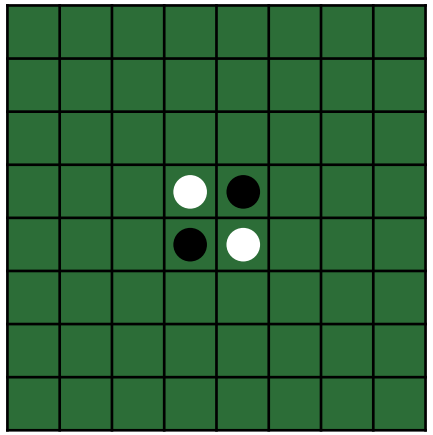
黑　　白　　棋

实验二

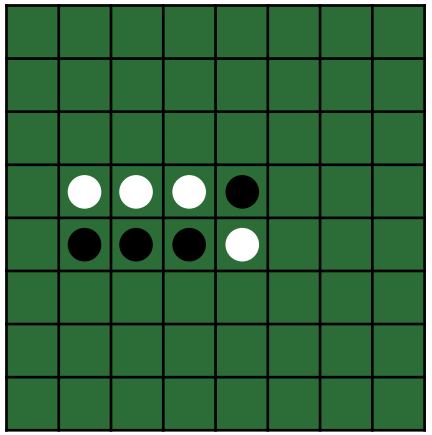
实验时间：2024年4月1日　00:00
截止时间：2024年4月15日 23:55

教师：王亚星　　助教：张鑫、胡泰航

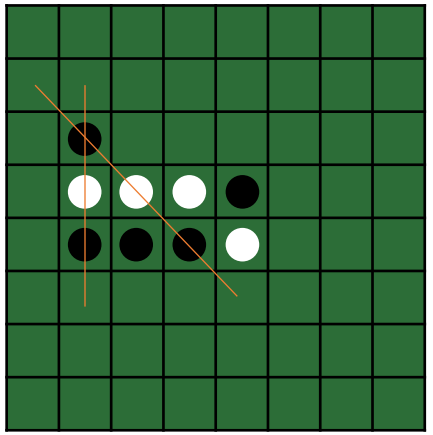
游戏背景与规则



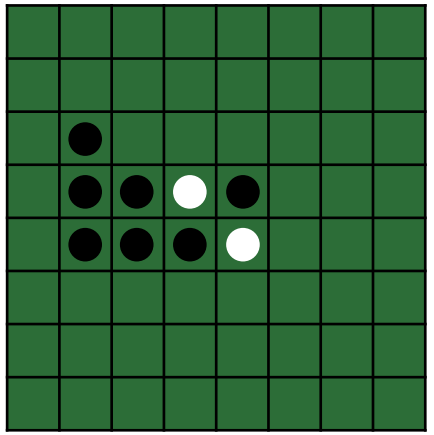
8×8棋盘
黑棋先手



假设案例
(实际不可能)



黑棋落子

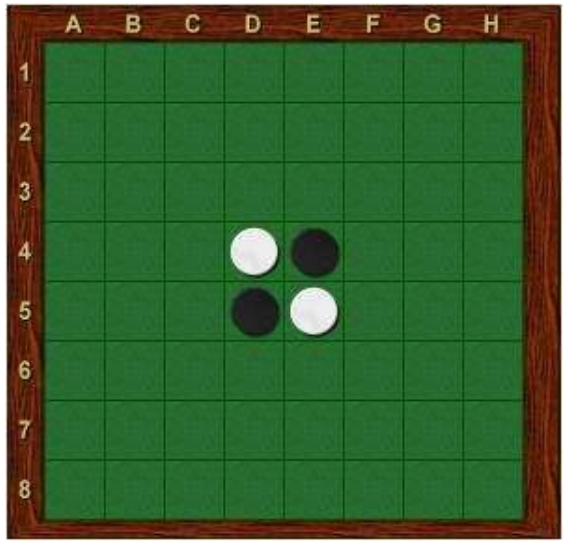


白棋**翻转**

基本规则：黑棋先手，**落子需有效（至少令对方棋子翻转）**，若没有位置落子，则由对方执棋

输赢判断
正常：双方都没有落子的地方，根据棋盘上落子数量决定输赢
异常：单方**退出**；**超时**不出未落子；**异常落子**（程序实现特有）

程序可视化展示



	A	B	C	D	E	F	G	H
1	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
2	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
3	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
4	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
5	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
6	(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
7	(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
8	(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

```python
# 导入棋盘文件
from board import Board

# 初始化棋盘
board = Board()

# 打印初始化棋盘
board.display()
```

```
  A B C D E F G H
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . O X . . .
5 . . . X O . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .
统计棋局: 棋子总数 / 每一步耗时 / 总时间
黑    棋: 2 / 0 / 0
白    棋: 2 / 0 / 0
```

O：白棋

X：黑棋

.：未落子

工具接口（utils）

`ReversiBoard`

棋盘类

`_move()`

根据位置落黑/白子并翻转

`get_legal_actions()`

获取某种颜色合法落子序列

（两种实现思路）

`Game`

游戏类

`run()`

进行游戏，并判断输赢

（建议看看异常判输的内容）

`HumanPlayer` `color`

人类**玩家类**

`get_move()`

根据当前棋盘做出选择

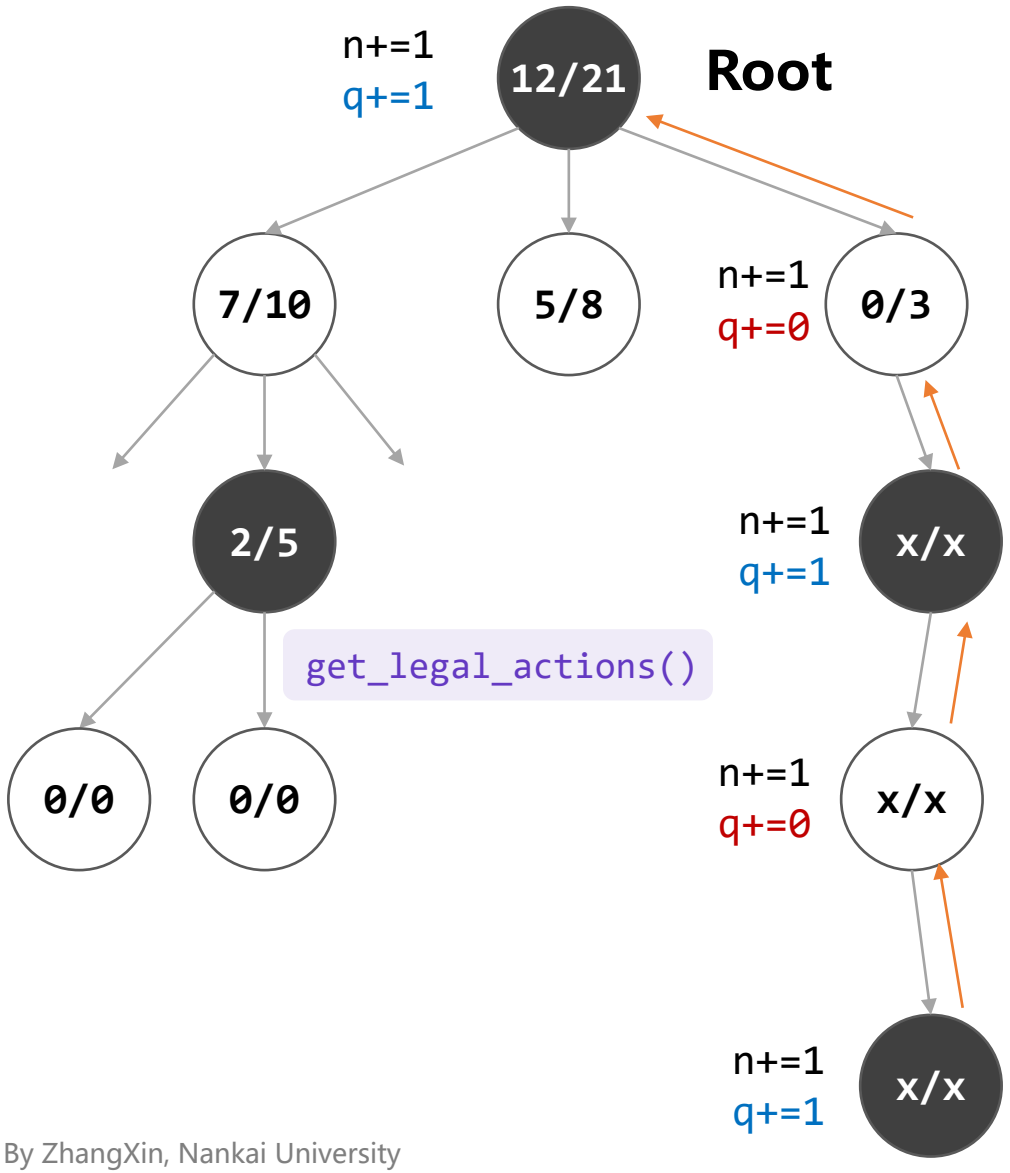
`RandomPlayer` `color`

随机**玩家类**

`AIPlayer` `color`

任务：利用蒙特卡洛树搜索

UCTSearch (MCTS+UCB)



1. SelectPolicy(UCB)

$$UCB = \frac{w_j}{n_j} + c\sqrt{\frac{2\ln n}{n_j}}$$

exploit explore

7/10

$$\frac{7}{10} + \sqrt{\frac{log(21)}{10}} = 1.252$$

TreeNode: parent children color reward visit_num

2. Expand

3. SimulatePolicy

`Game` `run()`

4. BackPropagate

Assume black player is lose.

5. Return

Choice form root.

By ZhangXin, Nankai University

注意事项

1. 深拷贝问题（deepcopy）
2. 颜色交换问题

Thanks

CONTENT

目　录

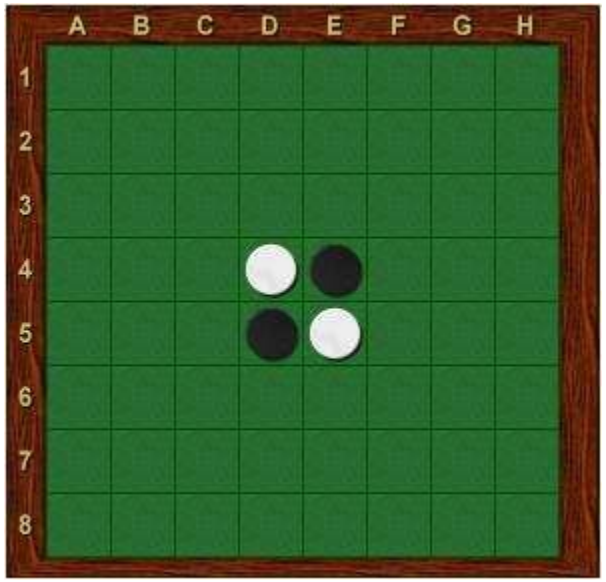
问题描述

Problem description

问题描述
Problem description

黑白棋(reversi),也叫苹果棋，翻转棋，是一个经典的策略性游戏。一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，变为己方棋子，故又称“翻转棋”。棋子双面为红、绿色的成为“苹果棋”。它使用8*8的棋盘，由两人执黑子和白子轮流下棋，最后子多方为胜。

游戏规则：
棋局**开始时**黑棋位于E4和D5，白棋位于D4和E5，如下图所示



游戏规则

Game Rules

(1) 黑方先行，双方交替下棋。
(2) 一步合法的棋步包括:
- 在一个空格处落下一个棋子，并且翻转对手一个或多个棋子;
- 新落下的棋子必须落在可夹住对方棋子的位置上，对方被夹住的所有棋子都要翻转过来，可以是横着夹，竖着夹，或是斜着夹。夹住的位置上必须全部是对手的棋子，不能有空格;
- 一步棋可以在数个（横向，纵向，对角线）方向上翻棋，任何被夹住的棋子都必须被翻转过来，棋手无权选择不去翻某个棋子。

(3) 如果一方没有合法棋步，也就是说不管他下到哪里，都不能至少翻转对手的一个棋子，那他这一轮只能弃权，而由他的对手继续落子直到他有合法棋步可下。
(4) 如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。
(5) 棋局持续下去，直到棋盘填满或者双方都无合法棋步可下。
(6) 如果某一方落子时间超过 1 分钟 或者 连续落子 3 次不合法，则判该方失败。

https://www.bilibili.com/video/BV1Dy4y1h7dy/

2

实验基础

Experimental Basis

实验基础

Experimental Basis

棋盘类

```python
# 导入棋盘文件
from board import Board

# 初始化棋盘
board = Board()

# 打印初始化棋盘
board.display()
```

```
  A B C D E F G H
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . O X . . .
5 . . . X O . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .
统计棋局: 棋子总数 / 每一步耗时 / 总时间
黑    棋: 2 / 0 / 0
白    棋: 2 / 0 / 0
```

O: 白棋
X: 黑棋
. : 未落子

棋盘的坐标转换

棋盘坐标 E4, 转化为坐标形式就是 (3, 4), 坐标数值大小是从 0 开始，到 7 结束。

Board 类中，提供以上两种坐标的转化方法：

- `board_num(action)`: 棋盘坐标转化为数字坐标。
  - action: 棋盘坐标，e.g. 'G6'
  - 返回值: 数字坐标，e.g. (5, 6)
- `num_board(action)`: 数字坐标转化为棋盘坐标。
  - action: 数字坐标，e.g. (2, 7)
  - 返回值: 棋盘坐标，e.g. 'H3'

	A	B	C	D	E	F	G	H
1	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
2	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
3	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
4	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
5	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
6	(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
7	(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
8	(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

- 重要方法
- 1. 获取合法落子坐标，可以调用board类封装的get_legal_actions获取当前局面下的可行路径，调用的参数为棋子颜色，即’X’或者’O’，返回值值为合法的落子坐标，用list()方法可以获取所有的合法坐标。

```
# 棋盘初始化后，黑方可以落子的位置
print(list(board.get_legal_actions('X')))
```

```
['D3', 'C4', 'F5', 'E6']
```

实验基础

Experimental Basis

- `_move(action, color)`： 根据 color 落子坐标 action 获取翻转棋子的坐标。
  - action: 落子的坐标，e.g. 'C4'
  - color: 下棋方，'X' - 黑棋，'O' - 白棋
  - 返回值: 反转棋子棋盘坐标列表

重要方法

2. 落子 & 翻转：使用board类封装的_move函数进行模拟对局，传入参数为落子坐标以及棋子颜色。board封装的move方法会自动翻转棋子，并返回反转棋子的坐标。

```
# 打印初始化后的棋盘
board.display()

# 假设现在黑棋下棋，可以落子的位置有：['D3', 'C4', 'F5', 'E6']，
# 黑棋落子 D3 ，则白棋被翻转的棋子是 D4。

# 表示黑棋
color = 'X'

# 落子坐标
action = 'D3'

# 打印白方被翻转的棋子位置
print(board._move(action,color))

# 打印棋盘
board.display()
```

```
  A B C D E F G H
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . O X . . .
5 . . . X O . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .
统计棋局: 棋子总数 / 每一步耗时 / 总时间
黑   棋: 2 / 0 / 0
白   棋: 2 / 0 / 0

['D4']
  A B C D E F G H
1 . . . . . . . .
2 . . . . . . . .
3 . . . X . . . .
4 . . . X X . . .
5 . . . X O . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .
统计棋局: 棋子总数 / 每一步耗时 / 总时间
黑   棋: 4 / 0 / 0
白   棋: 1 / 0 / 0
```

实验基础
Experimental Basis

- **对弈方法**
- 1. 人机对局：通过get_legel_actions获取机器玩家的合法位置，随机选取其中一个位置进行落子。
- 2. 人类对弈：人类玩家自行使用不同棋子颜色，通过调用get_move函数传递不同颜色棋子的行动轨迹。
- 3. AI对弈：自行实现类似于AlphaGo一样的Agent与玩家进行游戏。

- 博弈实质：**玩家采取不同的策略对get_legel_actions的返回合法位置进行选择。**

实验基础
Experimental Basis

- 对弈方法
- 1. 人机对局：通过get_legel_actions获取机器玩家的合法位置，随机选取其中一个位置进行落子。get_move进行行动，random_choice定义策略。

```
def __init__(self, color):
    """
    玩家初始化
    :param color: 下棋方，'X' - 黑棋，'O' - 白棋
    """
    self.color = color
def random_choice(self, board):
    """
    从合法落子位置中随机选一个落子位置
    :param board: 棋盘
    :return: 随机合法落子位置，e.g. 'A1'
    """
    # 用 list() 方法获取所有合法落子位置坐标列表
    action_list = list(board.get_legal_actions(self.color))

    # 如果 action_list 为空，则返回 None,否则从中选取一个随机元素，即合法的落子坐标
    if len(action_list) == 0:
        return None
    else:
        return random.choice(action_list)
```

```
def get_move(self, board):
    """
    根据当前棋盘状态获取最佳落子位置
    :param board: 棋盘
    :return: action 最佳落子位置，e.g. 'A1'
    """
    if self.color == 'X':
        player_name = '黑棋'
    else:
        player_name = '白棋'
    print("请等一会，对方 {}-{} 正在思考中...".format(player_name, self.color))
    action = self.random_choice(board)
    return action
```

实验基础
Experimental Basis

- **对弈方法**
- 2. 人类对弈：人类玩家自行使用不同棋子颜色，策略由人类玩家产生，无需定义策略成员函数，只需在get_move中接收用户输入数据并检测是否合理即可。

```python
def get_move(self, board):
    """
    根据当前棋盘输入人类合法落子位置
    :param board: 棋盘
    :return: 人类下棋落子位置
    """
    # 如果 self.color 是黑棋 "X",则 player 是 "黑棋"，否则是 "白棋"
    if self.color == "X":
        player = "黑棋"
    else:
        player = "白棋"

    # 人类玩家输入落子位置，如果输入 'Q'，则返回 'Q'并结束比赛。
    # 如果人类玩家输入棋盘位置，e.g. 'A1'，
    # 首先判断输入是否正确，然后再判断是否符合黑白棋规则的落子位置
```

```python
while True:
    action = input(
            "请'{}-{}'方输入一个合法的坐标(e.g. 'D3'，若不想进行，请务必输入'Q'结束游戏。): ".format(player,
                                                                    self.color))

    # 如果人类玩家输入 Q 则表示想结束比赛
    if action == "Q" or action == 'q':
        return "Q"
    else:
        row, col = action[1].upper(), action[0].upper()

        # 检查人类输入是否正确
        if row in '12345678' and col in 'ABCDEFGH':
            # 检查人类输入是否为符合规则的可落子位置
            if action in board.get_legal_actions(self.color):
                return action
        else:
            print("你的输入不合法，请重新输入!")
```

实验基础
Experimental Basis

- Game 类：实现棋盘对弈，成员变量如下：

- 属性：
  - `self.board`：棋盘
  - `self.current_player`：定义当前的下棋一方，考虑游戏还未开始我们定义为 None
  - `self.black_player`：定义黑棋玩家 black_player
  - `self.white_player`：定义白棋玩家 white_player

- 我们只需要定义玩家类型(AI, 人机, 人类等)，定义玩家角色，运行对局即可（e.g.下图为定义一场对局代码示例）

```python
# 导入黑白棋文件
from game import Game

# 人类玩家黑棋初始化
black_player = HumanPlayer("X")

# 随机玩家白棋初始化
white_player = RandomPlayer("O")

# 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()
```

```python
# 导入黑白棋文件
from game import Game

# 随机玩家黑棋初始化
black_player = RandomPlayer("X")

# 随机玩家白棋初始化
white_player = RandomPlayer("O")

# 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()
```

实验基础
Experimental Basis

- 对弈方法
- 3. AI对弈：自行实现类似于AlphaGo一样的Agent与玩家进行游戏。
- 本次实验最终编码要求即通过使用蒙特卡洛搜索算法实现AI player。实现类似于随机玩家random_choice一样的选择策略，将get_move的action更改为通过MCTS得到的行动坐标。

```
def get_move(self, board):
    """
    根据当前棋盘状态获取最佳落子位置
    :param board: 棋盘
    :return: action 最佳落子位置, e.g. 'A1'
    """
    if self.color == 'X':
        player_name = '黑棋'
    else:
        player_name = '白棋'
    print("请等一会，对方 {}-{} 正在思考中...".format(player_name, self.color))

    # -----------------请实现你的算法代码--------------------------------------

    action = None
    # ------------------------------------------------------------------------

    return action
```

3

蒙特卡洛树搜索原理

MCTS’s principle

蒙特卡洛树搜索原理

MCTS's principle

- **Introduction:**
- **MCTs是一种启发式的搜索算法，广泛应用于各种棋类游戏中，著名的AlphaGo的原理中就有MCTs的身影，其主要思想与强化学习RL很像，是考虑在当前环境之下，根据某一策略采取某一个行动，获得最大化的收益。**
- **类似于人类专家在下棋的时候一般一眼算多步，就是不仅仅考虑到当前的局势，还会在大脑中模拟下棋后可能的对局。MCTS主要进行若干次模拟，将结果保存到蒙特卡洛树中，根据模拟结果选择最佳的动作。**

蒙特卡洛树搜索原理

MCTS's principle

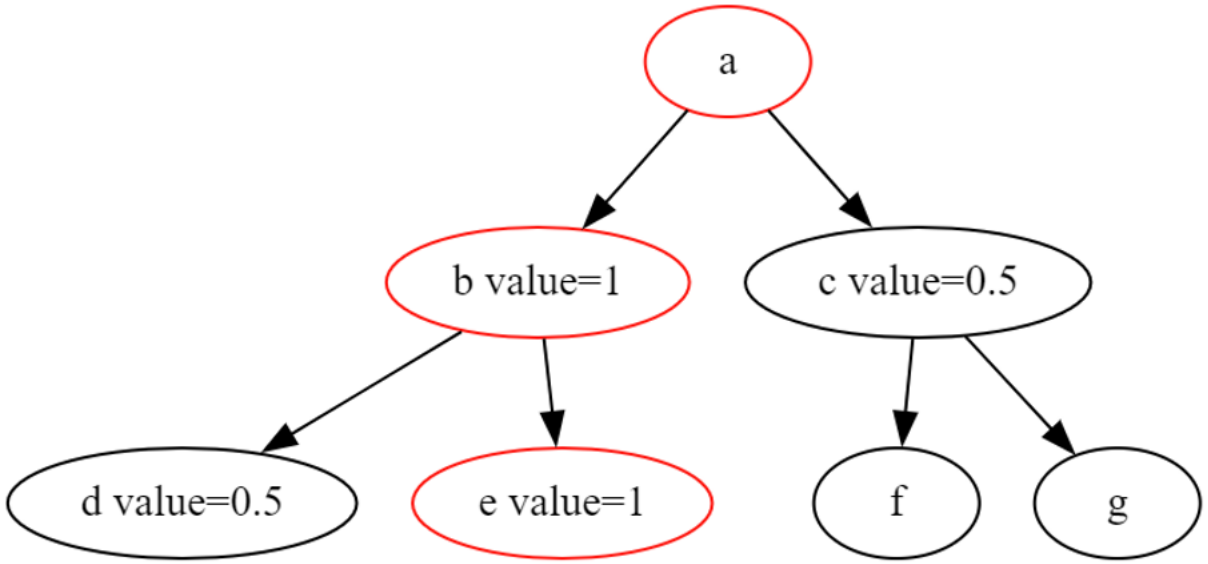
- **Process: MCTS的一轮迭代共有四个阶段：选择，扩展，模拟，反向传播。**

- **那么问题就是：**
- **1. 当前的环境如何定义？可以定义为当前棋局**
- **2. 有哪些行动是合法的？通过get_legal_action获得。**
- **3. 如何选择最佳行动？通过收益选择。**
- **4. 如何计算收益？选择胜率尽可能大的行动。**
- **5. 如何计算选择举动的胜率？通过模拟整个棋局走向。**

蒙特卡洛树搜索原理

MCTS's principle

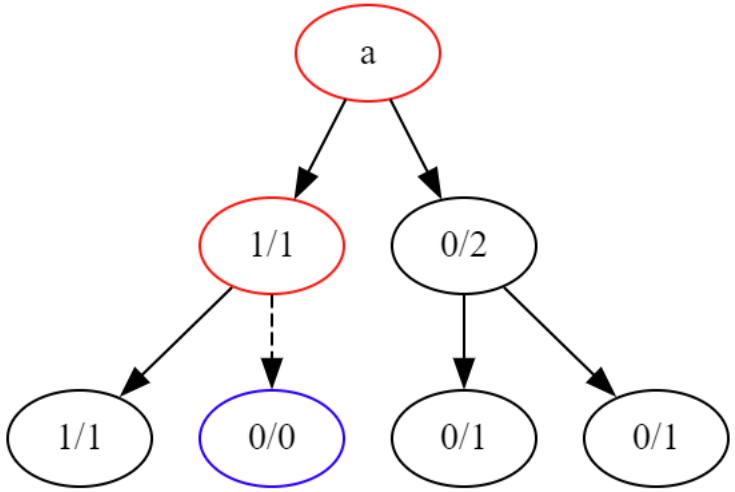
- **Step 1：选择**
- **MCTS的主要载体是一棵树，根节点代表当前棋局状态，子节点代表在当前棋局下，某方所有可能的行动方案。叶节点是尚未启动模拟的任何潜在子节点。**
- **选择操作的目的是从根节点开始，向叶节点进行遍历，在某一节点做选择时，每次选择价值较高的子节点，直到选择到叶节点或者未拓展完全的节点为止。**

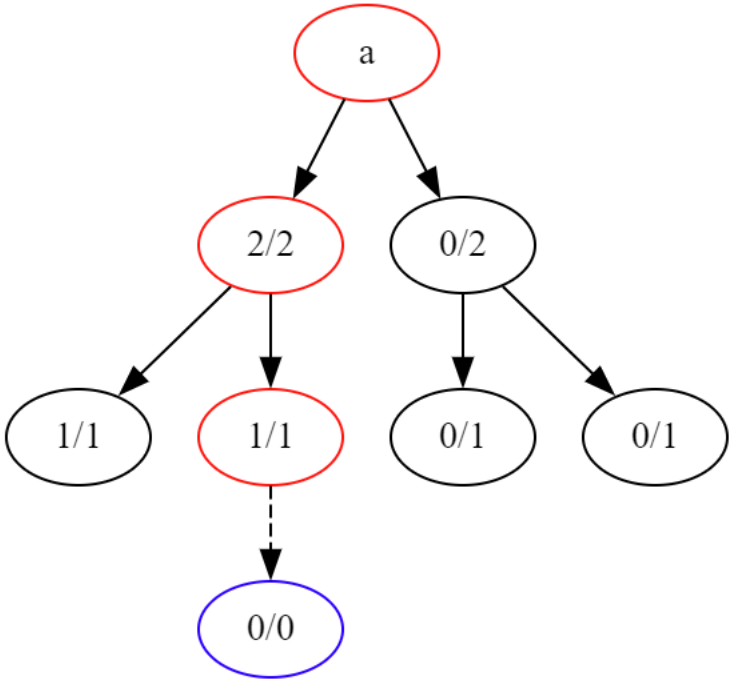


蒙特卡洛树搜索原理

MCTS's principle

- **Step 2：扩展**
- 扩展操作的目的是对已经过模拟的节点进一步的产生全新的子节点的过程，在根节点对应的棋局的情况下，产生从根节点到当前选择的叶节点的树链上进行所有对应行动后的全新棋局的可能路径。对于搜索树而言“逐层扩宽”。
- 一个节点是未扩展完全的节点当且仅当对于当前情况下，还有潜在的行动路径。





蒙特卡洛树搜索原理

MCTS's principle

- **Step 3：模拟**
- 在经过扩展后，以当前扩展后的新局面为基础，模拟完成正常对局，得到一个结果，即在当前情况下，经过模拟后的胜负关系。并更新当前节点的“价值”。

- 考虑刚刚选择的依据“价值”，如何定义选择依据？使用UCB做决策：

$$UCB = \bar{X}_j + c\sqrt{\frac{2\ln n}{n_j}}$$

- 其中第一项为在前 j 轮下的收益，可以定义为当前节点在前 j 轮的胜率，第二项表示置信区间，n表示当前节点的父节点被访问的次数，n_j表示为当前节点选择的次数，c为参数可调整。故UCB在这里可以改写为(w_j代表获胜的次数)：

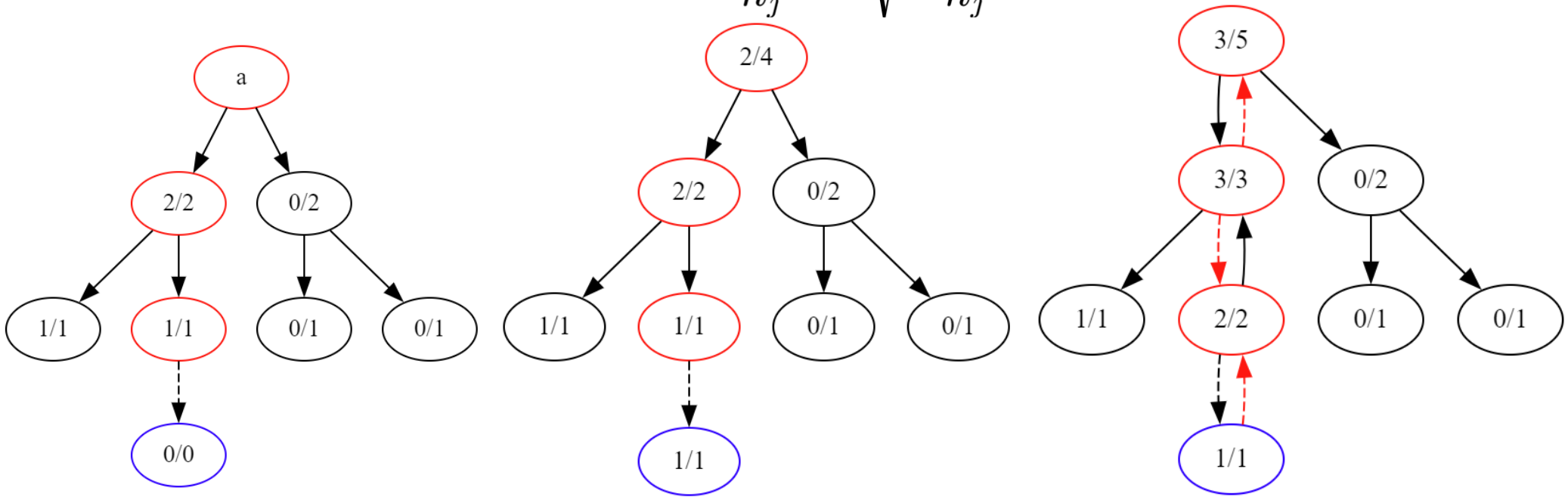
$$UCB = \frac{w_j}{n_j} + c\sqrt{\frac{2\ln n}{n_j}}$$

蒙特卡洛树搜索原理

MCTS's principle

- **Step 4：反向传播**
- **对于搜索树上任意一个节点，其价值代表选择该节点可能的收益，所以我们在模拟结束之后，需要将模拟结果从当前节点沿树链向上更新至根节点。依据UCB公式，每个节点维护两个变量，代表访问次数n_j，以及获胜次数w_j。**

$$UCB = \frac{w_j}{n_j} + c\sqrt{\frac{2\ln n}{n_j}}$$



蒙特卡洛树搜索原理

MCTS's principle

- **Hint：**
- **维护搜索树的树形结构，维护MCTS节点类，添加成员函数，例如扩展，选择等。实现MCTS整体流程，构建搜索博弈树。**
- **如何判断是否完全展开？调用get_legal_actions获得合法的行动，有合法行动即未完全展开。**
- **在当前节点的所有孩子中选择UCB最大的节点？枚举遍历所有的孩子节点即可。**
- **游戏分轮进行，故树形结构相邻两层代表不同玩家的行动。**
- **模拟的过程？潜在方案：调用random_players模拟对弈过程。或其他方法均可。**
- **AI Player的策略？计算根节点所有孩子的UCB值，选取最大的UCB。**
- **可能需要自行实现大部分的代码，面向对象的思想会帮助你很好的设计代码。**

结果提交

Submission

结果输出

Result output

- 经过测试 AI 玩家实现人类玩家与 AI 玩家对弈之后，在左侧 `提交作业` 的标签中，把**整个 AIPlayer 转化为 main.py** 文件进行`系统测试`。
- 你可以选择初级、中级或者高级对手进行对弈，对弈时请勾选 main.py 文件。
- 能通过测试就可以提交作业。

谢谢