Markov Decision Process

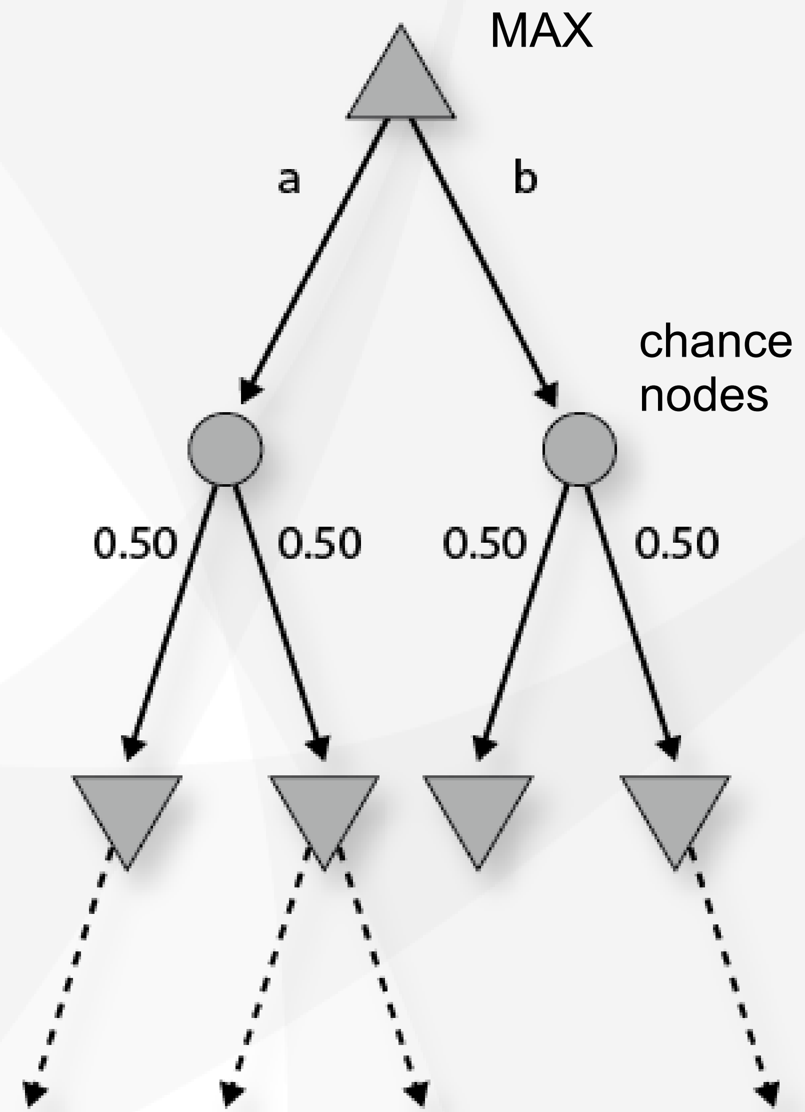
"In which we examine methods for deciding what to do today,
given that we may decide again tomorrow"

Miro Mannino

Course of Artificial Intelligence, University of Bologna 2012/13

Uncertain outcomes

- Outcomes controlled by a **chance node**
  - Randomness
  - Is not guarantee that you'll follow the optimal path
- Utility of states are uncertain
  - We must calculate an **expected value**
  - We can do a weighted average of possible outcomes (expectation)



Expectiminimax search

- Calculate the expected utility in the case we play optimally
  - weighted average of the children of a chance node

def $expmm(s)$:

$$\max_a expmm(move(s,a)) \text{ if } s = MAX$$

$$\min_a expmm(move(s,a)) \text{ if } s = MIN$$

$$\sum_r P(r)\, expmm(move(s,r)) \text{ if } s = CHANCE$$

r is the label of a possible arc from a chance node

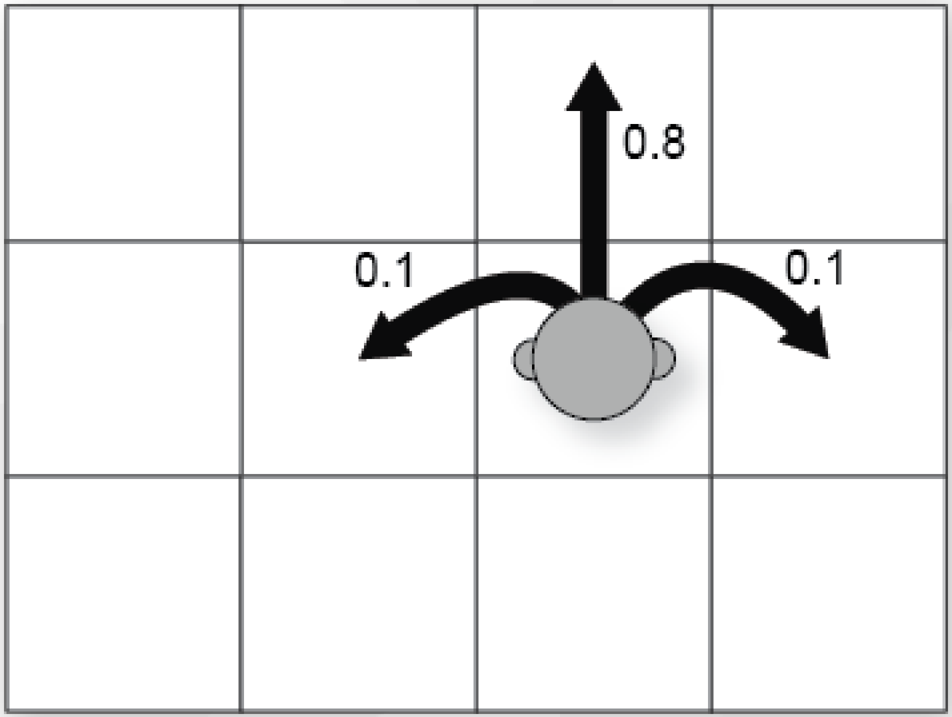
- With trees without MIN nodes, the algorithms is called expectimax search.
  - We'll work with these trees

More info at "Artificial Intelligence: a modern approach 3rd ed" (pag. 171+)

Grid World example

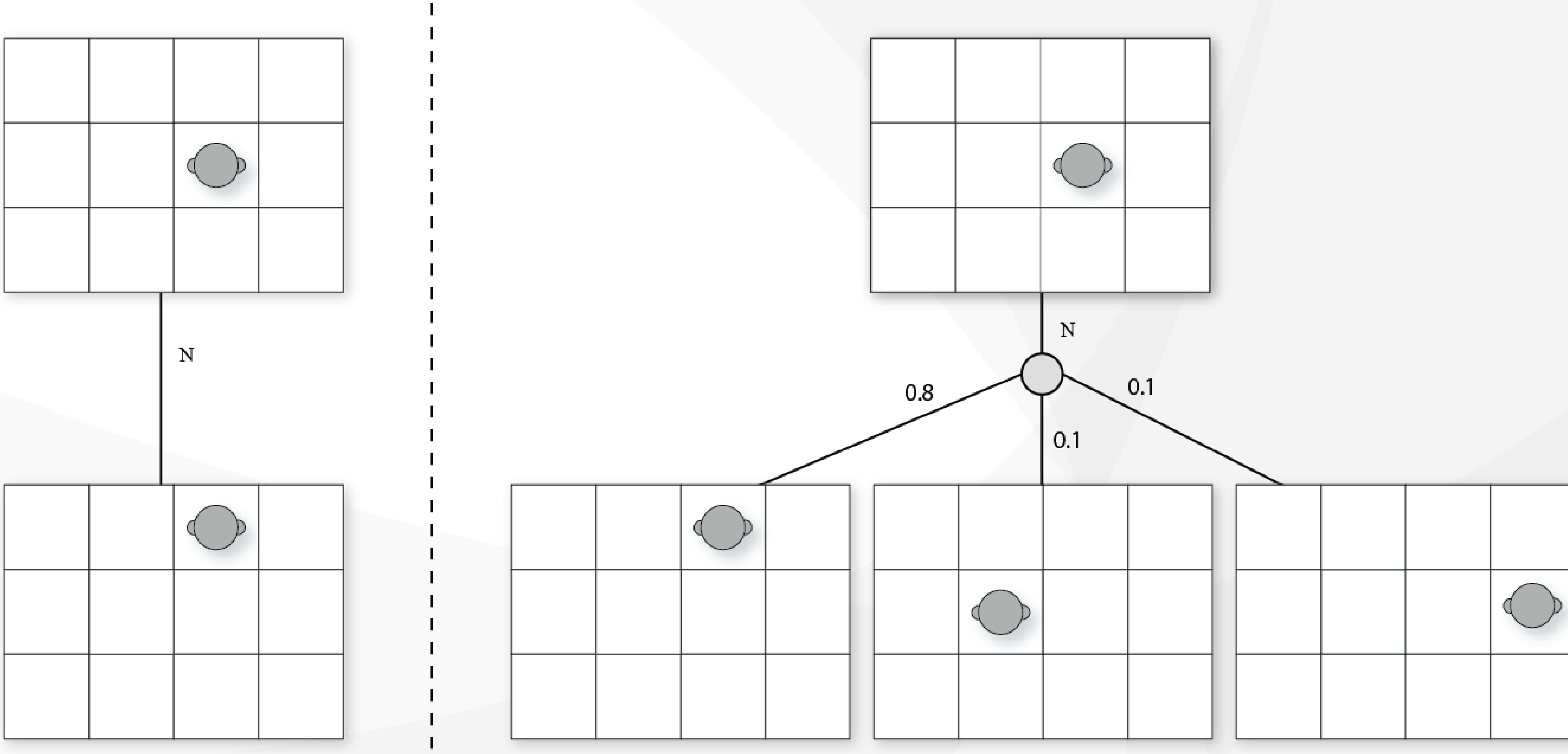
- Environment: 4x3 grid
  - Fully observable
  - Stochastic (we know the probabilities)

For example the action NORTH can lead in three different cells



This example can be found at “Artificial Intelligence: a modern approach 3rd ed” (pag. 645)

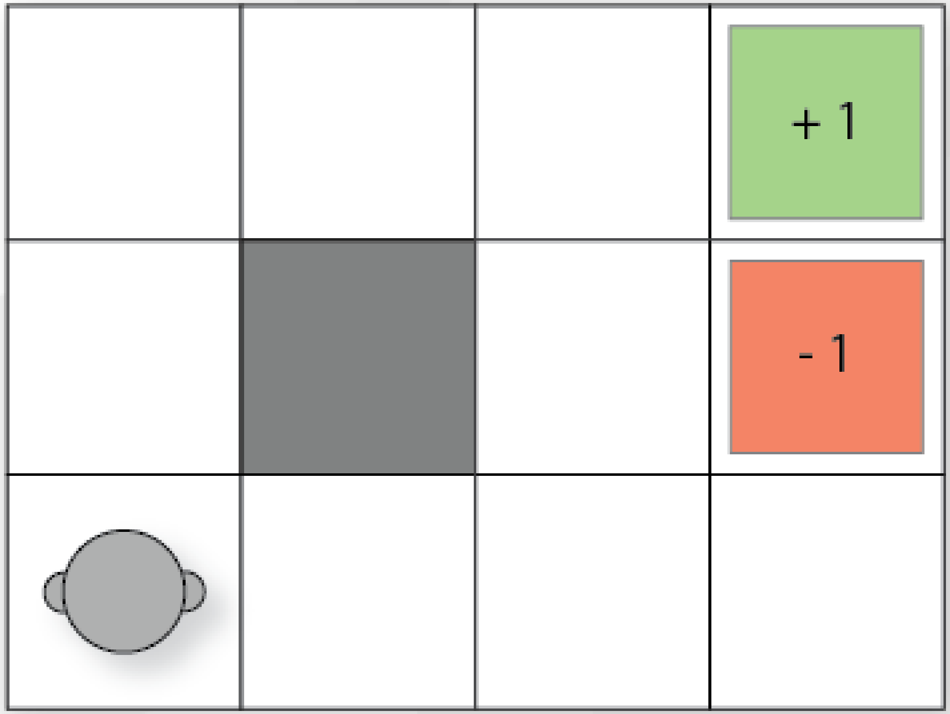
Deterministic vs. Stochastic



The action are unreliable, we have a chance point that execute an arbitrary action

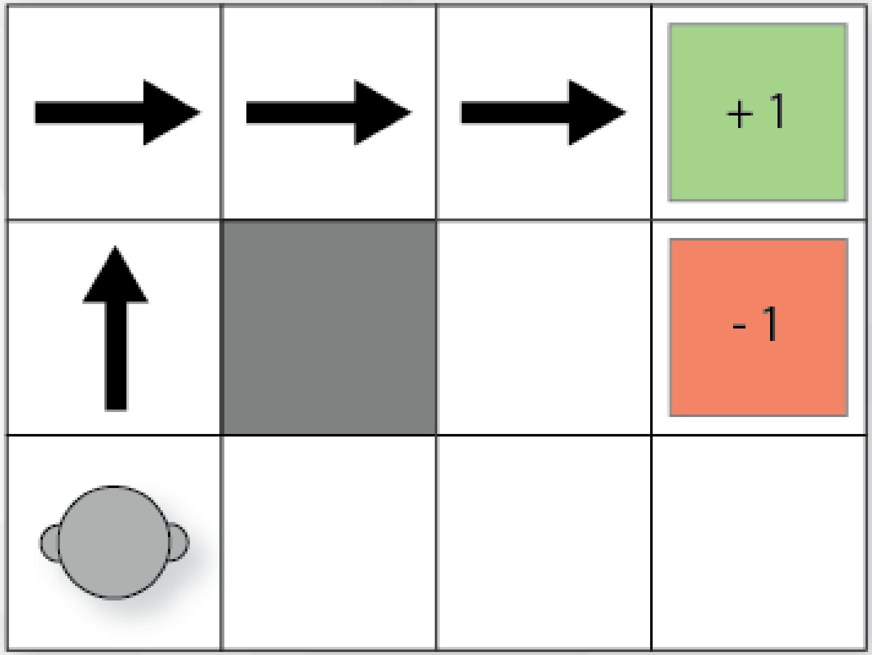
Grid World example (2)

- Various types of cell
  - **Exit**: our goal, with a positive reward, no actions
  - **Pit**: what we want to avoid, with a negative reward, no actions
  - **Wall**: unattainable cell, no reward
  - **Void**: a negative reward to represent the step cost
- Goal:
  - Maximize the sum of rewards
  - Hence, reach a terminal state (i.e. an Exit or a Pit)



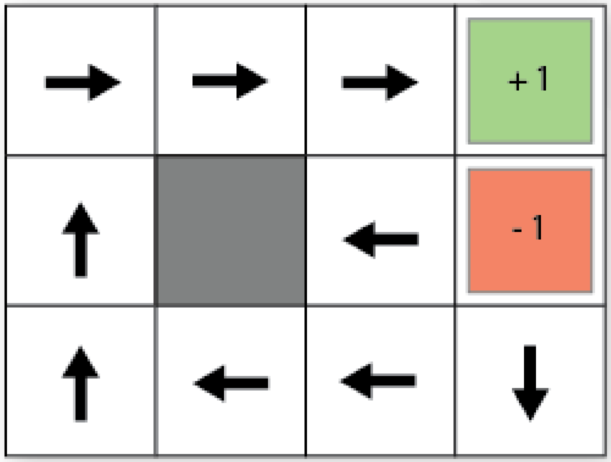
Needs for a policy

- The plan [North, North, East, East, East] is not a solution
  - Caused by the non-deterministic environment
  - A sequence of actions is not sufficient

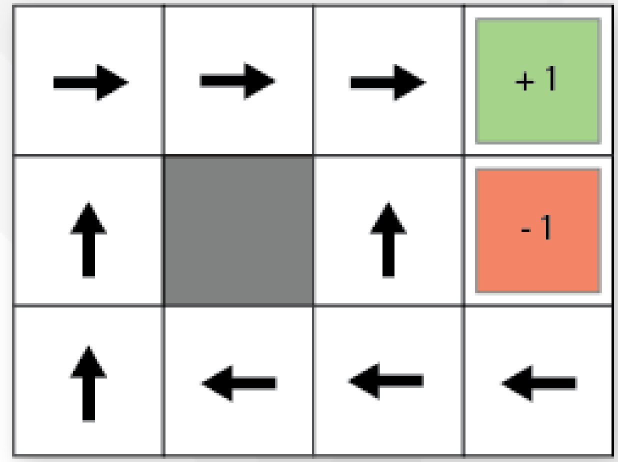


- The solution can't be a plan, but a **Policy**
  - For each state s, the policy π says the action we should do

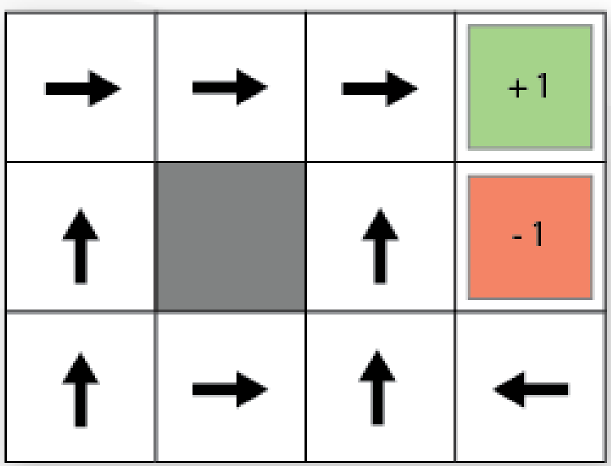
Policy examples



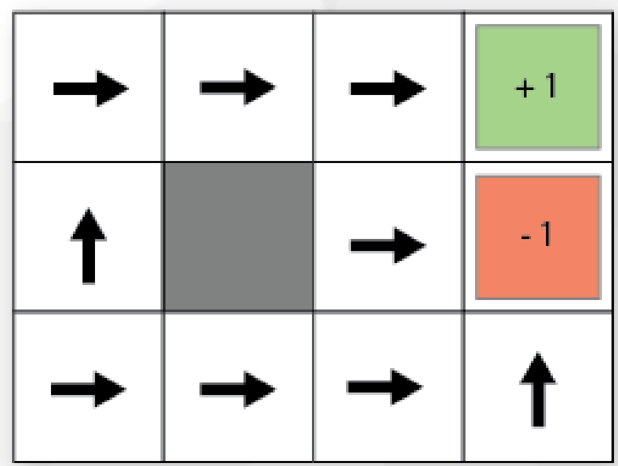
R(s) = -0.01
where s is a void cell



R(s) = -0.04
where s is a void cell



R(s) = -0.09
where s is a void cell



R(s) = -2.00
where s is a void cell

Markov Decision Process

- An MDP is defined by:
  - A set of action $a \in A$
  - A set of states $s \in S$
  - A transition function $T(s, a, s') = P(s' | s, a)$
  - A reward function $R(s, a, s')$
    - Sometimes it is simply: $R(s')$ or $R(s)$
  - An initial state s_0
  - Some terminal state $(s_{1,} s_{2,} s_{3,} \ldots, s_n)$
    - No action allowed from a terminal state

- Why we use the term "Markov"?

Markov chain

- A sequence of random variables

$$X_{1,} X_{2,} \ldots, X_n$$

- All variables satisfy the Markov property

$$P\left(X_{n+1}=x \middle| X_1=x_{1,} X_2=x_{2,} \ldots, X_n=x_n\right)=P\left(X_{n+1}=x \middle| X_n=x_n\right)$$

that is, given the present state, the future and past states are independent

Markov chain and MDP

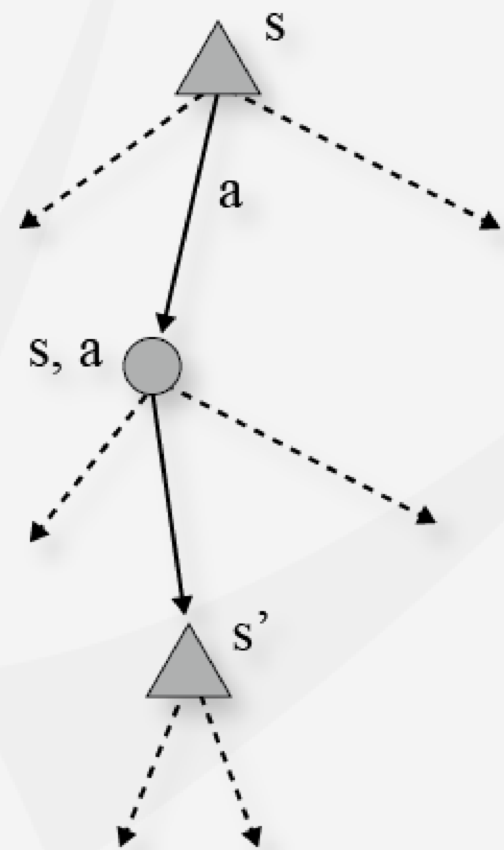
- With the MDP the property is similar

$$P\left(S_{n+1}=s_{n+1}|S_0=s_0, A_0=a_0, S_1=s_1, A_1=a_1, \ldots, S_n=S_n, A_n=a_n,\right)$$
$$=P\left(S_{n+1}=s_{n+1}|S_n=s_n, A_n=a_n\right)$$

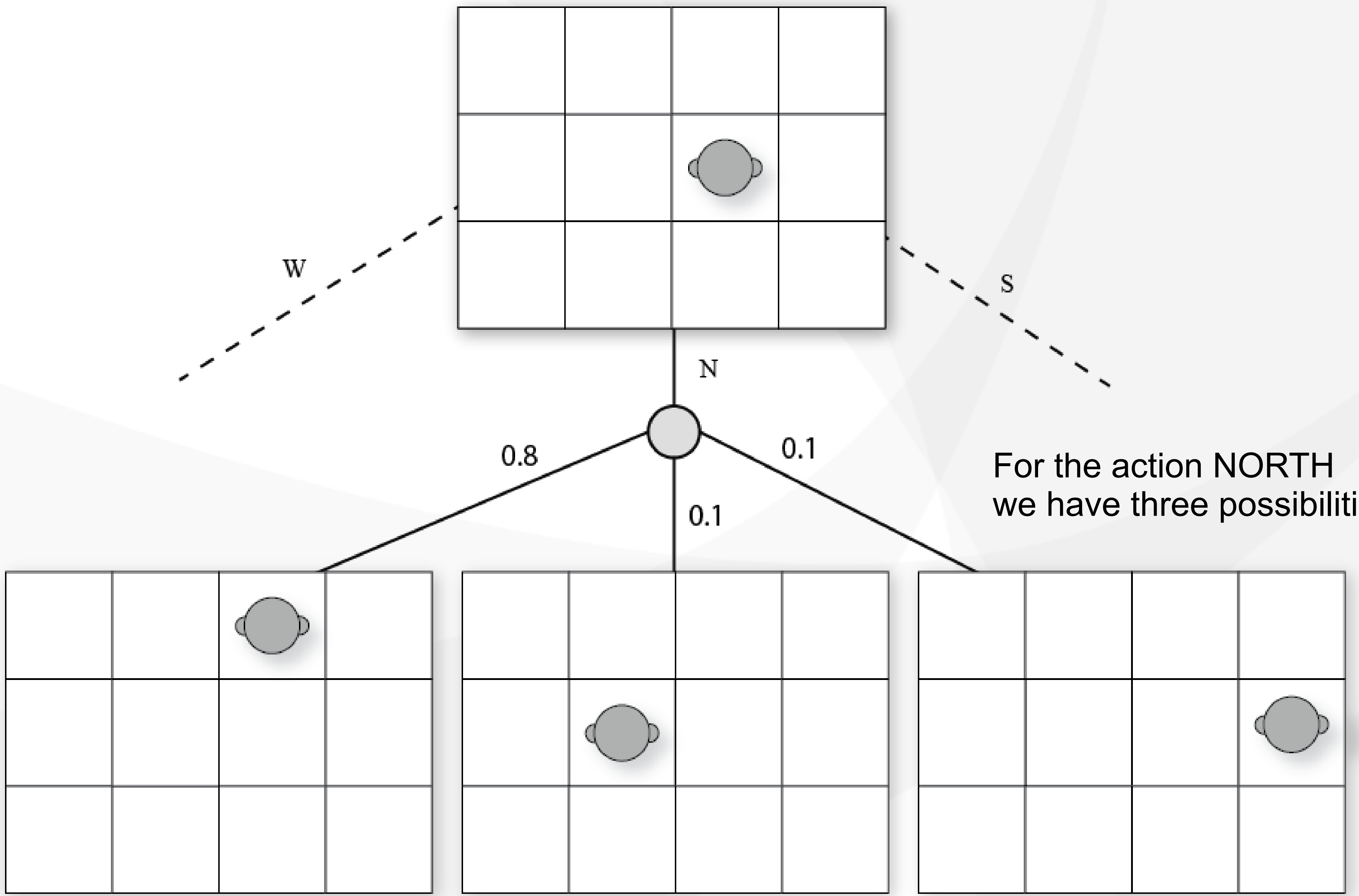
- From a state s_n, The outcome of an action a_n only depends on the current state s_n and not the history.

MDP Search tree

- From state s you can do some actions
- The action a leads us to a **q-state** s,a
  - The q-state represent a state where we have decided the action to do, but we haven't done the action yet, we don't know in which state we'll land.
- With some probability we'll land to the state s' where we can do another action
  - Each arc from a q-state s,a has a probability: $P(s'|s,a)$



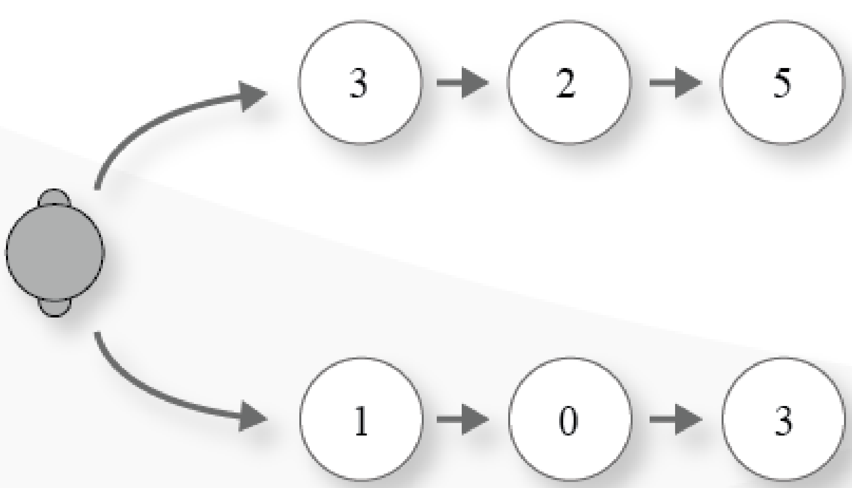
MDP Search tree (2)

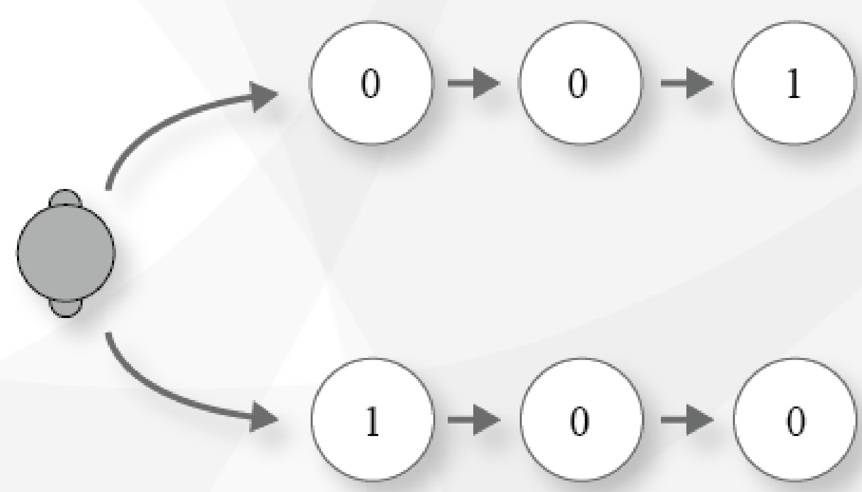


For the action NORTH
we have three possibilities

Utilities of state sequences

- The agent should prefer the "best" sequences, two cases.





- We need "More" because we want to maximize the sum of rewards
- We need "Now" because the non-determinism influences our outcomes

Utilities of state sequences (2)

- Two ways to define the utility of a state sequence:
  - Additive rewards:

    $$U([s_0, s_1, s_2, \dots, s_n]) = \sum_{i=0}^{n} R(s_i)$$

    don't satisfy our “Now” concept

  - Discounted rewards:

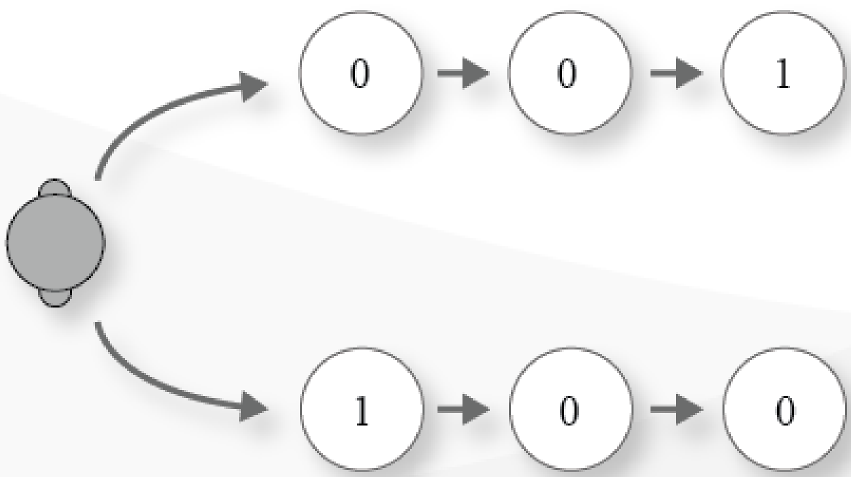
    $$U([s_0, s_1, s_2, \dots, s_n]) = \sum_{i=0}^{n} \gamma^i R(s_i)$$

    satisfies the two concepts “Now” and “More”

    γ is the **discount factor**, a value between 0 and 1

Discounted rewards

- With the discounted rewards:
  - with discount factor of 0.5



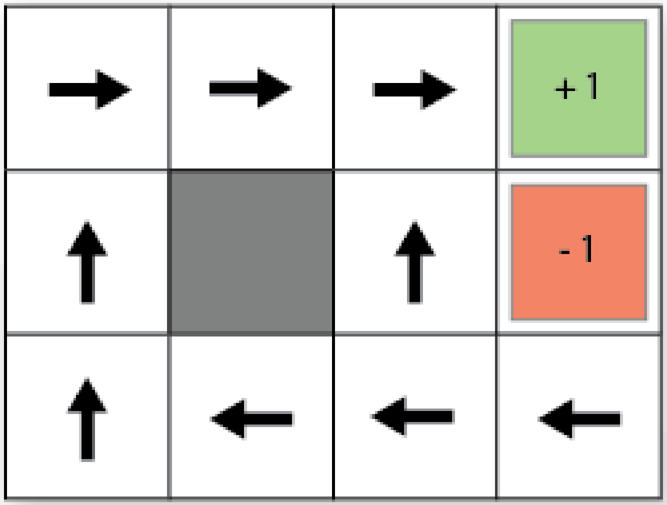
$$U([0,0,1]) = 0*1 + 0*0.50 + 1*0.25 = 0.25$$

$$U([1,0,0]) = 1*1 + 0*0.50 + 0*0.25 = 1$$

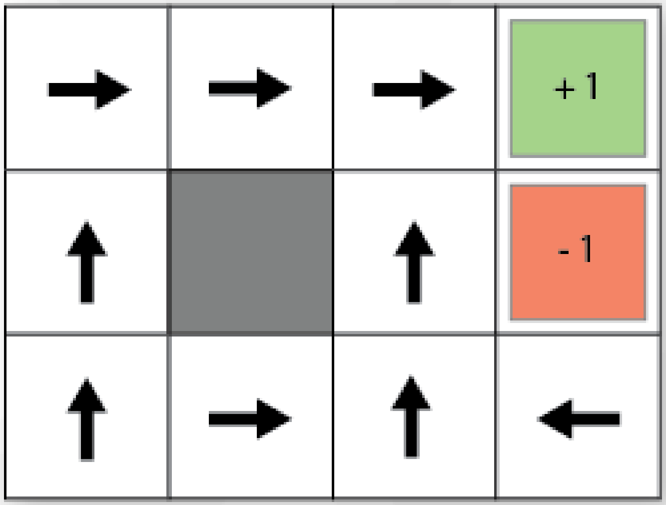
$$U([0,0,1]) < U([1,0,0])$$

Discounted rewards (2)

- In our example:
  - with $R(s) = -0.04$ for the void cells



discount factor = 1



discount factor = 0.8

- with 0.8 the agent prefers to take the risk to fall into the pit. in the other path, the big reward is too far

Discounted rewards (3)

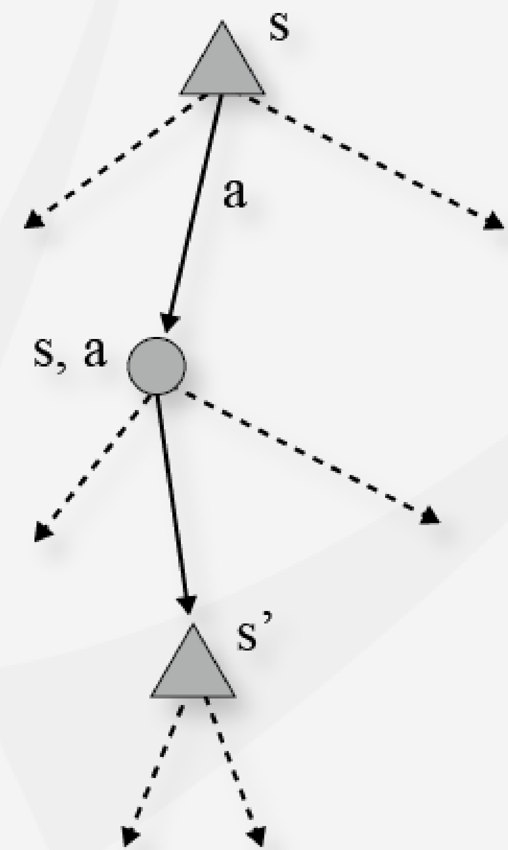
- What if the game doesn't finish?
  - We can use an approach similar to the “depth-limited search”
    - Gives a **non-stationary policy** (i.e. π depends on the time left)
  - We can use the discounted rewards
    - With $\gamma < 1$ we have:

$$U([s_0, s_1, \ldots]) = \sum_{i=0}^{\infty} \gamma^i R(s_i)$$

      converges (it's just a geometric series)
    - With γ we set our “horizon”: the smaller is γ and the smaller is the horizon.
    - It's a **stationary policy**

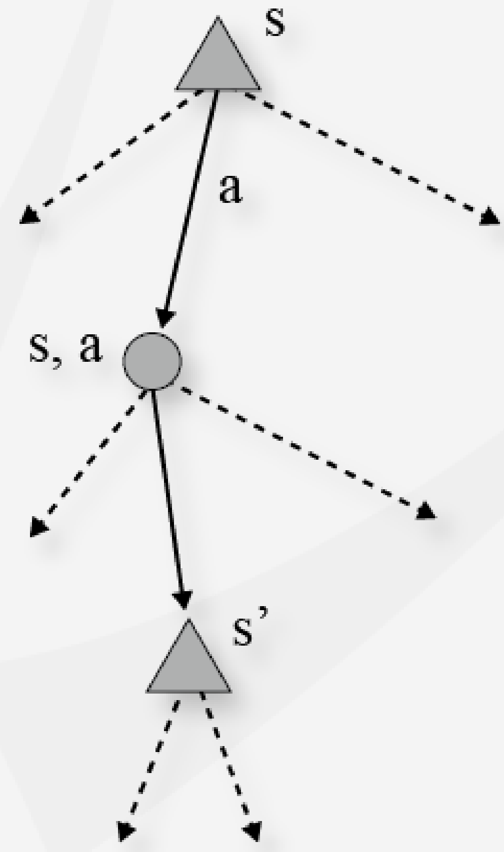
Utilities

- The utility $U^{\pi}(s)$ of a state s:
  - Is the expected utility starting from s and acting as the policy π *wants.*
- The utility $Q^{\pi}(s,a)$ of a q-state s,a:
  - Is the expected utility starting from s and doing the action a
  - It's called **q-value**
- With π^* we denote an optimal policy
  - $U^*(s)$ and $Q^*(s,a)$ denote the latter utility definitions, but using an optimal policy



Calculate Q^*

- From s,a we can go into some states
  - i.e. in the grid world we can go forward with a probability of 0.8, left or right with a probability of 0.1
  - to compute the $Q^*(s,a)$ we need a sum average
  - we define it recursively, using the utilities of the states below (U^*)



$$Q^*(s,a) = R(s) + \gamma * \sum_{s'} P(s'|s,a) * U^*(s')$$

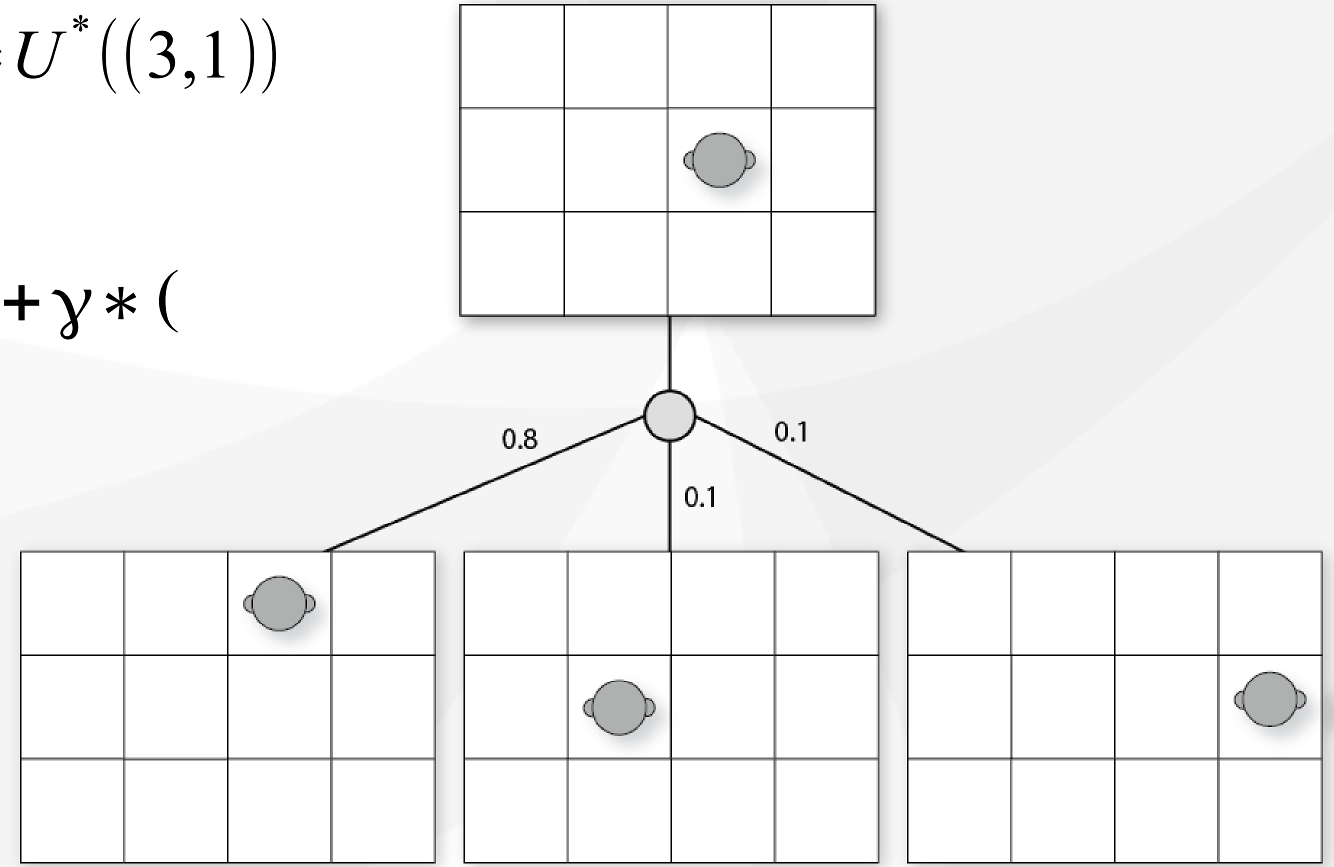
Hence, we can calculate U^* in this way

$$U^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

Calculate Q^* (2)

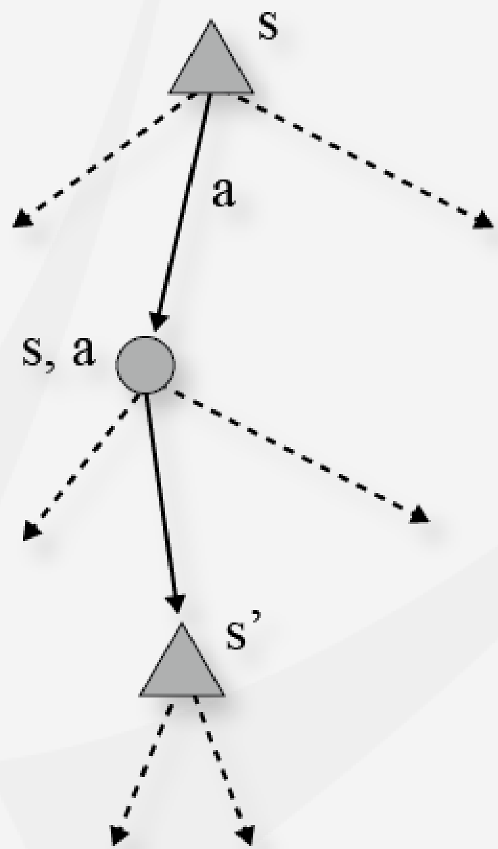
$$Q^*((2,1),N) = R((2,1)) + \gamma * ($$
$$P((2,0)|(2,1),N) * U^*((2,0))$$
$$+P((1,1)|(2,1),N) * U^*((1,1))$$
$$+P((3,1)|(2,1),N) * U^*((3,1))$$
$$)$$

$$Q^*((2,1),N) = -0.04 + \gamma * ($$
$$0.80 * U^*((2,0))$$
$$+0.1 * U^*((1,1))$$
$$+0.1 * U^*((3,1))$$
$$)$$



Calculate U^* *(The Bellman equation)*

- From s we have some possible actions
  - i.e. in the grid world we have NORTH, SOUTH, WEST, EAST
  - to compute the $U^*(s)$ we need the maximum average sum of the utility of the states below
  - we use the discounted rewards
  - the value of a terminal has only the reward $R(s)$



$$U^*(s) = R(s) + \gamma * \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * U^*(s')$$

Extract the policy from U^*

- If we have the optimal utility for all the states:
  - Calculated with:

$$U^*(s) = R(s) + \gamma * \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * U^*(s')$$

- We can calculate the optimal policy in this way:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) * U^*(s')$$

Extract the policy from Q^*

- If we have the q-values:
  - Calculated with:

$$Q^*(s,a)=R(s)+\gamma*\sum_{s'} P(s'|s,a)*U^*(s')$$

  - We know that the state utility can be compute as the following:

$$U^*(s)=\max_{a\in A(s)} Q^*(s,a)$$

  - Hence, we can extract the policy in this way

$$\pi^*(s)=\operatorname{argmax}_{a\in A(s)} Q^*(s,a)$$

Ways to Calculate $U*$

- Calculate the utilities $U*$ from a real tree is too expensive
  - Using the expectimax search
  - A lot of repeated states
  - What should we do with infinite trees?
- If the number of possible states is n, we have n possible Bellman equations.
  - But the equations aren't linear, because there is the “max” operator. The system of equations can't be solved with the linear algebra methods.
- Fortunately, we have two important algorithm:
  - Value Iteration
  - Policy Iteration

Value Iteration

- We start with $\forall s \in S . U_0(s) = 0$
- We update this values until it converges

$$U_{i+1}(s) = R(s) + \gamma * \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * U_i(s')$$

Bellman Update

- There is an unique solution
- The corresponding policy is optimal
- We can see this algorithm as propagating information through the state space by means of local updates
- It's just like a fixed point solution method

Value Iteration (notes)

- Complexity
  - Each iteration needs: $O\left(n^2 * A\right)$
  - It's slow!

- Notes
  - The policy can converge before the values do
  - For each state, the max rarely changes

Policy Iteration

- Alternates two steps:
  - **Policy evaluation**
    - Given the policy π_i we calculate $U_i = U^{\pi_i}$, that is the utility of states where we use the policy π_i

$$U_i(s) = R(s) + \gamma * \sum_{s'} P(s' | s, \pi_i(s)) * U_i(s')$$

these are called "simplified Bellman equations" because use a fixed policy

  - **Policy improvement**
    - Calculate π_{i+1} using the utility of states U_i , π_0 is random

$$\pi_{i+1}(s) = \underset{a \in A(s)}{argmax} \sum_{s'} P(s' | s, a) * U_i(s')$$

Policy Iteration (algorithm)

$repeat$

$U = policyEvaluation(\pi, U)$

$unchanged = true$

$foreach \;\; s \in S:$

$$if \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * U_i(s') > \sum_{s'} P(s' | s, \pi(s)) * U_i(s')$$

$$\pi(s) = \underset{a \in A(s)}{argmax} \sum_{s'} P(s' | s, a) * U_i(s')$$

$unchanged = false$

$until \; unchanged$

i.e. we continue until we reach a fixed point

Policy Evaluation

- At the *i-th* iteration we have π_i and these equations:

$$U_i(s_1) = R(s_1) + \gamma * \sum_{s'} P(s' | s_{1,} \pi_i(a)) * U_i(s')$$

$$U_i(s_2) = R(s_2) + \gamma * \sum_{s'} P(s' | s_{2,} \pi_i(a)) * U_i(s')$$

…

  - These equations are linear (no max operator)
  - Can be solved by a standard linear algebra methods $O(n^3)$

Policy Evaluation (2)

- $O(n^3)$ can be slow for large space states.
- We don't need the exact values
  - We can perform some number of Bellman update

$$U_{i+1}(s) = R(s) + \gamma * \sum_{s'} P(s' | s, \pi_i(a)) * U_i(s')$$

    - Repeated k times to produce the next utility estimation
    - Similar to the value iteration, but simpler

  - This algorithm is called **modified policy iteration**
    - It's often faster than the policy iteration and value iteration.
    - It's still optimal!

Bibliography

- Russell, Stuart, Peter Norvig, and A. Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs (1995).
  - Chapter: 5 (for expectiminimax)
  - Chapter: 17 (for MDP)
  - Chapter: 21 (for Q-States)
- Veness, Joel. Expectimax Enhancements for Stochastic Game Players. Diss. UNIVERSITY OF NEW SOUTH WALES, 2006.
- Ching, Wai Ki, and Michael K. Ng. Markov chains: models, algorithms and applications. Vol. 6. New York: Springer, 2006.
- Littman, Michael L., Thomas L. Dean, and Leslie Pack Kaelbling. "On the complexity of solving Markov decision problems." Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1995.
- CS188.1x: Artificial Intelligence. www.edx.org/courses/BerkeleyX/CS188.1x