一.

Value Iteration

流程：

1. $\forall s \in S$（对于所有状态），设初值 $V_0(s) = 0$
2. 重复如下更新操作

   $\forall s \in S,\ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$
3. 达到收敛条件

   $\forall s \in S,\ V(s) = V_{k+1}(s) = V^*(s)$

① 这个一个标准的马尔科夫决策过程(MDP)：

- 状态空间State：超级玛丽当前的坐标
- 决策空间Action: 上、下、左、右四个动作
- Action对State的影响和回报 P(State', Reward | State, Action)：本文认为该关系是已知的
  - 超级玛丽每移动一步，reward = -1
  - 超级玛丽得到宝箱，reward = 0并且游戏结束

利用价值迭代 (Value Iteration) 求解马尔科夫决策过程

首先我们定义超级玛丽当前位置的价值 $V(state)$：从当前state = (x, y)开始，能够获得的最大化Reward的总和。

初始化　第一轮迭代　第二轮迭代　第三轮迭代

```
class Agent:
    def __init__(self, env):
        self.env = env
        self.V = np.zeros(env.nS)
                Value iteration 过程
    def next_best_action(self, s, V):
        action_values = np.zeros(env.nA)
        for a in range(env.nA):
            for prob, next_state, reward, done in env.P[s][a]:
                action_values[a] += prob * (reward + DISCOUNT_FACTOR * V[next_sta
        return np.argmax(action_values), np.max(action_values)
                                    返回下标.  返回值
    def optimize(self):
        THETA = 0.0001
        delta = float("inf")
        round_num = 0

        while delta > THETA:
            delta = 0
            print("\nValue Iteration: Round " + str(round_num))
            print(np.reshape(self.V, env.shape))
            for s in range(env.nS):
                best_action, best_action_value = self.next_best_action(s, self.V)
                delta = max(delta, np.abs(best_action_value - self.V[s]))
                self.V[s] = best_action_value
            round_num += 1

        policy = np.zeros(env.nS)
        for s in range(env.nS):
            best_action, best_action_value = self.next_best_action(s, self.V)
            policy[s] = best_action

        return policy
```

② 例2见 note 4

二. Policy Extraction

1. MDP的最终目标是确定一个最优策略
2. $\forall s \in S,\ \pi^*(s) = \arg\max_a Q^*(s,a) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

   （选择不同动作action对应的Q状态中值最大的那个）

三. Policy Iteration

1. 值迭代慢，会进行很多多余计算；采用 policy iteration 可以保持最优性同时收敛更快

2. policy evaluation

(i) 用策略评估对当前策略进行评估。对于一个策略π，策略评估意味着计算所有状态s的Vπ(s)，其中Vπ(s)表示按照策略π从状态s出发的期望效益：

$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

(ii) 第 i 次迭代用 π_i 表示

为了计算收敛的 $V^{\pi_i}(s)$，采用类似值迭代的更新方法

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

(iii) policy improvement

由策略评估生成的状态的值进行策略提取

$$\pi_{i+1}(s) = \text{argmax} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

重复 (i), (ii), (iii) 直到收敛：

$\pi_{i+1} = \pi_i = \pi^*$

3. 例子见 note 4