**武汉大学计算机学院**

**本科生大作业报告**

**α-β剪枝实现五子棋AI**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：人工智能导论

指 导 教 师 ：陈磊

学 生 信 息 ：2019302080085 徐贝澄

2020年11月

# 郑 重 声 明

本人呈交的报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：徐贝澄 日期：2020年11月6日

目录

**一、问题描述 4**

**二、使用算法介绍 4**

**1. 极大极小搜索算法**4

**2. α-β 剪枝算法**4

**三、在五子棋中** α**-**β **剪枝算法的具体实现5**

**1. 静态评估函数**5

**2. α-β 剪枝算法**7

**3. 细节优化**8

**四、游戏界面及用户操作9**

**1. 游戏进行时**9

**2. 游戏结束后**10

1. **问题描述**

五子棋是全国智力运动会竞技项目之一，是一种两人对弈的纯策略型棋类游戏。它有两种玩法。玩法一：双方分别使用黑白两色的棋子，下在棋盘直线与横线的交叉点上，先形成五子连线者获胜。玩法二：自己形成五子连线就替换对方任意一枚棋子。被替换的棋子可以和对方交换棋子。最后以先出完所有棋子的一方为胜。

这次大作业，我主要实现了玩法一的AI，通过极大极小衍生的**α**-**β**剪枝算法实现了一个初步的人工智能棋手，与玩家对弈。

1. **使用算法介绍**
2. **极大极小搜索算法：**

极大极小搜索策略是考虑双方对弈若干步之后，从可能的步中选一个相对最好的走法来走，即在优先的搜索深度范围内进行求解。

首先，我们需要一个评估函数，来量化评估某次走步之后对于某一方来说，当前的棋局势态的优劣，这是一个静态函数。

接着，交替建立MAX和MIN层，使用宽度优先法生成规定深度的全部博弈树，对所有叶子节点用评估函数估计其函数值。然后自底向上逐级求倒推估计值。最终

,

最后根据选择最好的走步。

1. **α-β 剪枝算法**

**α**-**β** 剪枝算法是在极大极小搜索算法的基础上，在倒推的时候及时剪去一些无用分支来提高搜索的效率。

令 **α** 为极大层的下界；**β** 为极小层的上界。

**α** 剪枝：如果一个极小层当前的**β**值小于或等于了它任一先辈极大层节点的**α**值，那么倒推值就确定为这个****，并终止这个节点之后的搜索。

**** 剪枝：如果一个极大层当前的****值大于或等于了它任一先辈极小层节点的****值，那么倒推值就确定为这个****，并终止这个节点之后的搜索。

1. **在五子棋中 α-β 剪枝算法的具体实现**

**1. 静态评估函数**

评估函数主要根据棋型来给出，给每种可能的棋型赋分，然后遍历某一方的所有棋型，将得分相加得出总分，再算出它对手此时的得分，最后myScores - enemyScores \* RADIO 为当前对于自己来说局势的优势程度。RADIO是一个超参数，来调整攻击和防守的优先级。在本文的实现中，RADIO越大，进攻性越强。

棋型大致分为：连五、强四、弱四、强三、弱三、强二，其他的不予计分。

1. 连五：五子相连，得分最高

●●●●●

1. 强四：四子相连，两端不被堵，得分次之于连五

\_ ●●●● \_

1. 弱四，四子临近，但两端有被堵，或者敌人一子可以化解危机

○●●●●­\_

●●\_●●

●●●\_●

1. 强三，三子相邻，且不被堵，但敌人一子也可化解危机，得分与弱四大致相当

\_●●●\_

\_●●\_●\_

（5）弱三，三子相邻，且被堵，暂时不足为惧

○●●\_●\_

○●●●\_ \_

（6）强二，二子临近，两端不被堵，但暂时不足为惧，分最少

\_●\_ \_●\_

\_●\_●\_

\_●●\_

具体实现时，用一个Map对象存储各棋型和对应的分数：

1. // 分值map
2. shapeScores.insert({0,1,1,0,0}, 70);     // 强2
3. shapeScores.insert({0,1,0,1,0}, 60);
4. shapeScores.insert({0,0,1,1,0}, 70);
5. shapeScores.insert({0,1,0,0,1,0}, 50);
6. shapeScores.insert({1,1,0,1,0}, 300);    // 弱3
7. shapeScores.insert({0,0,1,1,1}, 500);
8. shapeScores.insert({1,1,1,0,0}, 500);
9. shapeScores.insert({0,1,1,1,0}, 5000);   // 强3
10. shapeScores.insert({0,1,0,1,1,0}, 4800);
11. shapeScores.insert({0,1,1,0,1,0}, 4800);
12. shapeScores.insert({1,1,1,0,1}, 4950);   // 弱4
13. shapeScores.insert({1,1,0,1,1}, 4900);
14. shapeScores.insert({1,0,1,1,1}, 4950);
15. shapeScores.insert({1,1,1,1,0}, 5000);
16. shapeScores.insert({0,1,1,1,1}, 5000);
17. shapeScores.insert({0,1,1,1,1,0}, 500000); // 强4
18. shapeScores.insert({1,1,1,1,1}, 99999999); // 连5

**2. -剪枝算法**

由于既需要求最大值，又需要求最小值，而这两种操作本质上逻辑类似，所以我通过在回传时取相反数的操作，来将极大极小搜索写进同一个算法中。更从实际的角度来讲，每一层相当于某一方在下棋，而他一定会使得自己落子后的评估值最大，而对于它的上一层，也就是对手来说，他的值越大，对手就会越不利，所以回传时取相反数符合逻辑。

接着，在向下拓展的时候，将本层****的相反数传给下一层的****，本层的的相反数传给下一层的****。那么每一层的****存了上上层的****，即当前的极大值，如果这一层有更大的，才更新；每一层的****存了上一层的-****，如果这一层得到的评估值大于这个，即上一层的-，那么在这个节点之后不会的大更好的结果，所以进行剪枝，直接返回**。**而接收下一层返回值的时候，要见它取反赋给**。**这方才实现了****-****剪枝算法。核心实现代码如下：

1. **int** AI::search(**bool** isAI, **int** depth, **int** Alpha, **int** Beta)
2. {
3. **if**(gameOver(AIList) || gameOver(humanList) || depth == 0) //
4. **return** evaluate(isAI);
6. **for**(**int** x = 0; x < SIDE\_lENGTH; x++)
7. {
8. **for**(**int** y = 0; y < SIDE\_lENGTH; y++)
9. {
10. //            qDebug()<< x<<' ' << y;
11. // 跳过周围没有棋子的位置，跳过已经下了棋的位置
12. **if**(AllList[x][y] || !hasNeighbor(x, y))
13. **continue**;
15. **if**(isAI)  // 轮到谁，就谁向下搜索
16. AIList[x][y] = **true**;
17. **else**
18. humanList[x][y] = **true**;
20. AllList[x][y] = **true**;
22. // 向下搜索一层
23. **int** value = -search(!isAI, depth-1, -Beta, -Alpha);
24. // 回溯，恢复状态
25. **if**(isAI)
26. AIList[x][y] = **false**;
27. **else**
28. humanList[x][y] = **false**;
29. AllList[x][y] = **false**;
31. **if**(value > Alpha)
32. {
33. **if**(depth == SEARCH\_DEPTH)
34. {
35. // 回传到了根节点，更新nextPoint
36. nextPoint.setX(x); nextPoint.setY(y);
37. }
38. **if**(value >= Beta)
39. // 减枝
40. **return** Beta;
41. Alpha = value;
42. }
43. }
44. }
45. **return** Alpha;
46. }
47. **细节优化：**

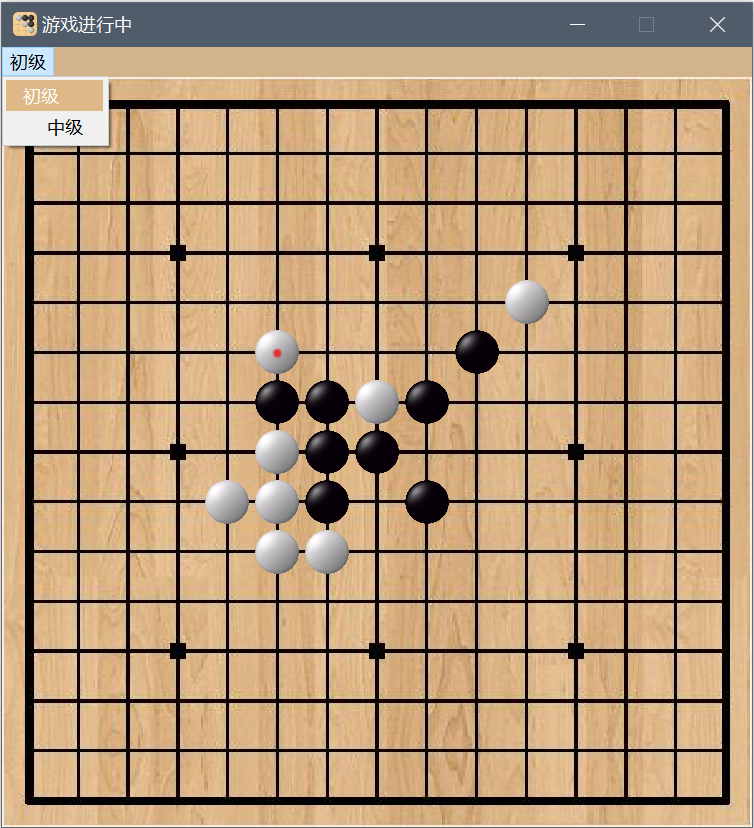
（1）在搜索时，跳过了周围八格没有已下棋子的格点，来提高搜索速度。

（2）在搜索的时候，优先搜索刚刚对方落子和上一次自己落子两个位置的周围位置。这些点最有可能是最优解，先找到了较优解，之后通过剪枝就能够大大减少搜索的花销。

（3）在评估某方棋局形势的时候，要存储已经计分的棋子点列，防止有些棋子在不同形状中被反复计分，导致评估偏差。

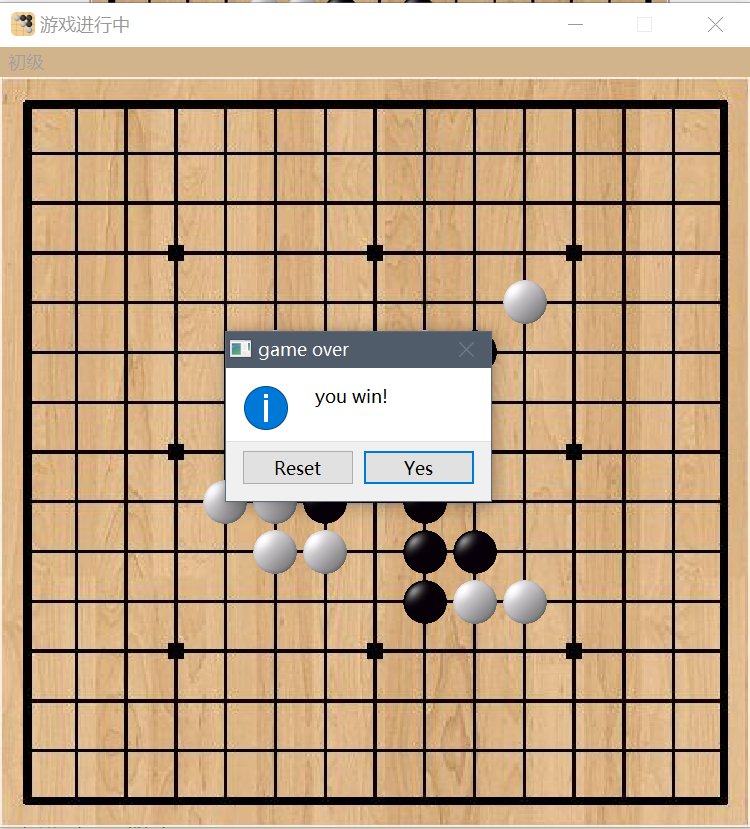
1. **游戏界面及用户操作**

我最终通过Qt来实现图形化用户界面，尽量复刻了经典五子棋棋盘外观。

1. 游戏进行时：

最上端的菜单栏可以调整难度，每次调整都会从头开始。初级和中级的区别在于剪枝搜索时中级搜索得更深，而相应的，要花的时间也更多。

1. 游戏结束后：

如果赢了或输了，会有弹窗提醒，这时候可以选择重新开始。

最终已将游戏打包成一个exe文件，直接点击即可运行。