|  |
| --- |
| **计算机专业类课程** |
| **实验报告** |
| **课程名称：C语言程序设计**  **学院专业：计算机科学与工程学院**  **学生姓名：李佳骏**  **学　　号：2020080901014**  **指导教师：俸志刚** |
| **日　　期：2021年 6月 13日** |
|  |
| 电子科技大学计算机学院实验中心 |

**电 子 科 技 大 学**

**实 验 报 告**

**实验一**

# 一、实验室名称：

电子科技大学基础实验大楼502

# 二、实验项目名称：

五子棋AI的实现

# 三、实验目的：

实现一个五子棋AI，使其拥有较强的棋力

# 四、实验主要内容：

实现一个五子棋AI：

1. 算法部分（核心）
2. UI部分
3. 模块化编程部分
4. 实现细节

# 五、实验器材（设备、元器件）：

编译命令：

"${fileDirname}\\\*.cpp",

                "-o",

                "${fileDirname}/${fileBasenameNoExtension}.exe",

                "-O3",

                "-g",

                "-m64",

                "-Wall",

                "-Wl,--stack=134217728",

                "-static-libgcc",

                "-std=c++17",

# 六、实验步骤

1. 问题描述：设计五子棋AI
2. 算法分析与概要设计：
   1. 评估函数
   2. 极大极小值搜索
   3. Alpha-Beta剪枝
   4. 启发式搜索
   5. 深度与广度剪枝
   6. Zobrist哈希记忆化
3. 核心算法的详细设计与实现：
   1. 评估函数：

棋型分为多种情况进行讨论。保证**不同级别**的棋型之间的差异至少是1个数量级，且考虑先后手 带来的常数级影响。

棋型得分是描述处于同一行/列/斜列的棋子状态，考虑的主要因素：连续的个数，对手棋子的位置，空位个数等

默认评估的棋型是**当前棋手已落子的状态**

局面较为胶着时，相同棋型下对手的评估分数要更高，因为下一步落子的一定是对手，相当于对手先手

局面较为缓和时，优先让己方先手评分更高，提高先手优势，防止AI“不敢下”

考虑上述因素后，我们对不同棋型的评分如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 活五 | 活四 | 冲四 | 活三 | 眠三 | 连续活二 | 散二 | 活一 |
| 4×107 | 1×106 | 4×105 | 5×104 | 100 | 60 | 3 | 1 |
| 1.7×107 | 1.2×105 | 3500 | 3000 | 120 | 72 | 5 | 1 |

最核心的排序：对手活四 > 对手冲四 > 己方活四 > 对手活三 ≫ 己方冲四 > 己方活三

对手剩余活四/冲四后，可以一步直接胜利

若己方剩余活四而对手无一步胜棋，对手无法阻止己方胜利

若双方均无无一手胜棋，而当前对手有活三，则对手必胜

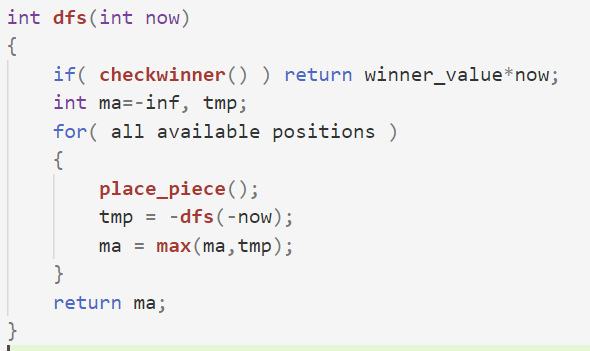
己方冲四，己方活三及剩余情况下，均可被防守阻止，根据连续棋子数目来确定评估分数，连续的越多评估函数越高，两侧空位越多分数越高

\*1：由于写法问题，本设计中连续落子的分数要比断开更高

* 1. 极大极小值搜索：

考虑这样一个博弈模型：每个人都可以选当前状态下的几个物品，先手取最大值，后手取最小值，问先手最多能取到多大的？

考虑用搜索解决，后手取最小值=将物品价值取反后求最大值，伪代码如下：



* 1. Alpha-Beta剪枝

Alpha-Beta剪枝是一种经典的极大极小值搜索优化算法。每次搜索时记录对自己而言的最优解alpha和对对手而言的最优解beta（对自己的最劣解）。

如果搜索的答案比beta还要大，对手不会让自己达到这一目的，转而选择之前搜索出来的最优解

如果搜索的答案比alpha大而比beta小，当前是更优的解法，我们更新alpha

进入到下一层时，-beta是对于下一层的最优解（如果不能更新，不如返回到上一层），-alpha是

代码实现：

。。放图

* 1. 启发式搜索

考虑alpha剪枝的过程，如果先搜索“可能的更优解”，那么alpha和beta的限制会更紧，后续搜索所用的时长也会缩短。实际操作中，我们把可用的位置，按评分从大到小排序即可。

* 1. 深度与广度与剪枝

代码中采用了一些非安全剪枝来提高效率：深度上，只搜索6层（己方4步，对手3步）。广度上，只搜索排序后的前五个位置。

在加入后文提到的Zobrist优化后，对后期局面的搜索速度会很快，因此在后期将**搜索广度**小幅度扩大以减小非安全剪枝带来的负面影响

* 1. Zobrist哈希记忆化

搜索过程中可能会遇到相同的棋盘，为减少重复的评分，我们对棋盘进行哈希和记忆化，Zobrist的哈希方法如下：

初始化两个棋盘分别代表黑子与白子，每个位置设定一个64位的随机数，每次落子后将哈希值异或上棋子颜色和位置对应的随机数棋盘，得到的就是新棋盘的哈希值。

用stl中的unordered\_map记录哈希值和棋盘评分

实际测试中，常数约减小1/2，还是比较可观的

1. UI的设计：

虽然UI并非本实验的核心，但自认为这个UI还是相当好用的 ~~即使它很丑~~

* 1. 双缓冲模块

Windows自带的清空命令行“cls”效果不好。清屏时会出现闪烁，伤害使用者的眼睛。

输出到终端的内容实则是把缓冲区内的内容输出，考虑建立两个缓冲区，交替输出内容：此时正在展示的（在终端内输出的）称为明缓冲区，另外一个缓冲区称为暗缓冲区。当需要输出新内容时，将之全部填入暗缓冲区，再将原暗缓冲区设置为明缓冲区。

实现：开一个大小为2的句柄数组，记录一个滚动变量now，在0和1之间每次都异或即可。

* 1. 换色

由于我们采用了自己手写的双缓冲，不能用原printf内自带的换色方式了，需要找系统函数。

**SetConsoleTextAttribute ：对接下来输入缓冲区的内容换色**

**SetConsoleCursorPosition ：切换缓冲区**

**WriteConsoleA：向缓冲区内输入字符**

**SetConsoleCursorPosition：设置在缓冲区内输入的位置**

……

* 1. 重载输出

有了换色功能，考虑**重载printf函数**。

代码中实现了一个dprintf 函数，它负责输出内容到缓冲区中，并可以设置是否在输出后切换缓冲区。

如果需要式样化字符串，利用库函数sprintf将内容输出到临时数组中，再调用dprintf输出即可

* 1. 额外功能:

~~不写不知道一写吓一跳~~

棋谱评估函数中实现的细节，是真的比我原料想的要多。为了方便调试我搞了如下三个函数方便调试：

int **inputmap**() 设置初始棋盘

void **savecurmap**() 保存当前棋盘到文件中

void **inform\_out**() 和 void **push\_inform**() 输出局面评估信息到文件中

对棋子下棋的编号进行了记录

增加了记录双方用时的功能

1. 模块化编程：

前端UI和后端算法的模块都需要分开，否则不同模块的代码堆到一起，难以调试和发现问题

实验初期对文件模块化后，采用include相互调用，导致无法实现模块间的**相互**调用，只能拓扑序调用，难以使用

后续调整过程中用如下形式实现相互调用：

1.创建一个公用头文件func.h，每个模块都include这个头文件。func.h中用extern前缀声明需要在模块间调用的函数，再在每个模块中写函数的定义。

2.对于全局变量，先在func.h中用extern声明，再在其他任意一个文件中进行**定义**（可以创建一个新文件）。没有**定义**的变量会导致编译无法通过！！！

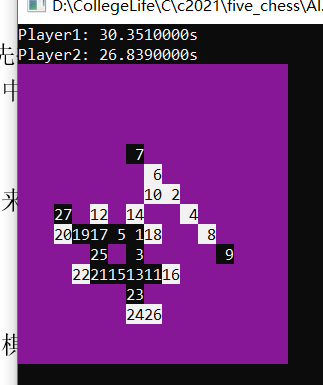
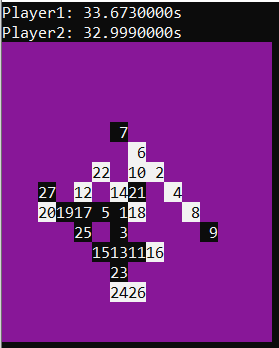
3.编译命令修改为\*.cpp（同时编译所有.cpp的后缀文件）

1. 实现细节：

实现时真是bug多多，评估过程中有奇奇怪怪的bug，不过在一番调试后，棋谱评估的效果还比较理想

# 七、实验数据及结果分析：

电脑对下的过程中，先手黑棋已经能下出比较规律的棋型了，几乎是先手必胜的

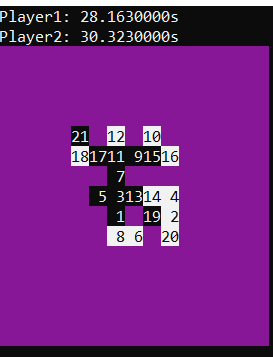
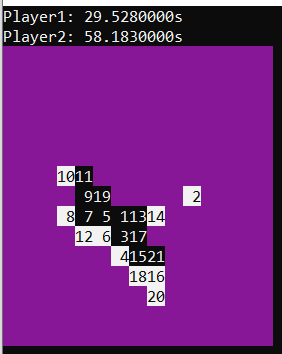
先手黑棋围出“环”或“十字”后，后手白棋无力阻挡黑棋的连环攻势

搜索深度为6层（己方4步，对手3步）。广度为（5+f(总落子步数)）

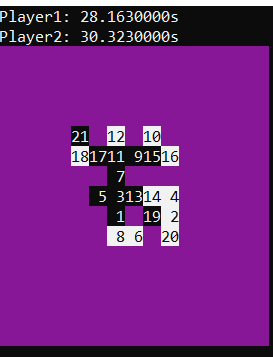
f(总落子步数)是单调不减的一个函数

即使在棋局后期提高了每一层的搜索广度，上述棋局平均每步的时间仅为2s，且耗时较多的是前期棋盘较空局面时的搜索，占据总运行约50%的总时间

我和电脑下，根本下不过电脑 ~~倒也可能是我棋艺不精~~

电脑先手真是步步紧逼，几乎没有疏漏

 我先手终于赢了一局，十分勉强

# 八、总结及心得体会：

收获都在实验步骤里。

总结一下：

1. 双缓冲的简单应用
2. Alpha-beta剪枝搜索和启发式评估函数\
3. 模块化编程
4. 如何调bug

调bug要多用**输出调试**，gdb调试这种大代码，远没有输出调试好用。调代码就像做物理生物实验，首先对可能产生的bug进行推测，再验证。遇到难以解决的bug考虑重构。

# 九、对本实验过程及方法、手段的改进建议及展望：

算杀和缓存优化已经没有时间写了，十分可惜。

但现在代码中大多数的bug应该都被我调出来了，电脑操作几乎没出现明显失误

题外话：

此次大作业感受很好 收获很多

工程的前期：下不过AI难过，因为自己棋艺不精。下得过AI也难过，因为AI写的太弱了

工程的后期：下不过AI开心，因为是AI太强了。 下的过AI也开心，说明我棋技提高了

~~乐观而自信~~

**报告评分：**

**指导教师签字：**