Gobang Algo 开发文档

王华强 2018.1.13 于比赛后

简单说明

核心算法在 algo_rebuild.h, weight.h, forbidden_move.h中.

具体各部分作用下面有介绍.

请务必浏览文档中的补充说明部分!!!

算法说明

支持:

1. 禁手判断
2. 棋谱保存

(悔棋懒得写啦)

使用了以下算法:

1. AlphaBeta剪枝
2. zobrist hash 置换表
3. 历史启发搜索
4. 迭代加深搜索
5. 只针对改变部分的快速局面评分函数
6. 同步进行胜手优先搜索(没开)
7. 贪心启发搜索搜索(没开)

因为使用多线程会导致跨平台出现麻烦, 所以并没有考虑多线程算法

否则, 可以同时开启VCF, 并将基本的AlphaBeta剪枝替换成极小窗口搜索来多线程执行

若没有特别说明, int形的函数表示一个操作的, 如 SetUpBoard(), 返回0代表正常执行.

名如 Weight()的函数会返回名称所对应的值.

形如 Is_Tested()为真时返回1, 否则返回0

有特殊返回值的函数会另行说明

程序在windows10下使用gcc编译通过

/* Todo: // 调整hash表 // 通过棋盘变化赋权 补注释 // _Judgewin // _StartTimer(4) */

各组件说明

主程序 gobang.c

```
//Copyright (c) 2017-2018 Augustus Wang
//自注释风格的函数和变量名命名，应该不用太多注释....
//主程序
#define TIMEIT
#define TEST
#include "timer.h"          //性能分析，计时相关
#include "support.h"        //棋盘定义，数据结构定义，基础情况判断以及其他共用部分，写成头文件以方便单元测
```

```
试
#include "forbidden_move.h" //禁手判断

#include "algo_basic.h"  //大猩猩下棋
#include "algo_linear.h" //一层两岁小孩下棋(别看了，没用的，注释也不存在的)
#include "algo_point.h"  //一层三岁小孩下棋(别看了，没用的，注释也不存在的)
// #include "algo_final.h"//Alphabet剪枝加各种优化
#include "algo_rebuild.h" //Alphabet剪枝加各种优化(重构)(算法核心)
#include "printboard.h"   //显示棋盘所用的函数
#include "socket.h"       //自动对战接口定义
```

主算法 algo_rebuild.h

```
// Copyright (c) 2017-2018 Augustus Wang
// AlphaBeta剪枝和历史启发为基本框架，其他部分通过头文件引入

//常量定义
#define ALGO_FINAL 3
#define EDGE 3

//程序设置
#define LEVEL 12 //最大搜索层数
#define NEABOR 2 //for GetAroundPosition(), 落子时只考虑已有棋子边NEABOR个点
//(6,2)is recommended;
#define THINKINGUPPERBOUND  //设置启发上界
#define DEEPLEVELUPPERBOUND //在层数较深时降低启发上界
#define DEEPLEVEL           //定义"较深的层数"
#define WEIGHTHEURISTIC
#define ENABLEHASH //激活置换表
#define RANDFST    //首步随机落子
// #define DEFENDMODE //白棋前十步采取守势

#define ENABLEFBDMOVE   //禁手探测(会极大拖慢速度)
#define TIMELIMIT 14800 //迭代加深时间限制

#include "zobrist.h"   //哈希
#include "support.h"   //全局变量，共用函数
#include "heuristic.h" //启发式搜索框架
#include "weight.h"    //部分更新估值函数
#include "killfirst.h" //胜手深搜(未启用)
#include "rand_move.h" //随机落子
#include "greedy.h"    //贪婪启发(未启用)

//全局变量
int GetAroundPosition();          //查找周围的可用位置
int showweight[BOUNDRY][BOUNDRY]; //权重数组，调试使用
time_t _starttime = 0;            //计时函数使用，用于控制迭代加深搜索的时间
int best_weight_found;            //显示当前局面评分
int maxlevel = LEVEL;             // 搜索层数控制
int thinkingupperbound = 200;     //搜索宽度
int deepleveluppperbound = 200;   //深层搜索宽度
int deeplevel = 4;                //"深层"
int maxneabor = NEABOR;           //落子时只考虑已有棋子边NEABOR个点
int defendmode = 0;               //白棋特殊操作
```

主支持库 support.h

```
//Copyright (c) 2017-2018 Augustus Wang
//support.h
//Judgewin, and some basic parameters.
```

```
//棋盘定义，数据结构定义，基础情况判断以及其他共用部分，写成头文件以方便单元测试
#ifndef _SUPPORT_H
#define _SUPPORT_H

#define BOUNDRY 15   //棋盘大小
#define BLACK 1      //黑方
#define WHITE 2      //白方
#define OUTOFBOARD 3 //棋盘边界，超出棋盘范围的点用3表示

#define NORMAL 0 //旧的判断函数用的，懒得改掉了
#define CLEAR 1  //旧的判断函数用的，懒得改掉了

#include "lazy.h"    //一些调试用的函数
#include "charlib.h" //处理用户输入输出的文本
#include <malloc.h>

#define INF 10000000
#define NINF -1000000

//结构和变量定义
//global vars.
int colornow;                    //当前应走子的颜色
int fstmove = 1;                 //是否为第一步
int board[BOUNDRY][BOUNDRY];     //棋盘数据
int printboard[BOUNDRY][BOUNDRY]; //绘图板
int weight[BOUNDRY][BOUNDRY];    //权重变化,注意权重为double

//settings
int _usesimpletest = 0;
int set_savelog = 0;

struct move
{
    int a;
    int b;
};

struct movenode //历史启发搜索用排序节点
{
    int a;
    int b;
    int weight;
    int history_score;
};

//定义4个方向以简化代码
int direction[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
```

Acknowledge

在五子棋的开发过程和比赛过程中, 得到了许多人的帮助, 在此一并致谢.

整体设计思路参考了王小春的<PC游戏编程>一书.

LFZ同学在开学前就完成了基本的五子棋程序, 在设计思路上提供了很大的建议.

SunKai(弈心作者)的网站上提供了很多有价值的参考资料.

WXP同学帮忙发现了hash表的一个严重bug, 在此致以最真挚的感谢(这是真救命了啊.....).

在研究这个算法的过程中, 与LYZ, CX同学进行了深入的探讨, 交流和测试.

HY同学提供了算法设计上的很多建议.

补充说明

```
**当前代码存在严重bug!**
```

如果想要使用此算法, 请务必关闭哈希表(极大减速)或将哈希表的校验位调整至64位(当前为32位)!

如果想要使用此算法, 请务必关闭哈希表(极大减速)或将哈希表的校验位调整至64位(当前为32位)!

如果想要使用此算法, 请务必关闭哈希表(极大减速)或将哈希表的校验位调整至64位(当前为32位)!

关闭哈希表只要使用:

```
#undef ENABLEHASH
```

调整校验位需要较大改动, 因为比完了我也懒得改了.

这个bug在比赛的前一周才被发现, 但是一直没有找到原因. 感谢WXP同学, 在比赛结束后的讨论中发现了这个问题.

因为运气比较好, 比赛中一路收割一层的对手, 最终成功吃鸡(雾), 所以Hash的冲突没有过大的影响.(但是这个bug仍然导致输了好几盘TT).

关于hash table bug的详细讨论

在重写局面估值和禁手函数之前, 由于运算速度过慢, 节点数不多, 哈希冲突仍可接受.

然而, 在重写局面估值和禁手函数之后, 运算速度指数提升, 节点数骤增, 导致32位校验位远远不足(校验位大小与节点数在相近量级), 出现大量冲突, 导致速度变慢和估值严重偏差.

调整校验位到64位可解决这个问题.

在比赛中使用的是有bug的版本. 击败示例程序需要关闭hash.....
