**同化棋 实验报告**

本项目用控制台应用实现，从功能实现上看，大致分为界面与交互，菜单与功能实现，计算机ai几大部分。下面将展开介绍。

**整体思路**：

使用二维数组int map[7][7]储存盘面，以正负1分别表示玩家与电脑（方便后续程序，在己方数字前加负号就能表示对方棋子），0代表空位。初始时，在四角放置棋子。程序开始直接由主函数main进入菜单界面，提供“新游戏”“读盘”“存盘”“退出”功能，在下棋过程中也可以随时输入数字0调出菜单界面以及继续游戏。开始游戏后，玩家从键盘输入要移动棋子和目标位置（要求坐标的两数之间用一个空格隔开），与电脑（使用minmax算法，并用alphabeta剪枝和预先一步贪心排序进行优化，搜索5步）轮流落子，玩家可输入数字9悔棋，每次落子后判断是否结束，结束则判断胜负并退出游戏。

**界面**

程序使用二维数组map[7][7]储存盘面，以正负1分别表示玩家与电脑（方便后续程序，在己方数字前加负号就能表示对方棋子），0代表空位。由print（map）函数负责输出盘面，整体使用拷贝自word文档的制表符，上方显示双方棋子数，并添设了横纵坐标方便判断位置。示例：

cout << " ━━━━━━━━━━━━━━━━━━━━━━━━━ " << endl;

cout << " 玩家: " << cnt1 << " VS 电脑: " << cnt2 << endl;

cout << " ━━━━━━━━━━━━━━━━━━━━━━━━━ " << endl;

cout << " 1 2 3 4 5 6 7 " << endl;

cout << " ┏ ━ ┳ ━ ┳ ━ ┳ ━ ┳ ━ ┳ ━ ┳ ━ ┓" << endl;

cout << " 1 ┃";

for (int i = 0; i < 7; i++)

if (map[0][i] == 1) cout << " ●┃";

else if (map[0][i] == -1) cout << " ○┃";

else cout << " ┃";

主菜单也同样采用此设计 ：

cout << " ┏━━━━━━━━━━━━━━━┓ " << endl;

cout << " ┃ 欢迎来玩同化棋┃" << endl;

cout << " ┃ 1.新开局 ┃" << endl;

cout << " ┃ 2.读盘 ┃" << endl;

cout << " ┃ 3.存盘 ┃" << endl;

cout << " ┃ 4.退出 ┃" << endl;

cout << " ┗━━━━━━━━━━━━━━━┛ " << endl;

cout << " 返回请输入其他数字" << endl;

**交互**

交互均通过用户从键盘输入数字实现，采用无限循环结构仅当用户输入的起始坐标与目的坐标均正确才能跳出循环。先用字符数组接收，防止各种非法输入的情形，再转到整数型。打印出棋盘后，有“请输入要移动的棋子的坐标；调用菜单请输入0，悔棋请输入9”字样 。正确输入后，即改变棋盘并打印出来，同时交由电脑决策。

**各功能的实现**

1. 悔棋功能

使用int数组now[7][7], last[7][7]，在每一轮后与map[7][7]传递更新储存上一轮的棋盘，悔棋时，即把last的数据拷贝到map中，然后提示继续下棋

1. 存盘读盘功能

使用文件流操作，建立“remember.txt”文件储存盘面信息（棋子摆放与双方棋子数），存盘示例：

ofstream fout;

fout.open("remember.txt");

if (!fout) {

cout << "存盘失败，请稍后再试" << endl;

return;

}

else {

fout << cnt1 << " " << cnt2 << endl;

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 6; j++) {

fout << map[i][j] << " ";

}

fout << map[i][6] << endl;

}

fout.close();}

1. 新游戏功能

只需将棋盘全部置零，在四角放置棋子，并改变棋子数为2，打印棋盘

1. 判断结束

依据规则，在一方落子后，判断另一方能否继续落子，若不能，则棋局结束。将空格算作这一方的棋子，所以只需将另一方的棋子数与25对比，大于等于25判定另一方胜利。轮流落子，在每一步棋后都进行判断，从而杜绝程序卡住不能结束的情况。

**AI设计**

程序采用minmax算法，即由电脑模拟双方落子，模拟己方采用利益最大的下法，模拟玩家采用使电脑利益最小的下法，形成博弈。

在优化后向后搜索五步。而为了优化程序，最初仅使用alpha-beta剪枝优化，在botzone的1s限制下能搜索四步。之后进一步优化，首先将本回合所有能走的方法都列出来，然后按照一步贪心的结果排序，优先吃子多且相邻的走法，取大概前40位的选择（经过试验基本不会在40种以外有最优解）。以及，将结果重复的走法删去（中盘阶段大约节约0.05s+）。实现五步搜索。

对于盘面的判断，设计评估函数。首先以双方棋子数之差为最优先的考虑。后考虑被我方两层棋子包围的空位，极有可能算作我的棋子数，可加入对空位的判断；考虑矩形是较优的形状，镂空形状、阶梯形都易被同化，可加入对形成矩形的判断；自己和对方的行动能力也是影响因素，考虑双方能去到的空格的数量。