前言：

本文是根据的文章Introduction to Monte Carlo Tree Search by Jeff Bradberry所写。
Jeff Bradberry还提供了一整套的例子，用python写的。
board game server
board game client
Tic Tac Toe board
AI implementation of Tic Tac Toe

阿袁工作的第一天 - 蒙特卡罗树搜索算法 - 游戏的通用接口board 和 player

阿袁看到阿静最近在学习蒙特卡罗树搜索算法。急忙凑上去问：“蒙特卡罗树搜索算法是干什么用的？”
"蒙特卡罗树搜索算法是一种方法（或者说框架），用于解决完美信息博弈。我现在学习一个蒙特卡罗树搜索算法的变种：UCT算法，用于提供一种通用的游戏对弈解决算法。"

> 注: perfect information games (完美信息)博弈，指的是没有任何信息被隐藏的游戏。

"通用的游戏对弈算法，是对任何游戏都有效，是吗？"
"简单的说，是这样的。重要的一点是，算法并**不用了解**游戏的**领域知识**。"
"领域知识？不是很好理解。难道连游戏规则也不知道，就可以赢吗？"
"游戏的领域知识。举个例子，国际象棋中每个棋子的子力，比如皇后的子力是10，车是5等等。这些就是领域知识。在通用的情况下，马的走法-这样的规则，也算是领域知识。"
"有点糊涂了！AI算法该如何下子呢？"
"用面向对象的逻辑来说，我们可以给游戏定义有一个通用接口(board)，具体的游戏只能实现这个接口，不能提供其它的信息。"
"对于程序猿来说，这就容易理解多了。我们可以先看看这个接口(board)，都应该定义什么样属性和方法。"
"首先，有一个num_players属性，返回游戏的玩家数。"
"嗯，让我想想，游戏开始的时候，需要一个方法start，启动一个游戏。"

"很好，这个方法需要返回一个state对象，用于记录游戏当前的状态。state对象的内容，外部是不可知的。使用board自己可以解释。"
"然后，需要显示棋盘的状态。这样，board就需要提供一个display方法，返回当前的状态或者是棋盘状态。"
"对。应该有个方法返回谁是该下子的玩家:current_player."
"当前玩家是一个AI玩家（也就是对弈算法的使用者），怎么知道如何下子呢？这里需要许多的领域知识吧？"
"一个技巧是让board根据历史的状态列表，返回当前允许的所有下法：legal_actions。"
"再加上一个is_legal(action)，来判断一个下法是否合适。"
"下来应该是根据现在的action，返回下一个游戏状态，next_state。"
"为了判断胜负，需要一个winner方法。"
"如果有了赢家，board需要返回一个winner_message信息。通知玩家谁胜了。"
"看起来不错！我们总结一下board接口的内容。"

```
class Board(object):
    '''
    Define general rules of a game.
    State: State is an object which is only be used inside the board class.
        Normally, a state include game board information (e.g. chessmen positions, action index, current action, current player,
etc.)
    Action: an object to describe a move.
    '''

    '''
    num_players: The player numbers of the board.
    '''
    num_players = 2

    def start(self):
        '''
        Start the game
        Return: the initial state
        '''
        return None

    def display(self, state, action, _unicode=True):
        '''
        Dispaly the board
        state: current state
```

```
        action: current action
        Return: display information
        '''
        return None

    def parse(self, action):
        '''
        Parse player input text into an action.
        If the input action is invalid, return None.
        The method is used by a human player to parse human input.
        action: player input action texxt.
        Return: action if input is a valid action, otherwise None.
        '''
        return None

    def next_state(self, state, action):
        '''
        Calculate the next state base on current state and action.
        state: the current state
        action: the current action
        Return: the next state
        '''
        return tuple(state)

    def is_legal(self, history, action):
        '''
        Check if an action is legal.
        The method is used by a human player to validate human input.
        history: an array of history states.
        Return: ture if the action is legal, otherwise return false.
        '''
        return (R, C) == (state[20], state[21])

    def legal_actions(self, history):
        '''
        Calculate legal action from history states.
        The method is mainly used by AI players.
        history: an array of history states.
        Return: an array of legal actions.
```

```
        '''
        return actions

    def current_player(self, state):
        '''
        Gets the current player.
        state: the current state.
        Return: the current player number.
        '''
        return None

    def winner(self, history):
        '''
        Gets the win player.
        history: an array of history states.
        Return: win player number. 0: no winner and no end, players numbers + 1: draw.
        '''
        return 0

    def winner_message(self, winner):
        '''
        Gets game result.
        winner: win player number
        Return: winner message, the game result.
        '''
        return ""
```

"另外，我们需要定义一个player接口，玩家主要是下子，所以需要一个get_action方法。"
"当一个玩家下完子后，需要通过一个update方法通知所有的玩家，状态要更新了。"

```
class Player(object):
    def update(self, state):
        '''
        Update current state into all states.
        state: the current state.
        '''
        self.states.append(state)

    def display(self, state, action):
```

```
        '''
        Display board.
        state: the current state.
        action: the current action.
        Return: display information.
        '''
        return self.board.display(state, action)

    def winner_message(self, msg):
        '''
        Display winner message.
        msg: winner infomation
        Return: winner message
        '''
        return self.board.winner_message(msg)

    def get_action(self):
        '''
        Get player next action.
        Return: the next action.
        '''
        return action
```

> 注：方法: diplay and winner_message用于向游戏的客户端提供board的信息。这样隔离了客户端和board。

阿袁工作的第2天 - 蒙特卡罗树搜索算法 - MonteCarlo Player

阿袁和阿静继续关于蒙特卡罗树搜索算法的讨论。
阿静说道，“在编写一个人工智能游戏对弈的应用中，至少需要两个具体的player，一个是human player，一个是MonteCarlo player。”
"human player向人类玩家提供了一个交互界面。"
“对，MonteCarlo player是一个AI player，也是我们要讨论的重点，MonteCarlo player在实现get_action中，通过board，模拟后面可能下法；并根据模拟的结果，获得一个最优的下法。”
"我们先从一个简单的问题开始：一个游戏下法的组合可能是一个很大的数，我们如何控制这个模拟行为是满足一定时间上的限制的。"
“对于这个问题，解决方法有一些。这里，我们允许一个参数calculation_time来控制时间。每次模拟一条路径，模拟完后，检测一下是否到

时。”

“一条路径就是从游戏的当前状态到对局结束的所有步骤。如果这些步骤太长了呢？”

“尽管游戏的下法组合数会很大。但是一个游戏的正常步骤却不会很大哦。我们也可以通过另外一个参数max_actions来控制。”

“明白了。代码大概是这个样子。”

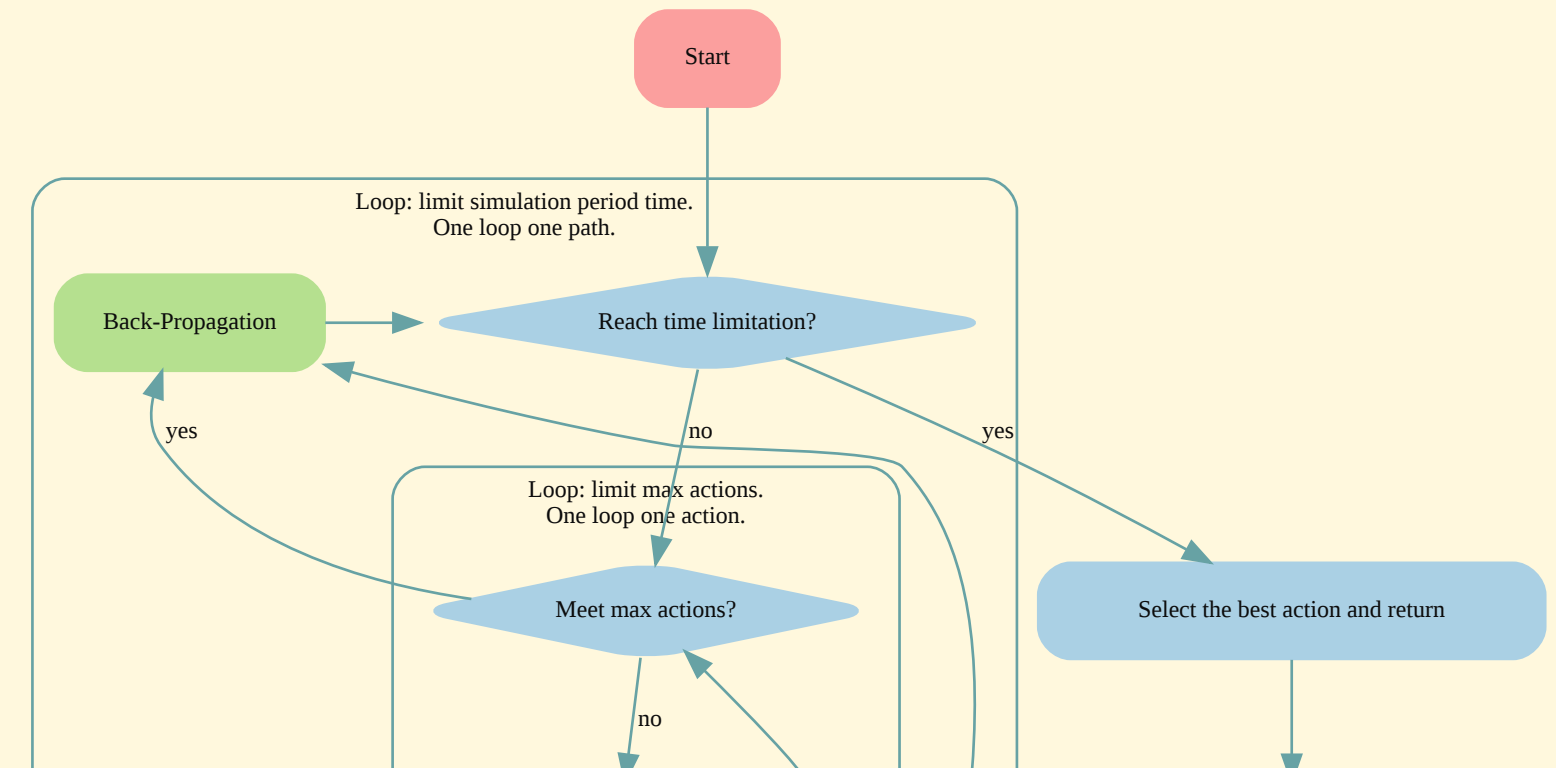
```
class MonteCarlo(object):

    def __init__(self, board, **kwargs):
        # ...

        self.calculation_time = float(kwargs.get('time', 30))
        self.max_actions = int(kwargs.get('max_actions', 1000))

        # ...

    def get_action(self):
        # ...

        # Control period of simulation
        moves = 0
        begin = time.time()
        while time.time() - begin < self.calculation_time:
            self.run_simulation()
            moves += 1

        # ...

    def run_simulation(self):
        # ...

        # Control number of simulation actions
        for t in range(1, self.max_actions + 1):
            # ...

        # ...
```

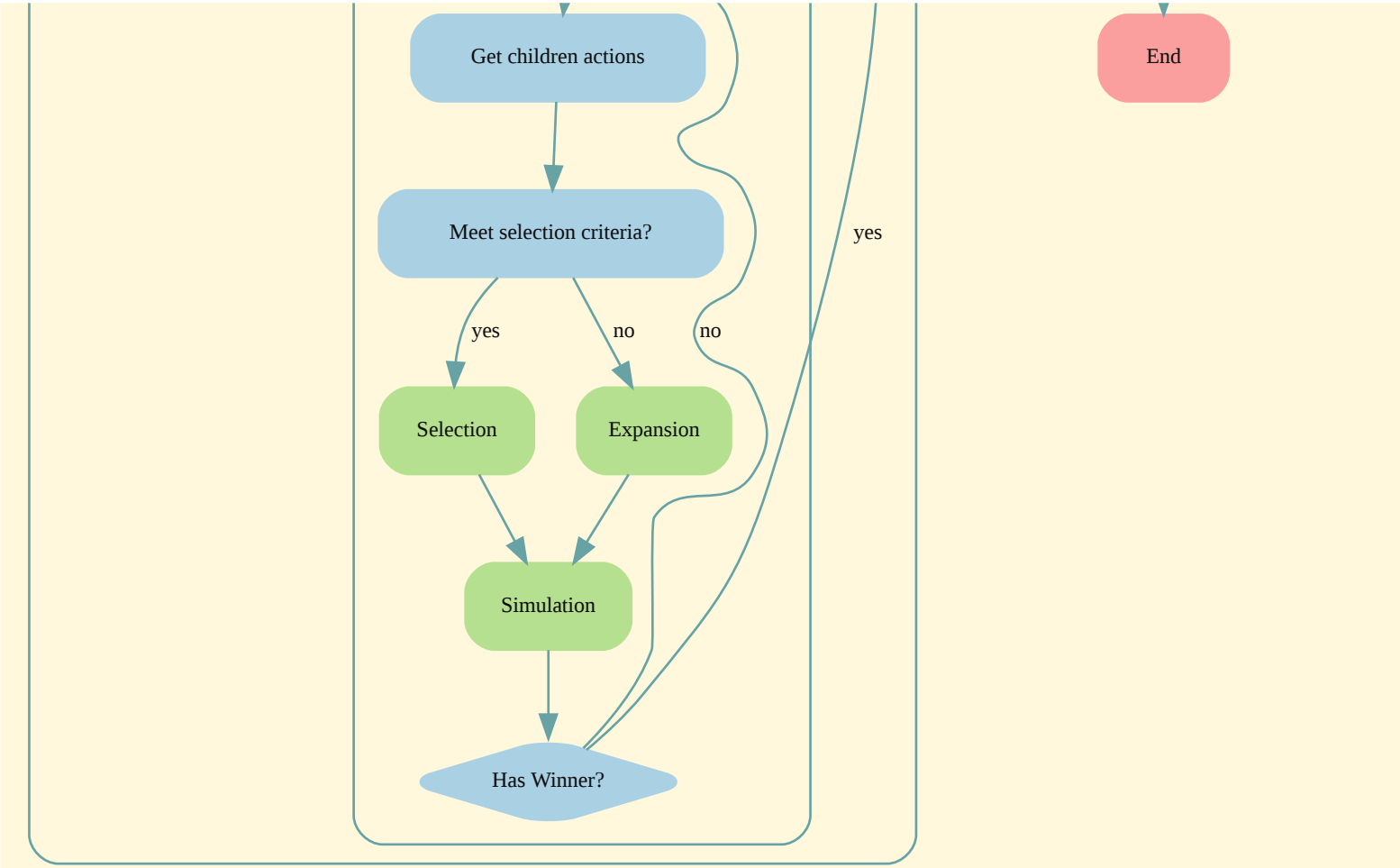
> 注：为了易于理解，我简单地重构了源代码，主要是rename了一些变量名。

"今天时间有些紧张，明天我们讨论蒙特卡罗树搜索的步骤"

阿袁工作的第3天 - 蒙特卡罗树搜索 - 蒙特卡罗树搜索的步骤

阿袁昨天晚上，也好好学习了蒙特卡罗树搜索。今天，他开始发言。
"蒙特卡罗树搜索是一个方法，应该是来自于蒙特卡罗方法。这个方法定义了几个步骤，用于找到最优的下法。"
“严格的说，蒙特卡罗树搜索并不是一个算法。”
“是的。所以蒙特卡罗树搜索有很多变种，我们现在学习的算法是蒙特卡罗树搜索算法的一个变种：**信任度上限树**(Upper Confidence bound applied to Trees(UCT))。这个我们明天研究。”
“好，今天主要了解**蒙特卡罗树搜索方法的步骤**”
"从文章上看一共有四个步骤。"
"是的。分别是选举(selection)，扩展(expansion)，模拟(simulation)，反向传播(Back-Propagation)。"
“我们看看这张图。绿色部分是蒙特卡罗树搜索的四个步骤。”





Monte Carlo Tree Search Steps

“**选举(selection)**是根据当前获得所有子步骤的统计结果，选择一个最优的子步骤。”
“**扩展(expansion)**在当前获得的统计结果不足以计算出下一个步骤时，随机选择一个子步骤。”
“**模拟(simulation)**模拟游戏，进入下一步。”
“**反向传播(Back-Propagation)**根据游戏结束的结果，计算对应路径上统计记录的值。”
“从上面这张图可以看出，选举的算法很重要，这个算法可以说是来评价每个步骤的价值的。”
“好了。今天，我们了解了蒙特卡罗树搜索的步骤。”
“明天，可以学习Upper Confidence bound applied to Trees(UCT) - 信任度上限树算法。”

阿袁工作的第4天 - 蒙特卡罗树搜索 - Upper Confidence bound applied to Trees(UCT) - 信任度上限树算法

一开始，阿静就开始讲到。

“信任度上限树算法UCT是根据统计学的信任区间公式，来计算一个步骤的价值。这个方法比较简单，只需要每个步骤的访问数和获胜数就可以了。”

“信任区间公式的是什么呢？”

阿静写下信任区间公式。

置信区间(confidence intervals)

$$\bar{x}_i \pm \sqrt{\frac{z \ln n}{n_i}} \tag{1}$$

$where:$

$\bar{x}_i$: the mean of choose i.

n_i : the number of plays of choose i.

n : the total number of plays.

z : 1.96 for 95% confidence level.

阿静进一步解释道。

“置信区间是一个统计上的计算值，如果z使用1.96，可以使置信区间的置信度达到95%。也就是说：有95%的信心，样本的平均值在置信区间内。”

“UCT算法使用了置信区间的**上限值**做为每个步骤的**价值**。”

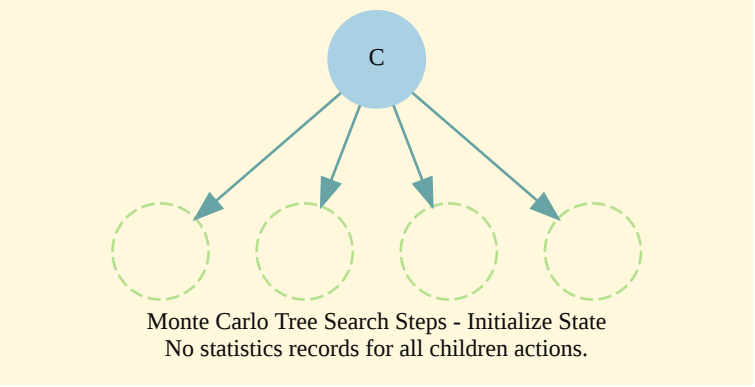
“使用**置信区间的上限值**带来的一个好处是：如果当前选择的最优子步骤在多次失败的模拟后，这个值会变小，从而导致另一个同级的子步骤可能会变得更优。”

“另外一个关键点是**选举的条件**，文章中的选举条件是当前所有子步骤都有了统计记录（也就是至少访问了一次，有了访问数。）。”

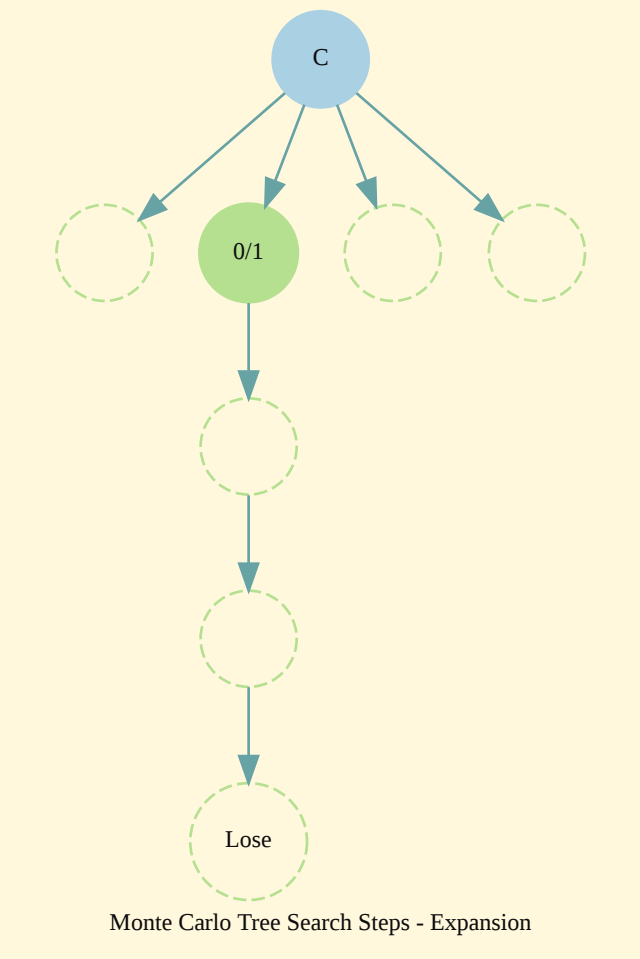
阿袁工作的第5天 - 蒙特卡罗树搜索 - 图形化模拟 Upper Confidence bound applied to Trees(UCT) - 信任度上限树算法

阿袁今天做了一天功课，画了一些图来说明UCT算法的过程。

- 首先，初始状态下，所有的子步骤都没有统计数据。

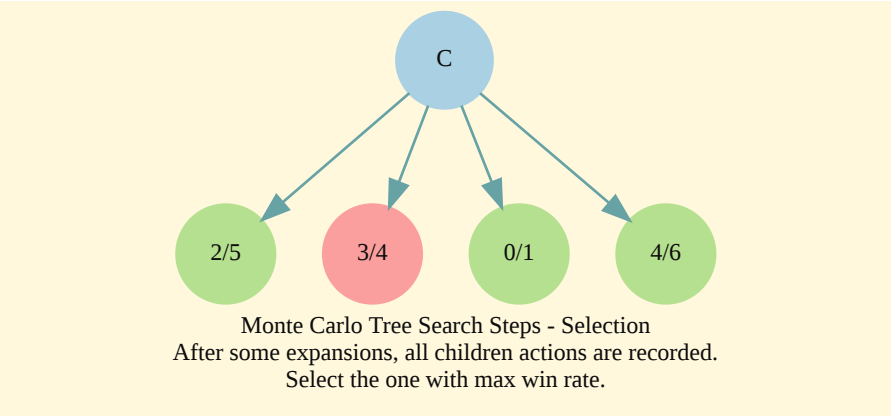


- 所以，先做**扩展(Expansion)**，随机选择一个子步骤，不停的**模拟(Simulation)**，直到游戏结束。然后**反向传播(Back-Propagation)**，记录扩展步骤的统计数据。



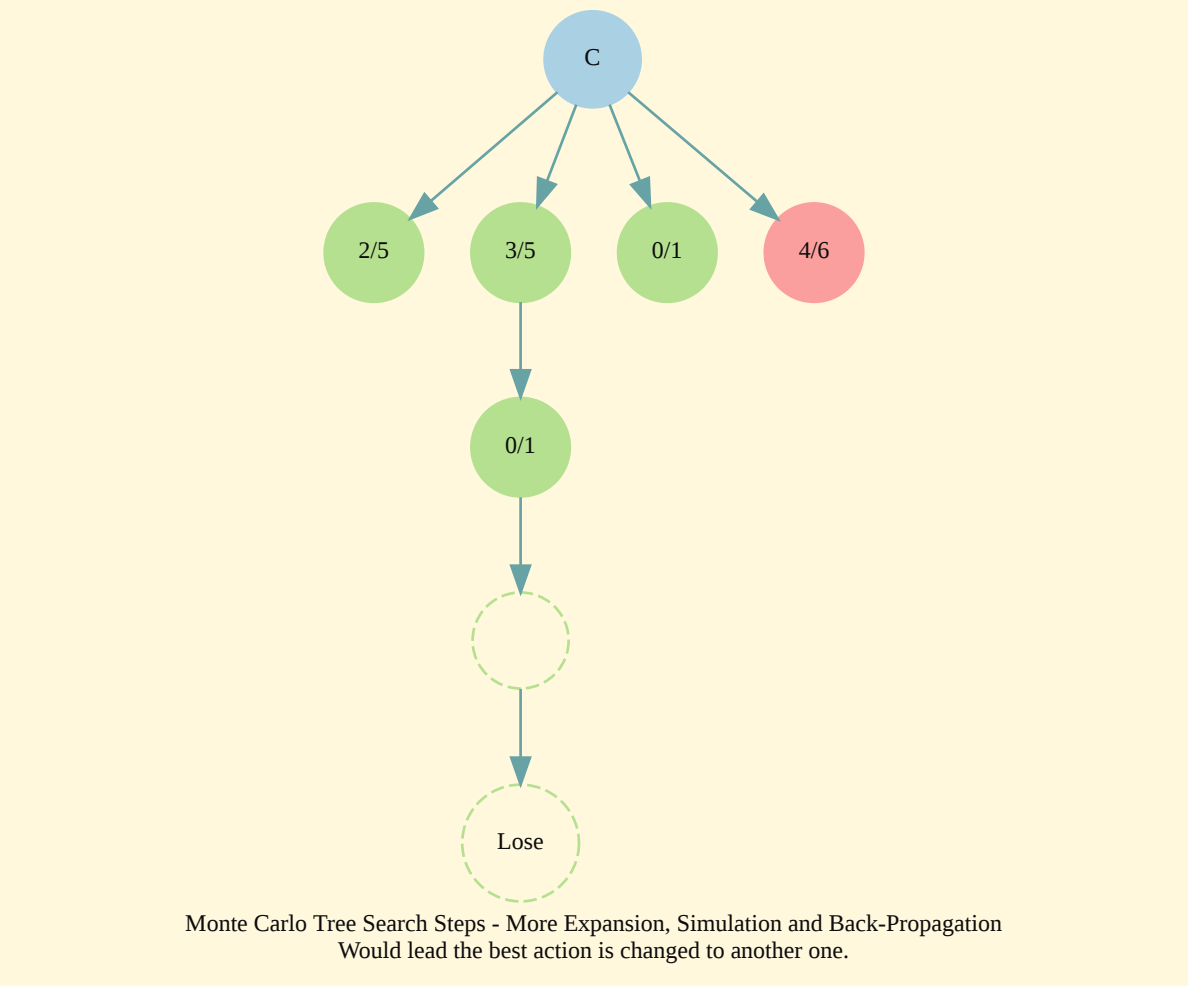
Monte Carlo Tree Search Steps - Expansion

- 多次**扩展(Expansion)**之后，达到了**选举(selection)**的条件，开始**选举(selection)**，选出最优的一个子步骤。



- 继续**扩展(Expansion)**，**模拟(Simulation)**，**反向传播(Back-Propagation)**

  下图说明以前最优的子步骤，可能在多次扩展后，发生变化。



Monte Carlo Tree Search Steps - More Expansion, Simulation and Back-Propagation
Would lead the best action is changed to another one.

阿袁的日记

2016年10月X日 星期六

这周和阿静一起学习了蒙特卡罗树搜索的一些知识。基本上了解了蒙特卡罗树搜索的步骤和使用方法。
发现在使用蒙特卡罗树搜索方法中，有许多可以优化的地方。比如：

- 步骤价值计算
  - 是否可以在没有赢的情况下，计算价值？

- 是否可以计算一个步骤是没有价值的，因而可以及早的砍掉它。

还有许多问题：

- 是否AI程序可以理解规则?比如，理解马走日。
- 是否AI程序可以算出一些领域规则。开局的方法、子力计算等。

参考

- Introduction to Monte Carlo Tree Search by Jeff Bradberry
- Confidence interval

评论列表

#1楼 2017-04-10 10:48 gaot

谢谢楼主，写的非常好

支持(0) 反对(0)