第 3 章

对抗与博弈：井字棋

3.1 教 学 目 标

（1）能够设计实现对抗性游戏的程序框架。

（2）能够分析研究 Minimax 决策的优缺点，并提出改进思路。

（3）能够理解并应用对抗搜索的技术。

3.2 实验内容与任务

TicTacToe 游戏，也叫“井字棋”，是一款简单的双人游戏。在一个 3×3 的 9 个方格构成的棋盘中，两个玩家轮流选一个格子布放棋子，如果一方的棋子占据了水平、竖直或对角线上的 3 个格子，则取得胜利。请设计一个游戏的软件框架，包含一个应用 Minimax 决策的 AI 玩家, 能跟人类玩家对弈。图 3.1 是执棋子 × 的玩家游戏先行且下到第 5 步时的图，下面轮到执棋子 ○ 的玩家下棋。

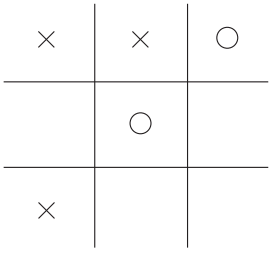

图 3.1　井字棋游戏（轮到玩家 ○ 下棋）

3.3 实验过程及要求

（1）实验环境要求：Windows/Linux 操作系统，Python 编译环境，random 等程序库。

（2）依次实现 Game、TicTacToe、Minimax_player、query_player 等各模块。

（3）进行人机对战、机机对战、人人对战的测试。

（4）将 TicTacToe 棋盘扩大，观察并讨论 Minimax_player 的表现。

3.4 相关知识及背景

从 20 世纪 50 年代人工智能学科初创，到 21 世纪深度学习技术的广泛应用，博弈游戏一直是人工智能研究的重要领域，五子棋、国际象棋、围棋领域的人工智能分别战胜了人类。其中 Minimax 决策树是博弈的基本手段，应用在各种人工智能博弈中。由于博弈游戏的规模决定了决策树的大小，也决定了应用 Minimax 策略的计算时间复杂性，因此完全应用 Minimax 策略的游戏只能适应小型游戏。井字棋游戏就是一个小型游戏，其决策树的高度只有 9，应用 Minimax 策略可以处理。本实验给出了一个博弈游戏的框架，并应用 Minimax 决策树实现人工智能玩家参与井字棋游戏。

3.5 实验教学与指导

本次实验有游戏模型、棋局状态、玩家 3 个主要对象，玩家根据模型知识和当前棋局，提供策略给出下一步行棋。

3.5.1 Minimax 决策

游戏的决策策略构成一棵决策树：不妨假设进行井字棋游戏的两个玩家名字是 Max 和 Min，当前局面下轮到 Max 行棋。根结点是玩家 Max 面对的棋局状态，称为 Max 结点。根据 Max 的行动选择，决策树生成多个子结点，构成了玩家 Min 面对的状态，称为 Min 结点。再下一层经 Min 选择行动后又是 Max 结点。假设有一个对棋局 s 的评分函数 Minimax(s)，值越大对 Max 越有利，则 Max 一定会选择一个行动，获得所有子结点中的最大的 Minimax 值，而 Min 则会选择一个行动，为所有子结点中具有最小的 Minimax 值，如图 3.2 所示。

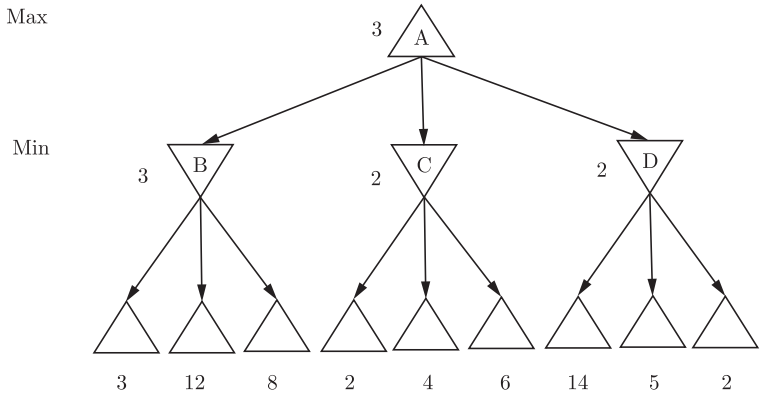

图 3.2 Minimax 决策树

根据上述关于 Minimax 值的描述，一个局面的 Minimax 可以由其子结点的 Minimax 得到，因此这是一个递归定义，还需要对叶子结点也就是终止局面的 Minimax 值进行定义。下面介绍一个比 Minimax 更直接但粗糙一点的局面评价函数，也就是局面的效用函数 Utility(s)，即

$$\text{Utility}(s)=\begin{cases}1, & \text{如果 } s \text{ 是 Max 胜的终局状态}\\ -1, & \text{如果 } s \text{ 是 Max 负的终局状态}\\ 0, & \text{如果 } s \text{ 是非终局状态，或者是平局状态}\end{cases} \tag{3.1}$$

规定终局状态的 Minimax 值等于其效用值，因此在井字棋游戏中，Minimax 函数可以定义为

$$\text{Minimax}(s)=\begin{cases}\text{Utility}(s), & \text{终局}\\ \max_{a\in\text{Action}(s)}\text{Minimax}(\text{Results}(s,a)), & \text{Max 结点}\\ \min_{a\in\text{Action}(s)}\text{Minimax}(\text{Results}(s,a)), & \text{Min 结点}\end{cases} \tag{3.2}$$

须强调的是本实验中，使用 Minimax 决策的玩家每次轮到自己行棋时都会新建一棵决策树进行计算，把自己当作 Max，把对方当作 Min。

3.5.2　游戏模型

首先用一个四元组定义游戏状态：GameState = namedtuple('GameState', 'to_move, value, board, moves')。在井子棋中，to_move 是当前行棋的玩家。

为记录简便，此处直接用所执棋子代表两个玩家，用字符“X”或者“O”表示。utility 是对当前棋局对先行者的效用，如公式 (3.1) 定义。board 是当前的棋盘，记录每个位置 (x,y) 的棋子字符“X”或者“O”，用 Python 的词典类型表示。moves 是当前棋局下可行位置 (x,y) 的列表。

游戏模型和 Problem 模型类似，需要定义 actions, result 等方法。

```
class Game:
    def actions(self, state):          #某状态下的行为集合
        raise NotImplementedError
    def result(self, state, move):     #采取某行为后的新状态
        raise NotImplementedError
    def terminal_test(self, state):    #判定是否终止状态
        return NotImplementedError
    def utility(self, state, player): #此函数返回针对 player 的效用
        if player == self.initial.to_move:
          return state.utility
```

```
        else:
          return -state.utility
    def to_move(self, state):          #获取当前状态下哪个玩家行棋
        return state.to_move
    def display(self, state):          #显示当前状态下的棋盘
        print(state)
    def play_game(self, *players):     #players 是玩家列表
                                       #每个 player 函数确定自己的行棋
        state = self.initial
        while True:
            for player in players:
                move = player(self, state)  #player 决定一个行棋
                state = self.result(state, move)
                if self.terminal_test(state):
                    self.display(state)
                    return self.utility(
                        state,self.to_move(self.initial))
```

3.5.3 玩家

在 Game 定义的游戏规则下，由玩家提供某状态下的走法。由于其功能单一，因此玩家写成一个函数。minimax_player 是使用 Minimax 策略的玩家。

```
def minimax_player(game,state):
    player = game.to_move(state)

    def max_value(state): #Max 结点计算 Minimax
        if game.terminal_test(state):
            return game.utility(state, player)
        v = -np.inf
        for a in game.actions(state):
            v = max(v, min_value(game.result(state, a)))
        return v

    def min_value(state):  #Min 结点计算 Minimax
        if game.terminal_test(state):
            return game.utility(state, player)
        v = np.inf
        for a in game.actions(state):
            v = min(v, max_value(game.result(state, a)))
```

```
        return v

    return max(game.actions(state),
               key=lambda a:min_value(game.result(state,a)))
    #玩家将自己当作 Max，构建 Minimax 决策树，选择最大 Minimax 行棋
```

再提供一个玩家，通过界面询问人类的行棋，从而可以实现人机对战。

```
def query_player(game, state):
    print("current state:")
    game.display(state)
    print("available moves: {}".format(game.actions(state)))
    print("")
    move = None
    if game.actions(state):
        move_string = input('Your move? ')
        try:
            move = eval(move_string)
        except NameError:
            move = move_string
    else:
        print('no legal moves: passing turn to next player')
    return move
```

3.5.4　井字棋游戏的实现

井字棋游戏继承 Game，实现相应的方法如下：

```
class TicTacToe(Game):
    def __init__(self, h=3, v=3, k=3):
        self.h = h  #棋盘行数
        self.v = v  #棋盘列数
        self.k = k  #k 个棋子连线算胜利
        moves = [(x, y) for x in range(1, h + 1)
                 for y in range(1, v + 1)]
        self.initial = GameState(to_move='X',
                                 utility=0,
                                 board={},
                                 moves=moves)

    def actions(self, state):
        return state.moves
```

```
15
16      def result(self, state, move):
17          if move not in state.moves:
18              return state  #非法行棋时，将放弃行棋机会
19          board = state.board.copy()
20          board[move] = state.to_move
21          moves = list(state.moves)
22          moves.remove(move)
23          return GameState(
24              to_move=('O' if state.to_move == 'X' else 'X'),
25              utility=self.compute_utility(
26                  board, move, state.to_move),
27              board=board, moves=moves)
28
29      def terminal_test(self, state):
30          return state.utility != 0 or len(state.moves) == 0
31
32      def display(self, state):  #打印棋盘 state.board
33          board = state.board
34          for x in range(1, self.h + 1):
35              for y in range(1, self.v + 1):
36                  print(board.get((x, y), '.'), end=' ')
37              print()
38
39      def compute_utility(self, board, move, player):
40      #判断 0°、90°、±45° 四个方向是否 k 子连线，确定棋局效用值
41          if (self.k_in_row(board, move, player, (0, 1)) or
42                  self.k_in_row(board,move,player,(1, 0)) or
43                  self.k_in_row(board,move,player,(1,-1)) or
44                  self.k_in_row(board, move, player, (1, 1))):
45              return +1 if player==self.to_move(self.initial) \
46                     else -1
47          else:
48              return 0
49
50      def k_in_row(self, board, move, player, delta_x_y):
51      #从当前行棋位置开始，在方向 delta_x_y 上判断是否 k 子连线
52          (delta_x, delta_y) = delta_x_y
53          x, y = move
```

```
        n = 0
        while board.get((x, y)) == player:
            n += 1
            x, y = x + delta_x, y + delta_y
        x, y = move
        while board.get((x, y)) == player:
            n += 1
            x, y = x - delta_x, y - delta_y
        n -= 1
        return n >= self.k
```

3.6　实验报告要求

实验报告须包含实验任务、实验平台、实验原理、实验步骤、实验数据记录、实验结果分析和实验结论等部分，特别是以下重点内容。

（1）实现游戏模型和 Minimax 策略。

（2）完成人机对战系统测试。

（3）试验 Minimax 策略在更大规模问题上的应用，分析其局限性。

3.7　考核要求与方法

实验总分 100 分，通过实验报告进行考核，标准有如下 3 点。

（1）报告的规范性 10 分。报告中的术语、格式、图表、数据、公式、标注及参考文献是否符合规范要求。

（2）报告的严谨性 40 分。结构是否严谨，论述的层次是否清晰，逻辑是否合理，语言是否准确。

（3）实验的充分性 50 分。实验是否包含“实验报告要求”部分的 3 个重点内容，数据是否合理，是否有创新性成果或独立见解。

3.8　案例特色或创新

本实验的特色在于：培养学生应用对抗搜索方法解决博弈问题的能力，提高博弈系统的设计和实现能力，加强学生对人工智能算法时间复杂性的认识，同时提高学生针对复杂工程问题的建模能力和研究分析能力。