天津大学

alpha-beta 剪枝解决一字棋问题实验报告

学院名称 智能与计算学部

年　　级 2019 级

专　　业 计算机科学与技术

班　　级 3 班

学生姓名 王红阳

实验名称 alpha-beta 剪枝解决一字棋问题

成　　绩

一、实验内容

1. 一字棋问题介绍

"一字棋"游戏（又叫"三子棋"或"井字棋"），是一款十分经典的益智小游戏。它的棋盘是一个 3×3 的格子，很像中国文字中的"井"字，所以得名"井字棋"

规则：两方轮流下棋，首先将三子连成一线的一方获得胜利

2. 博弈树

双人完备信息博弈过程可用改进的状态空间图，并用有向树表示出来，这种树可称为博弈树。

博弈树与状态空间图中有向树表示所唯一不同的是在其结点下方的弧中可用符号以增加“与”、“或”语义，博弈树是一棵与/或树。

在博弈树中，那些下一步该 MAX 走步的结点称为 MAX 结点，该 MIN 步的结点称为 MIN 结点

博弈树具有如下特点：

（1）博弈的初始状态是初始结点。

（2）博弈树的“或”结点和“与”结点逐层交替出现。

（3）整个博弈过程始终站在某一方立场。所有能使自己一方胜利的终局都是本原问题，相应的结点是可解结点；使对方获胜的终局都是不可解结点。

3. 极小极大分析法（一字棋问题）

估价函数定义如下：

设棋局为 P，估价函数为 e(P)。

(1) 若 P 对任何一方来说都不是获胜的位置，则 e(P)=e(那些仍为 MAX 空着的完全的行、列或对角线的总数)-e(那些仍为 MIN 空着的完全的行、列或对角线的总数)

(2) 若 P 是 MAX 必胜的棋局，则 e(P)＝+无穷

(3) 若 P 是 MIN 必胜的棋局，则 e(P)＝-无穷

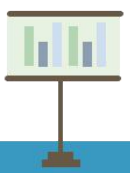
4. alpha-beta 剪枝

上述的极小极大分析法，实际是先生成一棵博弈树，然后再计算其倒推值，至使极小极大分析法效率较低。于是在极小极大分析法的基础上提出了 alpha-bata 剪枝技术。

该剪枝技术的基本思想或算法是，边生成博弈树边计算评估各节点的倒推值，并且根据评估出的倒推值范围，及时停止扩展那些已无必要再扩展的子节点，即相当于剪去了博弈树上的一些分枝，从而节约了机器开销，提高了搜索效率。

具体的剪枝方法如下：

(1) 对于一个与节点 MIN，若能估计出其倒推值的上确界 beta，并且这个 beta 值不大于 MIN 的父节点(一定是或节点)的估计倒推值



的下确界 alpha，即 alpha >= beta，则就不必再扩展该 MIN 节点的其余子节点了(因为这些节点的估值对 MIN 父节点的倒推值已无任何影响了)。这一过程称为 alpha 剪枝。

(2) 对于一个或节点 MAX，若能估计出其倒推值的下确界 alpha，并且这个 alpha 值不小于 MAX 的父节点(一定是与节点)的估计倒推值的上确界 beta，即 alpha>=beta，则就不必再扩展该 MAX 节点的其余子节点了(因为这些节点的估值对 MAX 父节点的倒推值已无任何影响了)。这一过程称为 beta 剪枝。

从算法中看到：

(1) MAX 节点(包括起始节点)的 beta 值永不减少；

(2) MIN 节点(包括起始节点)的 alpha 值永不增加。

在搜索期间，alpha 和 beta 值的计算如下：

(1) 一个 MAX 节点的 alpha 值等于其后继节点当前最大的最终倒推值。

(2) 一个 MIN 节点的 beta 值等于其后继节点当前最小的最终倒推值

5. 输赢判断算法思路

因为每次导致输赢的只会是当前放置的棋子，输赢算法中只需从当前点开始扫描判断是否已经形成三子。对于这个子的八个方向判断是否已经形成三子。如果有，则说明有一方胜利，如果没有则继续搜

索，直到有一方胜利或者搜索完整个棋盘

二、实验原理与步骤

1. 定义全局变量

WIN 表示游戏获胜；DRAW 表示平局； LOSS 表示游戏失败

定义 AI_MARKER 为'X'，表示它下的棋子为 X，并且用 X 表示它自己。

定义 PLAYER_MARKER 为 'O'，表示它下的棋子为 O，并且用 O 表示游戏方。

定义 EMPTY_SPACE 为 '-'，表示棋盘的该位置还没被占用

定义初始树深 START_DEPTH 为 0

2. 函数功能

Print_game_state() 打印游戏方的游戏结果

Wining_state() 定义所有可能获胜的方式（如某一行，某一列或对角线）

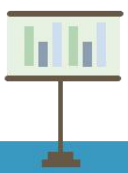
Print_board()　打印棋盘

Get_legal_moves() 获取所有可用的合法移动

Position_occupied()　判断该位置是否能被使用（即 AI 方或 player 方是否在这个位置下了棋）

get_occupied_positions() 获取给定一方，所有它下棋的位置

Board_is_full()　查看棋盘是否为空



Game_is_won()　查看该方是否赢得游戏

Get_opponent_marker()　切换下棋方

Get_board_state() 判断谁赢谁输

minimax_optimization ()　使用 alpha-beta 剪枝法，进行剪枝，确定 AI 下棋的位置

Game_is_done() 判断游戏是否结束

3. Main 函数

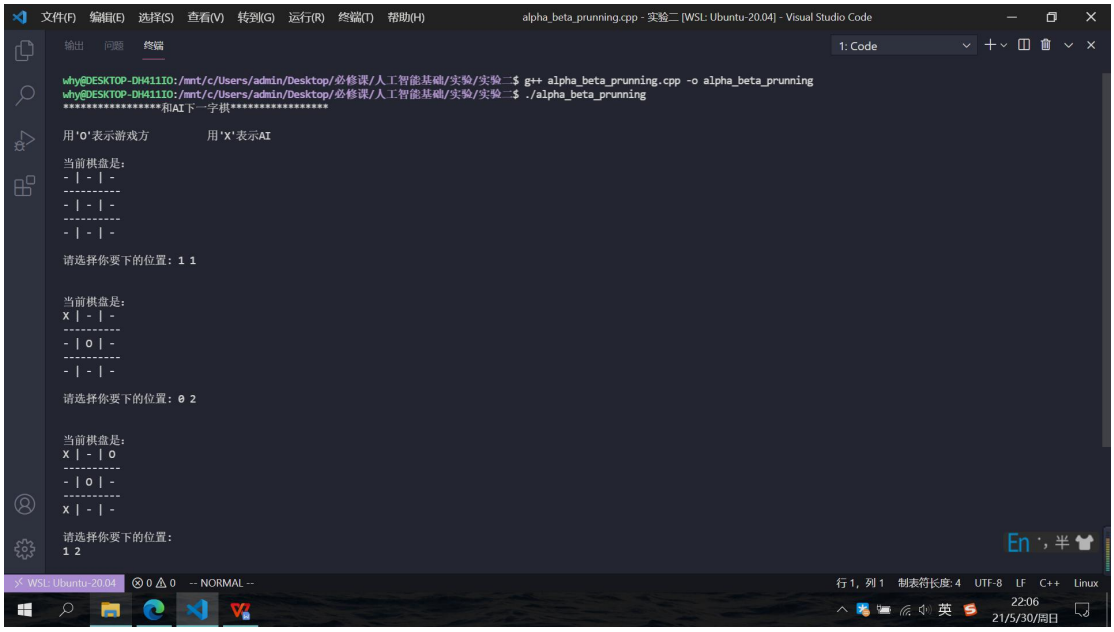
首先初始化棋盘，并用相应的符号表示出 player 方和 AI 方。

然后进行一个循环，player 方进行下棋（需判断其下的位置是否合理，棋盘是否满了等情况），然后轮到 AI 方下棋，直到游戏结束。

最后，判断游戏的获胜方，并输出相应结果。

三、实验结果与分析

实验运行如下图：


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
alpha_beta_prunning.cpp - 实验二 [WSL: Ubuntu-20.04] - Visual Studio Code
输出 问题 终端
1: Code
why@DESKTOP-DH411IO:/mnt/c/Users/admin/Desktop/必修课/人工智能基础/实验/实验二$ g++ alpha_beta_prunning.cpp -o alpha_beta_prunning
why@DESKTOP-DH411IO:/mnt/c/Users/admin/Desktop/必修课/人工智能基础/实验/实验二$ ./alpha_beta_prunning
**********和AI下一字棋**********
用'O'表示游戏方 用'X'表示AI
当前棋盘是:
- | - | -

- | - | -

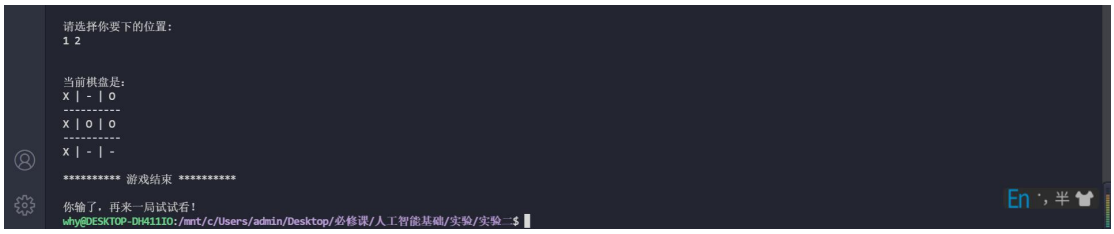
- | - | -
请选择你要下的位置: 1 1
当前棋盘是:
X | - | -

- | O | -

- | - | -
请选择你要下的位置: 0 2
当前棋盘是:
X | - | O

- | O | -

X | - | -
请选择你要下的位置:
1 2
WSL: Ubuntu-20.04
-- NORMAL --
行 1, 列 1 制表符长度: 4 UTF-8 LF C++ Linux
22:06
21/5/30/周日


请选择你要下的位置:
1 2
当前棋盘是:
X | - | O

X | O | O

X | - | -
********** 游戏结束 **********
你输了，再来一局试试看!
why@DESKTOP-DH411IO:/mnt/c/Users/admin/Desktop/必修课/人工智能基础/实验/实验二$

教师签字：

年 月 日