人工智能——围棋 AI 个人报告

小组名称：GiveMeA+
姓名：刘静静
学号：5120309669
班级：F1203025

在众多棋类游戏里面，围棋是少有的目前为止计算机程序未能打败顶级棋手的一类，正是因为其局面复杂多变难以评估，同时也存在着非常大的上升空间。我们这学期对围棋 AI 的探索和实现，不仅是对博大精深的围棋文化的探索，更是对人工智能博弈方法的更进一步认识。毫无疑问，这是大学以来最有趣难忘的一次项目，同时辛苦程度也是成正比的，不过这么多天以来，和队友们为着同一个目标努力的感觉特别充实与开心。围棋 AI 个人报告不仅是对我个人在整个项目中贡献内容的一个总结，对自己来说，这也算是对这么多天与小伙伴们共同日夜研究、实现与优化的这段经历留下一个纪念。

本报告主要从以下几个方面展开：主要负责部分，前期的尝试，后期的改进，个人总结，感想与建议。

一、 主要负责部分

我们小组的围棋 AI 主要的方法用的是 uct 方法，围绕这个主方法，我们还添加了许多方法与技巧，目的有两点：一是为了使一些明显的走法（比如提气、连接等等）能够通过匹配、评估等方式被快速识别出来；二是为了优化 uct 方法，通过剪枝使 uct 放弃一些可以判定为一定不好的点，通过排序使 uct 优先展开较好位置的点，通过加速使 uct 能模拟更多的盘数等等。uct 的主要实现由其她小伙伴们负责，我负责的是添加并测试各种方法与技巧，我的主要工作主要有：前期通过各种尝试，快速评估当前局面下不同落子位置的优劣，后期对 uct 方法做出各种优化工作，整合代码，得出不同版本的程序并测试比较。

二、 前期的尝试

1. 棋谱的导入——博大精深的围棋定式

围棋定式是经过棋手们长久以来的经验累积，形成在某些情况下双方都会依循的稳妥的下法。使用围棋定式的目的在于一开局就能根据定式，仿真棋手们的下棋思维，下出一个棋手们普遍赞同的好的位置。这就需要棋谱文件的支持，通过进入程序的时候读取棋谱文件，我们为自己保存了一个棋谱的数据结构，在每个局面要下子的时候，匹配棋谱树看有没有相同的局面，如果有就读取这个局面节点所连接的合理下法的节点。

一开始想到这个方法，是因为在前期学习围棋的时候在围棋教程网站和定式学习软件上，看到了每当我们在存在于推荐分支中的点位落子的时候，会进入某个分支，然后棋盘上会列举出下一步可以下在哪里的提示。我们也可以用类似的方法，把局面保存起来，给出针对这个局面比较优秀的落子位置。感觉这个方法和助教不久前刚给我们看的神经网络方法有点类似，不过复杂程度还没有那么高，我们还只是很基本的模式识别。

棋谱定式实现上难度主要体现在两个方面，一是如何读取并保存 SGF 棋谱文件上的信息，即“学习”，二是如何利用现有信息，匹配局面并选择下一步优秀的落子位置，即“实践”。SGF 是一个十分规范的棋谱文件类型，以文本文档的形式保存，为了加快读取，我先用了 Python 处理掉无关的信息，然后在程序中一个一个字符读取即可。保存棋谱文件上的信息我用的是一棵带有索引的棋谱树的数据结构，当匹配时，循环选取棋盘上 7*7 大小的位置，旋转翻转后的各种状态去和棋谱树种的格局进行匹配，当匹配成功时，就返回下一步合理落子位置的集合。具体的算法与数据结构会在小组的报告里给出。

我只在开局前期使用了棋谱，棋谱文件是一个定式大全，作为我们的定式，在开局阶段，局部格局基本都能在棋谱树中匹配到。

2. 基本的技巧——围棋入门必备

这一部分还是相当重要和基本的，对于围棋来说，有许许多多的技巧，人类棋手在入门阶段都需要掌握一些这样的技巧，下出一步步好棋。这一部分的实现也比较简单，大多数技巧只要循环+判断便可以得到，难点在于如何让这种暴力的搜索更加快，这便是代码水平问题了。还有一些技巧比较复杂，这时我们可以利用棋盘的棋串的各种数据对其进行处理。

目前我在项目中实现的技巧主要有：布局，提子，叫吃，紧气，连接，切断，做眼等等。

3. 局面的评估——最优最先搜索

上面的技巧，不论是棋谱定式还是基本技巧都会得到许多的点，不止一个，所以我们要对选出来的这些点进行选择。首先是评估，要为每一个技巧的匹配给予一定的分数奖励，然后是挑选出这些点中最好的点。这个挑选过程可以直接选分值最大的，但是这便受限于我们的打分规则，很难说应该给出怎样的分值最好，所以我们最后干脆就直接把这些都算是不错的位置按照我们给出的分数排个序，传到 uct 里去处理，让它从中挑选对我们最好的位置，这也充分利用了剩余的时间。

三、 后期的改进

1. 置换表优化——局面不再重复

置换表是棋类 AI 中非常常见的一种方法，配合 zobrist 键值，在哈希表中保存每个格局，每当搜到一个新的格局时，在置换表中查找是否已经存在，如不存在才创建新的格局，这样能够避免了重复格局的创建，在局数相同的情况下，能搜得更深。这部分主要的工作可以分为如何构建置换表，以及如何使用置换表。

Zobrist 键值是一个 64 位的无符号整数型数据，初始化就用 64 位随机数填充 `U64 zobrist[2][MAX_BOARDSIZE]`，下子和收子就让两个格局做异或运算，基本不会出现两个格局 zobrist 键值冲突的情况。

但是实现置换表需要更改原 uct 的数据结构，增加一类边节点，表示从这个格局到另一个格局做出的改动。增加结构体使 uct 的操作变慢，所带来的损失和置换表带来的提升还得平衡一下。

我尝试在 uct 创建子节点的时候先去查表，如果查到了就直接返回该格局的指针，如果

没有查到就新建一个节点，并更新置换表。显然只要最大最小的层数到达 3 层或 3 层以上，就应该会出现重复的格局，层数越多，或者模拟的次数越多，都能使重复的格局变多，命中率变大。我在测试的时候发现置换表的命中率还是相当高的，前期大概有差不多十分之一，后期甚至能达到三分之一，具体的实现过程和测试结果在小组报告中给出。

2. 位运算——更小与更快？

之前听说了位运算，就想用位数组来更换掉现有棋盘的数据结构，希望能达到更小更快的效果。当前用的是 char 类型的一维数组来保存棋盘上每个点的状态，char 是 1byte=8bit，但是棋盘上每个点常用状态只有三个（空，白子，黑子），只需要 2bit 就可以保存一个点的状态了，一个 int 数据就可以保存棋盘上一排的状态了，这样大大减小了棋盘的大小，这让我十分心动。

于是我写了几个小代码进行测试对比这两者的速度，结果发现在大部分操作（比如落子）上二者的速度相当，在匹配棋盘整体时位数组的优势相当明显，但是在获取棋盘某个点状态时，使用位数组就会特别慢，这是因为需要抽出一个函数来判断位数组表示的棋盘某个点的状态，但是参数的传递很耗时，这在操作频繁的情况下就看得出来差异了。现有的棋盘上判断某个位置状态的使用非常多，但是并没有匹配棋盘整体的操作，所以在不大规模改动棋盘的基础上，使用位数组显然是不合适的。

3. 有策略的 mc 模拟——不再是两个傻子

之前 mc 模拟的过程完全是双方纯随机地落子，这样并不能很好地还原真实情况，论文里提出双方应该是比较弱但是具有一定智慧的棋手在下棋，这样得到的结果才能更准。于是我们尝试在 mc 模拟里面加上了带有一点点策略的下法，包括了提子、pattern 等，不再是两个“傻子”在随便下棋。

这个修改同时也带来了一定的问题，mc 模拟的盘数是比较大的，每盘又有那么多步，每一步都会有策略，这就使得每一步时间加长，最后总的模拟盘数会下降很多，使棋力下降，直观地表现为我们的棋下得不紧凑。所以后来我们把加策略与不加策略的版本都跑出来进行各种测试，最终还是选择了不加策略保证模拟盘数足够大。

4. 打分策略的调整——看输赢还是看分数

我刚开始看 uct 代码的时候觉得十分不能理解为什么模拟完一盘之后，统计的是这盘是赢是输，而不是统计分数，因为到了后期会出现必赢或者必输的情况，这样模拟就没有什么意义了。于是我把代码调整了一下，变为统计分数，但是通过大量对局的测试发现棋力明显下降了。查了一下 paper 确实应该是统计输赢，因为这样收敛性更好。比赛那天听别的组说比较好的方法应该是前期看输赢，后期看分数，觉得这样也不错。

5. uct 模拟过程加速——盘数怎么都不嫌多

小伙伴们测试结果发现，一般在模拟到一定局数之后就收敛了，其实不必要模拟那么多，尤其是在有明显的比较好的点的时候，但是盘数还是模拟越多越好的。在 mc 模拟里面加策

略使盘数变少也能让人感觉到盘数还是十分重要的。于是我在 uct 里改代码结构，设法让重复相当频繁的步骤变得更快，比如利用 c 语言结构体的对齐加速，还有能少用函数传递参数就少用，再优化掉一些无意义的操作，结果还是能明显感觉盘数变多了，从 9.5 秒 12 万盘提升了 1 万盘，效果还是不错的。

6. 代码优化加速

加速的关键是找到性能的瓶颈，重点检查重复频率最高的部分。小伙伴们在 mc 里用了 random 取点，使性能得到了很大的提升。我在 mc 部分的加速主要是加快了 mc 内部策略匹配的速度，修改之前的策略代码，越快越好。

7. 代码整合

以上方法不是都能对棋力产生正面影响的，我们需要对它们进行排列组合，每种策略还需要加上一些步数限制，输出很多个程序比较它们的棋力。

四、 个人总结

在最后一天紧张激烈的围棋大战中，虽然和几个很强的小组相比我们的程序在棋力上还是有明显的差距，但是我们的战绩还是挺令人满意的。努力做围棋的这段时间，我们从四个围棋小白，到现在已经能熟练地辨死活看局势，我们的程序也从被屠城，提升到稳赢上一届第三第四，最后终于能打败上一届的第一名，总的来说都处于不断的进步当中。

但是和其它几个很厉害的小组比起来，我们的不足还是很明显的。首先是我们起步比较晚，大概从 12 周才开始着手，我们应该早点开始做，到了后期再把时间留出来不断调整。其次是我们参考的源代码不够多，不能拓宽思维，学习更多有用的方法。我们应该向一些小组学习，把课本上学到的知识应用于实践，更深刻理解人工智能这门科学。

五、 感想与建议

算起来连续做了差不多一个月的围棋 AI 了，似乎现在一闭上眼睛，眼前就出现一个个棋串，走这里能活，下那边会死……围棋 AI 之路充满了许多困难，每一次棋力的提升却都是那么振奋人心。电脑风扇呼啦啦一直转 写 AI 的热情并不会被偶尔的 bug 浇灭。刷 paper，看博文，写代码，赏比赛，再加上大家聚在一起讨论，基本每天都很开心。

围棋的确是一门博大精深的学问，充满了历史沧桑感，人类的智慧不容小觑。围棋 AI 的编写虽然不能使我们变成围棋高手，但是和以前比起来算是有很大进步了，之前没有接触过围棋，现在觉得围棋十分有趣，很能训练人的思维能力。

计算机十分强大，在围棋博弈上还存在很大的提升空间。通过这次项目，我们更多的收获在于人工智能的学习上，机器博弈的学习，搜索的基本方法等等，都能在实践中掌握。

我觉得围棋大战作为期末的大作业对于人工智能这门课来说是一种很好的方式，首先项目内容十分有趣，可以激发同学们的热情，其次在实践中学习非常高效，建议以后继续采用这样的形式，也期待围棋 AI 程序一年更比一年强。

最后非常感谢老师和助教的指导和帮助！