**实验报告**

|  |  |
| --- | --- |
| **课程名称** | **人工智能** |
| **实验项目** | **实验一五子棋游戏** |

|  |  |  |
| --- | --- | --- |
| **学院** |  | **计算机学院** |
| **系别** |  | **数据科学与大数据技术** |
| **班级/学号/姓名** |  | **大数据1902/2019011363/丁玲** |
| **班级/学号/姓名** |  | **大数据1902/2019010008/陈剑烽** |
| **班级/学号/姓名** |  | **大数据1902/2019010012/何强** |
| **实验日期** |  | **2021年5月5日** |
| **成绩** |  |  |
| **指导教师** |  | **蒋玉茹** |

## 实验目的

1.在解决五子棋问题中，深入体会盲目搜索策略和启发式搜索策略的基本原理；

2.能够运用编程语言实现典型的搜索算法；

1. **实验要求**

1、三人组成一个小组；

2、可以参考教师给定的wuziqi程序；

3、能够利用可视化手段展示五子棋的下棋过程；

4、实现至少三种下棋策略，描述每一种下棋策略；

5、设计合适的接口和类，在接口约束下，任意用户都可以写自己的下棋策略的对应类；假如使用了A用户的策略类和B用户的策略类，则程序运行之后，展示的是A和B对战的过程；

6、展示下棋策略的对战结果，比如：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 黑方 | 白方 | 赢方 |
| 1 | A | B | A |
| 2 | A | C | A |
| 3 | B | C | B |

7、编写控制程序，能够随机指定黑发和白方，能够随机设定先手落子的位置；运行得到50次对战结果，统计哪一个策略最好，分析原因；

8、描述系统的代码结构，代码编写要规范，要添加注释；

9、报告使用给定模板，除包含上述内容之外，还应指出参考资料，如书籍或者网站；

10、报告上传到课堂派指定位置；

11、工程打包上传到课堂派指定位置；

12、编写语言不限。

1. **实验内容**

本实验是为了让学生对于人工智能算法有进一步的认识，了解盲目搜索策略和启发式搜索策略的基本原理，真正理解机器智能的原理与方法，掌握相关的人工智能实现的技术路线和工具方法。

本实验将让学生以小组为单位合作实现一个五子棋游戏。

1. **实验过程**

**（1）环境构建：**

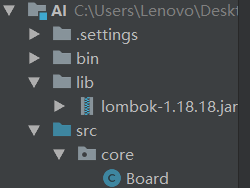
本实验代码部分利用到第三方jar包—lombok.jar

①lombok.jar下载

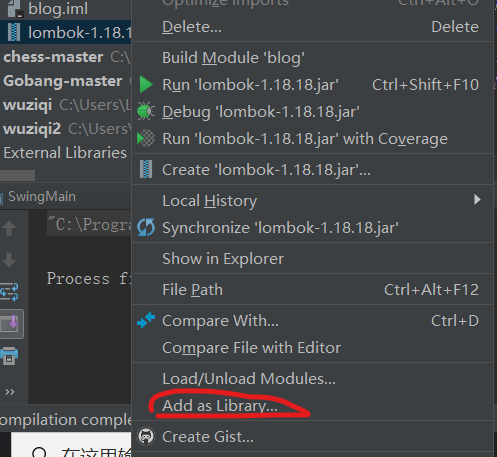
在官网下载该jar包，下载地址为<http://projectlombok.org/download>

②导入IDEA中

1. 在IDEA中创建一个lib包用来存放我们需要的jar包，将我们下载的jar包复制到lib下面



1. 将lombok.jar添加到我们项目的library中，即可用lombok中的内容。



**（2）代码结构设计：**

本工程包括两个package六个class，其中ui包中设计有BackGroundPanel、MarkerPanel和SwingMain三个类，core包中包括Board、ComputerPlayer、HumanMousePlayer、Player四个类；

Player类是选手类，其中包括Player的构造方法以及得到上一步棋子位置、tostring等方法和实现Comparable<Move>接口的Move类；

Board类是棋盘类，有包含两个Player对象的构造方法、含有一个Board对象的构造方法，此外设置了棋盘的位置、大小等，同时可以棋盘是否为空和判断棋子是否布满棋局；

ComputerPlayer类：继承Player类，是人机选手类，包括构造方法以及alphaBeta函数等可以实现人机选手的5步推算；

HumanMousePlayer类：继承Player类，是选手类，包括构造方法以及通过鼠标点击实现棋子布置的方法；

MarkerPanel类继承JPanel类，包含带有BackPanel类对象以及整形临时变量两个参数的构造方法以及其他用于可视化的函数；

BackGroundPanel类是五子棋下棋的可视化呈现类，继承系统的JPanel实现棋局的GUI界面的呈现以及棋盘背景的设置，包含一个无参数的构造方法；

SwingMain类是本项目的主类，使用到core包的Board、ComputerPlayer、HumanMousePlayer、Player类以及ui包的两个类,在已有类的基础上实现选择人机对战人机、人机对战玩家的功能，此外可以根据需要选择人机的等级(level 1-level 5 5个等级)进行五子棋的对局，并且在对局之后判定出此次棋局的赢的一方或是否打成平手等信息。

**（3）策略描述：**

我们主要运用了α-β剪枝与极大极小法来实现五子棋的人机对战、机机对战。在算法中，“-”代表没有棋子落下，表示有棋子落下的标志是通过主方法传参，可以为任何字符串且两位用户的不相同。对于AI走哪儿最好，就是要计算它走在某一个点后，计算局面的得分，然后取得分最大的那个点，这就是极大值搜索，但是不要忘了，这只是考虑的一步，搜索深度只有1，而老谋深算的都是考虑3步以上，也就是要考虑下了这一步后，对手下一步会怎么下。会玩的一定会下在我得分最小的那一个点上，这个得分相对于我而言，我的得分最小，那就是对手的最优策略了，这就是极小值搜索。但是随着搜索深度的增加，博弈树的枝点的数量是以指数增长的，搜索深度3与5的差别不是几倍的概念，是指数倍的概念，如果单单按照这种算法，电脑是承受不住的，所以，我们就用到了α-β剪枝进行处理。Ai要考虑三步的话，就是搜索深度为三，那就是搜索落在那一个点，3步都分后最大。这就能和老谋深算的选手对抗了。对于考虑的深远与否，可以把人分为高手与普通，机器是否也一样呢？或者说这一方面对于机器来说是否举足轻重？所以我们的策略是改变ai搜索的不同深度，让ai与不同深度的自己进行对抗，通过大量的对抗实践来获取答案。

我们的算法如下：

|  |
| --- |
| Int alphaBeta(Board board, int depth, int alpha, int beta, Player player)  游戏是否结束||搜索的递归的深度是否到边界  返回综合得分  判断所在的层是否为我方  Int v = min\_value  Int v = max\_value  对我们需要遍历的每一个候选步进行一个估值，并将得分较优的候选步放在集合前面  for (Board.Pos pos : childPos) { //遍历每一层的每个候选步  标记此位置已有棋子  递归下一个层的所有结点  判断当前层是否为我方  找到最佳位置  判断递归深度是否结束  获取最终最佳位置  获取当前层估值最大的位置  获取当前层估值最小的位置  比较同层节点的值与当前最优值  剪枝 |

减少可选的落子位置，限制距离为2。应用alpha-beta剪枝算法，根据往后推算步数的不同制定不同的策略。策略详细情况如下：

策略1：用字母A代替，即代码中的level1，往后推算1步。

策略2：用字母B代替，即代码中的level2，往后推算2步。

策略3：用字母C代替，即代码中的level3，往后推算3步。

策略4：用字母D代替，即代码中的level4，往后推算4步。

策略5：用字母E代替，即代码中的level5，往后推算5步。

**（4）参考资料来源：**https://github.com/pingao777/NewBe

1. **实验结果（对战结果）**

**（1）不同策略间对战结果**

本次实验采取三种策略进行实验结果的验证用字母代替分别为A,B,C

每两种策略之间对战五十次比较胜利次数，结果如下：

A,B对战50次 A赢22次 B赢28次

A,C对战50次 A赢12次 C赢38次

B,C对战50次 B赢17次 C赢29次 平局4次

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 黑方 | 白方 | 赢方 |
| 1 | A | B | B |
| 2 | A | C | C |
| 3 | B | C | C |

**（2）统计哪个策略最好并说明原因**

由上述统计结果可以得出A策略不如B策略，B策略不如C策略，综上所述在A、B、C三种策略中C策略最好。

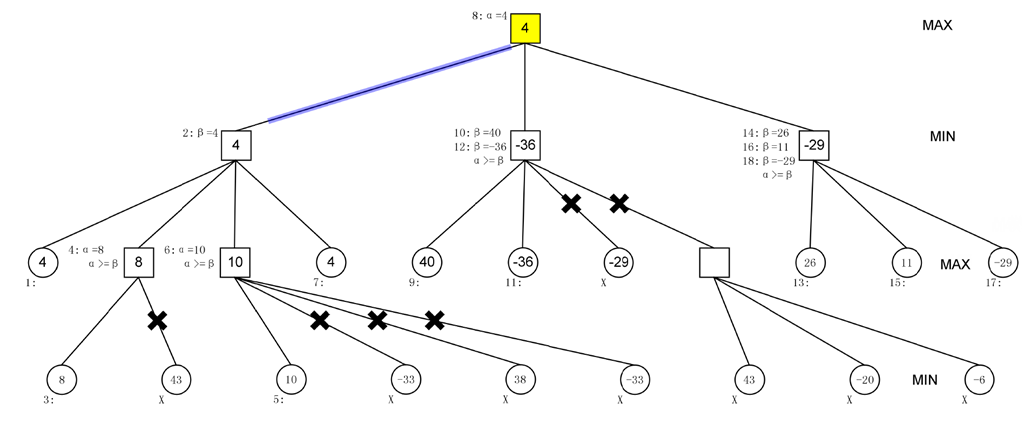
C策略好的主要原因是往后推算的步数与A、B策略相比较多，落子考虑较为全面，由此可推测往后推算步数越多，赢的可能性越大。其次是Alpha-Beta剪枝算法的作用。Alpha-Beta算法在机器博弈领域中最为重要的算法之一，它的工作原理是剪枝的思想，也就是我们可以不必构造和搜索最大深度内的所有节点，在构造的过程中，如果发现当前格局再往下不能找到更好的解，我们就停止在这个格局及以下的搜索也就是所谓的剪枝。之所以这个策略更好有两点原因一是限制落子范围：在当前棋局的所有棋子的最左、最右、最上、最下点的5格之内，不超过棋盘边界。这样在棋子较少的时候，搜索结点的数量大大减少。二是减少每层的搜索结点：在一方落子后，程序自动对新棋局下的可落子点进行落子估值，保存估值前几名的落子点，在拓展搜索树时，只考虑这些落子点作为新层结点，并进行下一步搜索，大大减少搜索范围，这样就大大提高了博弈是寻找落点的效率从而更优。

1. **实验总结**

**（1）收获**

1. 盲目搜索策略（非启发式搜索策略）：盲目搜索方法又叫非启发式搜索，是一种无信息搜索，一般只适用于求解比较简单的问题，盲目搜索通常是按预定的搜索策略进行搜索，而不会考虑到问题本身的特性。常用的盲目搜索有宽度优先搜索和深度优先搜索两种。同样的除此之外还有迭代加深搜索，而它的产生，就是为克服深度优先搜索陷入无穷分支死循环的缺点，这种方法也叫有界深度优先搜索方法。有界深度搜索的基本思想是：预先设定搜索深度的界限，当搜索深度到达了深度界限而尚未出现目标节点时，就换一个分支进行搜索。
2. 启发式搜索策略：所谓的启发式搜索策略，就是利用问题拥有的启发信息来引导搜索，达到减少搜索范围、降低问题复杂度的目的。但是由于启发式搜索只有有限的信息(比如当前状态的描述)，要想预测进一步搜索过程中状态空间的具体行为则很难。一个启发式搜索可能得到一个次最佳解，也可能一无所获。而这种缺点不可能由所谓更好的启发式策略或更有效的搜索算法来消除。所以通过大量实验总结我们可以得到，通常来说启发信息越强，扩展的无用节点就越少。引入强的启发信息，有可能大大降低搜索工作量，但不能保证找到最佳路径。因此，在实际应用中，最好能引入降低搜索工作量的启发信息而不牺牲找到最佳路径的保证。而由此也引入了一些估值函数等一系列计算节点重要性的计算策略来逐步完善相应的缺点。
3. 贪心策略：所谓的贪心策略，也就是不从总体上来考虑事物的总体情况，相反，它更侧重于对某时刻某一状态的策略的判断。以及就相对于某一情况而言做出的相应的选择。同时我们根据贪心策略所描述的也可以得到的一点总结就是，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。这样的话才能保证有效性和相对独立性。
4. Alpha-beta剪枝算法：Alpha-beta剪枝是对Minimax的补充和改进。采用Alpha-beta剪枝后，我们可不必构造和搜索最大深度D内的所有节点，在构造过程中，如果发现当前格局再往下不能找到更好的解，我们就停止在这个格局及以下的搜索，也就是剪枝。Alpha-beta基于这样一种朴素的思想：时时刻刻记得当前已经知道的最好选择，如果从当前格局搜索下去，不可能找到比已知最优解更好的解，则停止这个格局分支的搜索（剪枝），回溯到父节点继续搜索。

Alpha-beta算法可以看成变种的Minimax，基本方法是从根节点开始采用深度优先的方式构造格局树，在构造每个节点时，都会读取此节点的alpha和beta两个值，其中alpha表示搜索到当前节点时已知的最好选择的下界，而beta表示从这个节点往下搜索最坏结局的上界。由于我们假设对手会将局势引入最坏结局之一，因此当beta小于alpha时，表示从此处开始不论最终结局是哪一个，其上限价值也要低于已知的最优解，也就是说已经不可能此处向下找到更好的解，所以就会剪枝。以下是有关于此算法的大致图解：



同样以下是相关部分的一部分代码：

**private int alphaBeta(Board board, int depth, int alpha, int beta, Player player) {**

**// 游戏是否结束 | | 探索的递归深度是否到边界**

**if (board.status().isGameOver() || depth <= 0) {**

**return board.evaluate(this, this.depth - depth); //返回综合得分，自己的分-敌人的分**

**}**

**Board.Pos bestPos = null;**

**int v = (this == player) ? Integer.MIN\_VALUE : Integer.MAX\_VALUE; //敌方给最大，我方给最小**

**List<Board.Pos> childPos = sortChildPos(board); //搜索顺序排序提高剪枝效率**

**for (Board.Pos pos : childPos) { //遍历每一层的每个候选步**

**// System.out.println(pos.getRow()+"---------"+pos.getCol());**

**Board bd = new Board(board);**

**bd.mark(pos, player);**

**int w = alphaBeta(bd, depth - 1, alpha, beta, bd.getEnemy(player)); //换player，计算对方的分数**

**if (this == player) {**

**if (v < w) {**

**v = w; //**

**bestPos = pos;**

**if (depth == this.depth) { //一个候选步对其相应深度遍历结束**

**this.best = new Move(v, pos); //最佳位置**

**}**

**}**

**alpha = Integer.max(alpha, w); //极大极小法**

**} else { //敌方**

**if (v > w) {**

**v = w;**

**bestPos = pos;**

**}**

**beta = Integer.min(beta, w); //取当前层敌方最小的位置**

**// System.out.println(v+"---------"+w+"==="+beta);**

**}**

**// System.out.println(beta+"--------...."+alpha);**

**if (beta <= alpha) { //剪枝函数,剪掉同一层其他兄弟节点 ，在我方，更新β；在敌方，更新α**

**// System.out.println("执行了");**

**this.history[pos.getRow()][pos.getCol()] += 2 << depth;**

**break;**

**}**

**}**

**if (bestPos != null) {**

**this.history[bestPos.getRow()][bestPos.getCol()] += 2 << depth;**

**}**

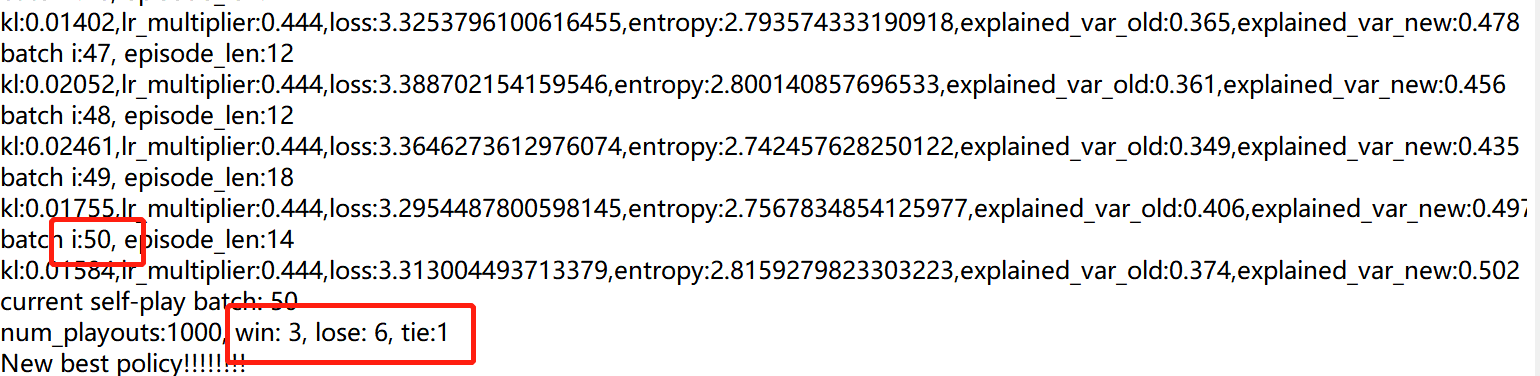
**return v; //当前层最好的位置得分**

**}**

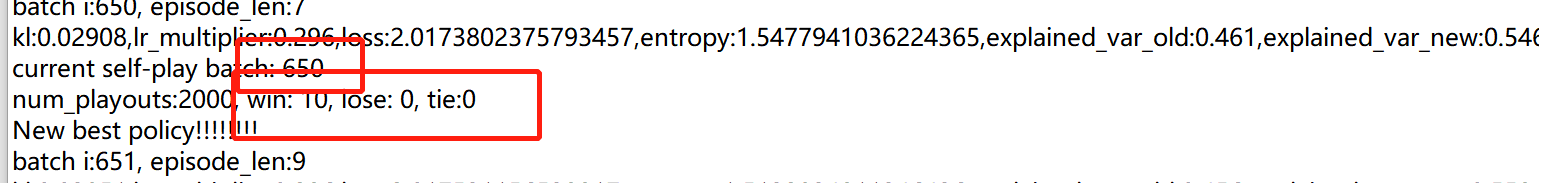
注：因为实验要求的原因，这部分代码仅作为研习相关算法的媒介。 实际情况下的编写时采用Java语言进行完成的。

1. 相关领域的发展：目前在这方面，强化学习也逐渐开始流行起来，就用alphago来举例子，通过一次又一次的不断学习，使得AI能够像人类一样积累相关的经验，我们也找到了相应的一些代码，进行了运行。下边是我们有关于运行结果的部分截图：

训练五十次的结果：



我们可以看到的是，前十局里边赢了三次，输了六次，同时还有一局平局。而当我们看第650次的时候



进行了十局的比赛中十局全胜，我们可以初步就认识道AI也在不断的训练中成长。

注：这部分是AlphaGO\_Zero和MTCS进行对弈的训练结果。

通过对于这部分的分析，我们也可以从中看出强化学习发展的基本原理，以及未来的发展趋势，而我相信这些也将会对我们有更多的启发。

**（2）遇到了什么问题，如何解决的**

（一）遇到的问题：

1. 权值的计算问题：

按照五子棋的下棋策略，我们需要计算相应节点的权值，以进行判断当前的下棋的状态以及下一步该走什么的决策。但是当我们计算的时候就会发现相应的部分无论走哪一步总会出现某些权值相差不大，无法进行权衡利弊的思考。更通俗的说，就是变成了一种下棋到后边的部分，很简单的赢得了棋局的胜利。而且下棋的步骤略显简单。

1. 由人机模式转化为机机模式的问题：

目前对于简单的五子棋的研究，大多都是属于人机模式的开发，即设计相关算法，由算法组成相关的下棋AI，通过判断人每一步的策略进行合适的判断与下棋。但是对于机机，也就是机器之间的博弈，相关的资料略少，甚至是无法找到相应的方法进行实现。如果单纯的使用互相获取节点的方法，难免会出现冗余和繁杂的情况，稍有不慎就容易引起更大的问题。同时我们在研究的时候发现，就对比贪心策略和alpha剪枝算法，将其写成两个类的话，难免会出现参数不一致的问题，这样的话更加对于代码的融合性有了极大的考验。很多时候会出现不兼容的特性。所以对于如何解决模式的问题成了我们面临的一个难题。

1. 多线程的使用问题：

在设计如何将人机模式转化为机机模式的过程中，我们考虑到了多线程的问题，即是否可以采用多线程的方式，策略一下棋，就开启相应的线程，策略二下棋，这时候，策略一的线程休眠，策略二进行下棋。

但是这样的话，虽然很好的解决相关问题，但是，面临着和上边一样的问题，使用多线程，大多都是单一策略，如果采用多策略，难免出现参数不一致的问题。所以对于多线程的使用仍然存在着问题。

1. 相关lombok插件的使用：

我们进行代码的运行时，报出了lombok找不到的问题。这样的话，有些部分就无法顺利的运行。所谓的lombok,其产生原因是因为以前的Java项目中，充斥着太多不友好的代码：POJO的getter/setter/toString；异常处理；I/O流的关闭操作等等，这些样板代码既没有技术含量，又影响着代码的美观。所以采用这样的方式能够很好的使得代码避免像上面说到的一些问题。但是当我们用的时候就发现，相应的模块缺失，而这也导致运行的失败。

（二）解决方法：

1. 对于权值的计算问题的解决，我们采用增加相应权值或者减少相应权值的方式来实现，举个例子，就是同样是下一步棋，这种方式既可以堵截对方，还能够链接自己的棋子，但另一种方法就显得有些落后了，所以这个时候我们进行特殊的权值分配。使得其能够完成相关的判断，也使得判断的精确度更加完美。
2. 对于模式转化的问题以及多线程的问题。我们采用的方式就是通过结合一些相应的java类库，进行一些函数的调用，这样的话就能够依靠Java强大的类库结合上我们相应的设计完成相关部分的内容，同时也减少了不兼容的问题。
3. 对于相关插件的使用问题，我们通过查找相关的资料，按照不同的编译器进行合理的配置。我们从官网找到相应的jar包，如果是eclipse的话直接导入并且进行配置修改，如果是idea的话，则先要进行lombok插件的安装，然后创建相应的mavean项目，并进行pom.xml的修改，创建相应的依赖。这样的话，经过调试，我们很好的解决了相关配置的问题。完成了相应的调试与编译。

**（3）心得**

本次实验，主要是进行有关于机器与机器之间的五子棋的设计。通过这次实验，初步的掌握了有关于启发式搜索和非启发式搜索等一系列相关算法的基本要点，并且能够尝试将其应用到相应的实际场景中。

同时，在应用算法的基础上，通过对相关资料的搜集与研习，慢慢感受到了计算机博弈等相关领域的发展，以及通过人工智能完成一些相应操作的大致步骤。也通过对于相关领域的学习，为之后的发展打下相应的基础。

再者，通过本次实验，从一开始的搜集相关资料到最后的完成相应任务，我们也慢慢从中体会到机器学习相关算法与人工智能之间的相辅相成的关系。就以强化学习的例子而言，虽然说现在可能读不懂相关的算法，不太能明白里边相应的一些数据用途，但是通过研习，也逐渐掌握学习训练的基本过程，以及其基本目的。

总的来说，通过本次实验，我们积累了相关方面的一些基本知识，也同时在解决遇到问题的过程中不断发现了一些新的，对于我们而言很有用的一些东西。无论是问题，还是成果，都将成为我们的经历。我们也从这其中学到很多相关的前沿知识。总之，这次实验，我们收获很多。我们也相信，在今后的不断学习中，我们也将会更加完善自己，取得更好的成果。