基于 Alpha-Beta 剪枝搜索的五子棋 AI

王世杰
2016010539
软件 72

黎思宇
2016010695
软件 71

摘要

在本次实验中，我们利用 Alpha-Beta 剪枝算法实现了一个五子棋的 ai 程序，并用启发式搜索增加了效率。

关键词

人工智能，Alpha-Beta 剪枝搜索，五子棋，Zobrist

1. 实验简介

五子棋是中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。通常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。

本次试验我们使用 Alpha-Beta 剪枝方法实现了一个五子棋人工智能程序，同时使用启发式搜索、迭代加深等方法提高了程序运行效率。

2. 程序设计与重要实现

2.1 程序逻辑

程序入口为 start.cpp，进入主程序循环，打印程序说明和输入样例，同时接受键盘输入确定 AI 先手还是用户先手。GameLoop 类为程序主类，其实例化对象 game 为主程序对象，通过 game.run()进入游戏主逻辑循环。

在 GameLoop 的构造函数中，实现了对五子棋棋局得分表的初始化，棋子颜色的分配，先手后手的分配，按照惯例，先手执黑。

GameLoop::run()函数为棋局的主逻辑循环，在每个循环内，通过调用 gameOver()函数判断当前棋局是否已经分出胜负，如果出现输赢或平局，则跳出游戏循环，开始等待下一局游戏；如果未分胜负，则判断当前轮到 AI 还是用户落子，并分别调用 aiLoop 和 userLoop 方法进行下棋操作。

userLoop 方法通过读取用户输入执行落子(move)或悔棋(regret)操作，如果为 move 操作，则判断落子是否合法，如果不合法(坐标超出棋盘或已被占用)则抛出异常并重新接受输入，若合法则调用 makeMove 方法落子；如果为 regret 操作，查看是否符合悔棋条件(至少有一子)，如果符合则调用 unMakeMove 方法撤销上一回合落子。

每个循环开始前都清除屏幕并重新绘制棋盘，起到刷新的作用。

2.2 程序逻辑

主要数据结构和变量定义在 define.cpp 中：

```
//定义棋盘
int chessBoard[GRID_NUM][GRID_NUM]
//棋盘每个位置的估值
int valueBoard[GRID_NUM][GRID_NUM]
//ai 的落子记录
vector<pair<int, int>> ai_steps;
//user 的落子记录
vector<pair<int, int>> user_steps;
//记录不同五子棋局势的分值
map<string, int> score_map;
//回合数
int turns = 0;
```

除此之外还有以下数据结构和变量：

```
//"score", "direction", "x", "y", "offset"为 key 列表，代表某个点所对应的五子棋形状得分
map<string, int> max_score_shape
//max_score_shape 的数组
vector<map<string, int>> score_all_arr_my
//max_score_shape 的数组
vector<map<string, int>> score_all_arr_enemy
//max_score_shape 的数组
//由于对称性取值为 1，2，3，4 代表四种方向
int direction
//代表棋盘上的一个点
pair<int, int> node
```

2.3 evaluate 模块实现方法说明

本实验中涉及到两种估值方法，分别为对于某一棋局状态的整体估值和对棋盘上每一个可落子位置的估值，前者用于 minimax 方法中对局面的估计，后者用于对落子位置的启发式搜索。

整体估值位于 evaluate.cpp 中，在 evaluate 函数中通过对每个已落子位置的四个方向的扫描估算局面，最终得分为 my_score - 0.1 * gamma * enemy_score，其中 gamma 作为算法超参数可以调控 AI 的进攻与防守偏向，gamma 越大 AI 越偏向防守。

对于每一个点，调用 calScore 函数，通过遍历其周边的点生成其参与的棋型，并在记录棋型得分的 map 中查找得分，最终取该方向上最高得分的棋型予以记录。如果该点在某方向上已经被记录在某个棋型中，则跳过该位置防止重复计算。通过对己方棋型的扫描和对方棋型的扫描，可以计算出当前棋局的总得分。

逐位置估值位于 BlankEvaluate.cpp 中，通过 evaluatePoint 函数对当前棋局每个未落子位置进行估值，储存在 valueBoard 中。其主要思路也是对己方棋型和对方棋型进行

分析并估值，类似于人的思维：对于每个可落子的点计算如果落子能够形成什么样的棋型，并对不同的棋型赋分。而与整体估值不同，逐点估值是将己方得分与对方得分相加，因为对于对方重要的点对我方也很重要(防守)，都是未来落子需要重点考察的位置。

2.4 gameover 模块实现方法说明

判断 gameover 时，我们只需要考虑上一次被下的点是否会导致产生五子连环。这样每次判断的复杂度即是$O(1)$具体实现上，可以通过从竖方向、横方向、左上斜方向、左下斜方向开始，查找是否形成以上一次点为中间点的五子连环。

2.5 createmove 模块实现方法说明

我们实现了两种 createmove，一种是简单的，一种是启发式的。具体区别如下：

简单的 createmoves 将产生所有已有点附近一格的点，这是考虑到五子棋中离已有点太远的点不大可能是最优解。

考虑到 Alpha-Beta 将顺序遍历 createmove 产生的点，若 createmove 能将最优点置于最前面，则可以增加 Alpha-Beta 剪枝次数，大幅度增快 Alpha-Beta 搜索速度。于是，我们参考了对空白点进行估值的 ai 下棋算法[1]，估值越高的点，即是最可能从其开始能搜索到最高分数的点。启发式的 createmove 中将按照空白点估值大小进行排序后，再进行返回。

2.6 searchmove 模块实现方法说明

我们在 searchmove 模板实现了六种 search 策略，分别为 minimax 搜索、Alpha-Beta 剪枝搜索、的启发式 alphabeta 剪枝搜索、Zobrist 散列表的 Alpha-Beta 剪枝搜索、带散列表的 Alpha-Beta 剪枝搜索、迭代加深搜索。

2.6.1 *迭代加深搜索*

本次作业中实现的迭代加深搜索，是为了使 ai 每步棋耗费相同时间。具体来说，我们设定的 ai 每步棋都耗费 2s，它将从 2 层深度开始搜索。若时间未耗尽，它将再次累加 2 层进行搜索，直到时间被耗尽。考虑到搜索中浅层耗时远远小于深层耗时，这种搜索方法并不会浪费多少时间。

2.6.2 *Minimax，Alpha-Beta，启发式搜索*

本次作业中实现的 MiniMax 和 Alpha-Beta 搜索均参考自《人工智能、一种现代的方法》[2]，二者关于搜索深度 n 均为指数复杂度。其中，令 minimax 算法中分子因子为 b 时，其复杂度为$O(b^n)$，则 Alpah-Beta 算法的分子因子大概是$\sqrt{b}$，因而能大幅度提高搜索性能。

实现中，MiniMax 和 Alpah-Beta 都可以仅用一个函数同时计算 max 和 min，这可以通过在递归时取负号来实现[3]，这样的话，考虑到每次下棋应该是两人先后下子，搜索深度应该是偶数，才能兼顾攻守[4]。

考虑到 Alpah-Beta 搜索剪枝次数和遍历空白点的顺序有关，我们使用了一种对空白点评分后排序的启发式搜索。评分越高的点越可能是最优点，因而我们将高评分的点排序在前面。经实验证实，这种启发式搜索可以加快十倍左右速度。

2.6.3 *Zobrist 散列表[5]*

我们发现，先下(3,3),再下(4,4)和先下(4,4),再下(3,3)，局面完全一样。因此我们可以通过存储某些局面的评分，来减少耗时的评分计算次数，从而加快搜索速度。

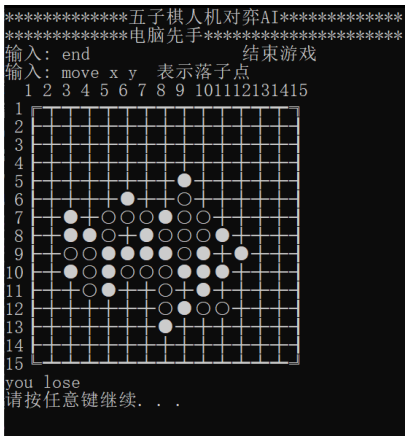
在实现中，我使用了常用于记录棋谱的 Zobrist 散列算法，其只需要在每步棋进行一次异或操作，就能有很好避免哈希碰撞，因此时间代价非常小。具体来说，它为每一个棋盘格子赋一个 64 位的随机数，并且我会存储两个 64 位的整数n_1, n_2。棋下在一个新格子时，我将用格子所在值 k 更新n_1，即是$n_1 = n_1 \wedge k$（异或），从该格撤销时也是一样的操作，更新为$n_1 = n_1 \wedge k$，白棋与 n2 操作一样。我们可以发现，使用异或操作的好处在于 2 点：1. 撤销方便，2 . 局面值n_1, n_2与具体下棋顺序无关。

于是我们只需要在 evaluate 时，建立(n_1, n_2)到当前局面 evaluate 值的散列表，就可以减少估值计算次数了。

在具体实现上，我使用的是 std::map<std::pair<long, long>, int>来建立这种联系。虽然 stl 中 map 并不是散列表，但哪怕存储量达到 4G 大小时，map 和哈希表 unordered_map 的单次查询、增加耗时差距仅在 100ms 左右[6]，而在该搜索算法实现中，evaluate 并不是性能瓶颈，搜索顺序剪枝次数、createmoves 的具体实现才是性能瓶颈，因此我直接使用了 map，实验下来，可以发现也能减少 20%左右用时。

3. 实验结果及评价

3.1 用户使用



支持人机对战、支持选择 ai 先走，或是人先走，支持悔棋

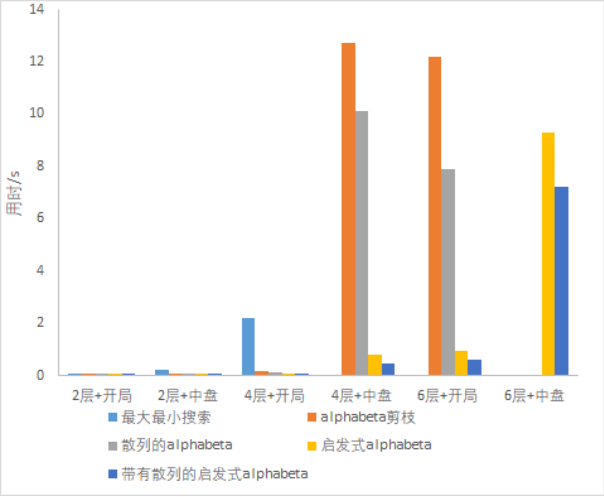
经我们人工测试，游戏中 ai 具有一定难度，即使人先走也有五六成次数失败，可以看出我们实现的 ai 已经能达到人类初学者以上的水平。

3.2 优化策略对比

本次我们共进行了五种搜索方案的对比：

1. minMax 搜索
2. alphabeta 剪枝搜索
3. 启发式 alphabeta 剪枝搜索
4. 散列表 alphabeta 剪枝搜索
5. 带散列表的 alphabeta 剪枝搜索

考虑到五种方案均为最大最小值搜索的优化手段，拥有一样的评价函数，相同层数时，搜索的结果应该是一样的，我们仅对比 5 者分别搜索 2、4、6 层开盘和中局的时间，时间越短说明该种方案搜索性能越强，也就代表该种方案更好。



（从第三列开始，最大最小值搜索无法在短时间内计算出，故不再显示，最后一列，alphabeta 剪枝和散列的 alphabeta 剪枝也无法在短时间内计算出，故不再显示。）

4. 额外功能

我们实现了以下额外功能：

1. 散列表优化方法
2. 迭代加深方法
3. 启发式搜索
4. 悔棋功能

5. 实验收获与体会

本次实验基于 C++实现了五子棋的 AI 程序。算法的主要思路是评估与搜索。通过对局面的评估与状态空间的搜索生成决策。通过本次作业，我们对于以五子棋为代表的对抗搜索博弈问题有了更深刻的理解，亲手实现相关的算法也加强了我们对于搜索的细节了解

另外，我们在实验中再次重温了算法设计的重要性，我们分析复杂度，通过将 gameover 算法从遍历改为查询上次着子，大幅增快了我们的搜索速度。

同时我们也查阅并了解了其他具有代表性的棋类智能方法，如蒙特卡洛模拟，AlphaZero 代表的神经网络方法等。由于时间与资源有限，本次试验并未深入研究这些方法，我们期待在未来有机会的话进行实现。

6. 分工

黎思宇：游戏逻辑主循环、搜索实现、测试实验

王世杰：整体局面估值、启发式估值、悔棋

7. 参考

[1] Maxuewei2 (2020, April 19th)五子棋估值算法 https://www.cnblogs.com/maxuewei2/p/4825520.html#daima

[2] StuartRussell, & PeterNorvig. (2004). *人工智能:一种现代方法*. 人民邮电出版社.

[3] 象棋百科全书网(2020 April 19th) 《对弈程序基本技术》专题 http://www.xqbase.com/computer/outline.htm

[4] lihongxun945(2020 April 19th) 五子棋 AI 教程第二版 https://github.com/lihongxun945/myblog/issues/14

[5] 宋天（2020 April 19th） Zobrist 散列 http://www.soongsky.com/othello/computer/zobrist.php

[6] Stpeace (2020 April 19th) map,hash_map 和 unordered_map 效率比较 https://blog.csdn.net/stpeace/article/details/81283650