Alpha-Beta剪枝求解五子棋AI问题

人工智能原理与技术课程

汇报人：龚哲飞 指导老师：王俊丽 2022.5.12

目录

content

第一部分

实验概述

实验概述

实验目的

熟悉和掌握博弈树的启发式搜索过程、α－β剪枝算法和评价函数，并利用α－β剪枝算法开发一个五子棋人机博弈游戏。

实验内容

以五子棋人机博弈问题为例，实现α-β剪枝算法的求解程序（编程语言不限），要求设计适合五子棋博弈的评估函数。

实验要求

1.要求初始界面显示15*15的空白棋盘，电脑执白棋，人执黑棋，界面置有重新开始，悔棋等操作。

2.设计五子棋程序的数据结构，具有评估棋势、选择落子、判断胜负等功能。

3.撰写实验报告，提交源代码（进行注释）、实验报告、汇报PPT。

第二部分

实验设计方案

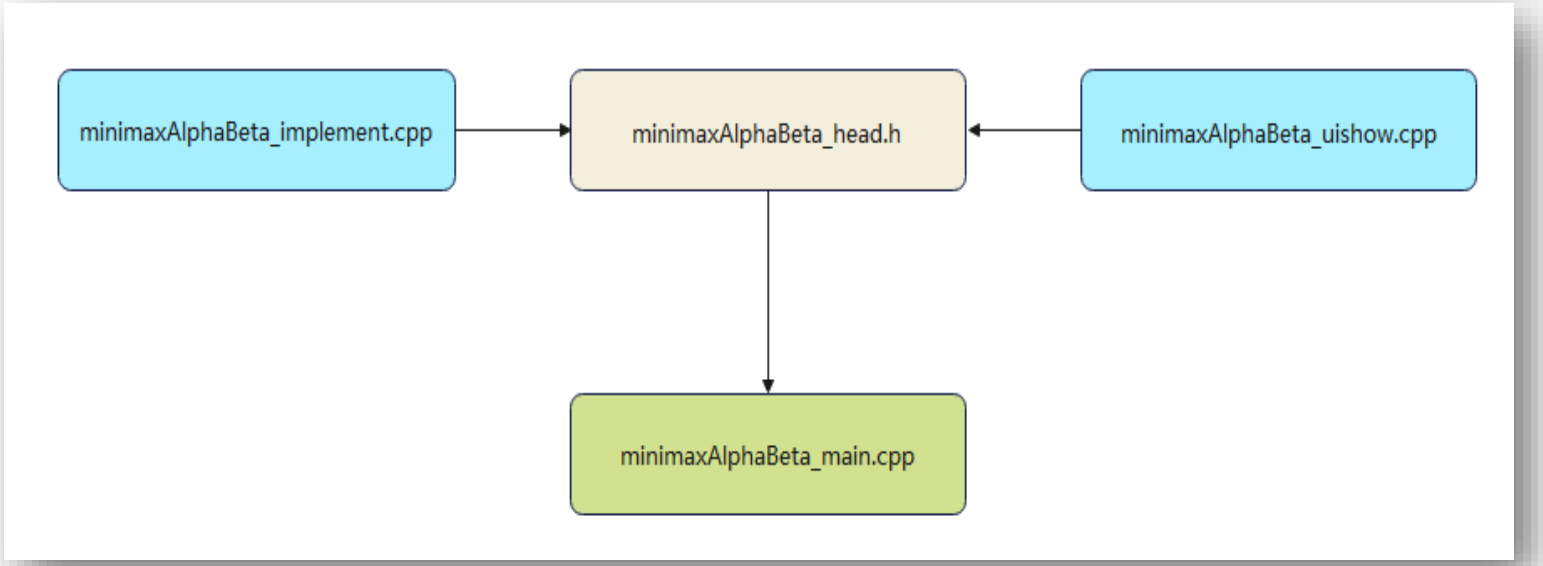
总体设计思路与总体框架

minimaxAlphaBeta_head.h

minimaxAlphaBeta_implement.cpp

minimaxAlphaBeta_main.cpp

minimaxAlphaBeta_uishow.cpp



总体框架

总体设计思路与总体框架

minimaxAlphaBeta_head.h

存储整个项目类、结构体、函数、宏定义的声明，作为项目的头文件使用。

内含*seearchAlphaBeta*类的具体实现函数，实现五子棋游戏的内部核心算法的求解。

minimaxAlphaBeta_implement.cpp

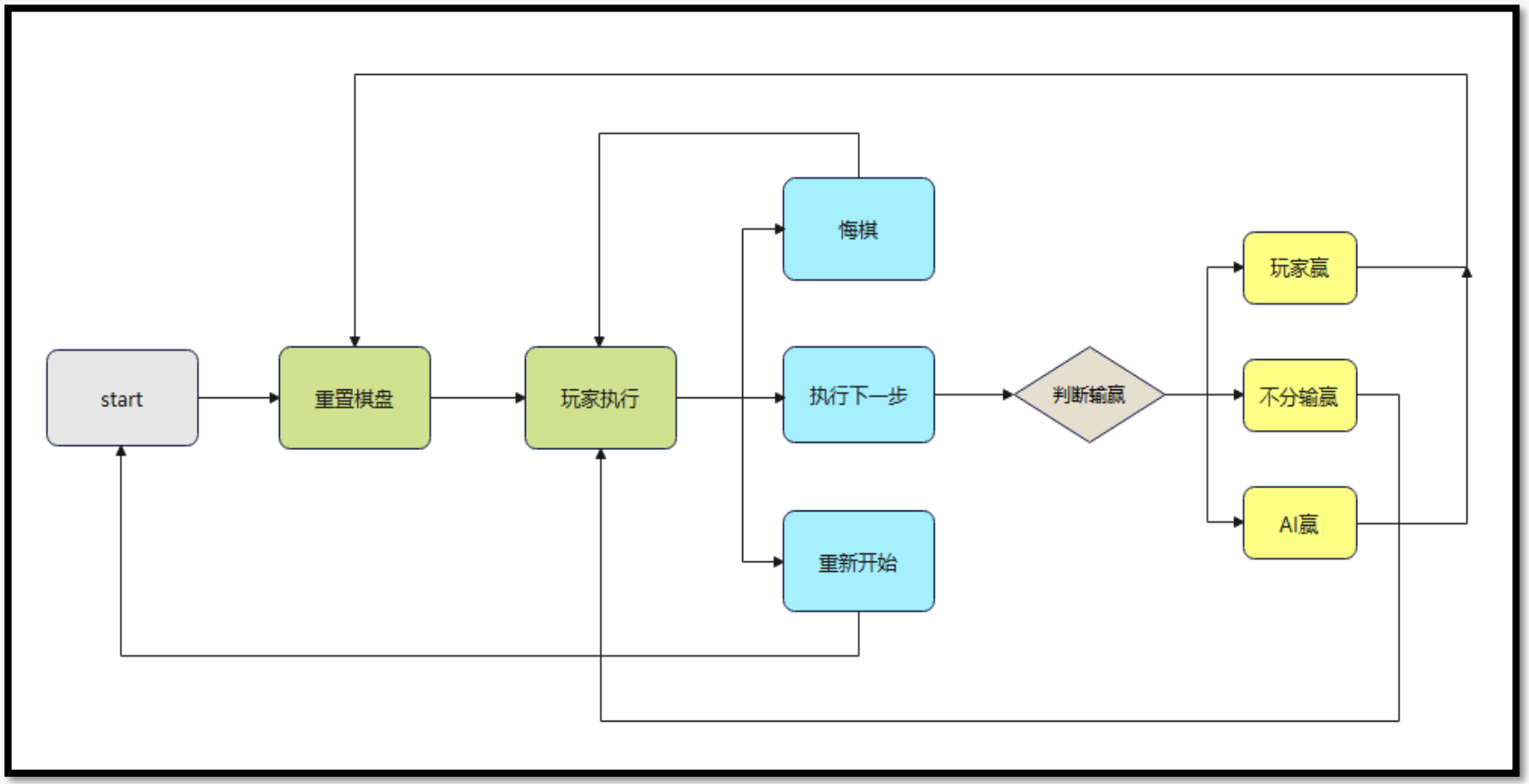
minimaxAlphaBeta_uishow.cpp

内含*showUI*类的具体实现函数，实现五子棋游戏的可视化界面输入输出。还包含部分整个游戏所需工具函数的具体实现。

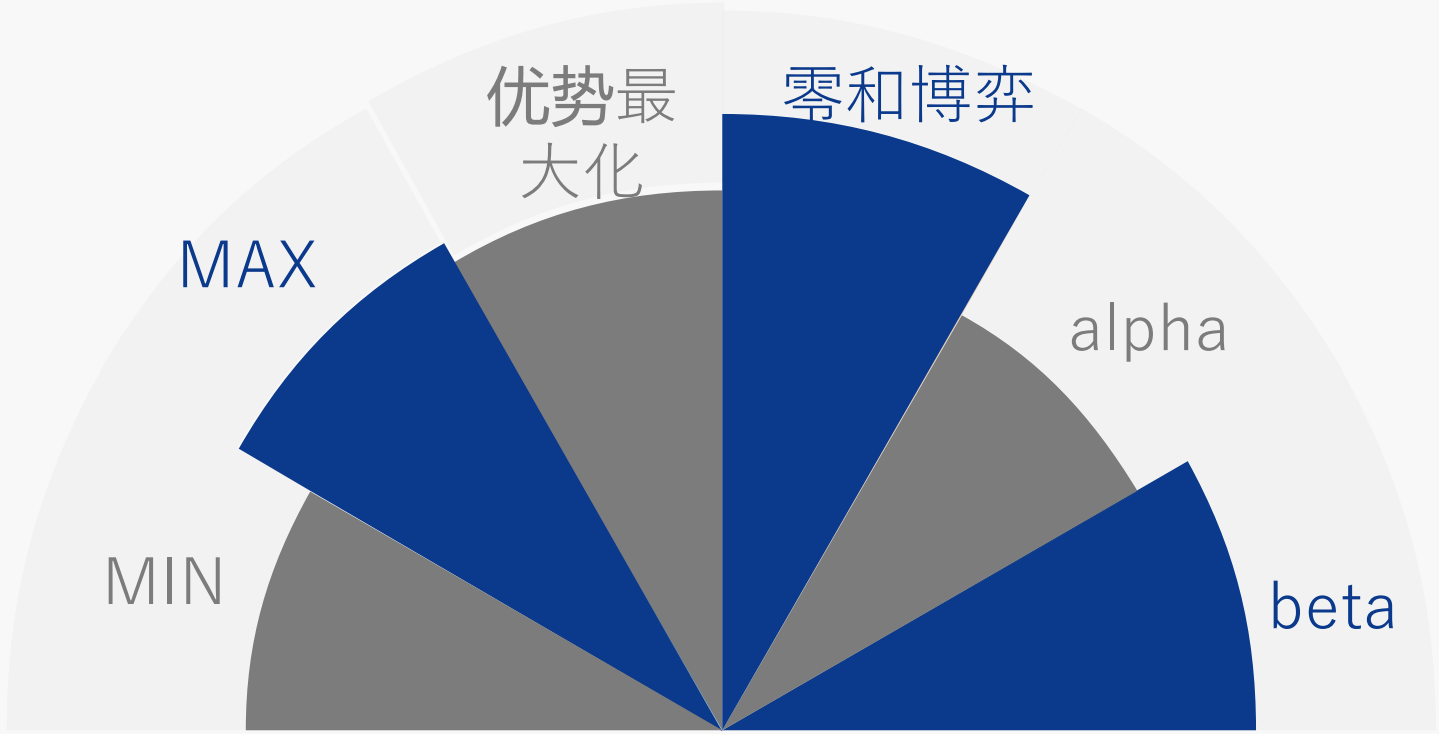
程序主函数所在地，调用各个类进行对五子棋问题的求解。

minimaxAlphaBeta_main.cpp

总体设计思路与总体框架



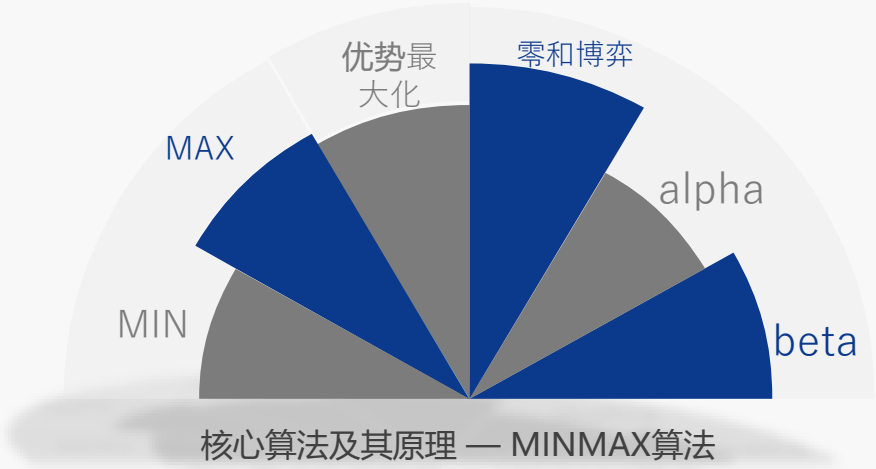
运行流程



核心算法及其原理 — MINMAX算法

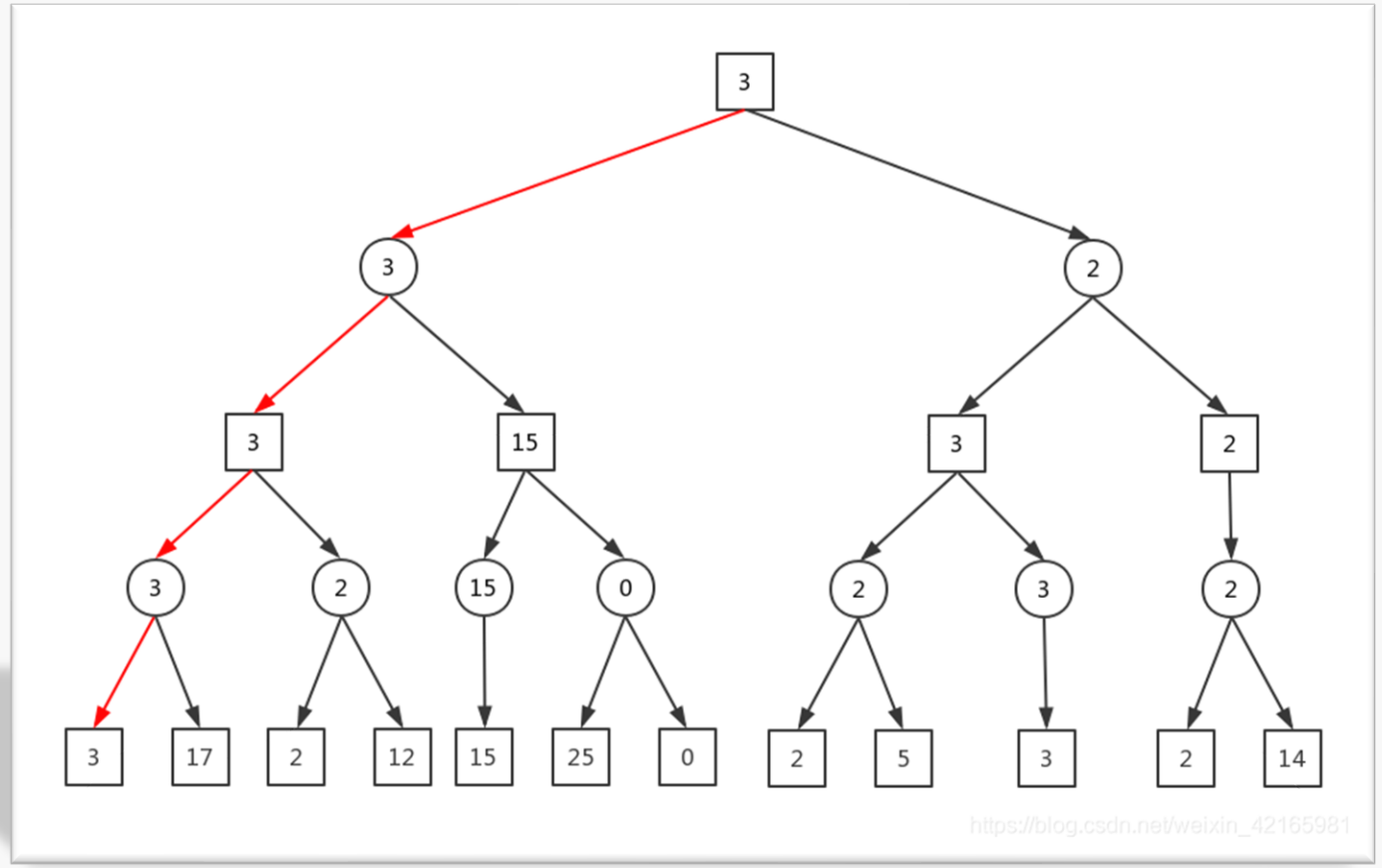
Minimax算法（亦称 MinMax or MM）是一种找出失败的最大可能性中的最小值的算法。

Minimax算法常用于棋类等由两方较量的游戏和程序。该算法是一个零总和算法，即一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的方法。而开始的时候总和为0。



核心算法及其原理 — MINMAX算法

MINMAX搜索

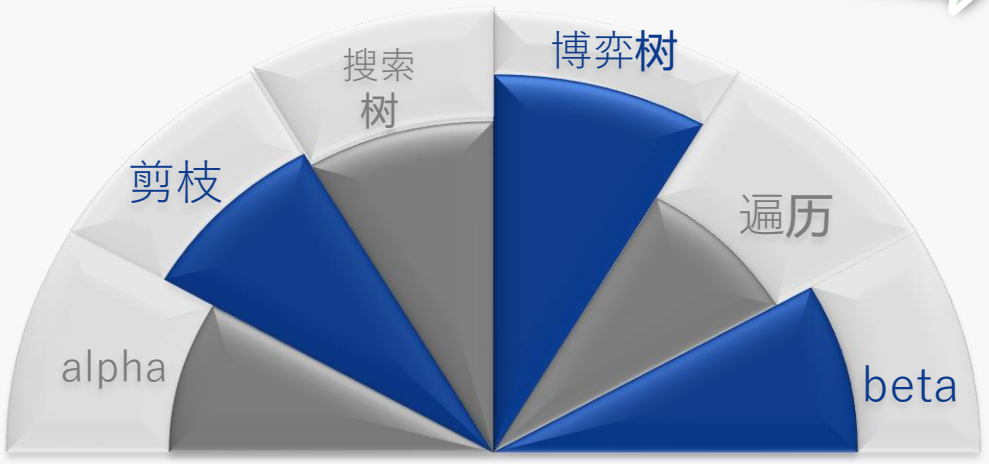


Alpha–beta(α – β)剪枝的名称来自计算过程中传递的两个边界，这些边界基于已经看到的搜索树部分来限制可能的解决方案集。其中，Alpha(α)表示目前所有可能解中的最大下界，Beta(β)表示目前所有可能解中的最小上界。

因此，如果搜索树上的一个节点被考虑作为最优解的路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N是当前结点的评估值）：

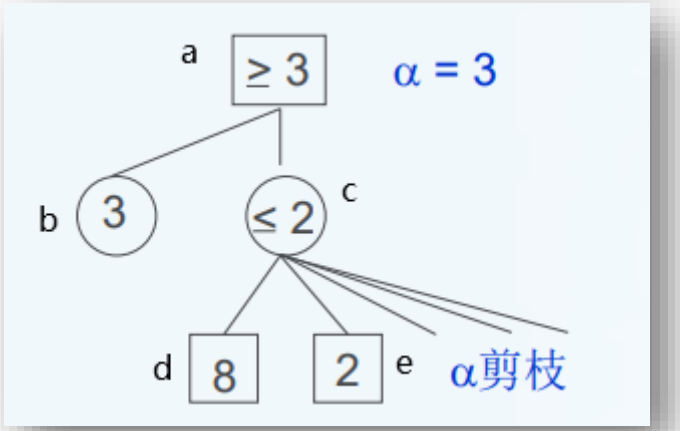
$\alpha \leqslant N \leqslant \beta$

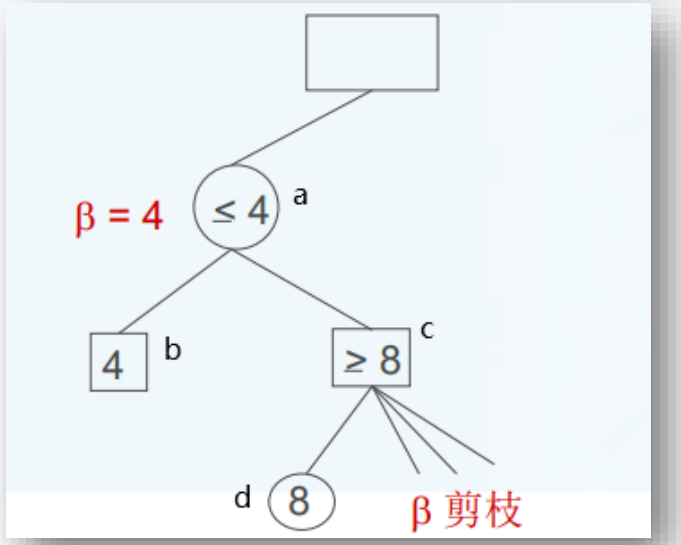
在我们进行求解的过程中，α和β 会逐渐逼近。如果对于某一个节点，出现了α > β >的情况，那么，说明这个点一定不会产生最优解了，所以，我们就不再对其进行扩展（也就是不再生成子节点），这样就完成了对博弈树的剪枝。

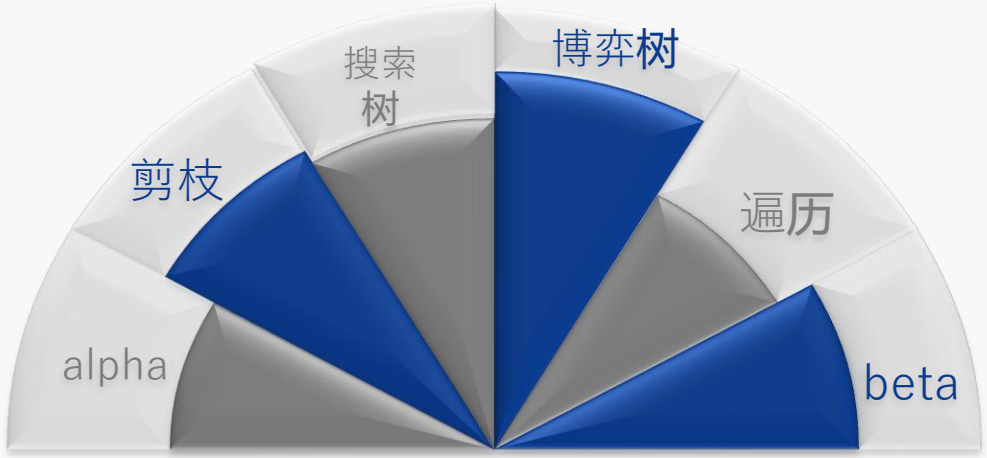


因此，在搜索最优解的过程中，每当需要搜索同级结点过程中，可以根据Alpha–beta进行剪枝，消除其余无需遍历的结点。

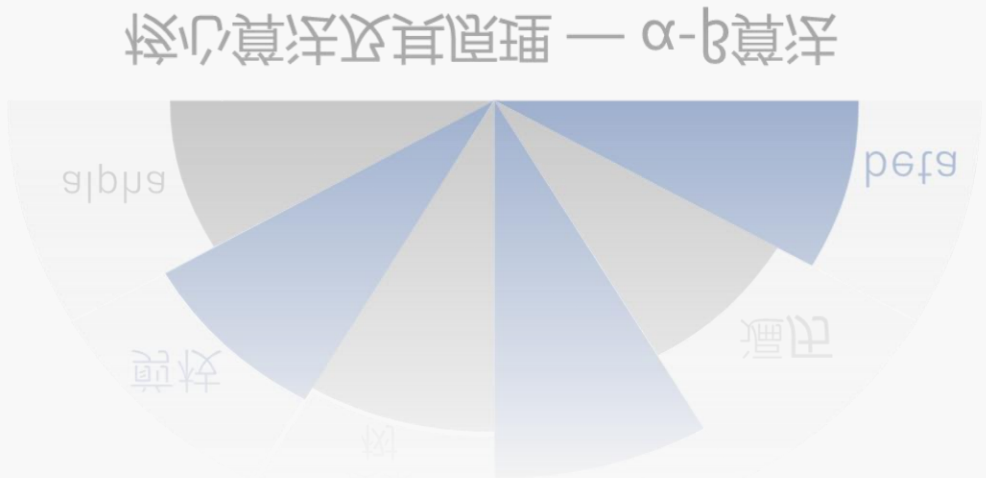
核心算法及其原理 — α-β算法







核心算法及其原理 — α-β算法



算法实现步骤

若遍历置叶子结点，则直接返回局面的全局评估值

若层数为奇数，则进入MIN结点判别
①调用增益评估函数，选择n个后继结点进行遍历
②依次遍历该结点的n个后继结点，并根据结点信息修改局面信息
③对每个后继调用方法alphabetaAlgorithm，返回value值
④取value和结点β值的较小值作为该结点新β值
⑤若该结点的α值大于等于该结点β，则无需继续遍历子结点（跳出循环）=> α剪枝

若层数为偶数，则进入MAX结点处理判别
①调用增益评估函数，选择n个后继结点进行遍历
②依次遍历该结点的n个后继结点，并根据结点信息修改局面信息
③对每个后继调用方法alphabetaAlgorithm，返回value值
④取value和结点α值的较大值作为该结点新α值
⑤若该结点的α值大于等于该结点β，则无需继续遍历子结点（跳出循环）=> β剪枝

评估函数算法设计

全面评估

棋型说明	权重		棋型说明	权重
白连五	1000000		黑连五	-10000000
白活四	50000		黑活四	-100000
白冲四	400		黑冲四	-100000
白活三	400		黑活三	-8000
白眠三	20		黑眠三	-50
白活二	20		黑活二	-50
白眠二	1		黑眠二	-3
白活一	1		黑活一	-3

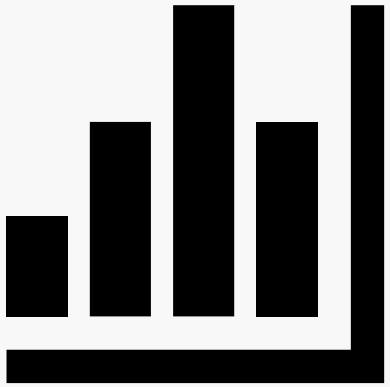
针对不同的局面给出不同的评估值，有利于使得五子棋AI更加智能，有效选择更有利的步骤。因此，根据查阅资料，对五子棋棋局的评估常采用六元组的形式进行判别，即对六个点位的信息进行思考，再次基础上发现五子棋局面总共分成如下情况(以黑棋为例):

①连五/长连：即存在五个或者六个连续的黑棋，即黑棋赢
②活四：有两个位置可以形成连五
③冲四：有一个位置可以形成连五
④活三：走一步可以形成活四
⑤眠三：走一步可以形成冲四
⑥活二：走一步可以形成活三
⑦眠二：走一步可以形成眠三
⑧活一：走一步可以形成活二

因此根据黑棋和白棋在以上不同情况，给予不同的分数。

评估函数算法设计

增益评估



对再是的历。位因计下；层法数索遍的。复右评每无个搜下点益重上-组若然点下向落增的左元，显结向续择分局；六出这子对继。选得棋-右；窗看，归要点数。以的原左滑以估递需结函可后对下；行可评为们些估可个前免上-进行因我某评个益每子避进，定增对落以位棋点果此确的指出可该的结结因以们是给，对向空出。，我数，想要方即可。的得长估要函估思需个四结果能内增评需估评此只四上后可间级行便评行据据有时数进，益进根分。-右前根所定指点时增，，得下子于一呈结此置此算左落

模块设计

```
//@function:alpha-beta剪枝搜索类
class searchAlphaBeta {
protected:
    NODE head;
    char board[BOARDWIDTH + BOARDADD][BOARDLENGHT + BOARDADD] = { '\0' };
protected:

    int getScoreFromString(string& get);                                                                                    //得到一个string串的的嗯
    int toneUpAssessIn(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& site);                                         //增益函数内部评估
    void toneUpAssess(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& site, char sort);                               //增益评估
    bool searchThreeMust(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& get, char chessSort);                        //白棋活三特判
    bool searchFiveMust(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& get, char chessSort);                         //五子连珠特判
    bool fiveMust(string& get, char chessSort);                                                                             //五子连珠特判in
    bool firstLayerAssess(char(*chessBoard)[BOARDLENGHT + BOARDADD], int& feeBack);                                         //第一层首先评估
    bool chessAround(int row, int col, char(*chess)[BOARDLENGHT + BOARDADD]);                                               //判断NONECHESS周围是否存在CEHSS
    bool findBorder(char(*chessBoard)[BOARDLENGHT + BOARDADD], BORDER& bor);                                                //找到搜索边界
    int eachGet(string& extract);                                                                                           //评价函数中对不同的情况返回不同得分
    int assessBoardValue(char(*chessBoard)[BOARDLENGHT + BOARDADD]);                                                        //全局评估函数
    bool seekBestPoint(char(*chessBoard)[BOARDLENGHT + BOARDADD], char chessClass, vector<NODESCORE>& result);              //找到较优的几个点进行针对性拓展
    int alphabetaAlgorithm(char(*chessBoard)[BOARDLENGHT + BOARDADD], int depth, int alpha, int beta);                      //进行alpha-beta剪枝搜索

public:
    searchAlphaBeta(char(*chessBoard)[BOARDLENGHT + BOARDADD]);                                                             //构造棋盘函数
    int alphabetaGetBack(void);                                                                                             //执行函数
};
```

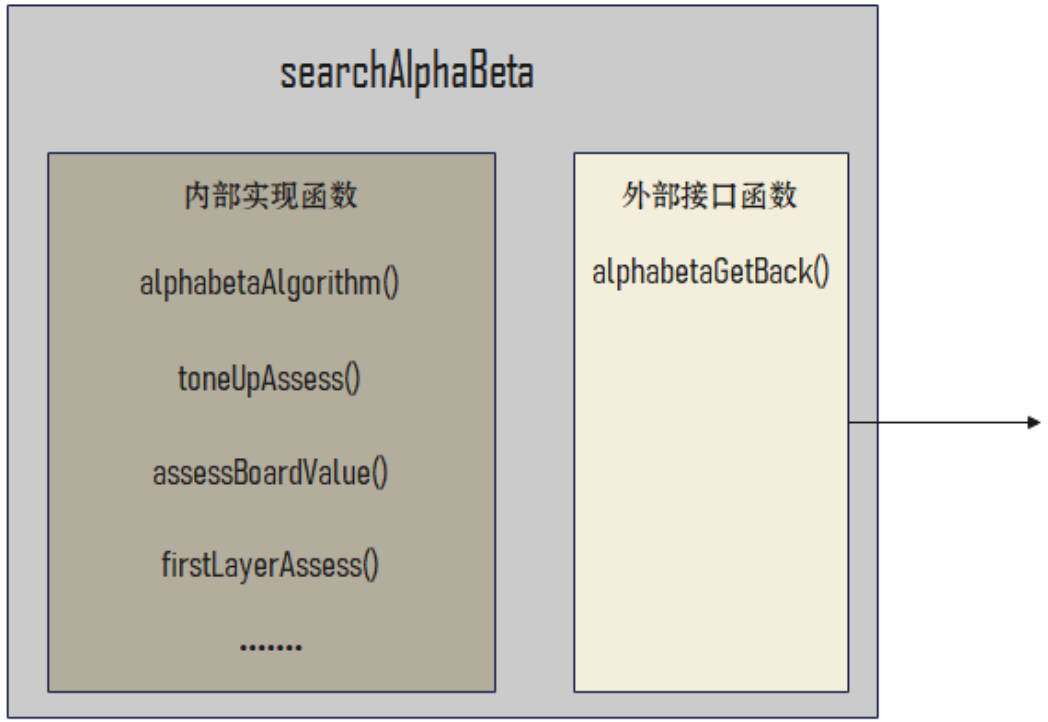
01 searchAlphaBeta模块

模块设计

```
//@function : 图像UI界面显示类
class showUI {
protected:

    //界面布局结构体定义
    SURFACE boardImage;
    SURFACE regretButton;
    SURFACE restartButton;
    SURFACE pictureSite;
    SURFACE winloseShow;

    //界面图片调用定义
    IMAGE backGround;
    IMAGE chessBoard;
    IMAGE chessBlack;
    IMAGE chessWhite;
    IMAGE chessWhitePre;

    IMAGE regretImage;
    IMAGE regretMouseImage;
    IMAGE restartImage;
    IMAGE restartMouseImage;
    IMAGE thinkingImage;
    IMAGE userImage;

    IMAGE winImage;
    IMAGE loseImage;

    bool findXY(int x, int y, int& boardX, int& boardY);          //找到xy所对应在棋盘中的位置
    void endShow(void);                                           //结束后只能按下重新开始按钮
public:
    showUI(void);                                                 //构造函数
    bool refresh(char(*chessBoard)[BOARDLENGHT + BOARDADD]);      //重置界面情况
    bool uiLoad(void);                                            //加载背景界面函数
    int uiGetBack(char(*chessBoard)[BOARDLENGHT + BOARDADD]);     //单步下棋执行函数
    bool addWhite(int site);                                      //在棋盘上添加白棋
    void winShow(void);                                           //成功显示
    void loseShow(void);                                          //失败显示
    void showThinking(void);                                      //切换图片
    void showNormal(void);                                        //切换图片
};
```

02 showUI模块

模块设计



01 searchAlphaBeta模块

02 showUI模块

创新算法--限定单结点向下遍历个数

根据递归向下剪枝可知，若每层不存在剪枝运算，将搜索结点个数将呈现指数级增长态势，因此随着个数的增加，我们将无法在一定时间内得到结果，也就丧失了人机对战的意味。故显然需要进行不完美的实时决策，对非终止结点打分。此时需要增益评估函数，对非终止结点进行完善的评估，选择出更为得分更高，更为优秀的结点继续向下遍历。

```
#define EACHSEARCH     20     // 定义每层的探索个数
```

```
//进行赋值返回
for (int i = 0; (i < (int)Total.size() && i + 1 <= EACHSEARCH); i++)
    result.push_back(Total[i]);
```

创新算法--空位落子策略

每次落子时，棋盘上的空白理论上均可落子，但根据查阅资料可知，仅在已有结点周围八个方向延申至多三层的结点对于本局有利，因此，设定chessAround函数，判定该空位周围是否在一定范围内存在棋子，即判断位置的拓展可行性。

```
#define DIRECTIONINNER 8    //8个方向上的节点进行拓展
//#define DIRECTIONOUTER 16
```

```
//@function:判断周围是否存在棋子
bool searchAlphaBeta::chessAround(int row, int col , char (* chess)[BOARDLENGHT + BOARDADD])
{
#ifdef DIRECTIONINNER
    for (int i = row - 1; i <= row + 1; i++) {
        for (int j = col - 1; j <= col + 1; j++) {
            if ((i >= 1 && i <= BOARDWIDTH) && (j >= 1 && j <= BOARDLENGHT) && (i != row || j != col)) {
                if(chess[i][j]!='*')
                    return true;  //存在棋子
            }
        }
    }
    return false;                 //不存在棋子
#endif
}
```

创新算法--优先队列增大剪枝概率

我们了解到alpha-beta剪枝将极大的优化效率，剪掉越多的结点可以获得更高效的搜索效率。对于alpha-beta剪枝有一个最优的剪枝状态，即优先搜索到极大或者较小得分值。

对于MIN结点，下一步走黑棋，首先搜索到的为得分最少的结点。当出现alpha剪枝时，便可在最大程度上剪去不必要结点。

对于MAX结点，下一步走白棋，首先搜索到的为得分最优的结点，当出现beta剪枝时，便可在最大程度上剪去不必要结点。

我们在seekBestPoint方法中实现这一过程。

```
//进行降序排序
if (chessClass == BLACKCHESS)          //黑棋升序-->易于剪枝
    sort(Total.begin(), Total.end(), compareAscend);
else if (chessClass == WHITECHESS)   //白棋降序-->易于剪枝
    sort(Total.begin(), Total.end(), compareDecend);

//进行赋值返回
for (int i = 0; (i < (int)Total.size() && i + 1 <= EACHSEARCH); i++)
    result.push_back(Total[i]);

return true;
```

创新算法--第一层初判

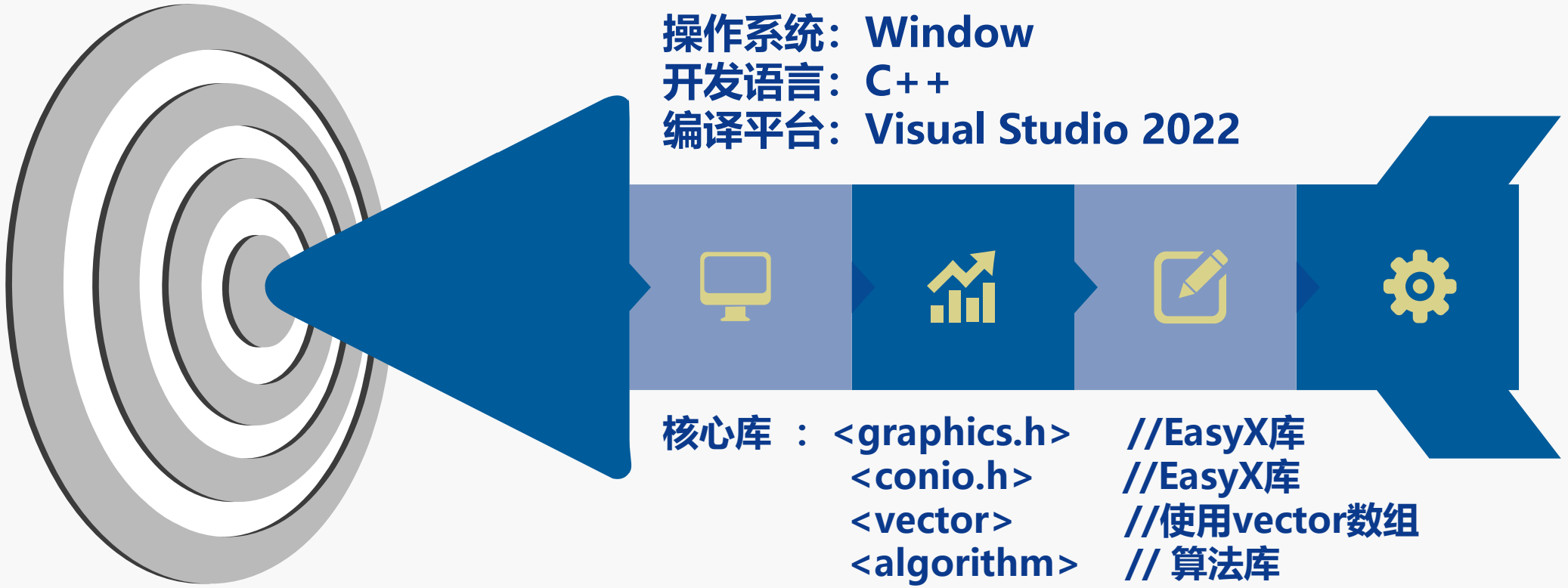
对于搜索，有些选择在第一层便可以得出，该选择可以为必走制胜步骤或必走抵抗步骤，而无需继续向下搜索。因此，在原有搜索的基础上，增加一个首层判断过程，将白棋能制胜的必走棋或抵抗黑棋的必走棋或白棋形成活四的必胜棋首先判断，若存在这样的必走棋，则无需再向下遍历，直接走该位置即可。

```
//@function:找到较优的探索结点（局部搜索+静态启发+剪枝）
bool searchAlphaBeta::seekBestPoint(char(*chessBoard)[BOARDLENGHT + BOARDADD], char chessClass, vector<NODESCORE>& result){ ... }

//@function:第一层白棋活三特判
bool searchAlphaBeta::searchThreeMust(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& get, char chessSort){ ... }

//@function:第一层五子连珠特判滑窗特判(五元组)
bool searchAlphaBeta::fiveMust(string& get, char chessSort){ ... }

//@function:第一层五子连珠特判
bool searchAlphaBeta::searchFiveMust(char(*chessBoard)[BOARDLENGHT + BOARDADD], NODESCORE& site, char chessSort){ ... }

//@function:第一层特判,如果存在必下棋，则返回必下棋局位置—>针对白棋视角
bool searchAlphaBeta::firstLayerAssess(char(*chessBoard)[BOARDLENGHT + BOARDADD], int& feeBack){ ... }
```

第三部分

实验过程

环境说明

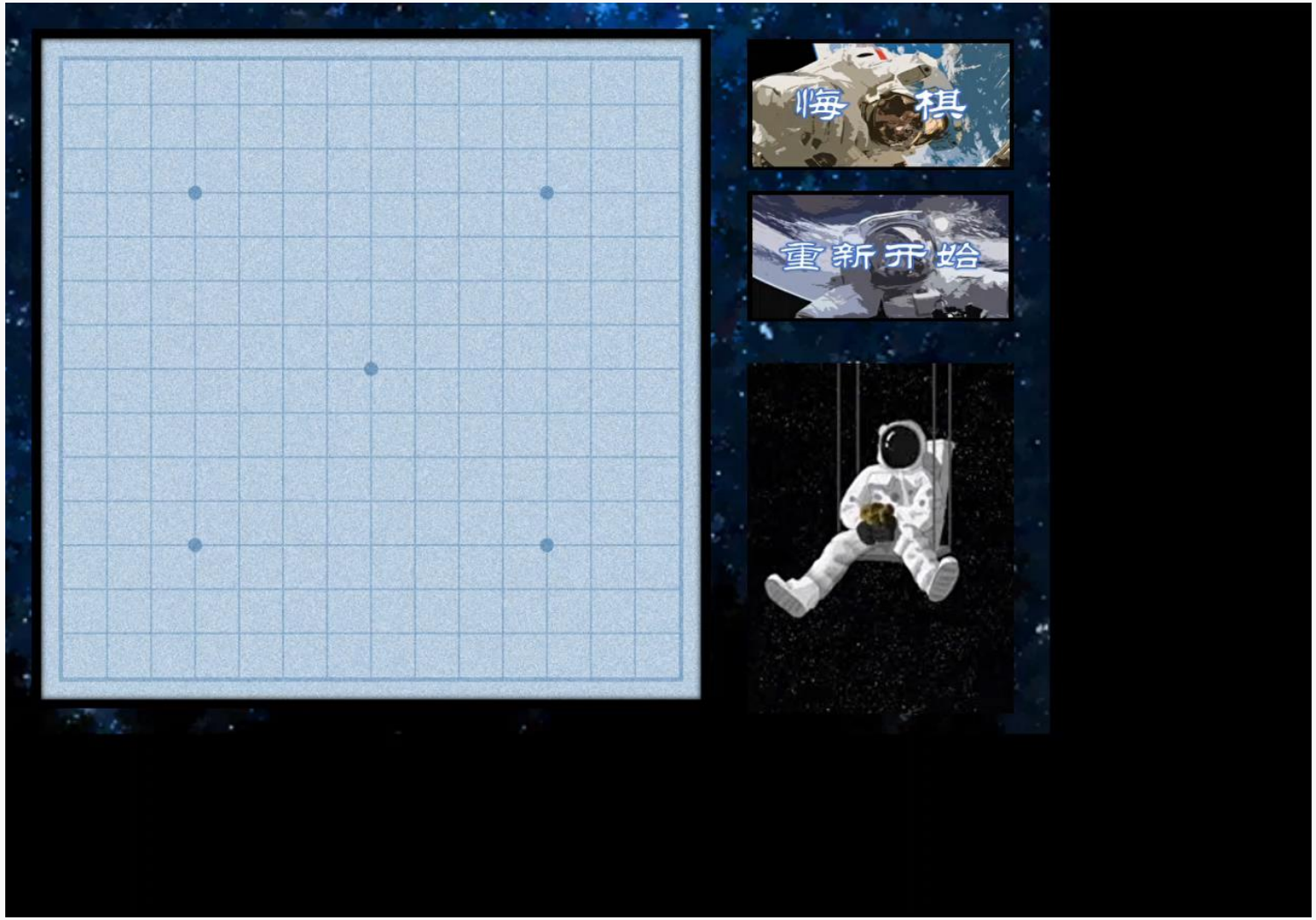


操作系统：Window
开发语言：C++
编译平台：Visual Studio 2022

核心库 ：

```
<graphics.h>    //EasyX库
<conio.h>       //EasyX库
<vector>        //使用vector数组
<algorithm>     // 算法库
<windows.h>     //sleep函数
<time.h>        //计算运行时间
```

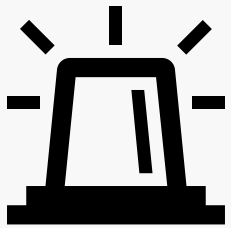
成果展示



结果分析

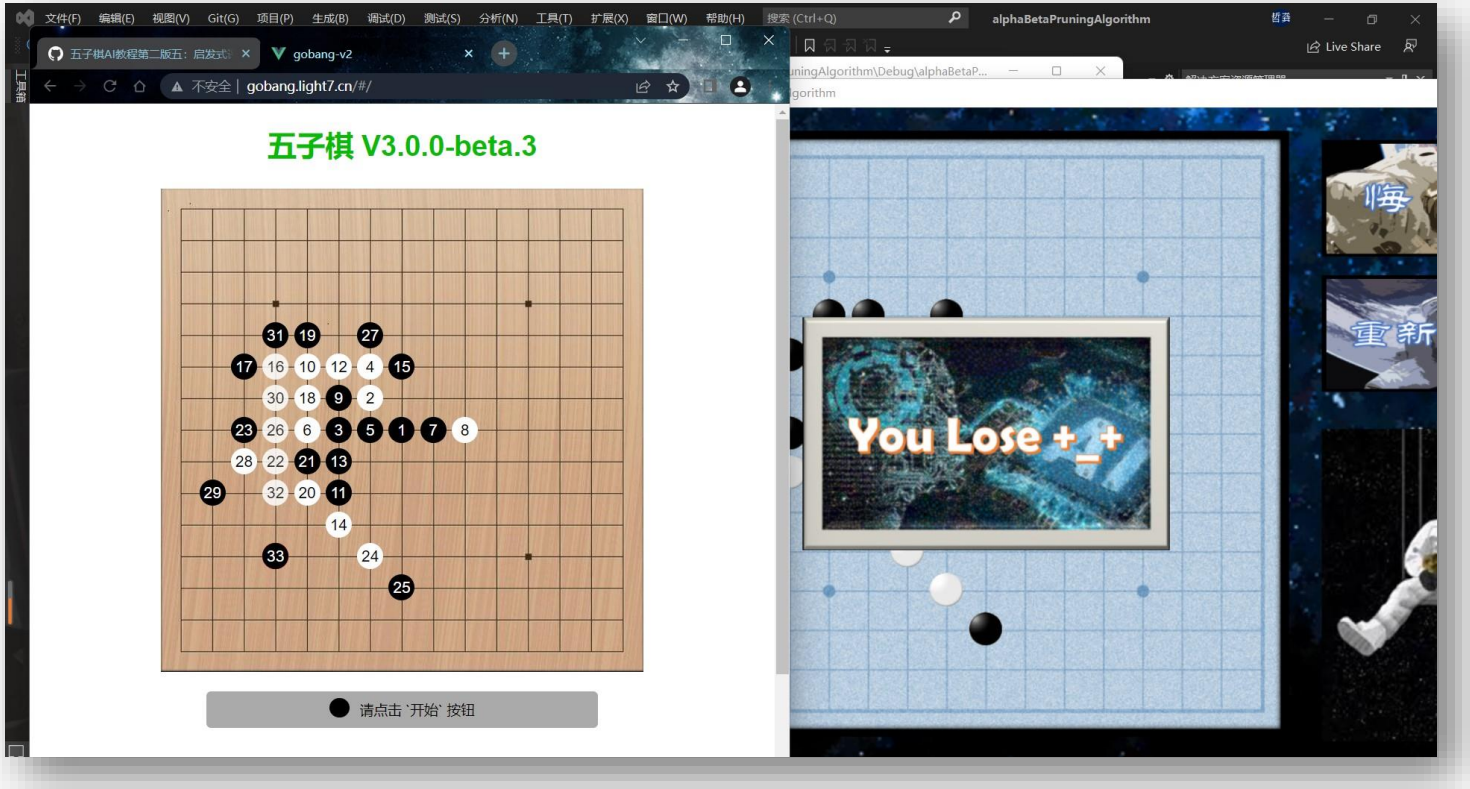
深度	每层至多个数	平均执行时长
3	20	2.5s
3	10	1s
4	10	6s
4	5	2.6s
6	4	7.8s
6	2	1.3s

根据表格可以看出，每层个数将极大影响计算个数，并且在个数不变，深度增加的情况下，搜索时间将呈现指数级增长。



但是经过测试发现，随着减少每层个数，由于采用逐层贪心，选择最优结点的方案，对AI智能的影响不大。并且若深度越深，AI的智慧增长情况显然能够弥补个数减少的影响。因此，其实深度为6层，每层个数为2便可以营造高速高智慧的AI，对局体验极好，暂时也无法赢过AI。

结果分析



战胜github star数最多的五子棋AI

根据实验结果可以看出，本次五子棋AI的实现效果极好，实现了高速反馈、高智选择的较高要求。根据分析，影响AI智慧的成败在于评估函数的选择和构建；影响AI速度的决定性因素在于α-β剪枝算法的实现程度。因此，可以看出，本次构建的评估函数是较为完备的；并且设计的剪枝优化算法也是能相当高效的剪去不必要的计算过程，提升整体效率。

第四部分

总结

总结

问题

整个实验思考最久的就是五子棋AI的优化上了。最初实现全体可落子位置的搜索时，程序执行效率极低，常常需要等待1-2min才能呈现结果，这也促使我进一步优化程序。在深入理解α-β剪枝算法后，阅读较多文献后，设计出的优化算法能较好的实现人机对局，且反应快，智商高。

体会

经过本次实验，我对于α-β剪枝算法的实现过程有了更加清晰的认识。评估函数的选择极大程度决定构造AI的智慧水平，因此选择较为完备合理的局面评估函数，将在一定程度上，提升AI的智慧水平。

今后

后续可以在评估函数的选择上提升，查阅更多文献，选择更完备的评估函数，进一步提升AI的智慧。还可以设计更好的优化算法，在进一步提升AI思考深度的基础上，保证较高的执行效率。

参考文献

[1] 赵美勇,宋思睿.博弈论算法在AI中的应用[J].计算机产品与流通,2019(09):278.

[2] 郑培铭，何丽．基于计算机博弈的五子棋AI设计[J]．电脑知识与技术,2016,12(33):80-81+90.DOI:10.14004/j.cnki.ckt.2016.4580.

[3]陈树彬，和昱旻，原菊梅．五子棋落子算法的研究[J]．电脑与信息技术,2021,29(05):49-51+94.DOI:10.19414/j.cnki.1005-1228.2021.05.014.

[4] 李昊． 五子棋人机博弈算法优化研究与实现[D]．大连海事大学,2020.DOI:10.26989/d.cnki.gdlhu.2020.000523.

[5] 沈雪雁. 基于蒙特卡洛树与神经网络的五子棋算法的设计与实现[D].沈阳化工大学,2021.DOI:10.27905/d.cnki.gsghy.2021.000057.

[6] 刘阳. 基于人工智能的五子棋专家系统研究和设计[D].电子科技大学,2015.

[7] 林华． 基于Self-Play的五子棋智能博弈机器人[D]．浙江大学,2019.DOI:10.27461/d.cnki.gzjdx.2019.000894.

感谢聆听

人工智能原理与技术课程

汇报人：龚哲飞　指导老师：王俊丽　2022.5.12