

**课 程 实 验 报 告**

**课程名称： 人工智能导论**

**报告主题： 井字棋α−β剪枝算法实现**

**专业班级： 计算机3班**

**学 号： U202015362**

**姓 名： 计胜翔**

**指导教师： 金 燕**

**报告日期： 2021.12.31**

**计算机科学与技术学院**

目录

[1题目分析 3](#_Toc91857631)

[1.1题目要求 3](#_Toc91857632)

[1.2需求分析 3](#_Toc91857633)

[1.3知识介绍 4](#_Toc91857634)

[2算法介绍 5](#_Toc91857635)

[2.1极大极小搜索算法 5](#_Toc91857636)

[2.2α−β剪枝搜索算法 10](#_Toc91857637)

[3程序设计 17](#_Toc91857638)

[3.1相关常量的定义和数据存储结构 17](#_Toc91857639)

[3.2交互界面设计 17](#_Toc91857640)

[3.3函数设计 18](#_Toc91857641)

[4实验结果及分析 20](#_Toc91857642)

[4.1实验结果测试 20](#_Toc91857643)

[4.2分析 22](#_Toc91857644)

[5心得体会 23](#_Toc91857645)

**井字棋α−β剪枝算法实现**

## 1题目分析

### 1.1题目要求

井字棋是一个很简单的游戏，双方在一个三行三列的棋盘上对弈，轮流走步，先后在空格上摆一个自己的棋子，若有一方先使自己的三个棋子同行、同列、对角线时，则判定这一方获胜，若步数执行完，无获胜方，则为平局。

本题要求使用α−β剪枝算法实现井字棋游戏，并分析α−β剪枝算法的优缺点。

### 1.2需求分析

本题的目标是基于α−β剪枝算法的井字棋游戏程序。要求游戏可玩，具有一定的、简单的交互性。重点在于α−β剪枝算法。

事务主要处理流程如图1所示。



图 1 程序流程图

### 1.3知识介绍

本题主要涉及到博弈搜索中的知识。

博弈，是一类具有智能行为的竟争活动。如下棋、沙盘推演……

博弈搜索，多智能体参与的一种搜索方法。首先需定义搜索的状态空间图，即构建博弈树。搜索的规则是双方交替进行。为评估每次搜索的效果，引入一个评估函数。一个智能体的搜索目标是找到一个节点，使得评估函数值极大化；而另一个智能体的搜索目标则是使评估函数值极小化。计算机象棋、计算机围棋等使用的就是博弈搜索法。

博弈搜索就是基于博弈树的搜索。博弈树有着以下几个特点：  
1、博弈树的初始状态是初始节点。

2、博弈树中的“或”节点、“与”节点逐层交替出现。本方走步的扩展节点是“或”关系，对方走步扩展的节点是“与”关系，双方轮流扩展节点。假设有两方博弈， 若MAX先走则处于奇数深度的节点都应由MAX走，所有偶数层都应该由MIN走。

3、整个博弈过程始终站在某一方的立场上。(MAX或MIN)

4、所有能使本方获胜的终局，相应的节点是可解节点，而所有对方获胜的终局都是不可解节点。

更重要的是博弈树的算法步骤，具体如下：（假设站在MAX方）

1、以当前状态为根节点，生成一棵博弈树（展开若干层）；

2、对当前展开的博弈树的每一个叶节点，利用估值函数给出估计值；

3、从叶节点开始，一层层回溯。过程中，利用极大/极小分析方，给出每一个节点的判定值；

4、MAX方选择下一层中判定值最大的节点，作为其下一状态。

其中，极大极小搜索算法和α−β剪枝算法是博弈搜索中的两种基本算法，其详细介绍见第2部分的算法介绍。

## 2算法介绍

本题的主要算法是极大极小搜索和α−β剪枝搜索算法。其详细介绍如下：

### 2.1极大极小搜索算法

#### 2.1.1算法原理

局面估价函数：给每个局面规定一个估价函数值f，评价它对于己方的有利程度。胜利的局面的估价函数值为+∞，而失败的局面的估价函数值为–∞。

Max局面：假设这个局面轮到己方走，有多种决策可以选择，其中每种决策都导致一种子局面。由于决策权在我们手中，当然是选择估价函数值f最大的子局面，因此该局面的估价函数值等于子局面f值的最大值，把这样的局面称为 max局面。

Min局面：假设这个局面轮到对方走，它也有多种决策可以选择，其中每种决策都导致一种子局面。但由于决策权在对方手中，在最坏的情况下，对方当然是选择估价函数值 f 最小的子局面，因此该局面的估价函数值等于子局面 f 值的最小值，把这样的局面称为min局面。

假设当前是MAX局面，即站在MAX方，需估算当前博弈树节点的估价函数值。

叶节点的估值方法如下：

1、对MAX有利的节点，其估价函数取正值；

2、对MIN有利的节点，其估价函数取负值；

3、使双方均等的节点，其估价函数取接近于0的值；

非叶节点的估值方法如下：从叶节点开始，向上回溯倒推，

1、对“或”节点（MAX节点）， 选其子节点中一个最大的得分作为其得分；

2、对“与”节点（MIN节点）， 选其子节点中一个最小的得分作为其得分。

如此一步步回溯（倒推), 直至求出初始节点的倒推值为止，这一过程称为极大极小搜索过程。

下面举例说明：

假设当前局面有如下博弈树，且已知其叶节点的估值：

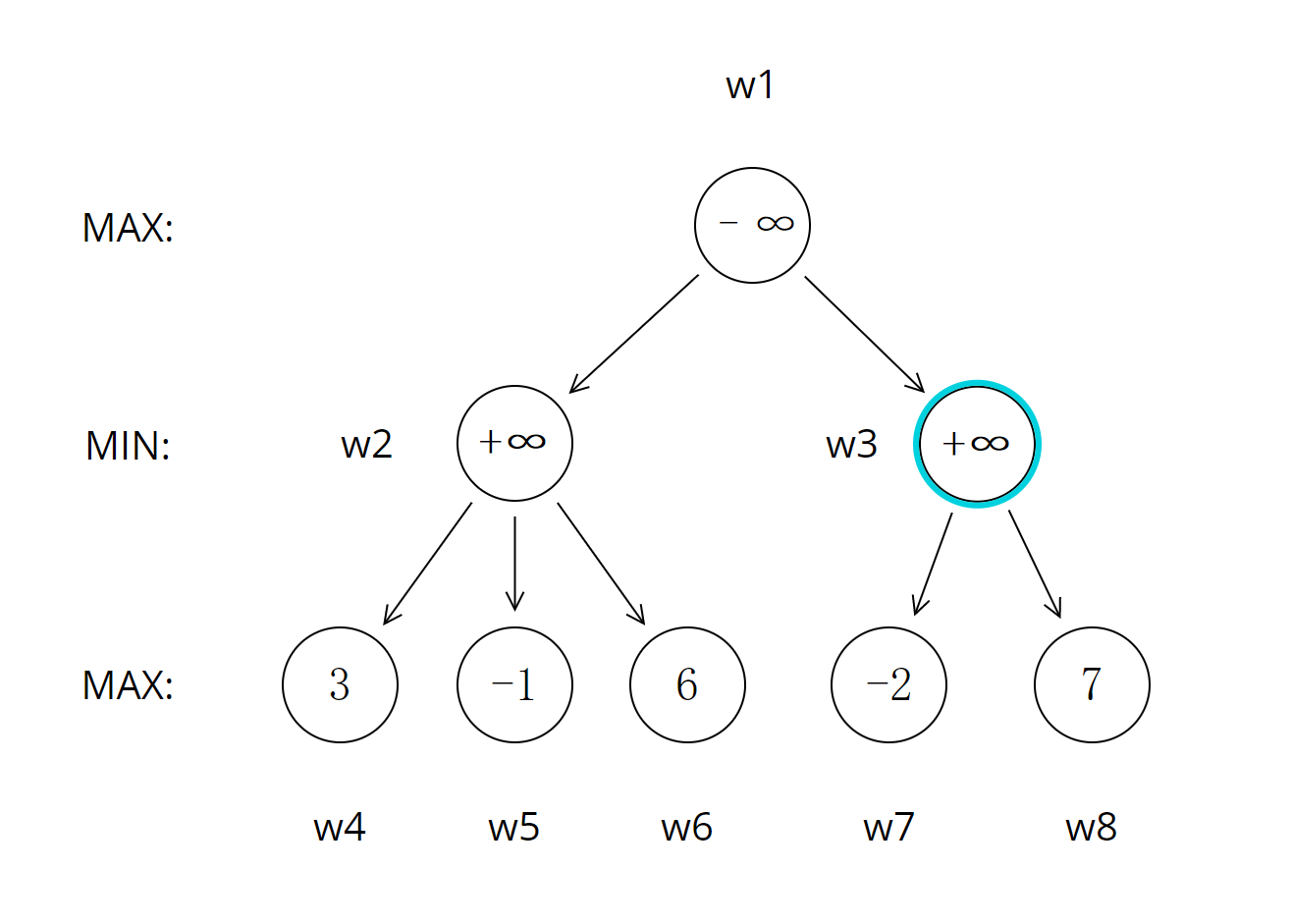


图 2：初始博弈树

上图的博弈树中，非叶节点的MAX节点（w1）的值置为-∞，而非叶节点的MIN节点（w2和w3）的值置为+∞。先通过倒推回溯得到w2和w3节点的估值，如下图所示：

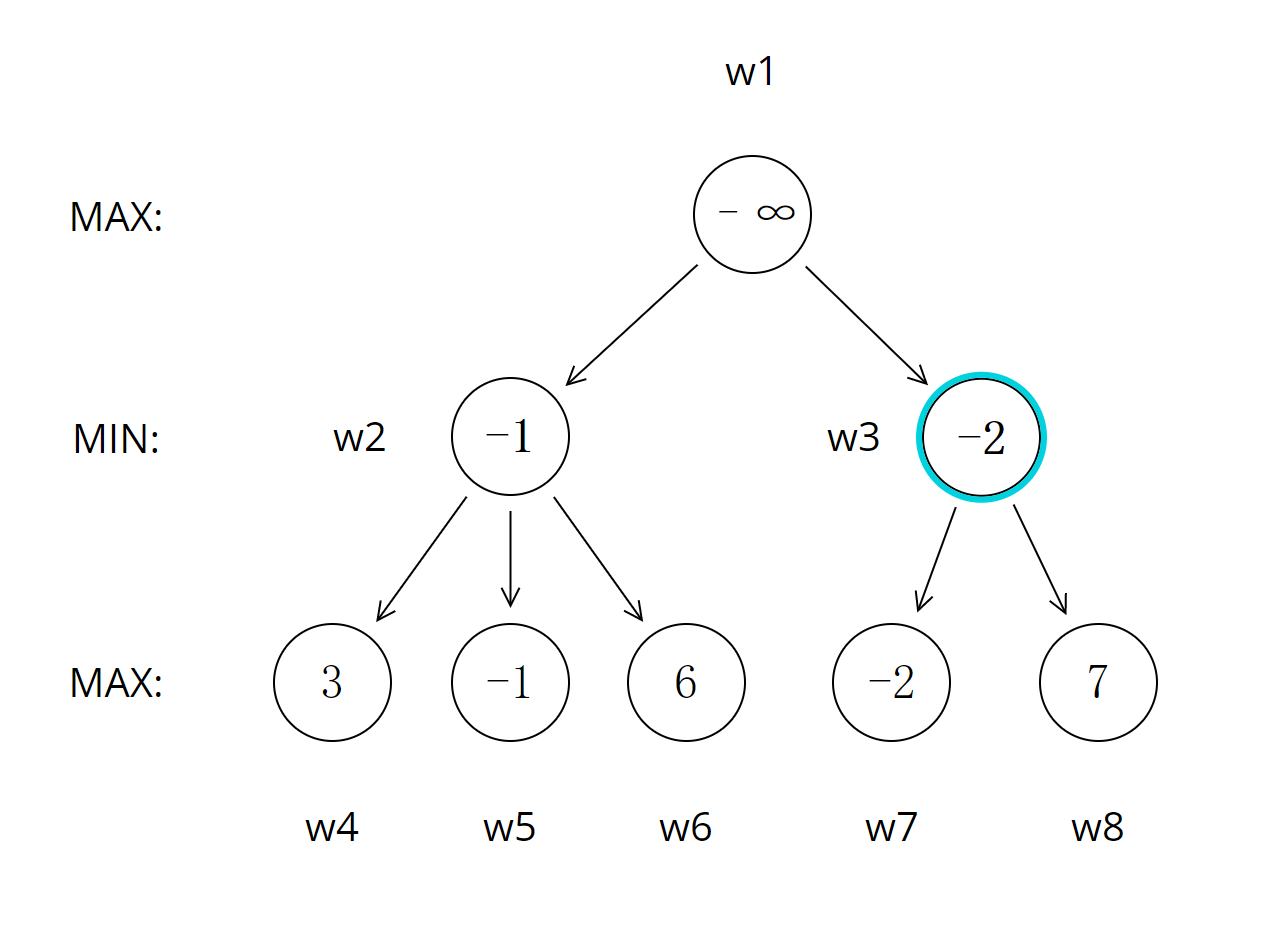


图 3：倒推w2、w3节点的估值

先通过倒推回溯得到w1节点的估值，如下图所示：

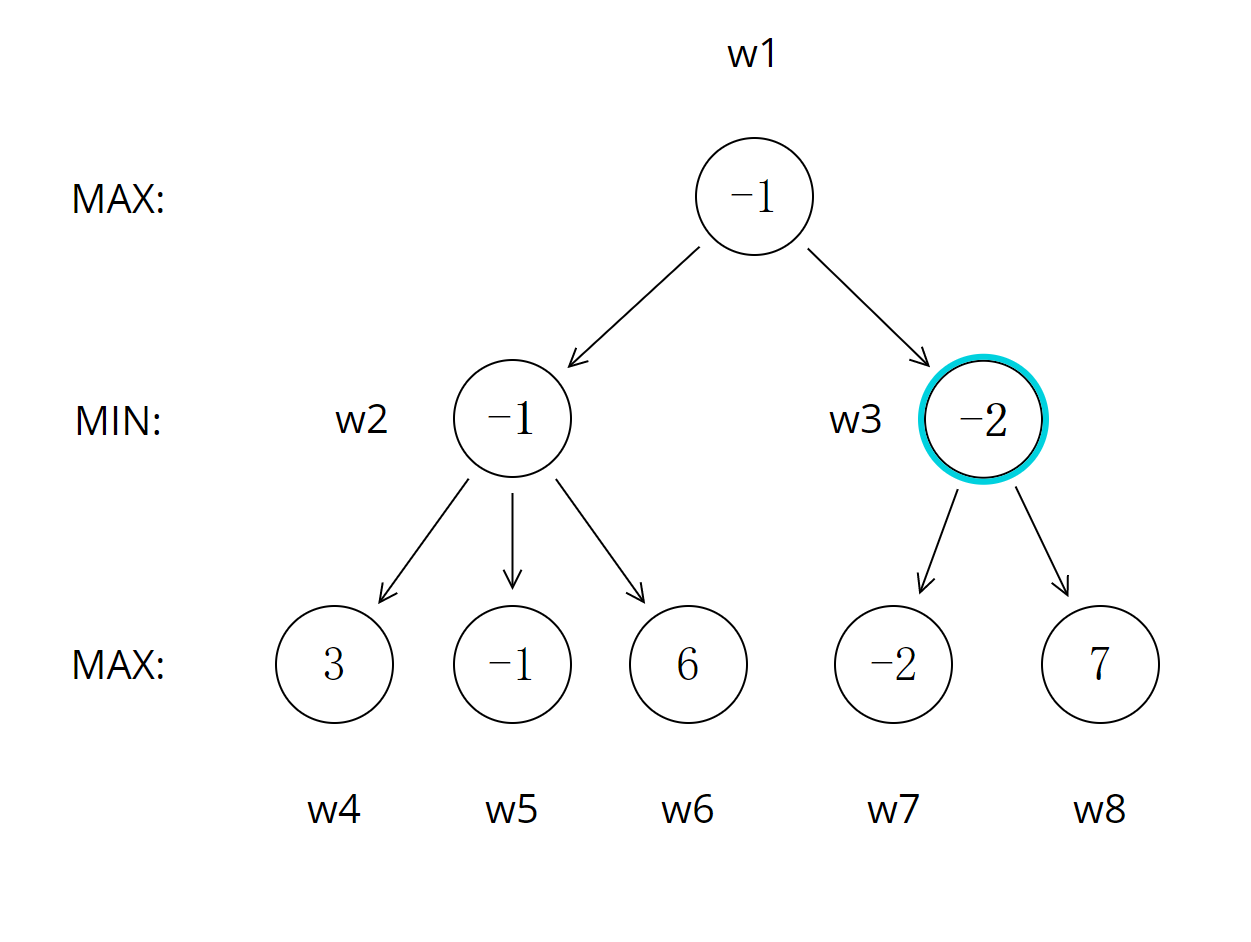


图 4：倒推w1节点的估值

由上述过程可以看到，估值采用倒推的方式，从最底层开始倒推到最顶层，即可得到最优的搜索路径，应用到本题的井字棋游戏中就可得到当前局面的最佳落子方式，且可以通过递归实现。

#### 2.1.2算法步骤

极大极小搜索算法的算法步骤用伪代码表示如下：

|  |
| --- |
| // player = 1表示轮到己方， player = 0 表示轮到对方  // cur\_node 表示当前局面(结点)  maxmin(player, cur\_node)  {  if 达到终结局面  return 该局面结点的估价值 f  end  if player == 1 // 轮到己方走  best = -∞ // 己方估价值初始化为 -∞  for 每一种走法 do  new\_node = get\_next(cur\_node) // 遍历当前局面 cur\_node 的所有子局面  val = maxmin(player^1, new\_node); // 把新产生的局面交给对方，对方返回一个该局面的估价值  if val > best  best = val;  end  end  return best;  else // 轮到对方走  best = +∞ // 对方估价值初始化为 +∞  for 每一种走法 do  new\_node = get\_next(cur\_node) // 遍历当前局面 cur\_node 的所有子局面  val = maxmin(player^1, new\_node); // 把新产生的局面交给对方，对方返回一个该局面的估价值  if val < best  best = val;  end  end  return best;  end  } |

#### 2.1.3在井字棋游戏中的应用

将上述极大极小搜索算法应用本题的井字棋游戏中。

本题中，最终局面只有3种结果：电脑胜利、人类胜利和平局。因此，在局面估价函数中，若电脑胜利，则估值（即函数返回值）为1；若用户胜利，则估值为-1；否则，即为平局，估值为0。并且在本程序中，用“X”表示电脑下的棋，用“O”表示用户下的棋。

下面举例说明，假设有一当前局面，其极大极小搜索和倒推回溯过程如下图所示：

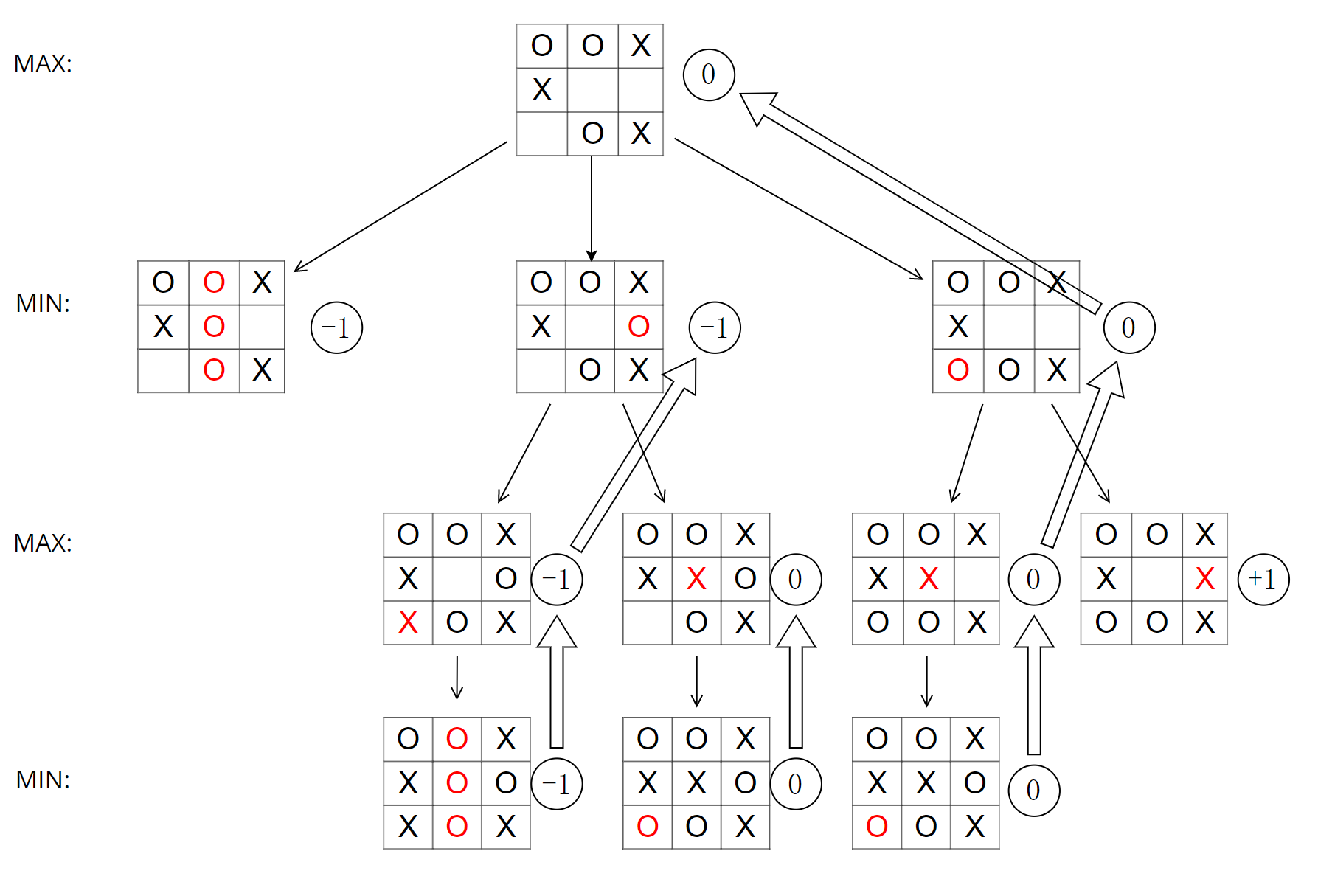


图 5：井字棋游戏中的极大极小搜索

将其简化成博弈树，如下图所示：

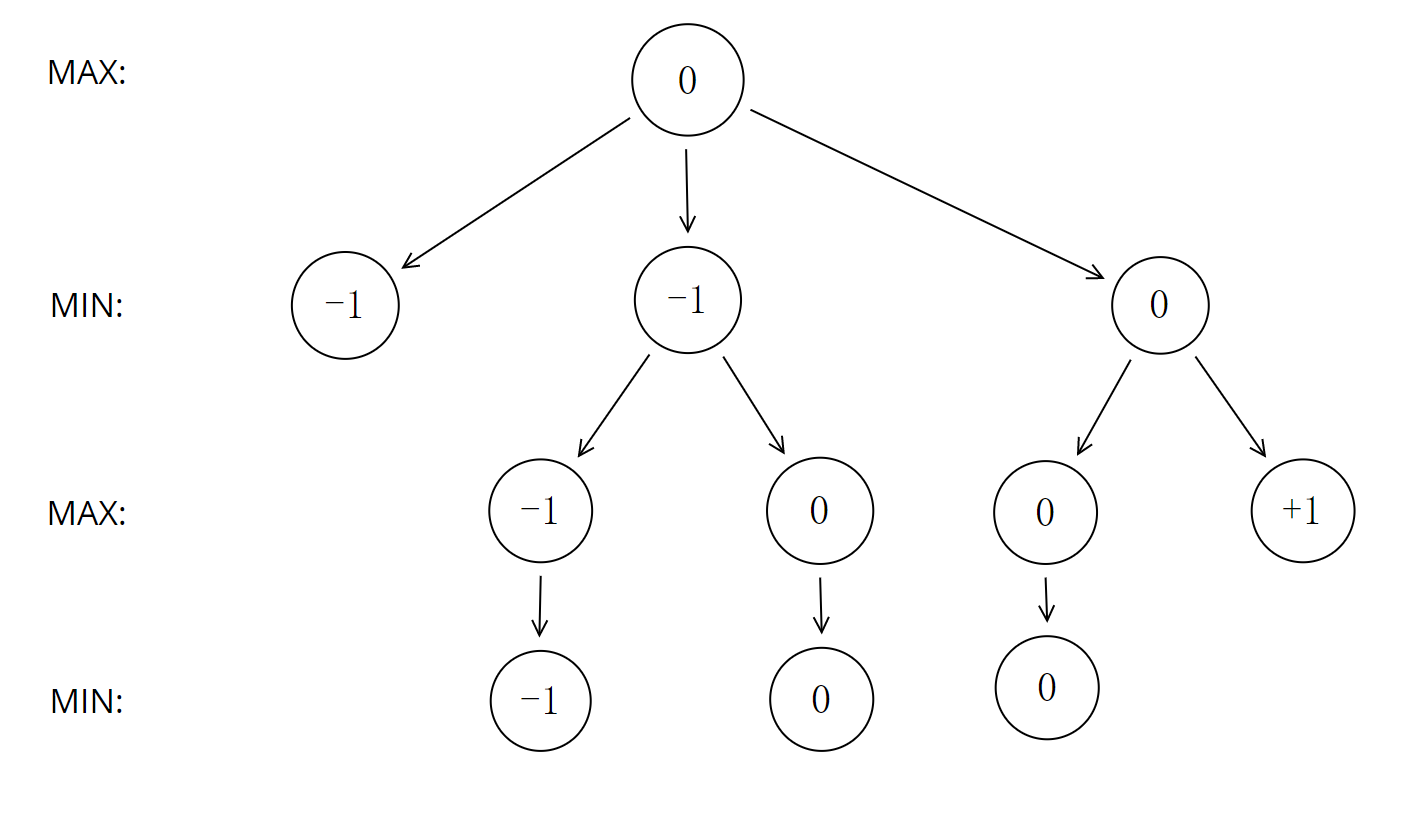


图 6：井字棋游戏中的博弈树

其中，往下搜索时，对于非叶节点，MAX节点的估值初始化为-2，以表示-∞，MIN节点的估值初始化为+2，以表示+∞。

本算法在本题的井字棋游戏中的具体实现详见3程序设计。

### 2.2α−β剪枝搜索算法

极大极小搜索算法实际上是一种暴力搜索、穷举的方法。但是，实际问题中的所有局面所产生的博弈树一般都是非常庞大的多叉树，若采用上述的极大极小搜索，需要计算所有的节点估值，时间复杂度会很高，所以并不能依靠暴力搜索来寻找最佳解法。因此需要用到一些剪枝手段，常用的比较初级的有α−β剪枝算法。其具体介绍如下：

#### 2.2.1算法原理

α−β剪枝算法的基本思想是：通过节点估计值的上下界，判断某些分支可以不搜索。

在搜索过程中，首先使得搜索树的某个分支达到最大深度（或规定的深度），然后倒推或节点（MAX节点）的估价函数值（记为α值，作为下界，），倒推与节点（MIN节点）的估价函数值（记为β值，作为上界，）。对任何一个节点，当其某一个后继节点的值给定时，就可以确定该节点的上界（与节点）或该节点的下界（或节点），从而判断其他分支是否可以不搜索。

所以α−β剪枝算法的两个规则是：

1、任何与节点的β值，如果小于等于其父节点的当前α值的下界，则对该节点以下的分枝可停止搜索；

2、任何或节点的α值，如果大于等于其父节点的当前β值的上界，则对该节点以下的分枝可停止搜索。

下面举例说明，假设有如下博弈树，且已知3个叶节点的估值：

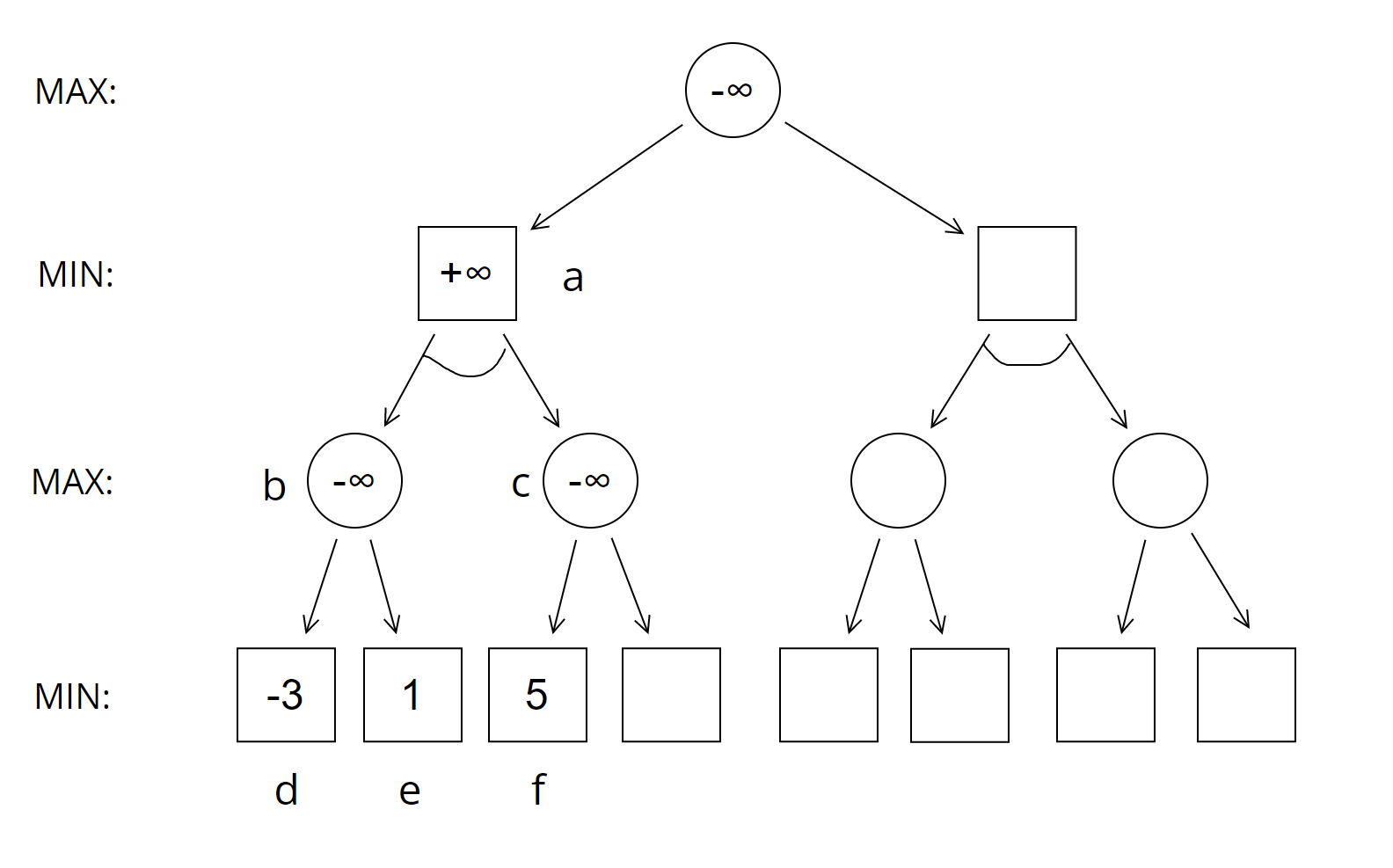


图 7：初始博弈树

第一步：由d节点的估值β=-3倒推，得到其父节点b的α值满足。

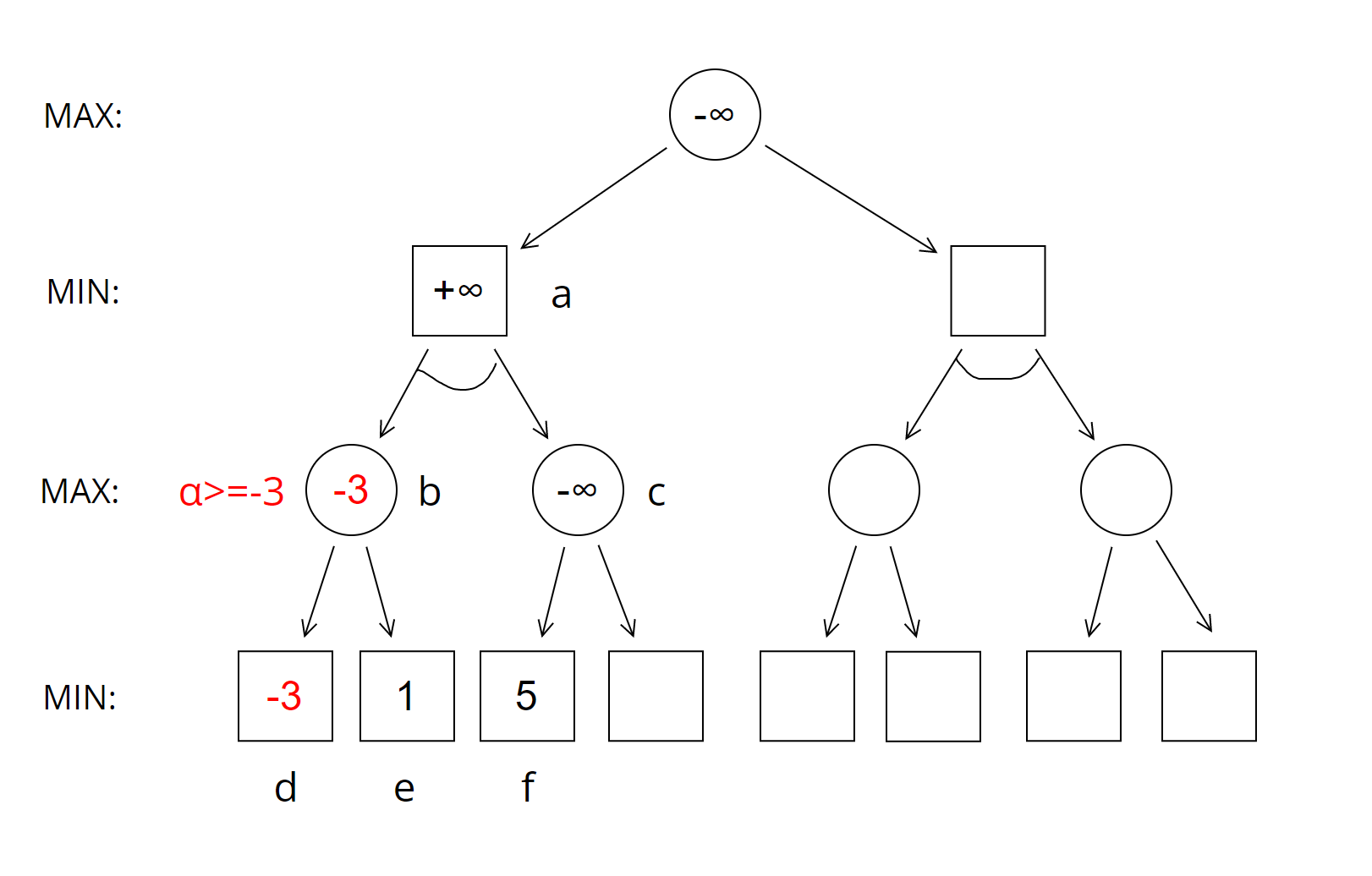


图 8：倒推得到b节点估值的下界

第二步：由e节点的估值β=1倒推，得到b节点的估值α=1。

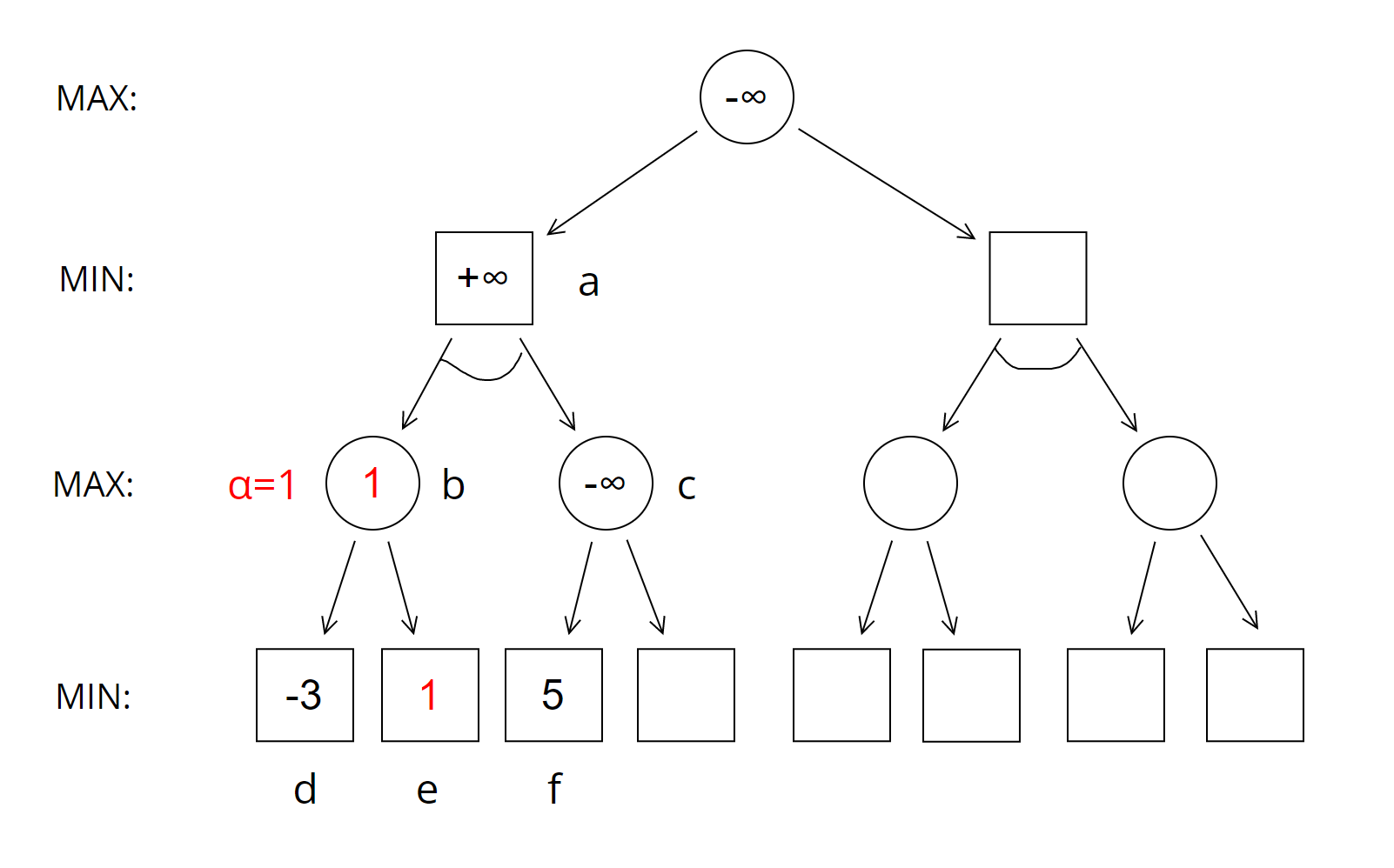


图 9：确定b节点的估值

第三步：由b节点的估值α=1倒推，得到父节点a的估值。

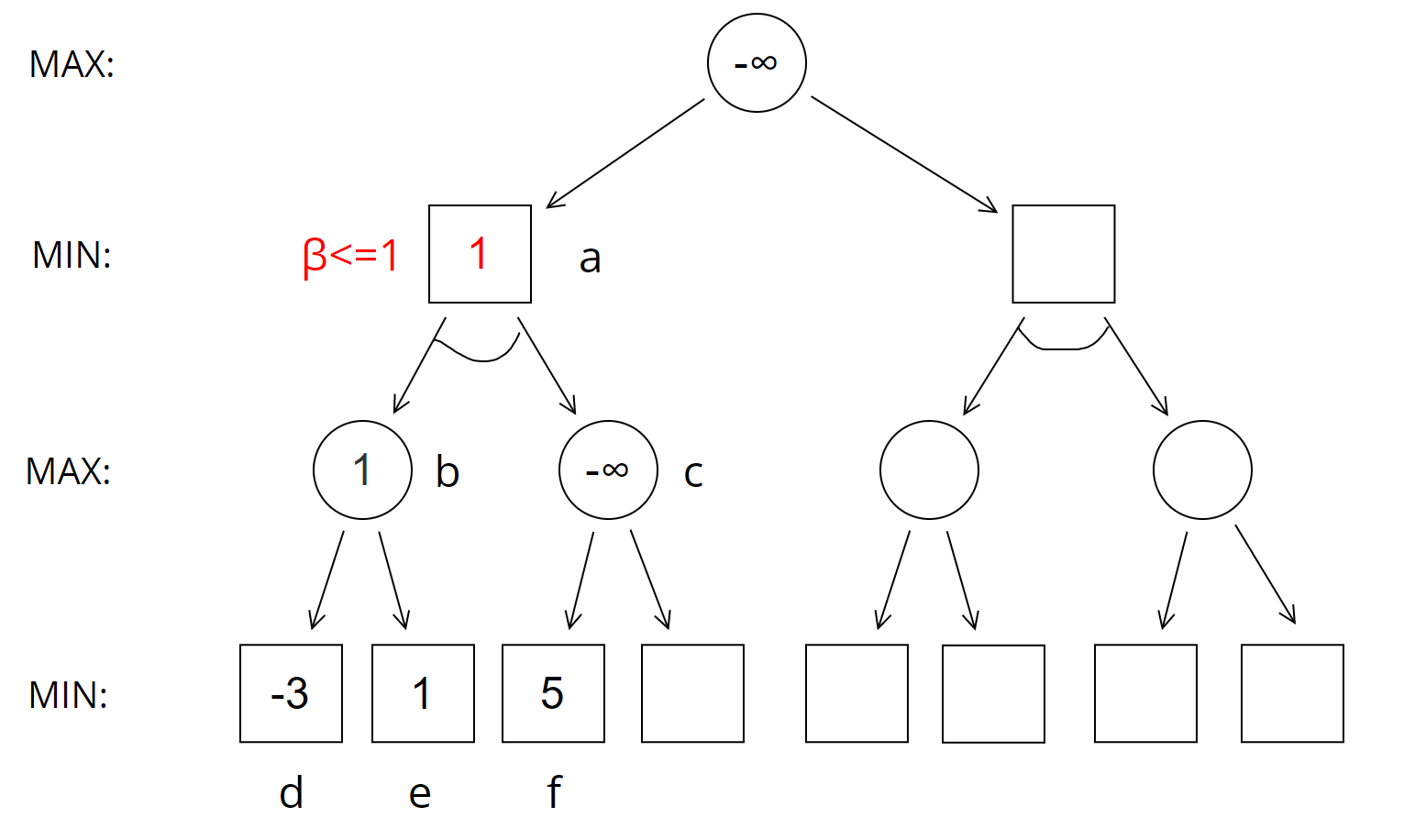


图 10：倒推得到a节点估值的上界

第四步：由f节点的估值β=5倒推，得到父节点c的估值。

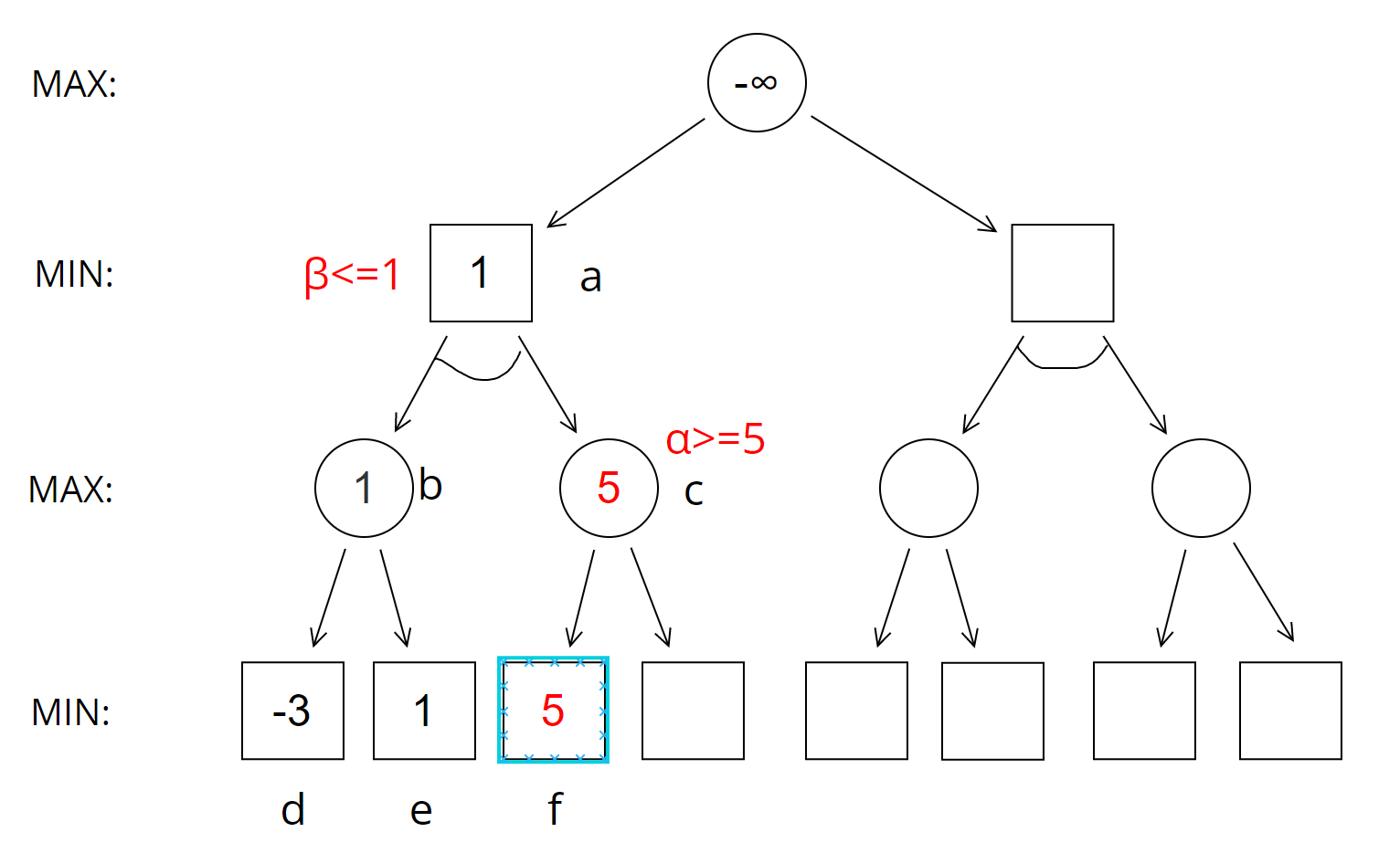


图 11：倒推得到c节点估值的下界

第五步：由于a节点的估值，而其子节点c的估值，c节点的下界大于a节点的上界，所以c节点的估值不可能降低其父节点a的估值。因此对于c节点的其他分支可以进行剪枝操作，不必搜索。同时确定c节点的估值α=5，a节点的估值β=1。

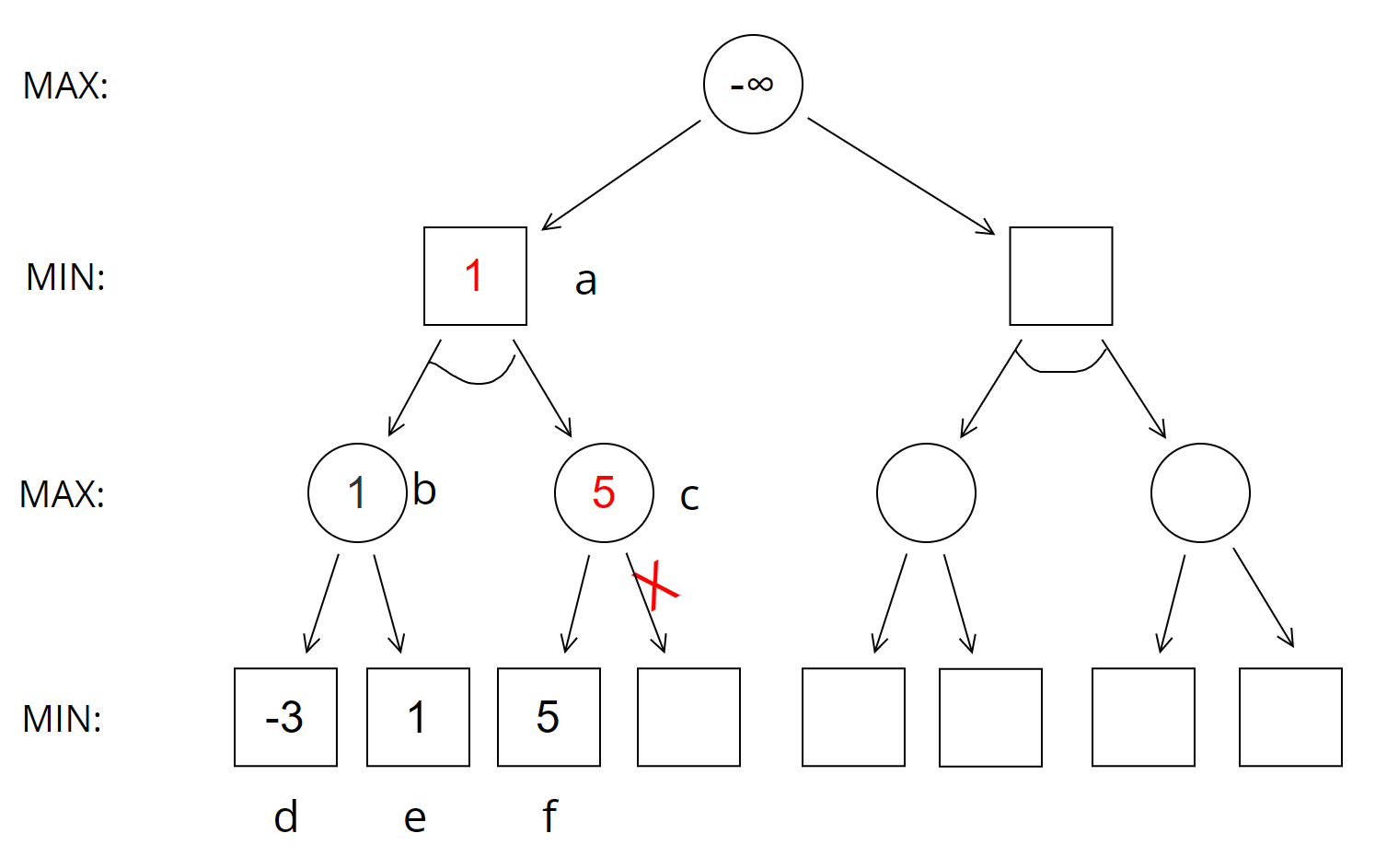


图 12：剪枝，确定c节点的估值

循环上述步骤，即可得到博弈树各节点的估值，并且剪掉一些不必搜索的分支，从而对极大极小搜索进行优化。

#### 2.2.2算法步骤

α−β剪枝搜索算法的算法步骤用伪代码表示如下：

|  |
| --- |
| // player == 1表示轮到己方， player == 0 表示轮到对方  // cur\_node 表示当前局面(结点)  //alpha表示父节点为或节点时其估值的下界，beta表示父节点为与节点时估值的上界  alpha-beta(alpha, beta, player, cur\_node)  {  if 达到终结局面  return 该局面结点的估价值 f  end  if player == 1 // 轮到己方走  alpha\_1 = -∞ // 己方或节点估值初始化为 -∞  for 每一种走法 do  new\_node = get\_next(cur\_node) // 遍历当前局面 cur\_node 的所有子局面  val = alpha-beta(alpha\_1, 0, player^1, new\_node); // 把新产生的局面交给对方，对方返回一个该局面的估值  if val > alpha\_1 //更新当前节点的估值  alpha\_1 = val;  if alpha\_1>=beta //剪枝  return beta;  end  end  return best;  else // 轮到对方走  beta\_1 = +∞ // 对方估值初始化为 +∞  for 每一种走法 do  new\_node = get\_next(cur\_node) // 遍历当前局面 cur\_node 的所有子局面  val = alpha-beta (0, beta\_1, player^1, new\_node); // 把新产生的局面交给对方，对方返回一个该局面的估值  if val < beta\_1 //更新当前节点的估值  beta\_1 = val;  if beta\_1 <= alpha //剪枝  return alpha;  end  end  return best;  end  } |

#### 2.2.3在井字棋游戏中的应用

将上述α−β剪枝搜索算法应用本题的井字棋游戏中。

由以上所述可知，只需在极大极小搜索算法上稍加改动即可。在搜索过程中，当遍历当前局面的所有子局面并进行估值时，在每次更新完估值后，只需加个判断，判断当前节点的β值是否小于等于其父节点的当前α值的下界，或当前节点α值是否大于等于其父节点的当前β值的上界，这样就可以完成α−β剪枝，优化时间复杂度。其余部分基本相同。

下面用2.1.3中所举的例子，其剪枝后的博弈树如下图所示：

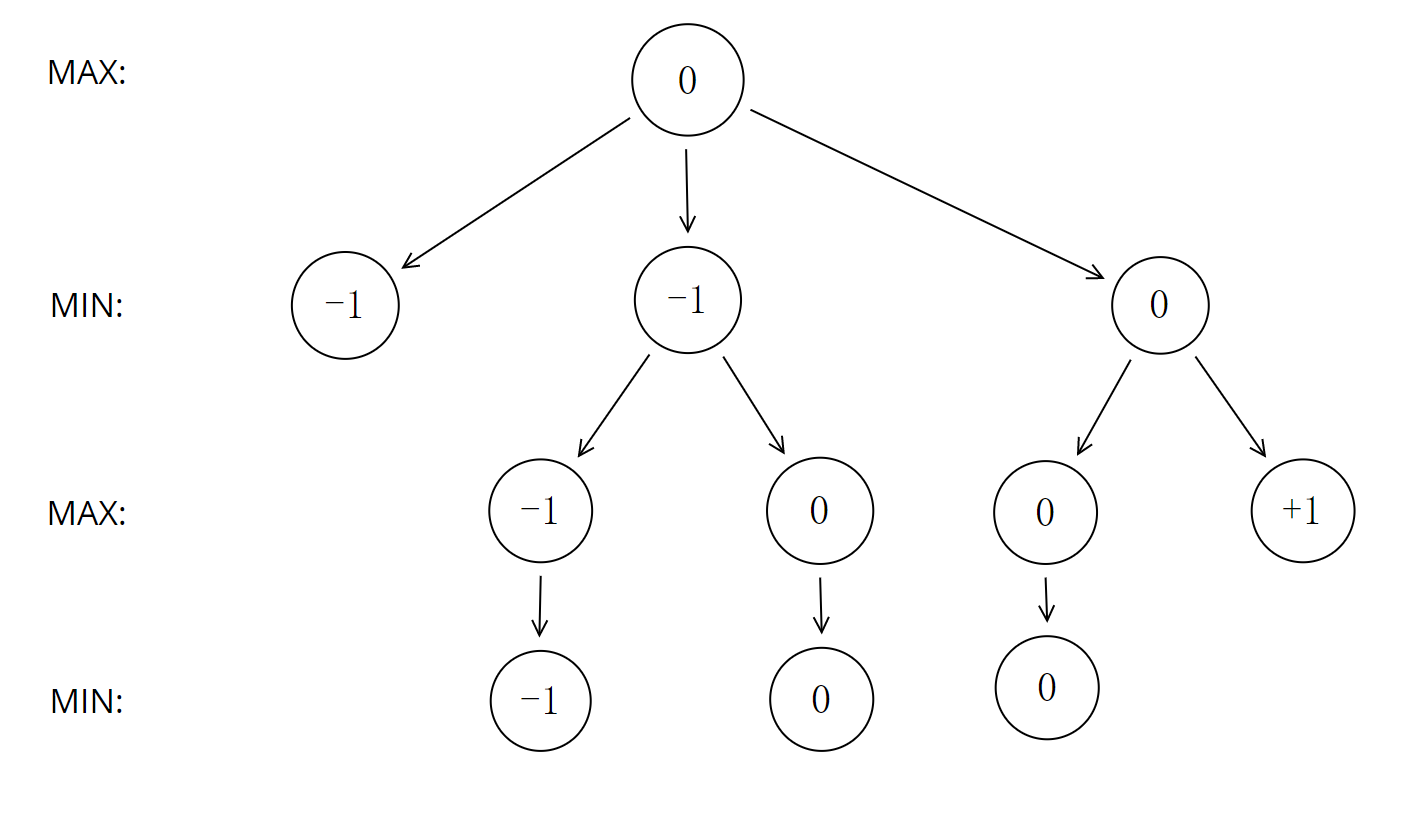




图 13：井字棋游戏中的博弈树剪枝

## 3程序设计

### 3.1相关常量的定义和数据存储结构

1、相关常量的定义

为了方便在程序中理解和使用相关的常量，在程序开头作如下定义：

|  |
| --- |
| #define COM 1 //表示电脑下的棋，打印时用‘X’表示  #define MAN -1 //表示人类下的棋，打印时用‘O’表示 |

2、数据存储结构

本程序中使用了两个全局变量，用于存储重要数据，如下：

|  |
| --- |
| int board[3][3]; //记录3X3棋盘，0表示空位置，1表示电脑下的棋，-1表示人类下的棋  int num; //遍历搜索时 记录已下的棋子个数 |

### 3.2交互界面设计

打印棋盘时，用“X”表示电脑下的棋，用“O”表示用户下的棋，空格表示该位置未被占领。

程序启动时，输出空棋盘，用户先手，程序等待用户输入位置坐标。

用户每输入一次坐标，用户回合结束，实时输出当前棋盘。

然后电脑根据算法做出决策，选择位置下棋，电脑回合结束。

……

重复上述步骤，直至游戏出现结果，并输出结果。

交互界面的棋盘样式如下图所示：

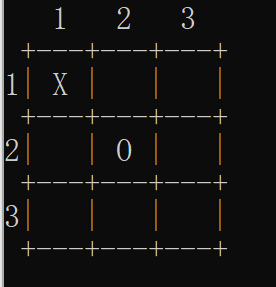


图 14：输出的棋盘样式

其中，棋盘的左侧和顶端是行列号。

### 3.3函数设计

1、void init(void)

功能：初始化棋盘

输入参数：空

返回值：空

具体内容：将数组board的所有元素置为0，并将棋子个数num置为0。

2、void show(void)

功能：打印棋盘

输入参数：空

返回值：空

具体内容：根据棋盘board打印棋盘，元素为0的对应位置用空格表示，元素为1的对应位置用“X”表示，元素为-1的对应位置用“-1”表示。

3、int value(void)

功能：局面估价函数

输入参数：空

返回值：为int类型。返回当前局面的估价函数值。

具体内容：当前局面中，若电脑胜利，则返回+1；若人类胜利，则返回-1；否则，返回0。

4、int alpha\_beta\_Sarch(int alpha, int beta, int player)

功能：α−β剪枝搜索，得到博弈树中当前节点的估值。

输入参数：（电脑为己方，人类为对方）

int alpha：当前或节点的父节点的当前估值；当前节点为与结点时，值为0。

int beta：当前与节点的父节点的当前估值；当前节点为或结点 时，值为0。

int player：玩家，为1表示当前是人类回合；为0表示当前是电 脑回合。

返回值：为int类型。返回博弈树中当前节点的估值。

具体内容：根据2.2中所述的α−β剪枝搜索算法，得到当前局面电脑的最佳估值。

## 4实验结果及分析

### 4.1实验结果测试

程序启动，游戏开始，输出空棋盘，等待用户输入。

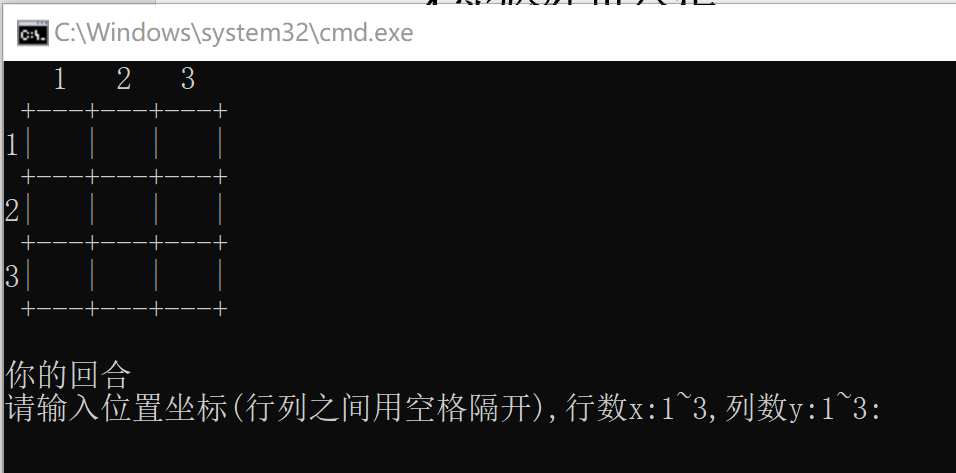


图 15游戏开始，等待用户输入

输入坐标（2，2），电脑做出相应对策。

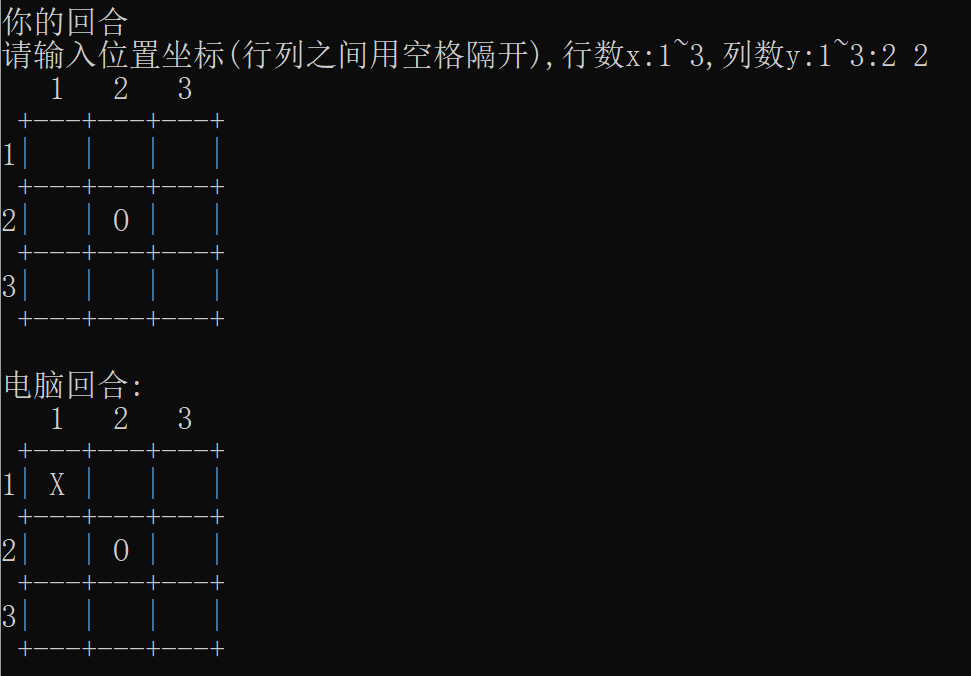


图 16输入坐标（2，2）

输入坐标（1，3），电脑做出相应对策。

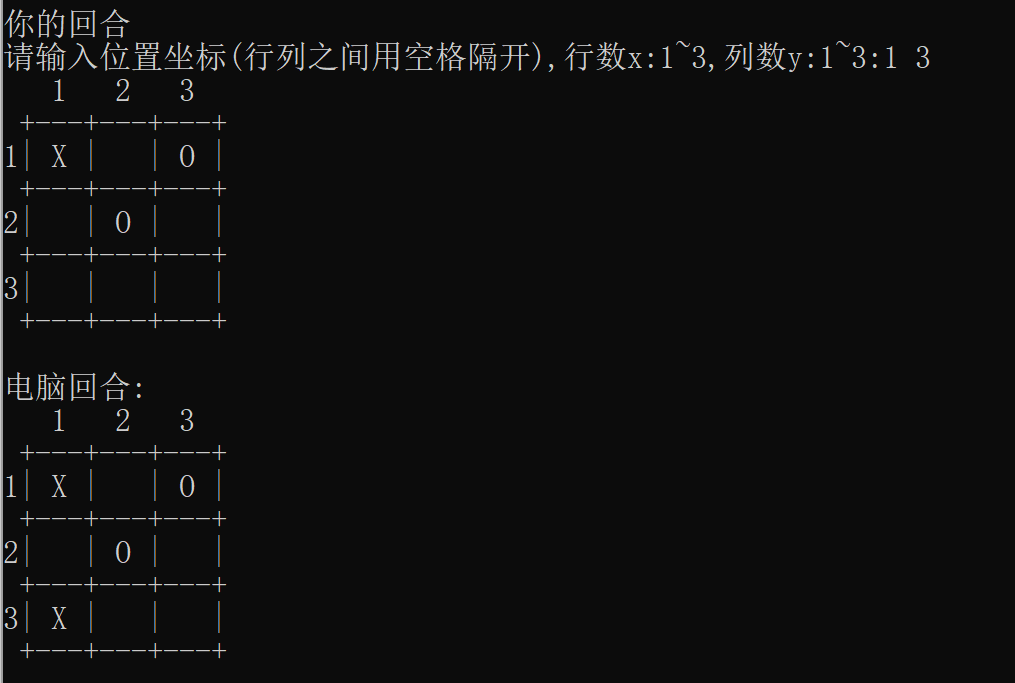


图 17输入坐标（1，3）

输入坐标（2，1），电脑做出相应对策。

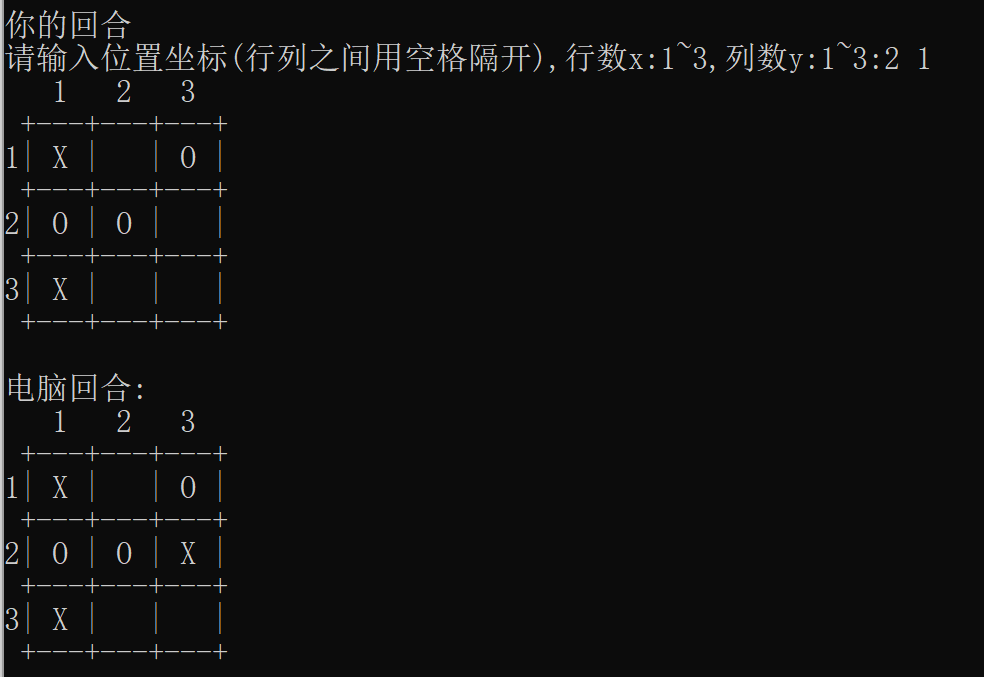


图 18输入坐标（2，1）

输入坐标（3，2），电脑做出相应对策。

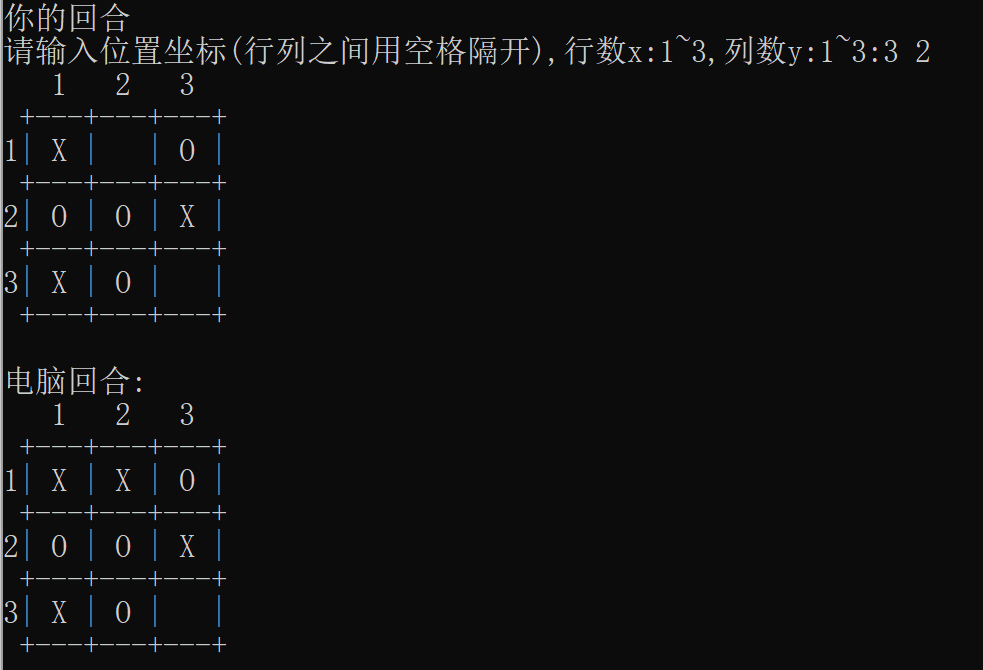


图 19输入坐标（3，2）

输入坐标（3，3），平局，游戏结束。

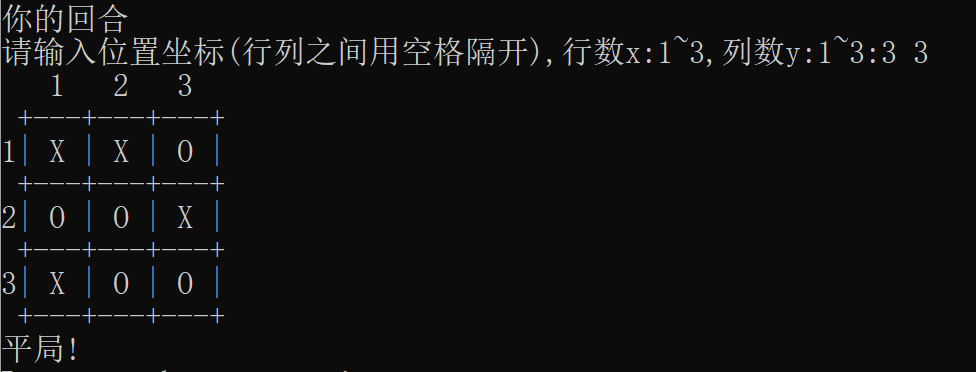


图 20平局，游戏结束

### 4.2分析

α−β剪枝搜索算法的优缺点如下;

1、优点：

理想情况下，使用α-β剪枝算法搜索博弈树时，可以剪去一些不必要的分枝，从而大大提高搜索效率。

2、缺点：

传统的α-β剪枝只是提供了剪枝的基本策略，保证了一定能得到正确解，但并没有保证它的效率。最坏的情况可能是从最坏的解按顺序搜到了最好的解，相当于没剪枝，还付出了额外的判断代价。

此外，α-β剪枝算法非常依赖对棋局的打分。像围棋这样十分复杂的情况下，对局面的判断非常复杂，棋子之间相互联系，不可能单独计算，所以对棋局打分非常困难。因此α-β剪枝算法并不适用于非常复杂的局面。

## 5心得体会

这次结课实验是我对课上所学的理论知识的实践，让我对本课程所学的人工智能有了更加深刻的体会和理解。

通过本次结课实验，我实现了基于α−β剪枝算法的井字棋游戏，对博弈树搜索中的极大极小搜索算法和α−β剪枝搜索算法有了更深的认识和理解，对人工智能的兴趣也更加浓厚。

实验的过程并不是一帆风顺。在学习完课件相关知识后，刚开始实验，就遇到了许多问题，如对搜索的过程和α−β剪枝的原理并不完全明白、不知道如何将算法应用到本题的井字棋游戏中等等。但是，在自己独立分析、与同学交流、上网查阅相关资料后，慢慢解决了问题，逐渐明确了实验目标和步骤。

本次实验所涉及到的算法知识只是人工智能中最基础的，人工智能还有许多知识和奥妙等着我们去探索。在人工智能飞速发展、影响巨大的今天，迫切需要我们学习人工智能知识，提升自身素养，培养创新能力。