课程作业 1

尧祥临 202518023406038 前沿交叉科学学院

1 任务介绍

在老师给的利用 MCTS 实现的黑白棋代码框架基础上，需要完成以下两个任务：

- ♣ 在蒙特卡洛树搜索中，补充实现 UCB 算法以选择最佳节点进行扩展和模拟。
- ♣ 原代码使用 Roxanne 策略进行模拟，尝试替换为随机模拟策略，并比较两种策略的效果差异。

2 任务一：补充实现 UCB 算法

2.1 UCB 算法介绍

任务是要在蒙特卡洛树搜索的节点选择中，如果节点没有未被访问过，则直接选择该节点；但是如果节点有未被访问过的子节点，则需要使用 UCB 算法来选择最佳子节点进行扩展和模拟。UCB 算法全称为上置信界算法（Upper Confidence Bound），其计算的公式如下：

$$UCB(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$

其中，$UCB(n)$ 表示节点 n 的 UCB 值，$U(n)$ 表示节点 n 获胜的次数，$N(n)$ 表示节点 n 被选中的次数，$\text{Parent}(n)$ 表示节点 n 的父节点。这里加号前面代表该节点当前的平均奖励，加号后面则给出了一个置信度上界。C 是一个平衡系数，通常取值为 $\sqrt{2} \approx 1.414$，但是这里为了去进一步探索平衡系数的取值对于模型效果的影响，我们将其设置为可调节的参数。该算法需要计算出各个节点的 UCB 值，然后选择值最高的节点进行扩展和模拟。这一算法的核心思想就是在选择子节点的时候，自动权衡探索（exploration）和利用（exploitation），从而提高搜索效率和效果。

2.2 代码实现

```
for k in node.child.keys():
    if node.child[k].n == 0: # 如果子节点未被访问过，则直接选择该节点
        best_move = k
        break
    else:
        '''作业任务1:补充实现UCB算法'''
        UCB = (node.child[k].w / node.child[k].n) + self.const_ucb * sqrt(log(node.n) / node
            .child[k].n) # Exploitation + Exploration
        if UCB > best_score: # 根据UCB算法，需要选择UCB值最大的节点
            best_score = UCB
            best_move = k
```

这里我设置了`self.const_ucb`为可调节的参数，来表示 UCB 算法中的平衡系数。

3 任务二：替换模拟策略

3.1 Roxanne 策略与随机策略

在原代码中，模拟搜索过程中 AI 使用的是 Roxanne 策略，这是一种启发式的策略，提前提供了一个较为合理的落子顺序。我们将其替换为随机策略，即在每一步模拟中随机选择一个合法的落子位置，为此需要另外构造一个随机模拟策略类`RandomPlayer`，并在`AIPlayer`类中将模拟策略从`RoxannePlayer`替换为`RandomPlayer`。

3.2 代码实现

```python
class RandomPlayer(object):
    ''' 构造随机落子策略类，与Roxanne策略类相对应，结构相似 '''
    def __init__(self, color):
        """
        随机落子策略初始化
        :param color: 执棋方
        """
        self.color = color

    def random_select(self, board):
        """
        采用随机策略选择落子策略
        :return: 落子策略
        """
        action_list = list(board.get_legal_actions(self.color))
        if len(action_list) == 0:
            return None
        else:
            return random.choice(action_list) # 从合法落子位置中随机选择一个位置，从而实现随机落子策略

    def get_move(self, board):
        """
        采用随机策略进行搜索
        :return: 落子
        """
        if self.color == 'X':
            player_name = '黑棋'
        else:
            player_name = '白棋'
        action = self.random_select(board)
        return action

```

```
class AIPlayer(object):
    ''' 蒙特卡罗树搜索智能算法 '''
    def __init__(self, color, time_limit = 2, const_ucb = 1.414):
...
        self.sim_black = RandomPlayer('X')
        self.sim_white = RandomPlayer('O')
```

可以看到，新构造的RandomPlayer类与原有的RoxannePlayer类结构相似，但是该类中落子策略函数变成random_select，这里返回的是合法落子位置中随机选择的一个位置，从而实现随机落子策略。

4 拓展与分析

虽然已经补充了 UCB 算法并用随机策略替换了 Roxanne 策略，但是为了更好的理解 UCB 算法中的平衡系数 C 对于模型效果的影响以及不同模拟策略的效果差异，这里我对AIPlayer类和最终的运行部分的代码做了部分调整和拓展。

```
class AIPlayer(object):
    ''' 蒙特卡罗树搜索智能算法 '''
    def __init__(self, color, time_limit = 30, const_ucb = 1.414, Player = RandomPlayer):
        """
        蒙特卡洛树搜索策略初始化
        :param color: 执棋方
        :param time_limit: 蒙特卡洛树搜索每步的搜索时间步长
        :param tick:记录开始搜索的时间
        :param sim_black, sim_white: 模拟黑白棋双方落子策略
        :param const_ucb: UCB算法中的平衡系数，默认设置为1.414
        :param player: 采用的落子策略，默认为Random策略，可以调整为随机策略
        """
        self.time_limit = time_limit
        self.tick = 0
        self.sim_black = Player('X') # 默认采用Random策略进行模拟搜索
        self.sim_white = Player('O')
        self.color = color
        self.const_ucb = const_ucb # 默认设置为1.414
...
# 人类玩家黑棋初始化
#black_player = HumanPlayer("X")
# 也可以选择一个AI玩家作为黑棋
black_player = AIPlayer("X", const_ucb = 2, Player = RoxannePlayer,)
# AI 玩家 白棋初始化
white_player = AIPlayer("O", const_ucb = 1.414, Player = RoxannePlayer)
# 游戏初始化，第一个玩家是黑棋，第二个玩家是白棋
game = Game(black_player, white_player)
# 开始下棋
```

```
game.run()
```

这里在AIPlayer类的初始化函数中增加了一个可调节的参数Player，用来表示采用的模拟策略类，可以选择RoxannePlayer或者RandomPlayer。同时在最终的运行部分代码中，可以通过调整const_ucb和平衡系数 C 的取值，来观察不同平衡系数和模拟策略对于模型效果的影响。为了进行简单的比较分析，这里分为两组：一组采用 Roxanne 策略，另一组采用随机策略，每组分别设置 $C = 1, 1.414, 2$，然后让 AI 进行循环对战，记录胜负情况，两两之间只进行两局对战，仅交换先后手位置。经过测试，结果如表 1所示：

表 1: 黑白棋对局结果（列为白棋配置，行为黑棋配置）

	Rox, 1	Rox, 1.414	Rox, 2	随机, 1	随机, 1.414	随机, 2
Rox, 1	白胜	白胜	黑胜	黑胜	黑胜	白胜
Rox, 1.414	黑胜	黑胜	黑胜	黑胜	白胜	黑胜
Rox, 2	黑胜	白胜	黑胜	黑胜	白胜	黑胜
随机, 1	黑胜	黑胜	白胜	黑胜	黑胜	黑胜
随机, 1.414	黑胜	白胜	白胜	黑胜	黑胜	白胜
随机, 2	黑胜	白胜	白胜	黑胜	白胜	黑胜

根据对局结果可以看出黑白棋的先手优势显著，黑棋整体胜率明显高于白棋。总体看来，Roxanne 策略的效果优于随机策略，这和其启发式的设计有关，而随机策略在 C 较大时表现有所提升，但仍然不及 Roxanne 策略。而平衡参数 C 的取值在这里简单的测试中并未表现出明显的差异，Roxanne 策略在黑白棋中表现更优，平衡参数 C 的影响需要进一步深入研究。