* (y - y3));
        if (len <= delta) {
            pos->row = row+1;
            pos->col = col;
            if (chessMap[pos->row][pos->col] == 0) {
                ret = true;
            }
            break;
        }

        //右下角判断
        int x4 = leftTopPosX+chessSize;
        int y4 = leftTopPosY + chessSize;
        len = sqrt((x - x4) * (x - x4) + (y - y4) * (y - y4));
        if (len <= delta) {
            pos->row = row + 1;
            pos->col = col+1;
            if (chessMap[pos->row][pos->col] == 0) {
                ret = true;
            }
            break;
        }

    } while (0);

    return ret;
}
```

_BLACK);

代码展示

- 玩家操作——落子：

```
//点击棋盘落子
if (chess->clickBoard(mx, my, &pos) ) {
    if (ispaused) {
        mciSendString("play resource/无效点击.mp3", 0, 0, 0);
        cout << "游戏已暂停！请点击“继续”按钮继续游戏。" << endl;
    }
    break;
}
```

```
//未“暂停”，正常落子
if (!ispaused) {
    if(playMode==1)chess->chessDown(&pos, CHESS_BLACK);
    else if (playMode == 2) {
        chess->chessDown(&pos, chessflag);
        chessflag = (chessflag == CHESS_BLACK ? CHESS_WHITE : CHESS_BLACK);
    }
}
```

chessDown(…):落子
插入棋子图片
并播放落子音效
人机对战时玩家一直下黑棋；
双人模式一直是玩家走棋，
用chessflag指示黑白子的变换

```
//落子
void Chess::chessDown(ChessPos* pos, chess_kind_t kind)
{
    mciSendString("play resource/down7.WAV", 0, 0, 0);

    int x = margin_x + chessSize * pos->col - 0.5 * chessSize;
    int y = margin_y + chessSize * pos->row - 0.5 * chessSize;

    if (kind == CHESS_WHITE) {
        putimagePNG(x, y, &chessWhiteImg);
    }
    else {
        putimagePNG(x, y, &chessBlackImg);
    }
    updateGameMap(pos);
}
```

代码展示

- 玩家操作——直接退出：

Man.cpp中
定义bool exitgame表示是否退出

```
bool exitgame = false;
```

```
//直接退出
if (mx >= 557 && mx <= 718 && my <= 390 && my >= 340) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    exitgame = true;
    return;
}
```

chessGame.cpp中
exitgame为真时退出

```
extern bool exitgame;
```

```
//对局：开始五子棋
void ChessGame::play()
{
    chess->init();

    while (1) {
        if (exitgame) {
            return;
            break;
        }
```

代码展示

- 玩家操作——暂停继续：

定义bool ispaused表示是否暂停
在人走棋时添加判断“是否处于暂停状态”,再继续走棋落子，否则不执行落子部分的代码；
点击继续键后状态恢复,正常走棋。
同时添加文字提示与暂停状态下无效落子的特定音效

```
//点击棋盘落子
if (chess->clickBoard(mx, my, &pos) ) {
    if (ispaused) {
        mciSendString("play resource/无效点击.mp3", 0, 0, 0);
        cout << "游戏已暂停！请点击“继续”按钮继续游戏。" << endl;
    }
    break;
}
```

```
//未“暂停”，正常落子
if (!ispaused) {
    if(playMode==1)chess->chessDown(&pos, CHESS_BLACK);
    else if (playMode == 2) {
        chess->chessDown(&pos, chessflag);
        chessflag = (chessflag == CHESS_BLACK ? CHESS_WHITE : CHESS_BLACK);
    }
}
```

```
//暂停
if (mx >= 557 && mx <= 718 && my >= 40 && my <= 90) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    cout << "游戏暂停！" << endl;
    ispaused = true;
}
//继续
if (mx >= 557 && mx <= 718 && my >= 115 && my <= 165) {
    ispaused = false;
    mciSendString("play resource/select.mp3", 0, 0, 0);
    cout << "欢迎回到游戏，现在请继续对局吧！" << endl;
}
```

代码展示

- 玩家操作——存档读档：

借助#include<fstream>实现

人机对战情况下为例：存档

以写的模式文件game_save.txt，遍历chessMap二维数组；

对每个位置将棋盘上的内容chessMap[i][j]写入文件；

空格分隔行尾换行；

输出文字提示存档成功。

```
//存档
if (mx >= 557 && mx <= 718 && my >= 190 && my <= 240) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    if (playMode == 1) {
        chess->saveGame("resource/game_save.txt");
    }
    else if (playMode == 2) {
        chess->saveGame("resource/game_save_pvp.txt");
    }
}
```

```
void Chess::saveGame(const std::string& filename) {
    std::ofstream outFile(filename);
    if (outFile.is_open()) {
        for (int i = 0; i < gradeSize; i++) {
            for (int j = 0; j < gradeSize; j++) {
                outFile << chessMap[i][j];
                if (j < gradeSize - 1) outFile << " ";
            }
            outFile << std::endl;
        }
        outFile.close();
        std::cout << "Game saved successfully!存档成功！" << std::endl;
    }
    else {
        std::cerr << "Failed to save game!读档失败" << std::endl;
    }
}
```

代码展示

- 玩家操作——存档读档：

借助#include<fstream>实现

人机对战情况下为例：读档

打开文件game_save.txt;

读取每行文件内容，每行都保存为一个字符串line，并用stringstream将每行含空格的字符串拆分，获取每个位置 棋子信息；

转换为整型存储到chessMap数组中

```
//读档
if (mx >= 557 && mx <= 718 && my >= 265 && my <= 315) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    if (playMode == 1) {
        chess->loadGame("resource/game_save.txt");
    }
    else if (playMode == 2) {
        chess->loadGame("resource/game_save_pvp.txt");
    }
}
```

```
void Chess::loadGame(const std::string& filename) {
    std::ifstream inFile(filename);
    if (inFile.is_open()) {
        std::string line;
        for (int i = 0; i < gradeSize; i++) {
            std::getline(inFile, line);
            std::stringstream ss(line);
            std::string cell;
            for (int j = 0; j < gradeSize; j++) {
                std::getline(ss, cell, ' ');
                chessMap[i][j] = std::stoi(cell); // 将字符串转换为整数
            }
        }
        inFile.close();
        std::cout << "Game loaded successfully!读档成功！" << std::endl;
```

代码展示

- 玩家操作——存档读档：

借助#include<fstream>实现

人机对战情况下为例：读档

最后更新UI界面

```
//读档
if (mx >= 557 && mx <= 718 && my >= 265 && my <= 315) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    if (playMode == 1) {
        chess->loadGame("resource/game_save.txt");
    }
    else if (playMode == 2) {
        chess->loadGame("resource/game_save_pvp.txt");
    }
}
```

```
    // 更新 UI 显示当前棋局
    for (int i = 0; i < 15; i++) {
        for (int j = 0; j < 15; j++) {
            if (chessMap[i][j] == CHESS_BLACK) {
                putimagePNG(margin_x + j * chessSize - 0.5 * chessSize,
                    margin_y + i * chessSize - 0.5 * chessSize, &chessBlackImg); // 绘制黑子
            }
            else if (chessMap[i][j] == CHESS_WHITE) {
                putimagePNG(margin_x + j * chessSize - 0.5 * chessSize,
                    margin_y + i * chessSize - 0.5 * chessSize, &chessWhiteImg); // 绘制白子
            }
        }
    }

}
else {
    std::cerr << "Failed to load game!" << std::endl;
}
```

代码展示

- 玩家操作——存档读档：

借助#include<fstream>实现

✓ 不同对局模式下存入、读取不同文件，支持分别保留不同的存档

```
//存档
if (mx >= 557 && mx <= 718 && my >= 190 && my <= 240) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    if (playMode == 1) {
        chess->saveGame("resource/game_save.txt");
    }
    else if (playMode == 2) {
        chess->saveGame("resource/game_save_pvp.txt");
    }
}
```

```
//读档
if (mx >= 557 && mx <= 718 && my >= 265 && my <= 315) {
    mciSendString("play resource/select.mp3", 0, 0, 0);
    if (playMode == 1) {
        chess->loadGame("resource/game_save.txt");
    }
    else if (playMode == 2) {
        chess->loadGame("resource/game_save_pvp.txt");
    }
}
```

代码展示

横竖斜四种大情况，每种情况都根据当前落子往后遍历5个棋子，有一种符合就算赢

- 胜负判定

```
bool Chess::checkWin()
{
    // 水平方向
    int row = lastPos.row;
    int col = lastPos.col;

    for (int i = 0; i < 5; i++)
    {
        // 往左5个，往右匹配4个子，20种情况
        if (col - i >= 0 &&
            col - i + 4 < gradeSize &&
            chessMap[row][col - i] == chessMap[row][col - i + 1] &&
            chessMap[row][col - i] == chessMap[row][col - i + 2] &&
            chessMap[row][col - i] == chessMap[row][col - i + 3] &&
            chessMap[row][col - i] == chessMap[row][col - i + 4])
            return true;
    }

    // 竖直方向(上下延伸4个)
    for (int i = 0; i < 5; i++)
    {
        if (row - i >= 0 &&
            row - i + 4 < gradeSize &&
            chessMap[row - i][col] == chessMap[row - i + 1][col] &&
            chessMap[row - i][col] == chessMap[row - i + 2][col] &&
            chessMap[row - i][col] == chessMap[row - i + 3][col] &&
            chessMap[row - i][col] == chessMap[row - i + 4][col])
            return true;
    }

    // “/”方向
    for (int i = 0; i < 5; i++)
    {
        if (row + i < gradeSize &&
            row + i - 4 >= 0 &&
            col - i >= 0 &&
            col - i + 4 < gradeSize &&
            // 第[row+i]行，第[col-i]的棋子，与右上方连续4个棋子都相同
            chessMap[row + i][col - i] == chessMap[row + i - 1][col - i + 1] &&
            chessMap[row + i][col - i] == chessMap[row + i - 2][col - i + 2] &&
            chessMap[row + i][col - i] == chessMap[row + i - 3][col - i + 3] &&
            chessMap[row + i][col - i] == chessMap[row + i - 4][col - i + 4])
            return true;
    }

    // “\“ 方向
    for (int i = 0; i < 5; i++)
    {
        // 第[row+i]行，第[col-i]的棋子，与右下方连续4个棋子都相同
        if (row - i >= 0 &&
            row - i + 4 < gradeSize &&
            col - i >= 0 &&
            col - i + 4 < gradeSize &&
            chessMap[row - i][col - i] == chessMap[row - i + 1][col - i + 1] &&
            chessMap[row - i][col - i] == chessMap[row - i + 2][col - i + 2] &&
            chessMap[row - i][col - i] == chessMap[row - i + 3][col - i + 3] &&
            chessMap[row - i][col - i] == chessMap[row - i + 4][col - i + 4])
            return true;
    }
    return false;
```

03 算法解释——估价函数

AI算法

AI落子的算法思路：

- 对棋盘的所有可能落子点，做评分计算，然后选择一个评分最高的点落子。
- 对于每一个可能的落子点，从该点周围的八个方向，分别计算，确定出每个方向已经有几颗连续的棋子。
- 同时考虑双方：计算玩家如果在这个位置落子会有多大的价值、AI如果在这个位置落子有大大价值，进行对比。

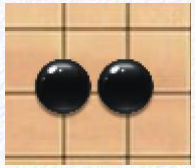
```
//AI思考落子位置
ChessPos AI::think()
{
    // 计算评分
    calculateScore();

    // 从评分中找出最大分数的位置
    int maxScore = 0;
    //std::vector<std::pair<int, int>> maxPoints;
    vector<ChessPos> maxPoints;
    int k = 0;

    int size = chess->getGradeSize();
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++)
        {
            // 前提是这个坐标是空的
            if (chess->getChessData(row, col) == 0) {
                if (scoreMap[row][col] > maxScore)          // 找最大的数和坐标
                {
                    maxScore = scoreMap[row][col];
                    maxPoints.clear();
                    maxPoints.push_back(ChessPos(row, col));
                }
                else if (scoreMap[row][col] == maxScore) {   // 如果有多个最大的数，都存起来
                    maxPoints.push_back(ChessPos(row, col));
                }
            }
        }
    }

    // 随机落子，如果有多个点的话
    int index = rand() % maxPoints.size();
    return maxPoints[index];
}
```

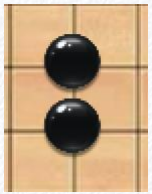
AI算法

calculateScore() 评分

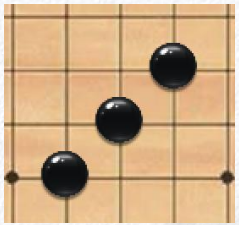
考虑棋型：

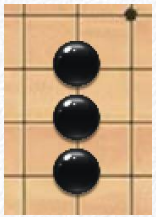
连二



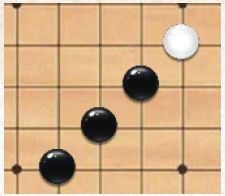


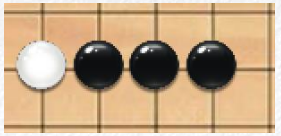
活三



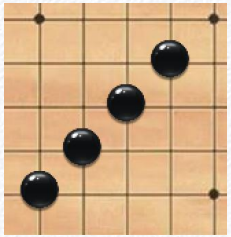


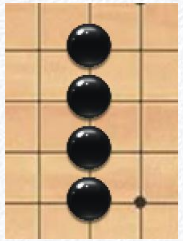
死三



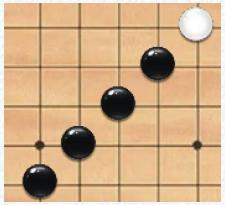


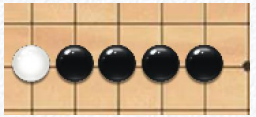
活四





死四





连五

AI算法

calculateScore() 评分

评分表（网上资料）

棋型	黑子	白字
连二	10	10
死三	30	25
活三	40	50
死四	60	55
活四	200	10000
连五	20000	30000

（改进：只关注局部落子形成的棋型，对全局的考虑偏少；特殊棋型如双死四）

AI算法

calculateScore() 评分

代码实现

Step1：始化评分数组scoreMap[i][j]，遍历空位（chess->getChessData(row,col)==0）

Step2：两层嵌套循环实现八个方向遍历（实际为四个方向+正反）

Step3：向每个方向延伸四个子，记录有多少连续的子(personNum,botNum)

Step4：根据评分表，将（row,col）位置的评分计入scoreMap[row][col]

使用i进行延伸

AI算法

calculateScore() 评分

代码实现——黑棋评分

- 遍历棋盘的每个空位

```
//AI算法：评估位置分数
void AI::calculateScore()
{
    // 统计玩家或者AI连成的子
    int personNum = 0;  // 玩家黑棋有多少个连续的子
    int botNum = 0;     // AI白棋有多少个连续的子
    int emptyNum = 0;   // 该方向空白位的个数

    // 清空评分数组
    for (int i = 0; i < scoreMap.size(); i++) {
        for (int j = 0; j < scoreMap[i].size(); j++) {
            scoreMap[i][j] = 0;
        }
    }

    int size = chess->getGradeSize();
    for (int row = 0; row < size; row++)
        for (int col = 0; col < size; col++)
        {
            if (chess->getChessData(row, col) != 0) continue;//落过子的跳过
```

- 四个方向、正反方向检查（使用i延伸）

```
for (int y = -1; y <= 0; y++) {
    for (int x = -1; x <= 1; x++)
    {
        if (y == 0 && x == 0) continue;// 原坐标不算
        if (y == 0 && x != 1) continue; //正反向判断，只需四个方向

        personNum = 0;
        botNum = 0;
        emptyNum = 0;

        // 每个方向延伸4个子
        // 黑棋评分--正向判断
        for (int i = 1; i <= 4; i++)
        {
            int curRow = row + i * y;//往上or下几行数
            int curCol = col + i * x;//往左or右几列数
            if (curRow >= 0 && curRow < size &&
                curCol >= 0 && curCol < size &&
                chess->getChessData(curRow, curCol) == 1) // 真人玩家的子
            {
                personNum++;
            }
            else if (curRow >= 0 && curRow < size &&
                curCol >= 0 && curCol < size &&
                chess->getChessData(curRow, curCol) == 0) // 空白位
            {
                emptyNum++;
                break;
            }
            else            // 出边界
                break;
```

使用i进行延伸

AI算法

calculateScore() 评分

代码实现——黑棋评分

- 遍历棋盘的每个空位

```
//AI算法：评估位置分数
void AI::calculateScore()
{
    // 统计玩家或者AI连成的子
    int personNum = 0;  // 玩家黑棋有多少个连续的子
    int botNum = 0;     // AI白棋有多少个连续的子
    int emptyNum = 0;   // 该方向空白位的个数

    // 清空评分数组
    for (int i = 0; i < scoreMap.size(); i++) {
        for (int j = 0; j < scoreMap[i].size(); j++) {
            scoreMap[i][j] = 0;
        }
    }

    int size = chess->getGradeSize();
    for (int row = 0; row < size; row++)
        for (int col = 0; col < size; col++)
        {
            if (chess->getChessData(row, col) != 0) continue;//落过子的跳过
```

- 八个方向、正反方向检查（使用i延伸）

```
//黑棋评分--反向判断
for (int i = 1; i <= 4; i++)
{
    int curRow = row - i * y;
    int curCol = col - i * x;
    if (curRow >= 0 && curRow < size &&
        curCol >= 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == 1) // 真人玩家的子
    {
        personNum++;
    }
    else if (curRow >= 0 && curRow < size &&
        curCol >= 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == 0) // 空白位
    {
        emptyNum++;
        break;
    }
    else  break;
}

if (personNum == 1) //连2
    scoreMap[row][col] += 10;
else if (personNum == 2)  //连3
{
    if (emptyNum == 1) //死3
        scoreMap[row][col] += 30;
    else if (emptyNum == 2) //活3
        scoreMap[row][col] += 40;
}
else if (personNum == 3) //连4
{
    if (emptyNum == 1) //死4
        scoreMap[row][col] += 60;
    else if (emptyNum == 2) //活4
        scoreMap[row][col] += 200;
}
else if (personNum == 4) //连5
    scoreMap[row][col] += 20000;
```

AI算法

calculateScore() 评分

代码实现——白棋评分

```
// 对白棋评分
emptyNum = 0;
for (int i = 1; i <= 4; i++)
{
    int curRow = row + i * y;
    int curCol = col + i * x;
    if (curRow > 0 && curRow < size &&
        curCol > 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == -1)
    {
        botNum++;
    }
    else if (curRow > 0 && curRow < size &&
        curCol > 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == 0) // 空白位
    {
        emptyNum++;
        break;
    }
    else            // 出边界
        break;
}

for (int i = 1; i <= 4; i++)
{
    int curRow = row - i * y;
    int curCol = col - i * x;
    if (curRow > 0 && curRow < size &&
        curCol > 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == -1)
    {
        botNum++;
    }
    else if (curRow > 0 && curRow < size &&
        curCol > 0 && curCol < size &&
        chess->getChessData(curRow, curCol) == 0) // 空白位
    {
        emptyNum++;
        break;
    }
    else            // 出边界
        break;
}
```

```
if (botNum == 0)                        // 普通下子
    scoreMap[row][col] += 5;
else if (botNum == 1)                   // 活二
    scoreMap[row][col] += 10;
else if (botNum == 2)
{
    if (emptyNum == 1)                  // 死三
        scoreMap[row][col] += 25;
    else if (emptyNum == 2)
        scoreMap[row][col] += 50;  // 活三
}
else if (botNum == 3)
{
    if (emptyNum == 1)                  // 死四
        scoreMap[row][col] += 55;
    else if (emptyNum == 2)
        scoreMap[row][col] += 10000; // 活四
}
else if (botNum >= 4)
    scoreMap[row][col] += 30000;    // 活五，最高优先级
```

04 功能展示与总结

功能展示

- 禁手问题

三三、四四、长连

```
//禁手判断
// 判断水平方向是否为活三
bool Chess::isLiveThreeHorizontal(int row, int col){ ... }
// 判断垂直方向是否为活三
bool Chess::isLiveThreeVertical(int row, int col){ ... }
// 判断斜向（从左上角到右下角）是否为活三
bool Chess::isLiveThreeDiagonal1(int row, int col){ ... }
// 判断斜向（从右上角到左下角）是否为活三
bool Chess::isLiveThreeDiagonal2(int row, int col){ ... }
// 判断是否为三三禁手
bool Chess::isDoubleThree(int row, int col){ ... }

// 判断是否为长连禁手
bool Chess::isLongLine(int row, int col){ ... }

// 判断水平方向是否为活四
bool Chess::isLiveFourHorizontal(int row, int col){ ... }
// 判断垂直方向是否为活四
bool Chess::isLiveFourVertical(int row, int col){ ... }
// 判断斜向（从左上角到右下角）是否为活四
bool Chess::isLiveFourDiagonal1(int row, int col){ ... }
// 判断斜向（从右上角到左下角）是否为活四
bool Chess::isLiveFourDiagonal2(int row, int col){ ... }
// 判断是否为四四禁手
bool Chess::isDoubleFour(int row, int col){ ... }
```

代码实现

用bool valid指示是否因触犯禁手规则而无法落子

假设黑棋落子后遍历各个方向，统计活三/活四的个数、是否长连

```
//落子
void Chess::chessDown(ChessPos* pos, chess_kind_t kind)
{
    if (chessflag == CHESS_BLACK) {
        if (isDoubleThree(pos->row, pos->col)) {// 三三禁手
            cout << "三三禁手，不能落子" << endl;
            mciSendString("play resource/无效点击.mp3", 0, 0, 0);
            valid = false;
            return;
        }
        if (isLongLine(pos->row, pos->col)) {//长连禁手
            cout << "长连禁手，不能落子" << endl;
            mciSendString("play resource/无效点击.mp3", 0, 0, 0);
            valid = false;
            return;
        }
        if (isDoubleFour(pos->row, pos->col)) {//四四禁手
            cout << "四四禁手，不能落子" << endl;
            mciSendString("play resource/无效点击.mp3", 0, 0, 0);
            valid = false;
            return;
        }

    valid = true;
    mciSendString("play resource/down7.WAV", 0, 0, 0);

    int x = margin_x + chessSize * pos->col - 0.5 * chessSize;
    int y = margin_y + chessSize * pos->row - 0.5 * chessSize;

    if (kind == CHESS_WHITE) {
        putimagePNG(x, y, &chessWhiteImg);
    }
    else {
        putimagePNG(x, y, &chessBlackImg);
    }
    updateGameMap(pos);
}
```

功能展示

- GUI界面

初始菜单页面UI设计：

支持查看左上角说明书、选择游戏模式（人机、双人）；

有“合法”退出程序的按键渠道

功能展示

- GUI界面

游戏页面UI设计：

内含基本功能——中途停止并继续、复盘，并提供直接退出按键

功能展示

- GUI界面

游戏页面UI设计：

对局结束时

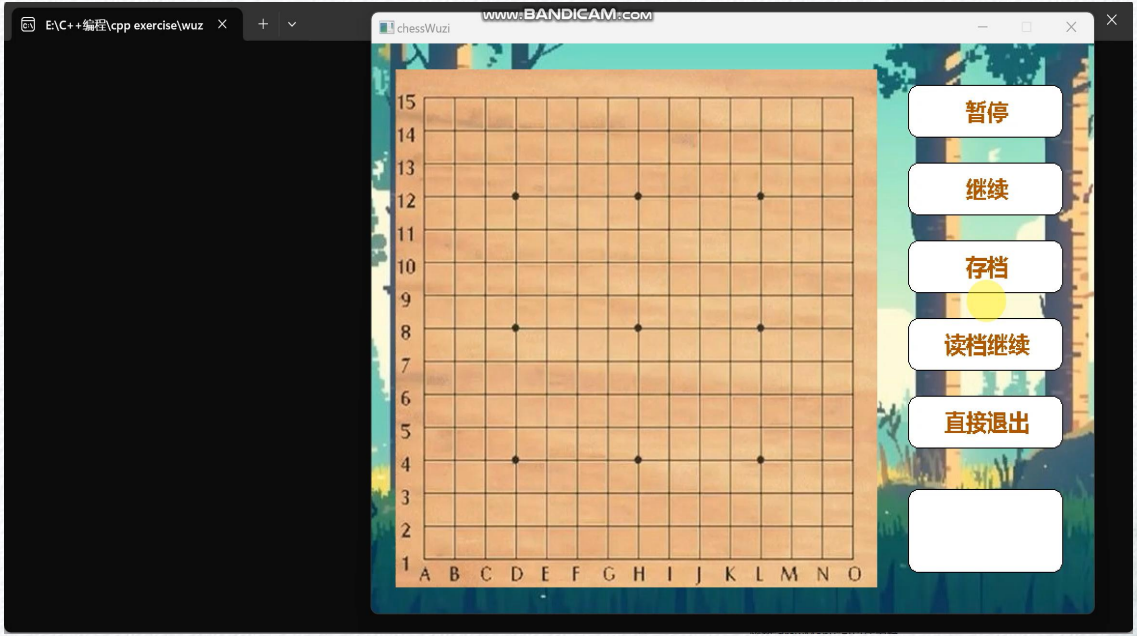
右下角为胜利播报显示板

功能展示

- 用户友好性设计

音效提示：正常点击落子、点击按钮、暂停下的无效点击

文字提示：暂停、继续、存档、读档的信息播报



总结

基础功能：

- 菜单、人机对战、落子、中途停止、复盘（文件实现读档存档）
- Easyx，GUI图形界面

算法：估价函数

补充功能：双人对战、BGM、不同模式下分别存档入不同文件

北京大学
PEKING UNIVERSITY

感谢倾听

2025年1月14日