人工智能基础: 编程作业1实验报告(2)

> 姓名：王博
> 学号：PB16020870

> 实验名称：五子棋人机对弈

1. 实验要求概述

- 棋盘为为十五路（15×15）棋盘，在其上实现一个五子棋AI。
- 设计一个评分函数对棋盘上局面进行评分。
- 利用评分函数生成一颗博弈树。使用minimax算法和Alpha- Beta剪枝策略实现一个固定搜索深度（搜索深度大于1）的人机对弈的五子棋AI。
- 结果的呈现为与AI棋手对弈一局的过程。
- 完成思考题。

2. 实验环境

- 操作系统：Windows subsystem for linux, Ubuntu 18.04
- 编译器：gcc 7.3.0
- 编译工具：CMake 2.8 +

3. 文件目录和编译运行说明

3.1 文件结构

```
.
├── CMakelists.txt      //CMake文件
├── bin
│   └── chess5          //可执行文件
├── docs
│   ├── lab2-chess.md   //文档源码
│   ├── output_history  //下棋输出历史文件
│   │   └── ...
│   └── pics            //下棋截图
│       └── ...
├── include
│   ├── Chess5AI.hpp    //下棋AI
│   ├── IOManager.hpp   //屏幕输出和文件输出
│   └── my_debug.hpp    //调试用的宏
└── src
    ├── Chess5AI.cpp
    ├── IOManager.cpp
    └── main.cpp        //执行入口
```

3.1 编译运行办法

- 首先，切换至本项目目录下，然后输入如下命令完成编译：

```
cmake .
make
```

- make成功后，可执行文件在bin/chess5。
- 运行方式1：命令行 在当前文件夹下，运行：

```
./bin/chess5
```

然后，每次输入两个整数，空格隔开，表示下棋位置。

- 运行方式2：浏览器可视化 在当前目录下，运行如下命令：

```
cd visualizer
./run.sh
```

即首先切换到visualizer文件夹下，然后调用 ./run.sh 运行。运行需要pyhon3，安装有 websockets 库。 接着用浏览器打开visualizer/chess.html，点击棋盘位置即可下棋。

4. 实验过程

4.1 评估函数设计和理由

- 评估函数设计
  - 评估函数在没有一方胜利时，是白色和黑色的 **单侧评估值 E** 之差。在有一方胜利时，评估值直接到达最大或者最小。

$$P_{r,j} = \begin{cases} E_{white} - E_{black}, & \text{没人赢,} \\ 9999999 & \text{AI赢,} \\ -9999999 & \text{玩家赢.} \end{cases}$$

  - 定义：单侧评估值：
    - 是该方行/列/对角线上有 **扩展为5连子潜力** 的2连，3连，4连个数的加权求和。简称**潜力2连**， **潜力3连**，**潜力4连**。如果潜力2连有n_2个，3连有n_3个，潜力4连有n_4个，没有5连子，则评估函数值为：

$$E_{color} = n_{4Color} * 8000 + n_{3Color} * 400 + n_{2Color} * 10;$$

  - 定义：扩展为5连子潜力：
    - 在该3连或者4连的两边，有空白或者友军棋子，使得有连成5连的可能。
  - 定义：潜力3连：
    - 由3个棋子和两个空格或者棋子构成。比如：

```
X  X  X [ ][ ] 三个棋子两个空格
[ ] X  X  X [ ] 三个棋子两边都有空格
[ ][ ] X  X  X 三个棋子两个空格
```

        - 下面的就不是潜力3连：

```
o [ ] x   x   x   o   三个棋子无法下到5连
```

        - 注意，同样三个棋子，可能对应多个潜力3连。
        - 比如 `[ ] [ ] x x x [ ] o` 就对应如下的2个潜力3连：

```
[ ] [ ] x   x   x
    [ ] x   x   x [ ]
```

    - 潜力4连，潜力2连的定义和潜力3连类似。

- 评估函数设计原因

    - 博弈树深度有限，评估函数需要尽可能真实的反映棋盘形势。
    - 已经不会有赢棋机会的地方，没有价值，不应该影响评估函数。
    - 同样是3/4连子，一边被阻挡一边能下，和两边都能下的赢棋机会是不同的。简而言之，要判断的不仅仅是3连，4连的个数，而是包含3连，4连的有赢棋可能的连续5个空位或棋子的个数。
    - 在计算评估函数值时，不用太在意下一步该谁，因为评估函数值只要保证在这一层评估时,不同选择的相对序关系接近真实即可。

4.2 算法思想：MIN-MAX算法

- 算法先自顶向下展开，再自底向上收集计算MIN/MAX。
- MAX节点，取其子节点的最大值作为本节点评估函数值。
- MIN节点，取其子节点最小值作为本节点的评估函数值。

4.3 算法思想：Alpha-Beta剪枝

- 配合MINMAX算法，通过剪枝减少每一层扩展的节点数量。
- 在一个max节点下面，初始是-999999，已经至少求完了一个min节点，该节点的min值为a，则其余min节点一旦小于a（其余min节点的子max节点小于a）则跳过该min节点。
- 同理，在一个min节点下面，初始值是999999，已经求出一个max节点，该节点max值为b，则其余max节点如果值超过b，跳过该max节点。

4.3 算法优化：子节点访问顺序

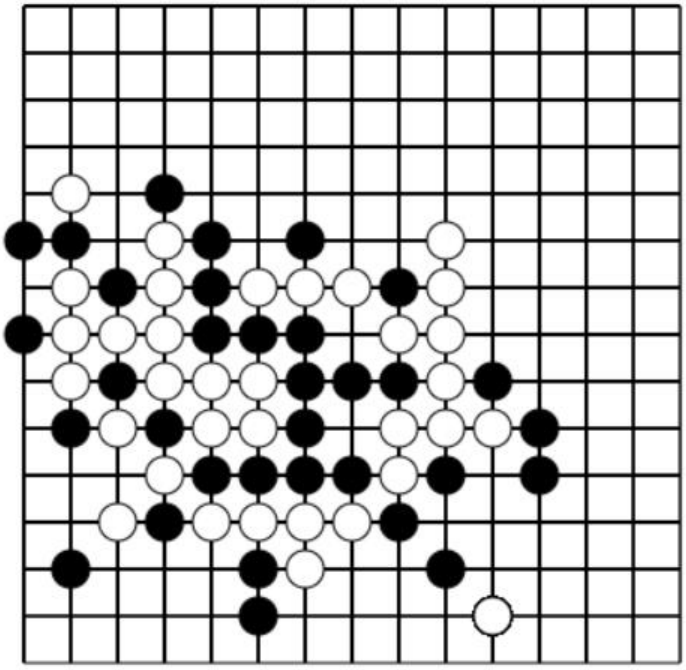
- 通过改变展开子节点的顺序，每次都先访问猜测的最优节点，则在最好情况下，其它子节点都会很快被剪掉。
- 关键问题是最优解点未知，于是按照猜测的值（评估函数值）进行排序。排序后，按照评估函数值从最好到最差进行展开。
- 所以需要在展开之前计算评估函数值。可以先临时将棋子放在棋盘上，计算完评估函数值之后再复原。

5. 实验结果

5.1 和AI对弈

- AI胜利1(白棋)

ws_input MSG: #RE: PATH Recieved: (10, 9)
ws_output CLOSED.
选择: (0, 0)



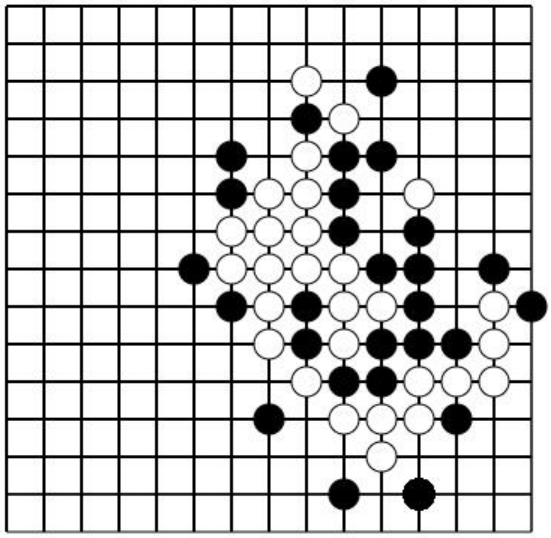
SHUTDOWN

输出文件:

```
AI       ME
[9,5]   [10,4]
[8,4]   [10,6]
[7,3]   [6,2]
[8,3]   [7,4]
[8,5]   [8,2]
[6,3]   [9,3]
[5,3]   [4,3]
[11,5]  [10,5]
[10,3]  [10,7]
[10,8]  [9,6]
[9,4]   [7,6]
[11,2]  [12,1]
[11,4]  [11,3]
[11,6]  [8,6]
[6,6]   [8,7]
[6,5]   [7,5]
[11,7]  [11,8]
[6,7]   [6,4]
[12,6]  [12,9]
[13,10] [13,5]
[9,9]   [8,10]
[7,8]   [8,8]
[8,9]   [5,6]
[9,10]  [10,11]
[9,8]   [9,11]
[9,2]   [12,5]
[8,1]   [7,0]
[7,2]   [5,4]
[6,1]   [5,0]
[7,1]   [9,1]
[4,1]   [5,1]
[6,9]   [6,8]
[7,9]   [10,9]
[5,9]
AI Win!
```

- AI胜利2(白棋)

ws_input MSG: #RE: PATH Recieved: (7, 10)
ws_output CLOSED.
选择: (7, 10)



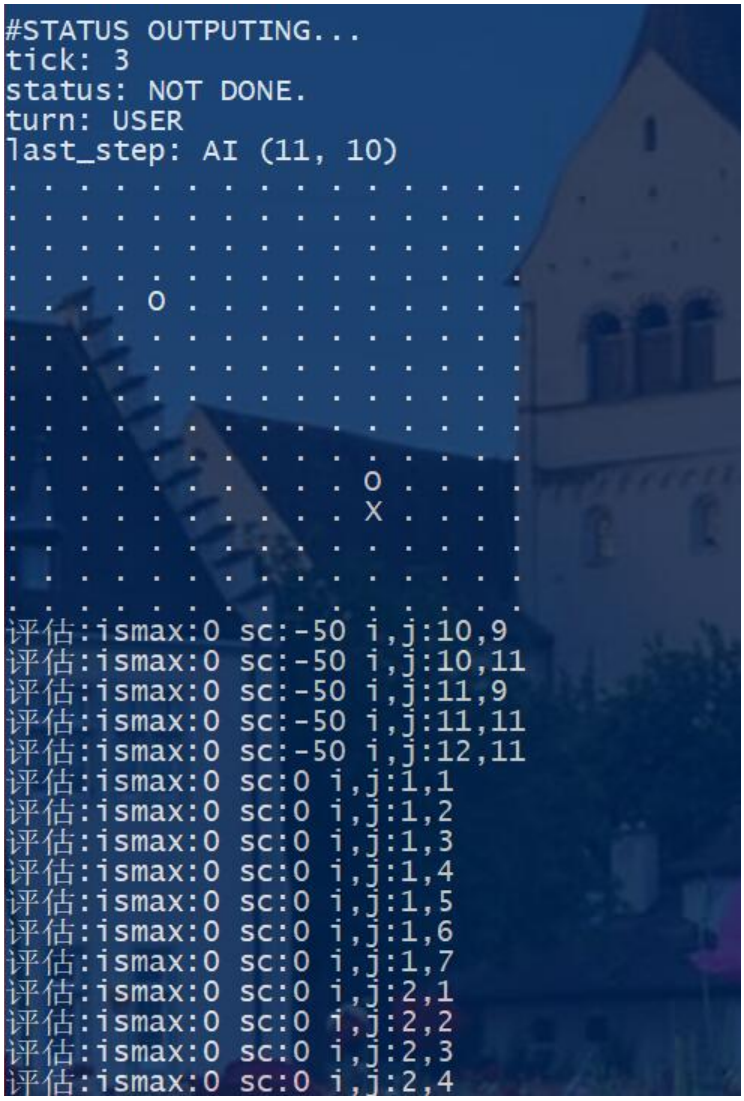
SHUTDOWN

输出文件:

```
AI       ME
[9,9]   [8,8]
[10,8]  [11,7]
[9,7]   [9,8]
[11,9]  [8,6]
[8,9]   [10,9]
[8,10]  [7,11]
[7,9]   [6,9]
[6,8]   [9,11]
[12,10] [13,11]
[11,10] [10,10]
[11,11] [11,12]
[10,12] [13,9]
[9,13]  [8,14]
[8,13]  [8,11]
[10,11] [6,11]
[5,11]  [9,12]
[10,13] [7,13]
[5,7]   [4,6]
[5,8]   [5,9]
[4,8]   [3,8]
[3,9]   [2,10]
[2,8]   [4,10]
[6,6]   [7,5]
[6,7]   [5,6]
[7,6]   [4,9]
[7,8]   [9,10]
[7,7]   [7,10]
```

```
[8,7]
AI Win!
```

- 剪枝顺序优化示意：



5.2 棋力分析

和AI下了多轮，AI屡战屡胜，棋力还可以。AI会守会攻，攻击时常常行程很多连3，防守时也很有意识，有3连棋就开始防守了；而且AI也很偏好那些同时给自己局面加分以及给对手减分的位置。
虽然很难下赢AI，但是自己棋力也有提高。

6. 思考题

- 思考搜索的深度对AI的决策效率有何影响？如何利用搜索深度提高AI的智能程度？
  - 搜索空间大小随着搜索深度增加指数上升，因此搜索深度上升，决策速度下降。
  - 在评估函数确定的情况下，AI的智能程度随着搜索深度的增加而提高，因为对棋局的评估更加准确了。
- Alpha-Beta剪枝法在减枝过程搜索效率与节点的排列顺序有很大关系。思考是否可以改进剪枝策略提高决策速度？
  - 可以。可以在展开博弈树子节点之前，先计算各个子节点的评估函数值，然后根据评估函数值排序，先展开评估函数值最好的那个节点。只要评估函数写得好，评估值的较优者，很可能也是指定深度博弈树中该节点孩子中较优者。该方案已经实现，使得可行博弈树深度增加了1~2.
- 思考是否有方法实现AI的自学习能力，让AI不在相同的地方犯错？本题只需要给出思路，不需要具体实现。
  - 计算一个唯一的棋盘编号值，然后将棋盘编号值作为key，最后的输赢局面对应的评估值（大正数或者大负数）存入一个特殊棋局评估值哈希表。并且，借助求出的博弈树，使用“反向传播”，将这个棋局的状态值在博弈树上回传，逐层使用比例不同的线性插值来修正棋局之前状态的评估值（越靠近最终局面的棋局，评估值修正的越多），并将修正后的值也同样存入或者替换哈希表项。
  - 下次，在计算评估函数时，首先根据当前棋局算出的唯一棋盘编号值，查哈希表，找到条目就用哈希表中的值代替。

6. 实验总结

通过本次实验，亲手实践了博弈树上的Alpha-Beta剪枝，体会到了算法的时间复杂度，和节点扩展顺序对于性能的影响。另外，也学习了使用python, websocket, js来做简易图形界面。