计算概论 A 大作业实验报告：黑白棋游戏

李安齐
北京大学元培学院 数据科学与大数据技术方向
2300017432

2024 年 1 月

1 引言

本学期的计算概论 A 所布置的大作业是黑白棋。我自 2023 年 11 月开始构思，并在当时写成了初步的游戏框架和文字界面。此后，进入期末复习周期，由于精力难以集中于大作业的编写，我将编写中心放在了计算机执棋算法的优化上。从最初通过吃子数决定落子，到基于自身游戏经验写成的静态估值表，再到在计算概论课上学完递归后摸索到的 MinMax 算法，我的计算机棋力得到了逐步的优化。同时，我还阅读了一些文献，了解并学习了 α-β 剪枝策略，以及 MCTS、机器学习、强化学习等 AI 算法的设计。但相关文献同时指出，未经大规模数据训练的机器学习模型表现不如经过优化的 MinMax 算法。所以，我最终决定，保留自己曾经写过的计算机执棋算法，将它们设成共计 5 档的难度梯度，并将最难的 3 档分别定为搜索 2、4、6 层的 MinMax 算法。

2 游戏介绍

本游戏采取了字符界面，通过在控制台手动输入的方式操控游戏。游戏语言设定为英文。本游戏的优点在于游戏界面简洁大方、没有过多修饰，各类功能一应俱全，操作逻辑贯穿始终，棋面局势一目了然。

2.1 界面设计

打开游戏，进入菜单界面，首行为黑白棋英文名称 Reversi，接下来共有 4 个选项：新游戏、读取存档、设置、退出。此外，还标注了作者信息，以及操作提示。

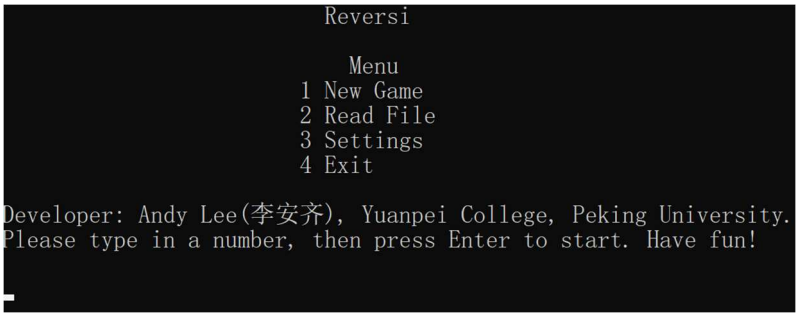

图 1 主菜单

2.1.1 新游戏

在主菜单界面选择 1，开启新游戏。首先选择对手，可以是双人对战，也可以

是人机对战。若选择双人对战，则直接进入对局，黑棋先手。若选择人机对战，则进一步选择人类玩家的执棋颜色（即决定先后手），进而选择计算机玩家的难度。难度共分 5 档：菜鸟、新手、中等、专家、大师。选择难度后进入对局。

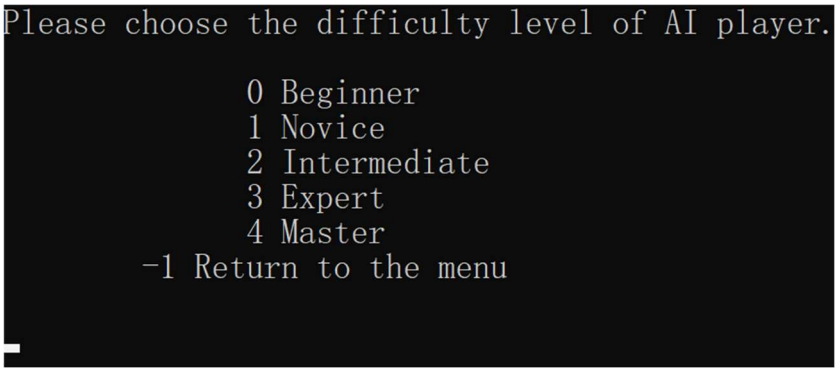

图 2 难度选择界面

2.1.2 读取存档

在主菜单界面选择 2，读取历史游戏存档。若没有历史存档，则会有相关提示，并要求玩家返回主菜单。若有历史存档，则允许玩家选择确认读档、不读档开启新游戏，或返回主菜单。

2.1.3 设置

在主菜单界面选择 3，进入设置。在此功能中，玩家可设置是否允许悔棋、机器提示两个功能。设置完成后可返回主界面。

2.1.4 退出

在主菜单界面选择 4，退出游戏。

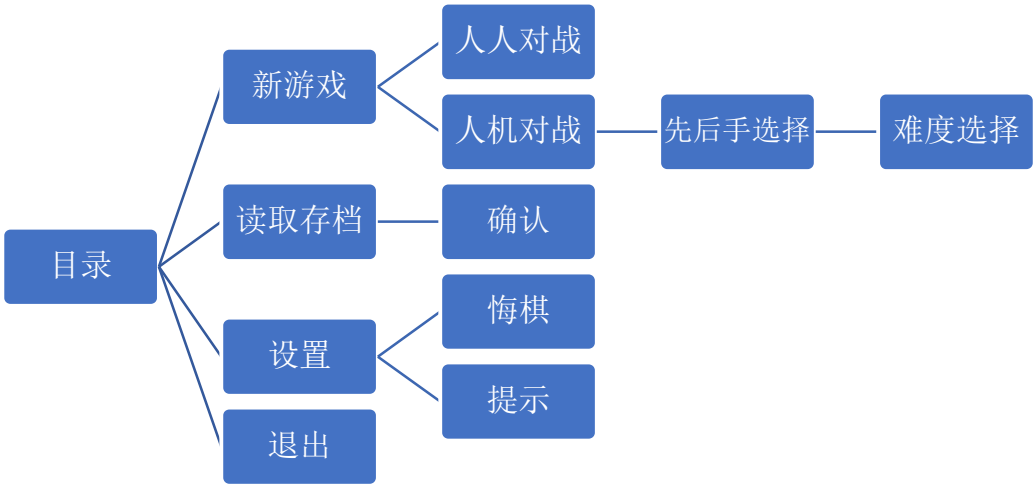

图 3 游戏功能思维导图

2.2 操作逻辑

本游戏的一大特点，就是操作逻辑符合人们日常生活使用电子设备的习惯。

首先，游戏的“返回”功能贯穿始终。进入游戏前，玩家可以在任意时刻输入-1 来返回主菜单；游戏过程中，为了符合游戏输入两个数的操作逻辑，输入-1 -1 也即可返回主菜单。

游戏的第二大特点，就是适应性强。玩家在任何场合进行不规范的输入，系统都会提示输入错误，并允许玩家重新输入。在设置界面，玩家可以随意、多次地修改相关设置，直至满意后输入-1 返回主菜单。

进入棋局后，玩家可以输入落子的横纵坐标，来操控棋局，这与线性代数课程中学习的内容有相同的逻辑。在设置中相关功能被允许的时候，玩家还可通过输入 0 0 来悔棋，或输入-2 -2 来获取计算机的落子建议。

在对局当中，刚落下的棋子会以黄色的实心或空心圆显示，给玩家更好的游戏体验。在与计算机对弈的过程中，对于可以在极短时间内给出结果的算法，系统会先“思考”1 秒时间再落子，允许玩家对自己落子后的局势有充足的时间加以了解。

在对局中，玩家可以实时地输入-1 -1 来结束对局。系统会保存包括棋面信息、历史走棋、人机难度、先后手在内的所有信息。故玩家在选择读档继续游戏后，包括悔棋在内的所有功能都可以正常运转。

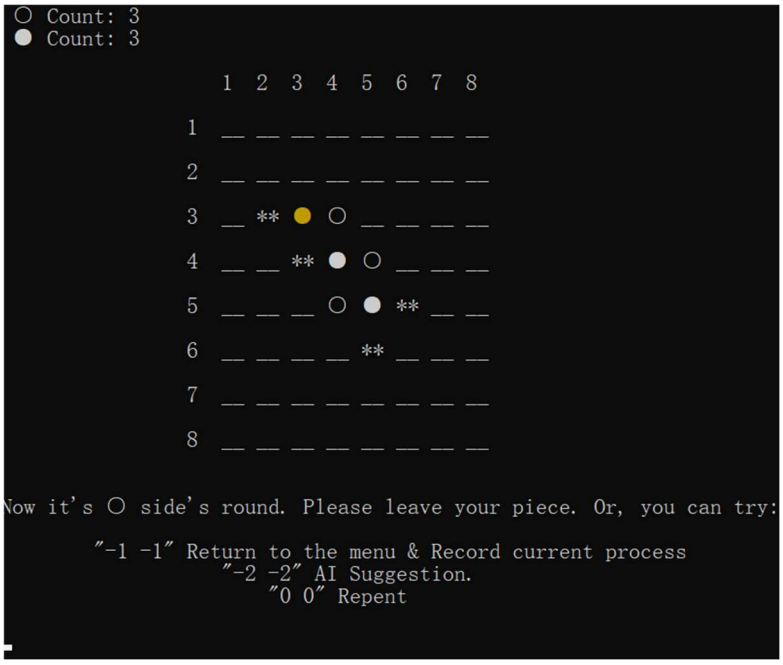

图 4 棋局界面

3 算法介绍

3.1 数据存储

代码使用了多元化的数据类型，用最合适的数据类型来存储相关的变量。例如，对于棋子颜色相关的变量，使用布尔型(bool)来表示黑或白棋；对于初等难度的算法可能搜到的多个等价的结果，用动态数组(vector)将所有结果加以存储；对于每一步棋，用结构体(struct)存储该步的相关信息，并在其中套用整型(int)数组来存储每步棋的吃子信息。

可以说，本代码的数据结构是复杂而有序，巧妙而恰到好处的。

3.2 函数架构

在这份代码中，所有的函数采取了驼峰命名法，简明而规范。

```
string boolToStr(bool b)
string numToColor(int num)
void setColor(int color)
void printBoard()
void leavePiece(int x, int y)
void findForCalc(int x, int y, int dir)
void calc()
void repent()
bool findForReverse(int x, int y, int dir)
void Reverse(int x, int y)
bool allZero()
void LeavePieceAccNumEaten()
void LeavePieceAccValueEasy()
void findStable(bool player)
int evaluateCurrBoard()
int LeavePieceSimulate(int x, int y, int depth)
void LeavePieceAccValueMedium()
void AISuggestion()
int personLeavePiece()
void endGame()
void saveProgress()
void startGame()
void initialization()
void settings()
void readProgress()
void intoMenu()
int main()
```

表 1 代码函数表。其中含显示、规则、人机三部分

3.2.1 显示逻辑部分

string boolToStr(bool b) 用于设置的显示，将逻辑值 b 转换为字符串"ON"或"OFF"；

string numToColor(int num) 用于棋子的显示，将数值 num 转换为黑白棋〞○〞〞●〞；

void setColor(int color) 用于提示上一步落子，改变输出的前景色；

void printBoard() 用于打印棋局界面；

void endGame() 用于在对局结束时给出提示语；

void startGame() 用于执行对局，并显示对局内的提示语；

void settings() 用于显示设置。

void intoMenu() 用于显示菜单并完成菜单逻辑。

3.2.2 规则逻辑部分

void leavePiece(int x, int y)　用于落子并记录该步相关信息；
void findForCalc(int x, int y, int dir)　用于与下面的函数合作，完成递归；
void calc()　用于寻找可以落子的点位；
void repent()　用于完成悔棋逻辑，依据历史每步记录；
bool findForReverse(int x, int y, int dir)　用于与下面的函数合作，完成递归；
void Reverse(int x, int y)　用于翻转，或服务于第 0 档人机，计算可吃子数；
bool allZero()　用于判断棋盘是否有落子点；
int personLeavePiece()　用于完成人类下棋步；
void saveProgress()　用于向文件输出以存档；
void initialization()　用于棋盘初始化；
void readProgress()　用于从文件输入以读档。

3.2.3 人机逻辑部分

void LeavePieceAccNumEaten()　用于第 0 档人机，根据吃子数落子；
void LeavePieceAccValueEasy()　用于第 1 档人机，根据静态估值表落子；
void findStable(bool player)　用于第 2+档人机，与下面的函数合作，寻找边角稳定子；
int evaluateCurrBoard()　用于第 2+档人机，为 MinMax 算法叶节点估值；
int LeavePieceSimulate(int x, int y, int depth)　用于第 2+档人机，MinMax 递归；
void LeavePieceAccValueMedium()　用于第 2+档人机，择 MinMax 最优者落子；
void AISuggestion()　用于机器提供建议，实际采用了第 3 档人机算法。

3.3 人机对战算法

本代码采用了 3 种不同的人机对弈策略，旨在为不同水平的玩家提供不同棋力的对弈者，加快玩家学习与适应游戏策略的速度，更好地服务于玩家。下面展开介绍三种策略。

3.3.1 吃子计数算法

顾名思义，吃子计数算法是遍历当前局面上己方所有行动力，计算其中每个落子点可以翻转对方的棋子数量，择最高者落子。

之所以为第 0 档难度选择该算法，主要是由于作者观察到许多初学者容易被局面上的吃子数蒙蔽，认为吃子越多该步棋最优；实则不然，黑白棋正是以惊心动魄的局面翻转为其特色。

我注意到，吃子计数算法在实现过程中，代码会与 Reverse()以及 findForReverse()函数高度相似。于是，我干脆添加了一些全局变量，用于区别模拟翻转与真实翻转，并记录每个点的吃子数。

另外，我使用动态数组(vector)记录了所有吃子数最高的落子点（如有相同），并在一定范围内生成了随机数，从这些点中随机挑选一个落子。

3.3.2 静态估值算法

在第 1 档难度中，我根据自己下黑白棋的经验设定了一张静态的棋面估值表。

计算机每次寻找估值最高的落点，用可变数组存储，并随机挑选落子（原理同上）。遵循本算法的计算机棋手已经可以展示出一定的策略性，例如避免劣势位置、优先占角占边等。

20	-10	10	5	5	10	-10	20
-10	-15	1	1	1	1	-15	-10
10	1	3	2	2	3	1	10
10	1	3	2	2	3	1	10
10	1	3	2	2	3	1	10
10	1	3	2	2	3	1	10
-10	-15	1	1	1	1	-15	-10
20	-10	10	5	5	10	-10	20

表 2 静态估值表

3.3.3 极小极大搜索算法（MinMax 算法）

本代码在算法层面上的主要工作放在了构建博弈树搜索算法上。在这份代码中，我采用了极小极大搜索算法。难度中的后三挡，分别对应了极小极大搜索算法的 2、4、6 三档搜索层数。

根据实验，搜索 2 层或 4 层，计算机往往能在 0.5s 内给出结论。而搜索 6 层，由于呈指数趋势上升的搜索量，计算机能基本控制在 45s 内给出结论。

对于极大极小搜索的叶节点，我引入了一个考虑周到的估值函数，下面对其进行简单介绍。

在黑白棋策略中，有几个关键概念：

行动力(Mobility)：局面上我方可落子的数量

稳定子(Stable Discs)：局面上永远不会另一方翻转的棋子，通常出现在边和角上。

于是，对于局面 S，可以引入极大极小搜索的叶节点估值函数 V(S):

$$V(S) = w_1 \cdot M + w_2 \cdot (Sd_1 - Sd_2) + w_3 \cdot \sum_{i=1}^{8}\sum_{j=1}^{8} \omega_{i,j} \cdot s_{i,j}$$

其中：

$$s_{i,j} = \begin{cases} 1 & color_{i,j} = my\_color \\ -1 & color_{i,j} = opp\_color \\ 0 & otherwise \end{cases}$$

估值函数由三部分的加权之和得到。三部分分别为：我方行动力、双方稳定子之差、双方静态估值之差。权值之比定为了 3:5:2。

参考文献

[1] 柏爱俊：几种智能算法在黑白棋程序中的应用
[2] 黄海通：黑白棋 AI 设计探究
[3] 彭之军：计算机博弈算法在黑白棋中的应用

致谢

感谢我的计算概论 A 教师刘譞哲，以及楼持恒、王启鹏等帮助过我的助教学长们。你们让初入燕园的我在这样一门计算机入门课上感受到了热爱、温暖和力量，我会继续努力向你们学习。

感谢我的室友毕晨博、王一川、胡云天同学，以及我的好友黄景暄、高梓钧、张轩瑞、李牧雨同学。你们为我写这份大作业提供了数不胜数的灵感和建议，这总让我备受鼓舞。