Chapter 17: Making Complex decisions

These slides are adopted from Berkeley course materials and Russell and Norvig textbook

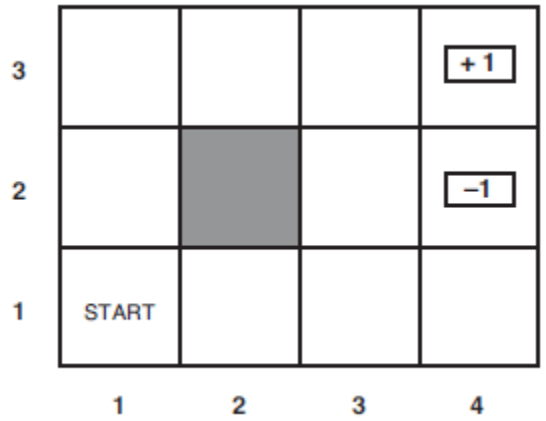
Sequential Decisions

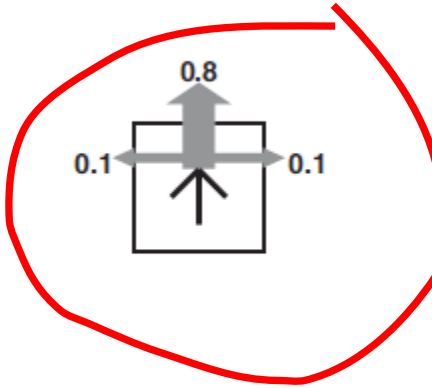
- In this chapter, we consider **sequential decision problems**, in which the agent's utility depends on a sequence of decisions.

- We focus on stochastic environments.

Example: Grid World

- The agent lives in a 4x3 grid.
- Walls block the agent's path





- The actions in every state ACTIONS(s): U*p*, *Down*, *Left*, and *Right*.
- We assume a fully observable environment (the agent knows where it is.
- However, actions are noisy (stochastic environment).
- Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction.
- If the agent bumps into a wall, it stays in the same square.
- For example:
  - 80% of the time, the action Up takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East

Example: Grid World

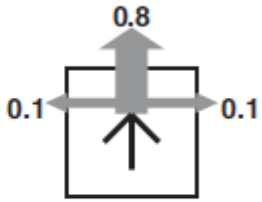
The **transition model P(s'|s,a)**

- P(s'| s, a) to denote the probability of reaching state s' if action a is done in state s.
- We will assume that transitions are **Markovian.**
- The probability of reaching s' from s depends only on s and not on the history of earlier states.

- The agent receives rewards R(s) each time step
  - Small "living" reward each step (can be positive or negative)
  - In the grid world example, R(s)=-0.04
  - Big rewards come at the end (good or bad)
  - For example:
  - In the grid world example, there are two terminal states having rewards +1 and –1.
  - All other states have a reward of –0.04.
- Goal: maximize sum of rewards

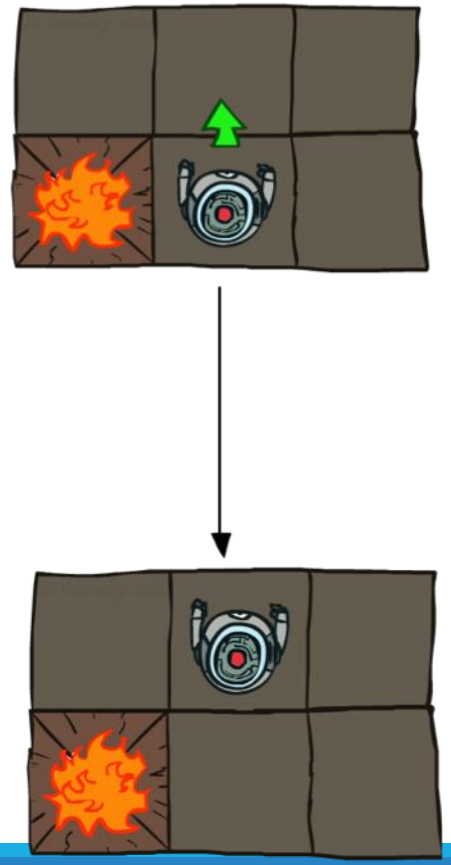
Example: Grid World

- If the environment were deterministic, a solution would be easy: [*Up, Up, Right, Right, Right*].
- Unfortunately, the environment won't always go along with this solution, because the actions are unreliable!

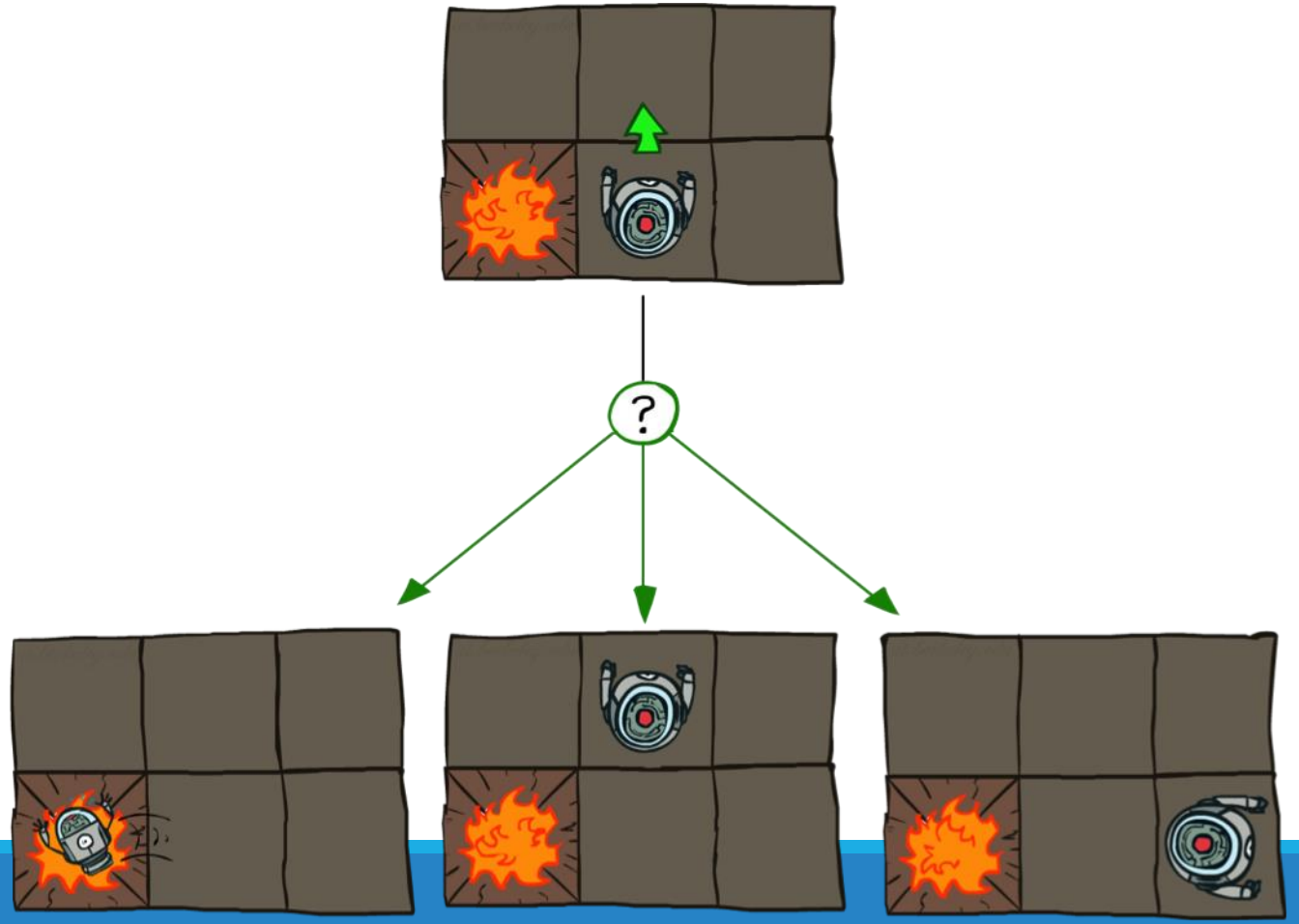


Grid World Actions

Deterministic Grid World



Stochastic Grid World



Markov Decision Processes (MDPs)

A sequential decision problem for a fully observable, stochastic environment

with a Markovian transition model and additive rewards is called a **Markov decision process** (**MDP).**

What is Markov about MDPs?

"Markov" generally means that given the present state, the future and the past are independent

For Markov decision processes, "Markov" means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

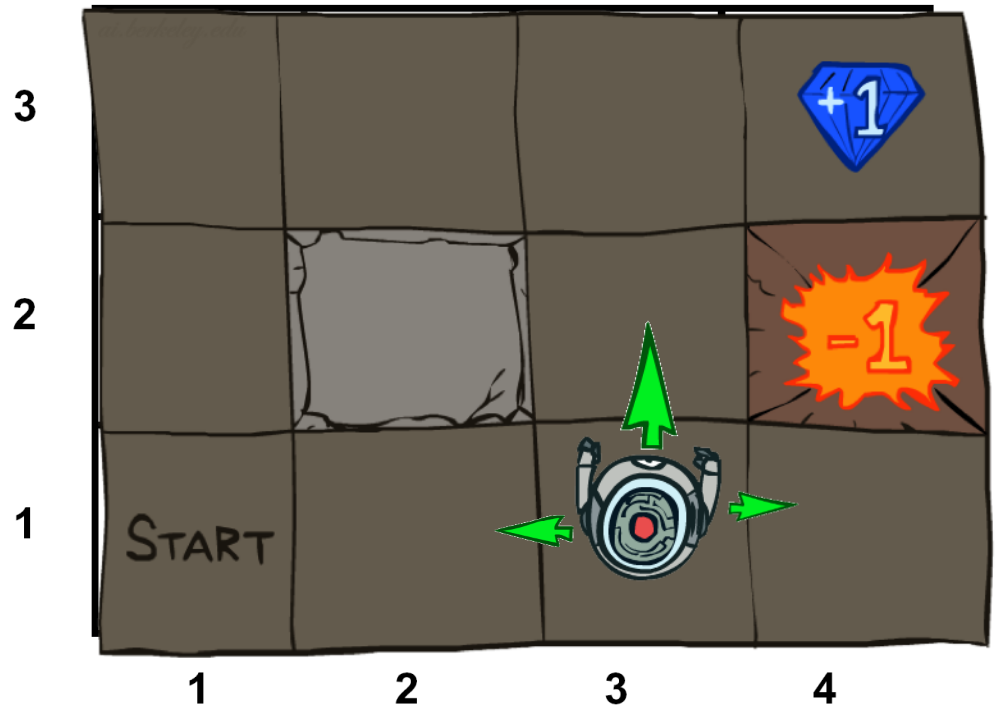
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

This is just like search, where the successor function could only depend on the current state (not the history)

Markov Decision Processes (MDPs)

An MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P(s' | s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$ General form
  - Sometimes just R(s)
- An initial state S0
- Maybe a terminal state



[Demo – gridworld manual intro (L8D1)]

Policy

- In deterministic search problems, we wanted an optimal plan, or sequence of actions, from start to a goal.

What does a solution to a sequential decision problem look like?

- Any fixed action sequence won't solve the problem, because the agent might end up in a state other than the goal.
- Therefore, a solution must specify what the agent should do for *any* state that the agent might reach.
- A solution of this kind is called a **policy.**

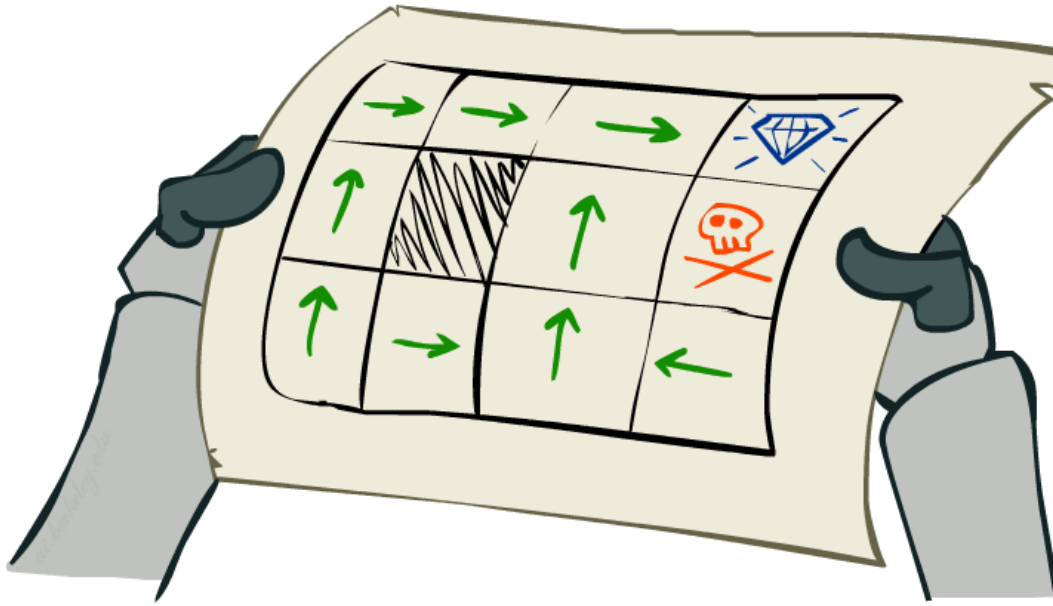
Policy

- In deterministic search problems, we wanted an optimal plan, or sequence of actions, from start state to a goal.

- For MDPs, we want an optimal

  policy $\pi^*: S \rightarrow A$

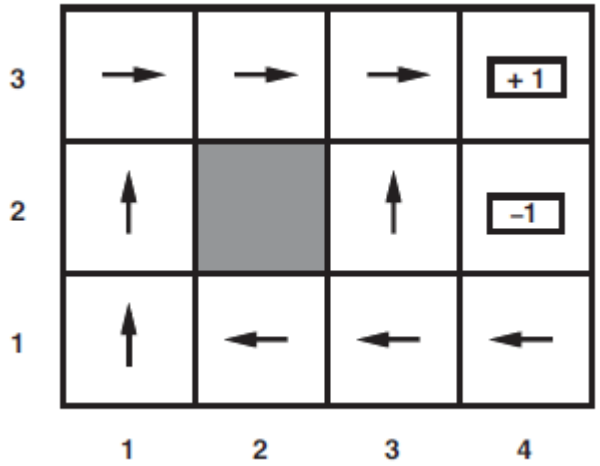
  - A policy π gives an action for each state
  - An **optimal policy** is a policy that yields the maximum expected utility.
  - A policy represents the agent function explicitly and is therefore a description of a simple reflex agent, computed from the information used for a utility-based agent.



Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Optimal Policies

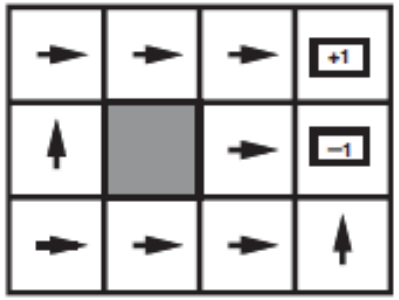
- Since the cost of taking a step is fairly small compared with the penalty for ending up in (4,2) by accident, the optimal policy for the state (3,1) is conservative.
- The policy recommends taking the long way round, rather than taking the shortcut and thereby risking entering (4,2).



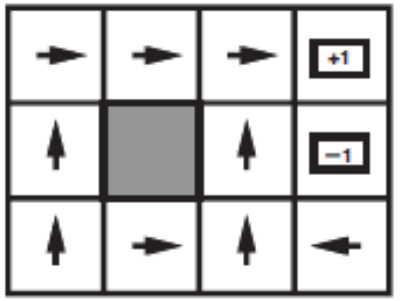
R(s)=-0.04

Optimal Policies

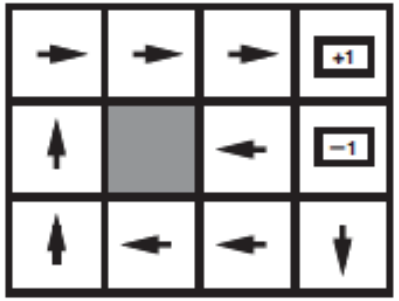
- The balance of risk and reward changes depending on the value of R(s) for the nonterminal states.
- For R(s) ≤ −1.6284, life is so painful that the agent heads straight for the nearest exit, even if the exit is worth –1.
- When −0.4278 ≤ R(s) ≤ −0.0850, life is quite unpleasant; the agent takes the shortest route to the +1 state and is willing to risk falling into the –1 state by accident.
- For (−0.0221 < R(s) < 0), the optimal policy takes *no risks at all*.
- Finally, if R(s) > 0, then life is positively enjoyable and the agent avoids *both* exits and the agent obtains infinite total reward because it never enters a terminal state.



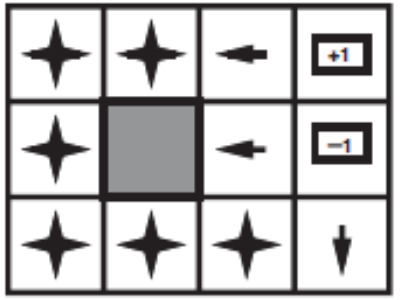
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



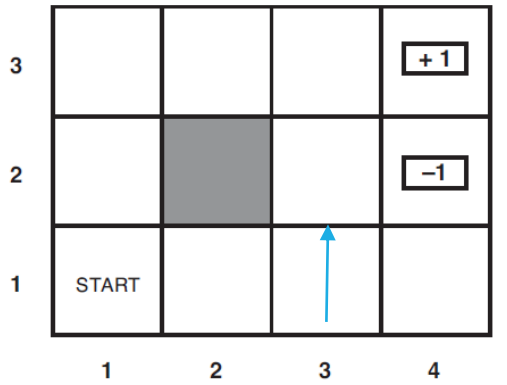
$-0.0221 < R(s) < 0$



$R(s) > 0$

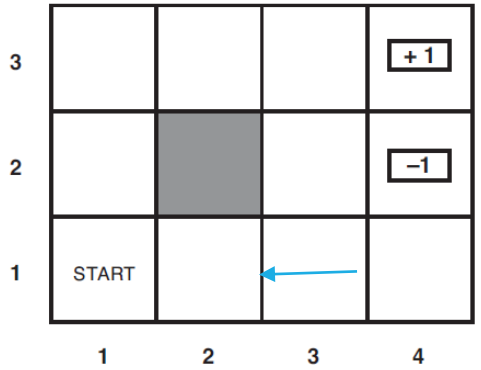
Finite vs. Infinite Horizons

- Finite horizon means that there is a *fixed* time N after which nothing matters (the game is over).
- It is similar to depth-limited search.
- For example, suppose an agent starts at (3,1), and N =3.



The optimal action is to go *Up*.

- On the other hand, if N =100, then there is plenty of time to take the safe ro
- The optimal action is to go *Left.*



Finite vs. Infinite Horizons

- If the optimal action in a given state could change over time, the policy is named nonstationary.
- Thus, finite horizons yield nonstationary policies.
- With no fixed time limit, there is no reason to behave differently in the same state at different times. Hence, the optimal action depends only on the current state, and the optimal policy is **stationary**.
- Infinite horizon does not necessarily mean that all state sequences are infinite; it just means that there is no fixed deadline.
- In particular, there can be finite state sequences in an infinite-horizon MDP containing a terminal state (for example: the grid world with no fixed N).

Utilities of Sequences

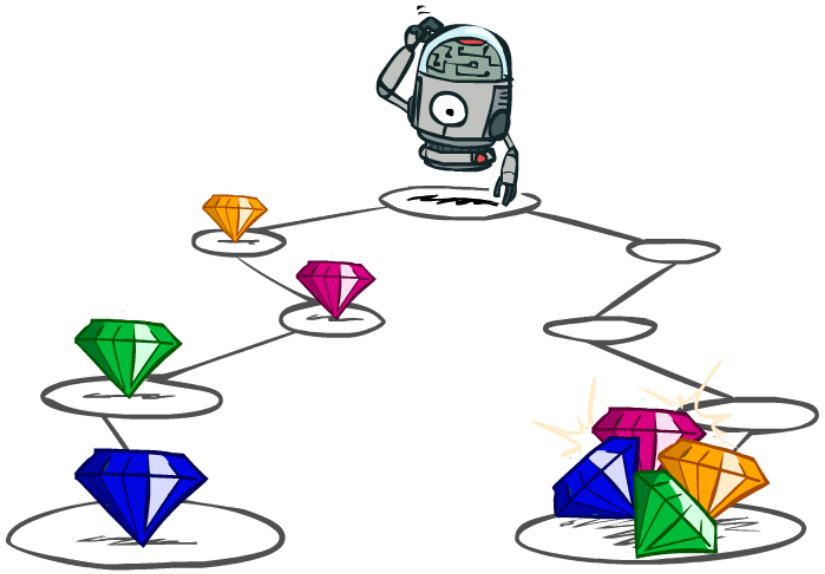
What preferences should an agent have over reward sequences?

[1, 2, 2] or [2, 3, 4]

More or less?

[0, 0, 1] or [1, 0, 0]

Now or later?



Utilities of Sequences

1. Additive Rewards

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots .$$

2. Discounted Rewards

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

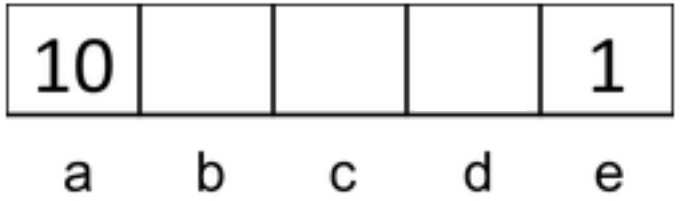
- **A discount factor** γ is a number between 0 and 1.
- The discount factor describes the preference of an agent for current rewards over future rewards.
- When γ is close to 0, rewards in the distant future are viewed as insignificant.
- When γ is 1, discounted rewards are exactly equivalent to additive rewards,

Example: discount of 0.5

- U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
- U([1,2,3]) < U([3,2,1])

Example: Discounting

Given:



- Actions: East, West, and Exit (only available in exit states a, e)
- Transitions: deterministic

10	<-	<-	<-	1

1. For $\gamma = 1$, what is the optimal policy?

10	<-	<-	->	1

2. For $\gamma = 0.1$, what is the optimal policy?

3. For which γ are West and East equally good when in state d?

$1\gamma = 10\,\gamma^3$

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?

- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

- If the environment contains terminal states *and if the agent is guaranteed to get to one*
- *eventually*, then we will never need to compare infinite sequences.
- A policy that is guaranteed to reach a terminal state is called a proper policy.

Optimal utilities and policies

- The expected utility obtained by executing π starting in s is given by:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

- U(s) and R(s) are quite different quantities
- R(s) is the "short term" reward for being in s.
- U(s) is the "long term" total reward from s onward.

- The optimal policy maximizes the expected utility.

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

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

γ=1, R=-0.04

34an ageb el best utility, ana 3auz ashof men el action elly btgebly a7sn future reward, fa hnst5dm el transition equation mn el robotics baa, w m3 est5dam markov assumption, hgeb el probability eny aro7 le state s` given eny kont fe state s, w khadt action a. w adrb el klam da fe el long term total reward bta3 s`, w e7na bn iterate 3la kol el states w n3ml sum, w el argmax btgbly el action elly 3ndu 27sn value (max).

de equation mohema gedan

Bellman Equation

- The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

- Bellman Equation:

el goz2 da hwa el bn7sb beh el optimal policy

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

w da el current reward.

da bygeb kol el states, recursive function.

el hadaf bt3na baa hwa ezay n7l el equation de, el mwdo3 s3b shwya bsbb wgod el max, lw mkantsh mwgoda kan hyb2a linear equation w han7lha 3ady.

Bellman Equation

- For n possible states, there are n Bellman equations, one for each state.
- The n equations contain n unknowns—the utilities of the states.
- So we would like to solve these simultaneous equations to find the utilities.
- However, the equations are *nonlinear*, because the "max" operator is not a linear operator.
- Whereas systems of linear equations can be solved quickly using linear algebra techniques, systems of nonlinear equations are more problematic.
- One thing to try is an *iterative* approach.

Value Iteration

1. Start with arbitrary initial values for the utilities,

2. Calculate the right-hand side of the Bellman equation, and plug it into the left-hand side—thereby updating the utility of each state from the utilities of its neighbors.

3. We repeat this until we reach an equilibrium. (Convergence)

Let Ui(s) be the utility value for state s at the i^{th} iteration.

The iteration step, called a **Bellman update**, looks like this:

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

ana babd2 en el utility bta3thom kolohom b 0
b3d keda est5ddm el belman equation ka update, enk kol state tst5dm el right hand side 34an t7sb el utility.
el meza en l value el gdeda de btb2a a7sn mn el initial 34an bt2rb mn el actual result.

Value Iteration Algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

update the new utility, using the previoius utility.

lw el gama b 1, 3omro ma hykhls, lw b 0, hy5ls b3d awl khatwa.

Value Iteration

If we apply the Bellman update infinitely often, the utilities will eventually converge.

In fact, they are also the *unique* solutions to the Bellman equations, and the corresponding policy is optimal.

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

Key note:

Policy may converge long before values do

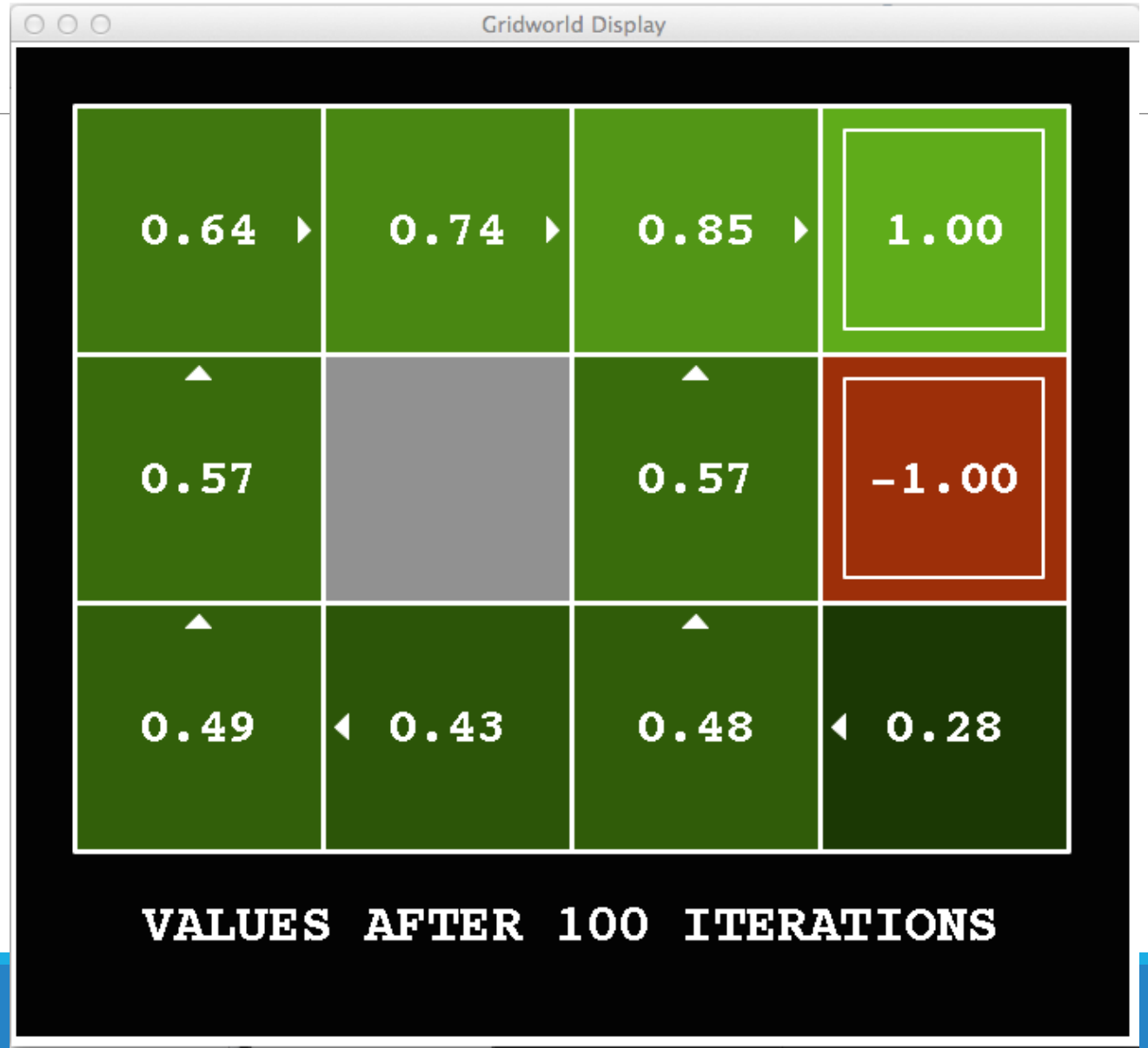
akbur mushkela lel 7war da eno bya5ud w2t kber.

k=12

Noise = 0.2
Discount = 0.9
Living reward = 0

el ashom homa el policy bta3tna, fa ana mohtm eny ageb el policy bs, fa lw bset 3la el fo2 wl t7t, el policy sabta(el ashom mbtt8yrsh) lakn el utilities homa el byt8yro.

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

tb hwa msh lw ana 3ndy tre2a a3rf beha el policy 3ltoul, msh hyfr2 m3aya el utility? bzbt, fa hwa da baa elly el policy iteration by3mlo.

Policy Iteration

If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.

Thus, we can directly find the optimal policies.

The **policy iteration** algorithm alternates the following two steps, beginning from some initial policy $\pi 0$:

1. Policy evaluation: given a policy πi, calculate $Ui = U^{\pi i}$, the utility of each state if πi were to be executed.

2. Policy improvement: Calculate a new MEU policy $\pi i+1$, using one-step look-ahead based on Ui.

The algorithm terminates when the policy improvement step yields no change in the utilities.

Policy Evaluation

At the i^{th} iteration, the policy π_i specifies the action $\pi_i(s)$ in state s.

This means that we have a simplified version of the Bellman equation relating the utility of s (under π_i) to the utilities of its neighbors:

el max msh hyfr2 m3ana delw2ty, l2n el max kan mwgod, 34an ana msh 3aref anhy a7sn, fa kont bab2a 3auz a3ml max 34an a3rf men a7sn choice lya, lakn ana hena msbt el policy, fa khlas msh m7tag el max, w bema en el max mb2ash mwgod, fa ana keda baa 3ndy linear

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

equation, fa hyb2a asr3 bkter awy fe el 7sabat bt3to.

These equations are *linear*, because the "max" operator has been removed.

For n states, we have n linear equations with n unknowns, which can be solved exactly in time $O(n^3)$ by the standard linear algebra methods.

Policy Improvement

Calculate the updated policy π_{i+1} using the evaluated utility values from the policy evaluation step $U^{\pi i}$:

$$\pi_{i+1}(\mathrm{s}) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s'|s,a) U^{\pi i}(s')$$

Policy Iteration Algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

lw mtghyrtsh, bn3ml return true khalas.

Value Iteration versus Policy Iteration

Both value iteration and policy iteration compute the same thing (all optimal values)

In value iteration:

- Every iteration updates both the values and (implicitly) the policy
- We don't track the policy, but taking the max over actions implicitly recomputes it

In policy iteration:

- We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
- After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
- The new policy will be better (or we're done)

Race Car Example

States={cool, warm, overheated}

Actions={fast, slow}

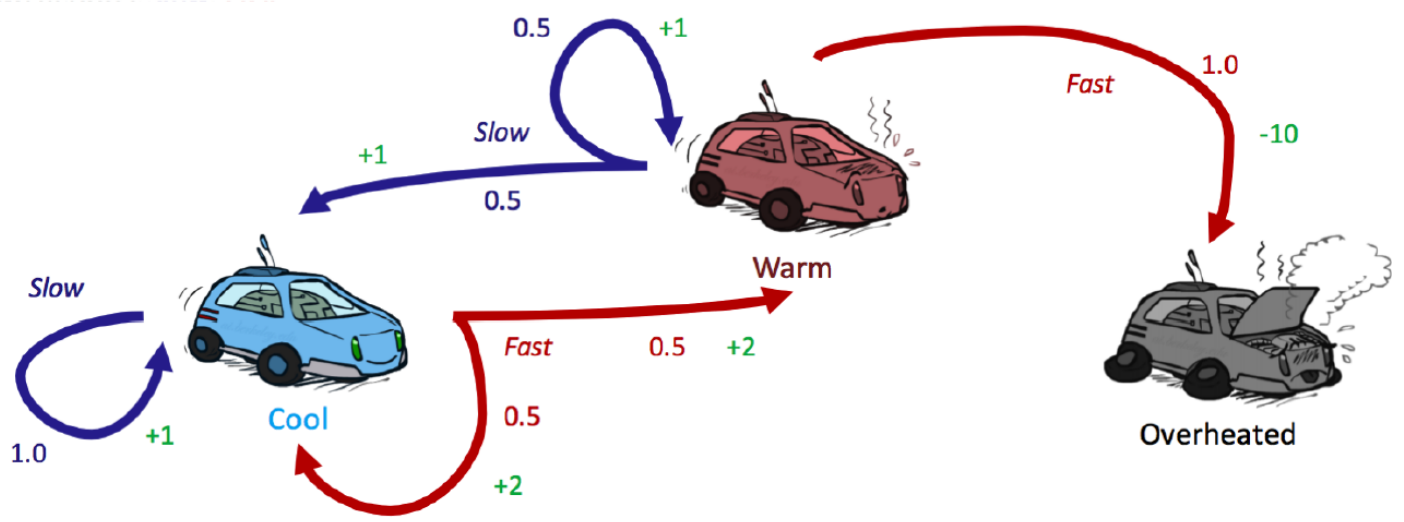
γ=0.5

Initially:

U0(cool)=0

U0(warm)=0

U0(overheated=0)



$$U_{i+1}(S) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$

Run two iterations of the value iteration algorithm.

Race Car Example

	cool	**warm**	**overheated**
V_0	0	0	0
V_1	2	1	0
V_2	2.75	1.75	0

Race Car Example

Apply policy evaluation for the given initial policy:

	cool	**warm**	**overheated**
π_0	*slow*	*slow*	–

Race Car Example

Policy Evaluation:

	cool	**warm**	**overheated**
V^{π_0}	2	2	0

Race Car Example

Apply policy iteration given the initial policy

	cool	**warm**	**overheated**
π_0	*slow*	*slow*	–

	cool	**warm**
π_0	*slow*	*slow*
π_1	*fast*	*slow*
π_2	*fast*	*slow*

Sections 17.1-17.3 from the textbook.