目 录

实践背景1

算法简介4

算法正确性证明4

算法流程4

系统环境与编译环境4

代码实现4

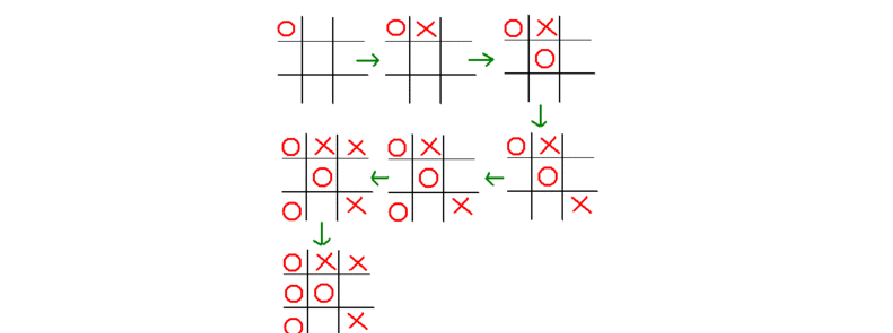
程序运行结果4

总结与感悟4

1. 实践背景

井字棋，又称OX棋、Tic-Tac-Toe、或圈叉棋，是一种由两人轮流在3x3格的九宫格上画圆圈（O）和叉（X）的游戏。第一个将自己的三个标记排成一线（可以是水平的、垂直的或者对角线的）的玩家即为胜者。如果所有的格子都被填满但没有玩家满足上述胜利条件，那么游戏就会以平局结束。

井字棋是一种极其简单的游戏，因此适合在学习人工智能、游戏理论或决策算法时作为实例进行研究。尽管它在人类看来可能比较无聊，因为只要双方都不出现错误，游戏总是以平局结束，但这种特性也使得它成为理论研究的好对象。



图一 井字棋的实现

在本课程问题求解章节，我系统地学习了双人零和博弈及其相关算法、数据结构，其中包括与或图以及alpha-beta剪枝。不过在课上学到的东西较浅，仅仅存在于理论阶段，于是我在课下自行查阅了相关文献，并将经典的、简单的双人零和博弈游戏——井字棋使用alpha-beta剪枝实现。在编写代码过程中，我发现了alpha-beta剪枝可以使用与或图来证明，于是将理论和项目综合在一起，完成了本篇报告。

1. 算法简介

Alpha-beta剪枝是一种搜索算法，用以减少极小化极大算法（Minimax）搜索树的节点数。这是一种对抗性搜索算法，主要应用于机器游玩的二人游戏（如井字棋、象棋、围棋）。当算法评估出某策略的后续走法比之前策略的还差时，就会停止计算该策略的后续发展。该算法和极小化极大算法所得结论相同，但剪去了不影响最终决定的分枝。

Alpha-Beta剪枝的主要思想是，它在搜索过程中跟踪到目前为止找到的最优的已知分数，并用它来剪去那些不可能影响最终结果的分支。具体来说：

Alpha：在MAX层（也就是在我们假设的“我们”正在做出决策的层面），我们会跟踪找到的最高的分数。这是到目前为止我们已经找到的最好的选择，也就是一个下限——我们知道我们可以至少达到这个得分。

Beta：在MIN层（也就是在我们假设的“对手”正在做出决策的层面），我们会跟踪找到的最低的分数。这是到目前为止我们已经找到的最坏的选择，也就是一个上限——我们知道无论怎样我们的得分不会超过这个数。

在搜索过程中，如果我们在一个MIN节点找到了一个得分低于alpha的选项，那么我们就可以停止在该节点下面的搜索，因为我们知道在这个节点上，“对手”会选择一个更高的得分，这将导致这个节点的得分至少与alpha相等，因此不会影响最终的结果。同理，如果我们在一个MAX节点找到了一个得分高于beta的选项，那么我们也可以停止在该节点下面的搜索，因为我们知道在这个节点上，“我们”会选择一个更低的得分，这将导致这个节点的得分至多与beta相等，因此也不会影响最终的结果。这种方法就叫做“剪枝”。

Alpha-Beta 剪枝的一个关键假设是，每个玩家都会选择对他最优的行动。也就是说，每个玩家都会选择能够让他们得分最高的行动。

图示

描述已自动生成

图二 Alpha-Beta剪枝例子

需要注意的是，Alpha-Beta 剪枝并不会影响最终结果的正确性，我们在下面会进行证明。也就是说，使用 Alpha-Beta 剪枝的 Minimax 算法将返回与不使用 Alpha-Beta 剪枝的 Minimax 算法相同的结果。然而，由于 Alpha-Beta 剪枝减少了搜索空间，它可以大大提高搜索的效率。

在最佳情况下，如果移动是按最优顺序进行的（也就是说，最好的移动总是首先被考虑），那么 Alpha-Beta 剪枝可以将 Minimax 搜索的复杂度从 O(b^d) 减小到 O(b^(d/2))，其中 b 是分支因子（每个节点的平均子节点数），d 是树的深度；但是在最坏的情况下，如果移动是按照最差顺序进行的（也就是说，最差的移动总是首先被考虑），那么 Alpha-Beta 剪枝将无法剪枝任何分支。

1. 算法正确性证明

在上面一段我们大致说明了Alpha-beta剪枝算法的原理、内涵等等，下面我们就来遵循着上述的内容进行其正确性的证明。

在上文我们已经陈述过alpha与beta的意义，即：Alpha在MAX层，是我们追踪最高的分数；Beta在MIN层，是我们追踪最低的分数。因此在搜索过程中，如果我们发现某一节点的可能最低分数（beta）小于等于祖父节点（MAX节点）的最高分数（alpha），那么我们就可以剪枝，因为MAX节点永远不会选择导致分数下降的路径，反之亦然。

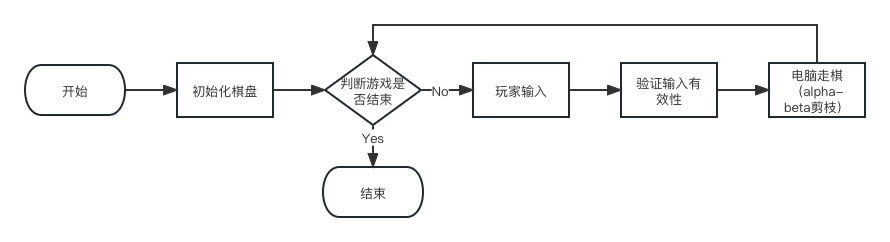
如果利用我们学过的另外一个重要的数据结构与或图来进行证明，那么在与或图中，可以将Max节点视为"AND"节点，将Min节点视为"OR"节点。如果我们在OR节点找到一个解（即找到一个比当前alpha值更小的beta值），那么我们就可以停止搜索其他子节点，因为OR节点只需要一个子节点为真（解）。同理，如果在AND节点我们发现一个子节点不是解（即找到一个比当前beta值更大的alpha值），那么我们也可以停止搜索其他子节点，因为AND节点需要所有子节点为真。

因此，Alpha-Beta剪枝的正确性在于它遵循了与或图的基本逻辑：Max节点（AND节点）需要所有可能的子节点都被考虑，而Min节点（OR节点）只需要一个子节点被考虑。任何在这个逻辑下被排除的搜索路径都是不必要的，因此Alpha-Beta剪枝不会错过任何可能的最优解。

1. 算法流程

具体的算法流程可以分为以下几个部分：

* 定义棋盘和相关的变量：定义了一个3x3的棋盘用来存储游戏的状态，棋盘上的每个位置可以是空的、玩家的棋子或者电脑的棋子。此外，还定义了一个Move结构，用于表示一步移动。
* 棋盘评估函数（evaluate）：评估函数用于计算当前棋盘的状态。如果一方获胜，函数返回相应的得分（对于电脑来说是10，对于玩家来说是-10），如果还没有获胜，函数返回0。
* 移动剩余检查函数（movesLeft）：此函数检查棋盘上是否还有空位，如果有，返回true，否则返回false。
* Minimax函数与Alpha-Beta剪枝（minimax）：这是实现Alpha-Beta剪枝的核心函数。这个函数以当前的棋盘状态、搜索的深度、最大化或最小化玩家、alpha值和beta值为输入，返回最优的评分。在搜索过程中，如果发现一种可能性不能改善已经找到的最佳结果，就停止对其的进一步搜索。
* 寻找最佳移动（findBestMove）：这个函数搜索所有可能的移动，使用minimax函数找到最好的移动。
* 打印棋盘（printBoard）：这个函数打印当前的棋盘状态。
* 检查游戏是否结束（isGameOver）：这个函数检查游戏是否结束，如果没有可用的移动，或者有一方获胜，游戏结束。
* 主函数（main）：这是程序的入口。它实现了游戏的主要逻辑：在游戏没有结束的情况下，轮流让玩家和电脑进行移动，玩家的移动是通过输入来决定的，而电脑的移动是通过调用findBestMove函数来决定的。游戏结束时，打印出最后的棋盘状态和游戏结果。



图三 算法流程图

1. 系统环境与编译环境

* 操作系统：macOS 13.0 (22A380)
* 编译环境：Apple clang version 14.0.3 (clang-1403.0.22.14.1)

1. 代码实现

此报告重点为alpha-beta剪枝算法，因此本代码中非相关部分不做深入讨论，仅详细分析电脑利用alpha-beta剪枝落子的代码。

**minimax函数：**

1. int minimax(int depth, bool isMax, int alpha, int beta) {
2. int score = evaluate();
3. if (score == 10)
4. return score - depth;
5. if (score == -10)
6. return score + depth;
7. if (!movesLeft())
8. return 0;
9. if (isMax) {
10. int best = -1000;
11. for (int i = 0; i < 3; i++) {
12. for (int j = 0; j < 3; j++) {
13. if (board[i][j] == EMPTY) {
14. board[i][j] = COMPUTER;
15. best = max(best, minimax(depth + 1, !isMax, alpha, beta));
16. board[i][j] = EMPTY;
17. alpha = max(alpha, best);
18. if (beta <= alpha)
19. return best;
20. }
21. }
22. }
23. return best;
24. }
25. else {
26. int best = 1000;
27. for (int i = 0; i < 3; i++) {
28. for (int j = 0; j < 3; j++) {
29. if (board[i][j] == EMPTY) {
30. board[i][j] = PLAYER;
31. best = min(best, minimax(depth + 1, !isMax, alpha, beta));
32. board[i][j] = EMPTY;
33. beta = min(beta, best);
34. if (beta <= alpha)
35. return best;
36. }
37. }
38. }
39. return best;
40. }
41. }

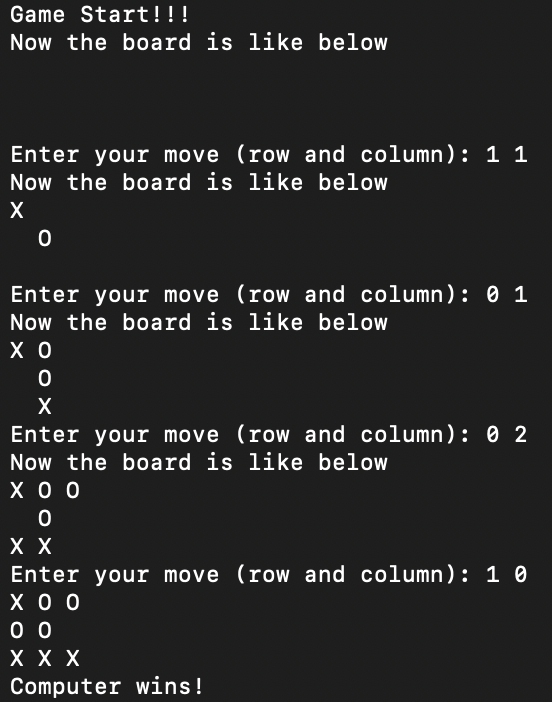
* 检查游戏是否结束：首先，函数会检查当前局面是否已经结束，也就是看是否有玩家赢得游戏，或者游戏是否平局（没有剩余的空位置）。这是通过调用 evaluate 函数来实现的。如果游戏已经结束，函数会返回对应的局面得分。
* 遍历所有可能的走法：如果游戏还没有结束，函数就会遍历棋盘上的所有空位置，模拟每一个可能的走法。这是通过一个二重循环来实现的，对棋盘的每一行和每一列进行遍历。
* 递归调用： 对于每一个可能的走法，函数会假设棋子已经放在那个位置，然后递归调用自身，以模拟接下来的游戏局面。在递归调用时，搜索深度 depth 加一，而 isMax 参数取反，以模拟另一方的回合。
* 计算最大或最小值： 函数会根据当前是电脑的回合还是玩家的回合，分别计算并返回最大或最小的局面得分。在电脑的回合，函数会尝试最大化局面得分，而在玩家的回合，函数会尝试最小化局面得分。
* 回溯： 在模拟走法的过程中，函数会暂时改变棋盘的状态。因此，在每一次递归调用之后，函数需要回溯，也就是将棋盘恢复到原来的状态。
* Alpha-Beta 剪枝： 在遍历所有可能的走法时，如果当前已经找到的最大值（alpha）大于或等于当前已经找到的最小值（beta），那么函数就可以提前返回，不需要再继续搜索。这是因为在这种情况下，我们已经知道这个走法不可能被选作最优解。

**findBestMove函数：**

1. Move findBestMove() {
2. int bestVal = -1000;
3. Move bestMove;
4. bestMove.row = -1;
5. bestMove.col = -1;
6. for (int i = 0; i < 3; i++) {
7. for (int j = 0; j < 3; j++) {
8. if (board[i][j] == EMPTY) {
9. board[i][j] = COMPUTER;
10. int moveVal = minimax(0, false, -1000, 1000);
11. board[i][j] = EMPTY;
12. if (moveVal > bestVal) {
13. bestMove.row = i;
14. bestMove.col = j;
15. bestVal = moveVal;
16. }
17. }
18. }
19. }
20. return bestMove;
21. }

* 初始化最佳值和最佳走法： 这个函数会初始化一个 bestVal 变量为 -1000 （这个值比任何可能的局面得分都要小），以及一个 bestMove 变量为 (-1, -1) （这是一个无效的走法）。
* 遍历所有可能的走法： 函数会遍历棋盘上的所有空位置，模拟每一个可能的走法。这是通过一个二重循环来实现的，对棋盘的每一行和每一列进行遍历。
* 调用minimax： 对于每一个可能的走法，函数会假设电脑已经在那个位置下了棋，然后调用 minimax 函数，计算这个走法对应的局面得分。
* 更新最佳走法： 如果一个走法的局面得分高于当前已知的最佳值，那么函数就会更新 bestVal 和 bestMove，将这个走法记作当前的最佳走法。
* 返回最佳走法： 函数会返回 bestMove，也就是电脑应该选择的最佳走法。

1. 程序运行结果



图四 程序运行截图

1. 总结与感悟

人工智能确实是一门深奥却又吸引人的学科。它的知识点深入理论，同时又密切关联实际问题，比如三子棋（Tic Tac Toe）问题，甚至更复杂的棋类问题。这些实际问题引发了我对人工智能的兴趣，尤其是在探索和解决这些问题的过程中。

本学期的《人工智能基础》课程中，我主要学习了问题求解、逻辑与推理、机器学习等知识，这些都为我打开了通往人工智能世界的大门。特别是在这次的课程实践中，我对Max-Min搜索算法有了更深入的理解，并成功地将其应用到了三子棋游戏中。

在项目过程中，我面临了各种挑战，包括C++程序设计，评估函数的实现，以及Alpha-beta剪枝算法的实施等等。经过不断地查阅博客、论文，以及反复地调试和优化，我成功地实现了Alpha-beta剪枝算法。这使得我编写的三子棋游戏程序运行速度大大提升，也提高了电脑对抗的难度。通过这个项目，我不仅深入理解了Minimax算法和Alpha-beta剪枝的原理，更实际掌握了如何将理论知识应用到实际问题中去。

**【参考文献】**

[1] Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. Artificial Intelligence, 6(4), 293-326.

[2] 朱小瑜. (2010). 基于Alpha-Beta剪枝的四子棋游戏研究. 计算机工程与应用.

[3] Marsland, T. A. (1986). A review of game-tree pruning. Artificial Intelligence Review, 1(1), 3-34.

[4] Russell, S. J., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.

**致 谢**

本学期的人工智能基础课给了我一个很好的开始，让我有机会系统地学习人工智能的基础知识，并有机会将所学知识应用到实践中；在完成这个实践的过程中，我深深体会到了理解和实践人工智能的重要性。

我能够顺利完成本学期的课程，并且学到很多关于AI的专业知识，离不开老师的谆谆教导以及严谨的教学，在此，我真挚地向廖士中教授表示真诚的感谢！