计算概论 A 大作业

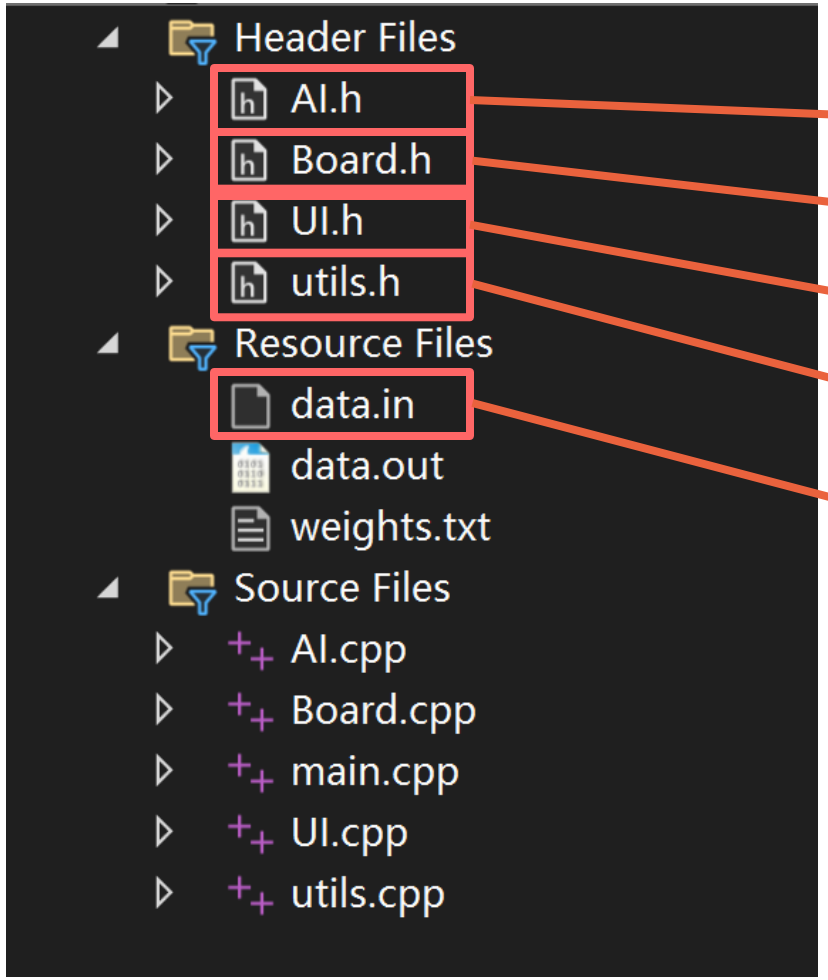
五子棋 *Renju*

目录

- 项目总览
- 棋盘类 Board.cpp
- AI模块 AI.cpp
- 图形界面与程序循环 UI.cpp

目录

项目总览 —— 项目结构



项目结构

- AI模块
- 棋盘类
- 图形界面
- 辅助函数(来自旧版)
- 其它：AI模型权重

项目总览 —— 基本功能的实现

【基本功能要求】

(1) 应包含一个菜单，用于支持如下操作（不限于）：新开始，存盘，读盘，退出 ✓

(2) 可以用字符实现画棋盘和棋子。 ✓

(3) 对弈双方一方是人类选手（如助教），另一方选手是计算机程序。 ✓

(4) 人类选手可以通过输入坐标或点击鼠标等方式实现落子。 ✓

(5) 程序要根据人类选手的输入，在棋盘上显示变化。程序根据算法决定下一步棋子所放的位置，并显示新的棋盘布局。允许中途停止游戏。 ✓

(6) 有存盘、读盘的功能（玩到一半，存储棋盘的状态） ✓

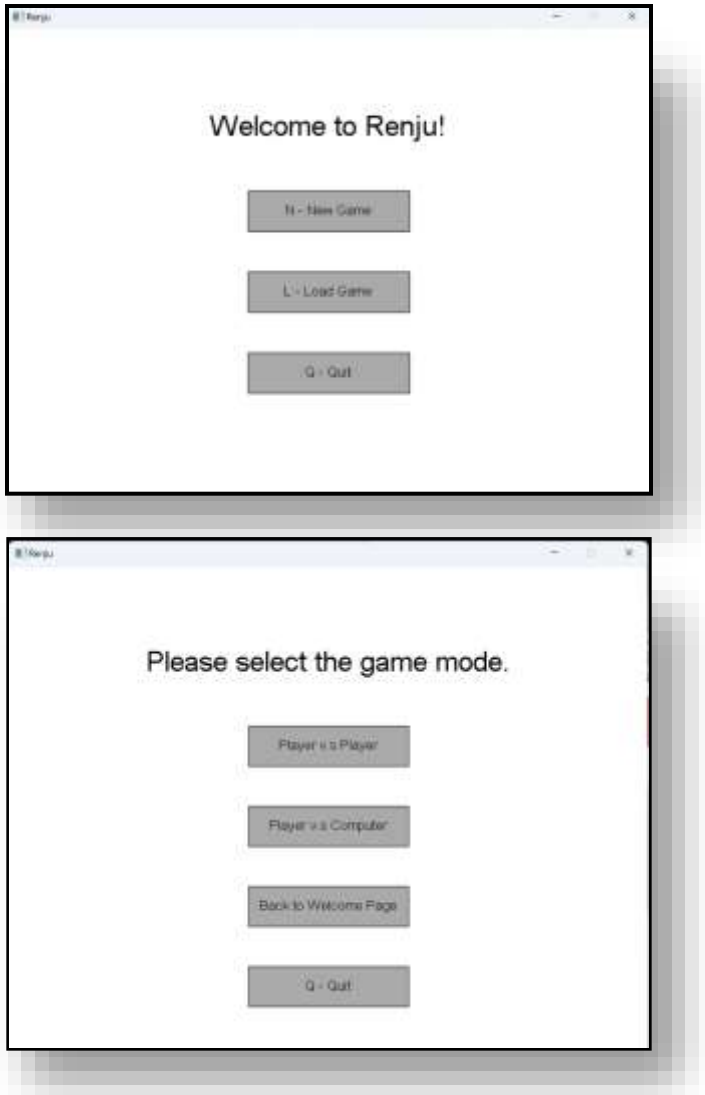
还有更多！

- 选择模式：PVP / PVC
- 图形界面
- 悔棋
- ……

项目总览 —— 基本功能的实现

菜单

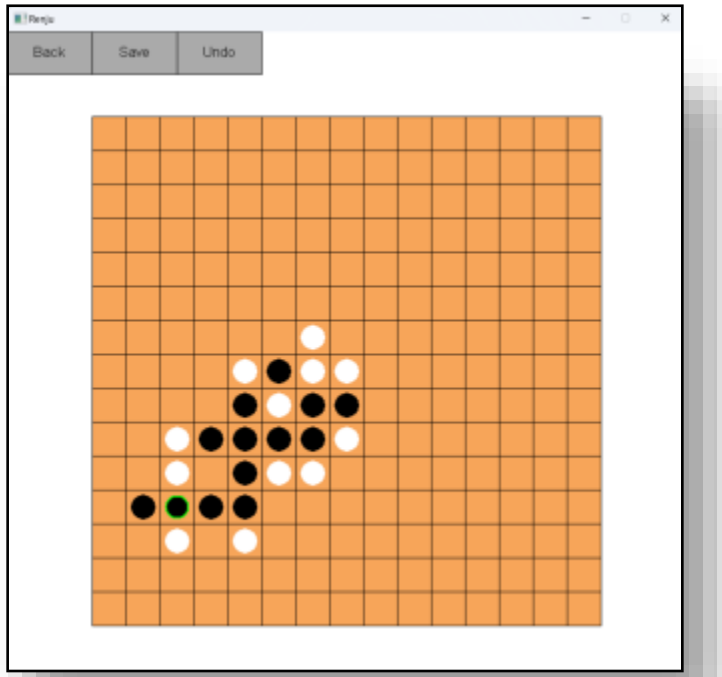
- 创建新游戏、读盘启动、退出游戏
- 选择模式 : PVP / PVC

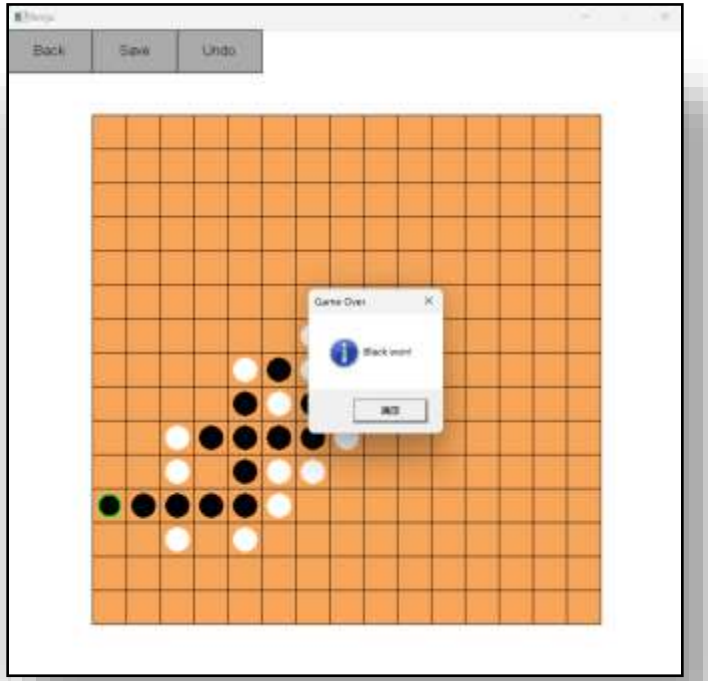


项目总览 —— 基本功能的实现

游戏界面

- 用鼠标点击实现落子
- 当前棋子高亮
- 悔棋
- 存盘
- 中途退出

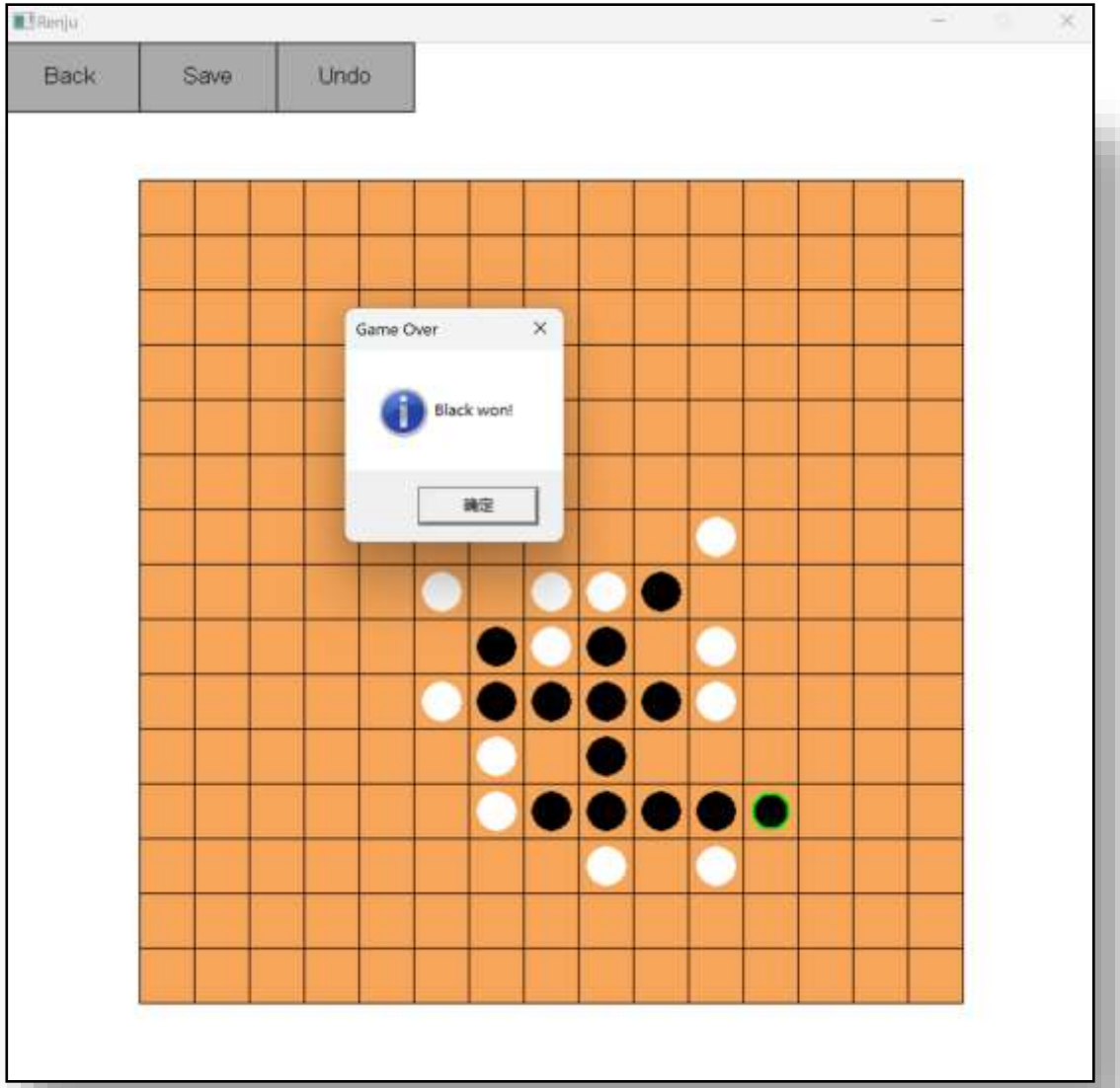




项目总览 —— 基本功能的实现

PVP 与 PVC 模式

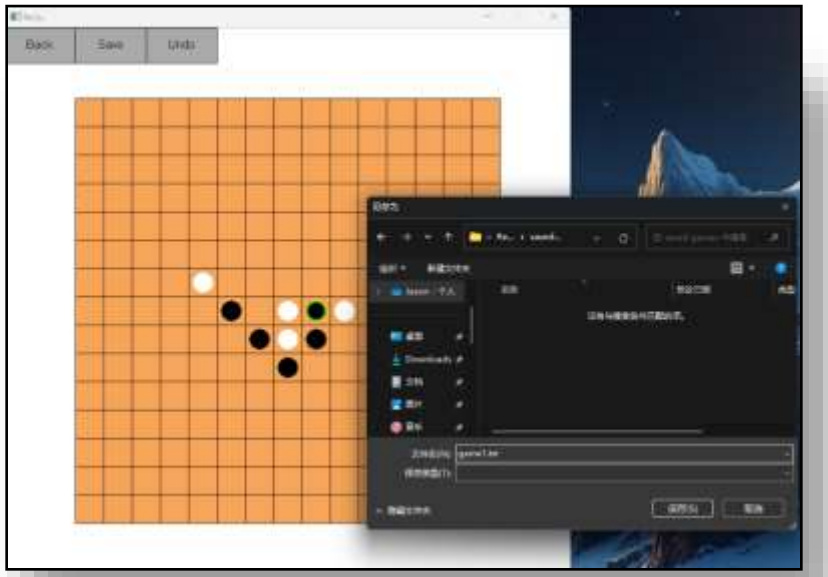
- 强大的AI

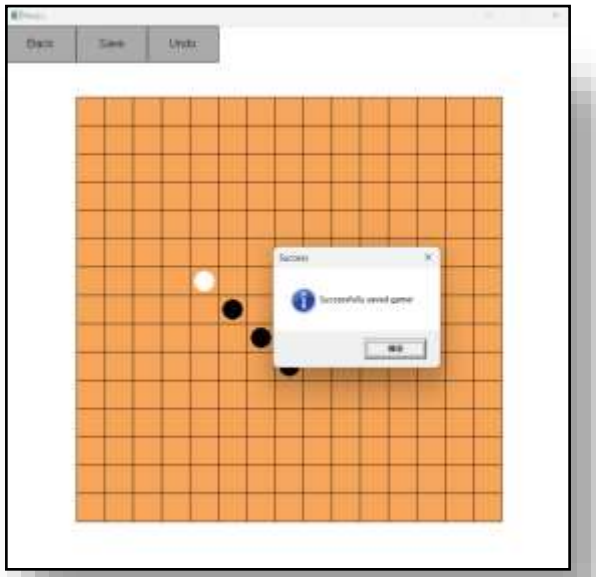


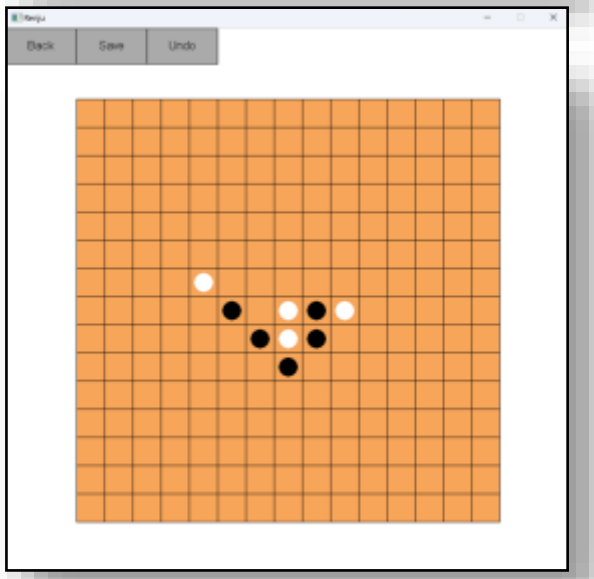
项目总览 —— 基本功能的实现

存盘与读盘

- 中途存盘
- 读盘启动游戏







目录

棋盘类

概述

- 封装了游戏逻辑模块，如禁手、胜负的判断(此处参考 Botzone 裁判代码 judge.cpp)。
- 棋盘类是 AI 模型搜索的基础，搜索时不断调用 doAction 与 undo 模拟落子，调用 win_end 判断胜负。
- 绘制棋盘与棋子、读盘存盘

目录

AI模块 —— AI.cpp

总览

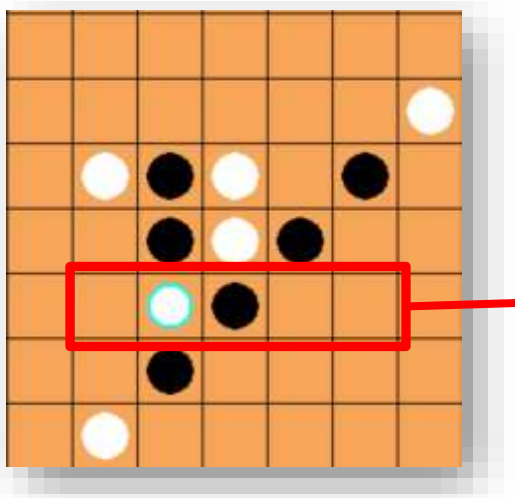
- 采用 **α – β** 剪枝的 Minimax 搜索算法
- 关键：设计一个好的估值函数
- 为防止超时，采用迭代加深搜索策略，超过2000 ms 强制中断思考

```
void read_weights();
void init_pos_weight();
double delta_SV(int board[][15], int x, int y, int player);
vector<Move> generate_moves(int board[][15], int player, int width);
double minimax(Board &b, int depth, int width, double value, double alpha, double beta, int player);
int action(Board &b, int player);
```

AI模块 —— 估值函数(SV)的设计

原始想法

- 由于五子棋的最终目标是形成五连珠，一个自然的想法是考虑棋盘上所有连续五个位置的状态。
- 这样的状态共有3^5种，每种状态赋予一定的权重(weights 数组表示)，求和即为整个棋盘的估值。估值越大代表白方越有优势。



事实上，这类似于一个单层卷积神经网络，具有一定的模式识别能力。

例如：如图所示的五个位置的状态可用三进制数 $(01200)_3 = 45$ 表示，对整个函数的贡献为 weights[45]

AI模块 —— 估值函数(SV)的设计

问题

- SV函数调用一次开销过大，极大限制 minimax 搜索层数。棋盘上连续五个位置共有572个。
- 243个权重，即 weights 数组难以确定。

AI模块 —— 估值函数(SV)的设计

优化：考虑 SV 的增量

- 注意到每增加一个棋子，整个棋盘变化不大。
- 每个棋子影响到的连续五个位置仅有20个，这启发我们考虑每下一步后SV函数的增量(用 delta_SV 函数表示)。
- 进一步优化：扫描棋盘时，状态编码可以减少重复计算。

$$(bcdef)_3 = 3 * ((abcde)_3 \bmod 81) + f$$

AI模块 —— 估值函数(SV)的设计

Weights 数组

- 尝试了一些策略后(如对 SV 或 delta_SV 强化学习)，决定手动调参。
- 随机初始化后手动调整，如为 ooooo 打高分，为 xxxxx 打低分等等。
- 实践表明，模型效果很好。

```
216    XXXOX -0.9
217    ...XX -2.24
218    O..XX -0.74
219    X..XX -5.61
220    .O.XX -0.19
221    OO.XX 0.0
222    XO.XX -1.84
223    .X.XX -7.71
224    OX.XX -1.48
225    XX.XX -150.0
226    ..OXX -0.24
227    O.OXX 0.46
228    X.OXX -1.32
229    .OOXX -0.14
230    OOOXX 0.15
231    XOOXX -0.47
232    .XOXX -0.28
233    OXOXX -0.42
234    XXOXX -0.36
235    ..XXX -10.0
236    O.XXX -0.32
237    X.XXX -160.0
238    .OXXX -1.16
239    OOXXX -0.19
240    XOXXX -4.14
241    .XXXX -180.0
242    OXXXX -4.34
243    XXXXX -10000.0
```

AI模块 —— 搜索算法

α - β剪枝的 Minimax 搜索

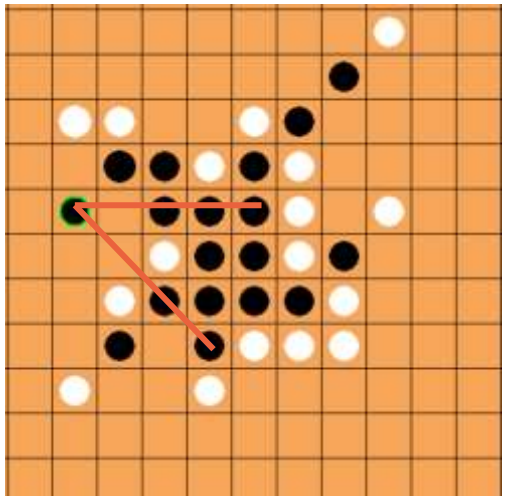
- 对于每个节点，计算所有可能走法的 `delta_SV` ，选出最优的若干个作为子节点，避免搜索树过宽。
- 由于我们计算 `delta_SV` 而非 `SV`，递归时需要多加一个参数 `value` 代表搜索路径上各节点 `delta_SV` 之和。
- 递归调用 `minimax(board, depth - 1, width, value + (1 - 2 * player) * move.score, alpha, beta, 1 ^ player);`
- 不同阶段动态调整搜索层数与宽度，由于可能的好走法逐渐增多，越往后搜索宽度需要越大。
- 迭代加深搜索以限制时间。

AI模块 —— 一些观察

有趣的观察

- 此模型没有加入任何棋谱，但是由于搜索层数较多(后期可达9 ~10层)，能识别出各种棋形，探索出许多下法(例如识别连四、活三等，故意制造连四以攻为守，制造四三等等)。
- 可以说，`weights` 数组是五子棋的一个粗糙的“世界模型”。简单的 `delta_SV` 函数迭代后产生复杂的行为。

例：制造四三



目录

图形界面与程序循环

概述

- 图形界面使用图形库 EasyX 制作。
- 包含五个界面 ：欢迎、选择游戏模式、选择先手后手、游戏与结束。
- MVC(模型-试图-控制器)设计模式，控制器类控制完整的程序循环。

计算概论 A 大作业

感谢观看！