计算概论 A 大作业

毛川学号：2300013218

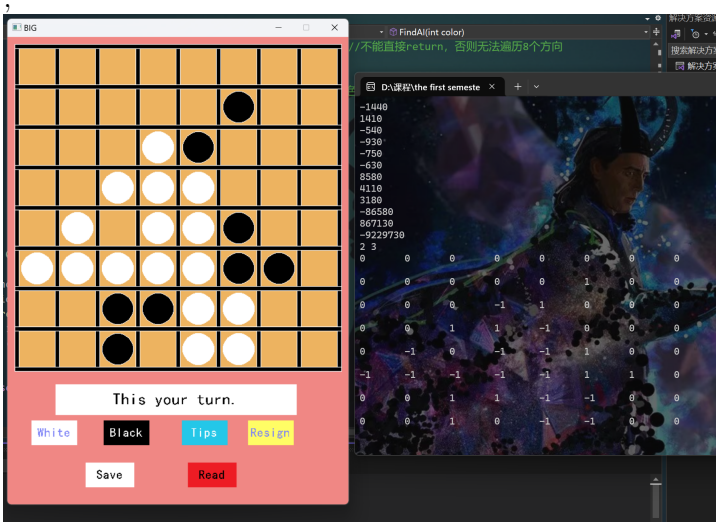
2023 年 12 月 29 日

0.1 完成总述

本次大作业（含报告）完成时间大约 50 小时，我认为对于玩家在黑窗口上算棋是一种“折磨”，于是想实现可视化。发现每步棋用键盘输入较麻烦，希望实现鼠标点击输入的功能。

开始使用一半时间完成 GUI 的学习和实际应用，同时实现落子，判定可行点等规则。之后阅读了一些资料，结合个人思考确定了一个 DFS 算法，在进行数十盘棋的对弈过程中调整参数。此后修改代码，通过剪枝提高搜索深度，最后又调整了 UI 界面，并添加文件的存档和读档功能。

;



0.2 具体过程

1.GUI 实现 首先安装了 EASYX，在 B 站学习了一些基本函数的使用方法，然后进行实际操作，计算规划棋盘的大小，颜色，格子的数目，变框的宽度等必不可少的元素之后，宏定义了如“MAX_NUM”等数据，方便后续使用。绘制好棋盘后，一个艰巨的任务是，绘制按钮，全局一共 64 个格点，有 60 个可以落子的位置，于是我定义了 64 个棋子按钮（以结构体）的方式实现，和几个功能按钮。如果鼠标在按钮中按钮会变红，这个过程通过不断绘制按钮完成，由于每一次循环足够快，人类视觉无法察觉出这种闪烁。

对于多种颜色，采用了一个枚举，每次 Drawbutton 不需要再写一遍每个元素的 RGB。mouseInButton 和 clickChessButton 这两个函数以鼠标信息为参数，如果鼠标在按钮里或点击按钮，进行下一步操作。

2. 规则实现 这里主要由两部分组成，其一为判断某一个点能否落子，另一个判断哪些棋会被翻转，它们都采用了朝八个方向深度优先搜索的递归函数。前者采用 judge，后者采用 judgerev 和 JudjeWSTurn 等函数。

3. 核心算法及剪枝 FindAI 这个体量较大的函数实现了 AI 的主要算法，我的算法思路核心来源于权重值和对方的行动力，将 AI 可以下的各步棋的 utility 排序，选择最优解。以下是我的效用方程：

$$utility = a\times(later\ weight\ sum - now\ weight\ sum) - b\times(opponent's\ opportunity) - c\times(opponent's\ highest\ utility)$$

$$a > 0 \& b > 0 \& c > 0$$

之后令我的 AI 与人类或电脑对弈，观察胜率和行棋的偏好，尝试改变参数 a、b、c，让 AI 变得更强。

但是我们发现，在棋局进行到中盘过程时，计算到的步骤数量指数级别增长，这大大制约了计算的深度，甚至无法在 1 秒内算到三步。这样就必须考虑剪枝的手段：将从第二步开始的可行解按照线性表达式的前两项排序，只保留其中的部分选点，从而进行更深的搜索。

0.3 遇到的困难

1. 第一次创建按钮时按钮上的文字不停闪动，影响美观，最后采取清屏代码得以解决。

2. 开始检测规则可行性的时候采用 random 算法，当完成了 DFS 搜索之后，接入到程序主体时遇到了 BUG，即如果人类已经无棋可下，AI 找不到落子的位置，经过整整一下午的 Debug，发现是全局变量的位置有问题，深入到第二步时返回值使第一步判断为无棋可下。最终通过微小调整得以解决。

0.4 作品亮点

1. 在众多小程序游戏网站上，黑白棋由于下棋者不易看到那些位置可以下棋（尤其是行动力非常小的时候），于是都给出可选点的提示。但是这些提示始终在棋盘上，影响下棋者计算的深度，所以我采用一个新的方法，对弈者可以选择开启提示和关闭提示。

2. 构建算法模型时采取博弈论的观点，从本 AI 的视角考虑下一步棋的时候，选取之后人类最优的落子位作为 AI 这一步棋效用评估的一项，这样可以尽可能减少 AI 受的损失。

3. 人类下棋时如果有事可以点击“save”保存，下次可以在新的程序运行时点击“read”读取上一次的棋局和双方的棋子颜色。

0.5 收获

1. 学会使用 EASYX 的一些简单的函数，实现最基本的可视化。2. 通过参考和类比资料，得到一种算法，并通过调整线性组合的参数，和剪枝来优化算法。3. 发现了写一个工程（游戏），有很大的趣味性，同时也能学到许多新知识。在设置变量，定义函数，调试 Bug 时都有独特的体验，假期准备继续尝试新事物。