Value Iteration
(Ch. 17.1-17.2)

Markov Decision Process

Last time, we discussed how random variables and utility functions could tell you the "best" action to take among options

We will continue this train of though, but now we will need to take multiple actions before we can stop

One way of framing this is called a Markov decision process (MDP)

Markov Decision Process

The “Markov” property is useful as it means the only thing that matters where we go next is our current state/position (not any previous)

Just as we have in the past, we will assume there is some uncertainty in the problem

A simple example of this would be a robot exploring a simple grid-world, but sometimes it does not go where it wants when moving

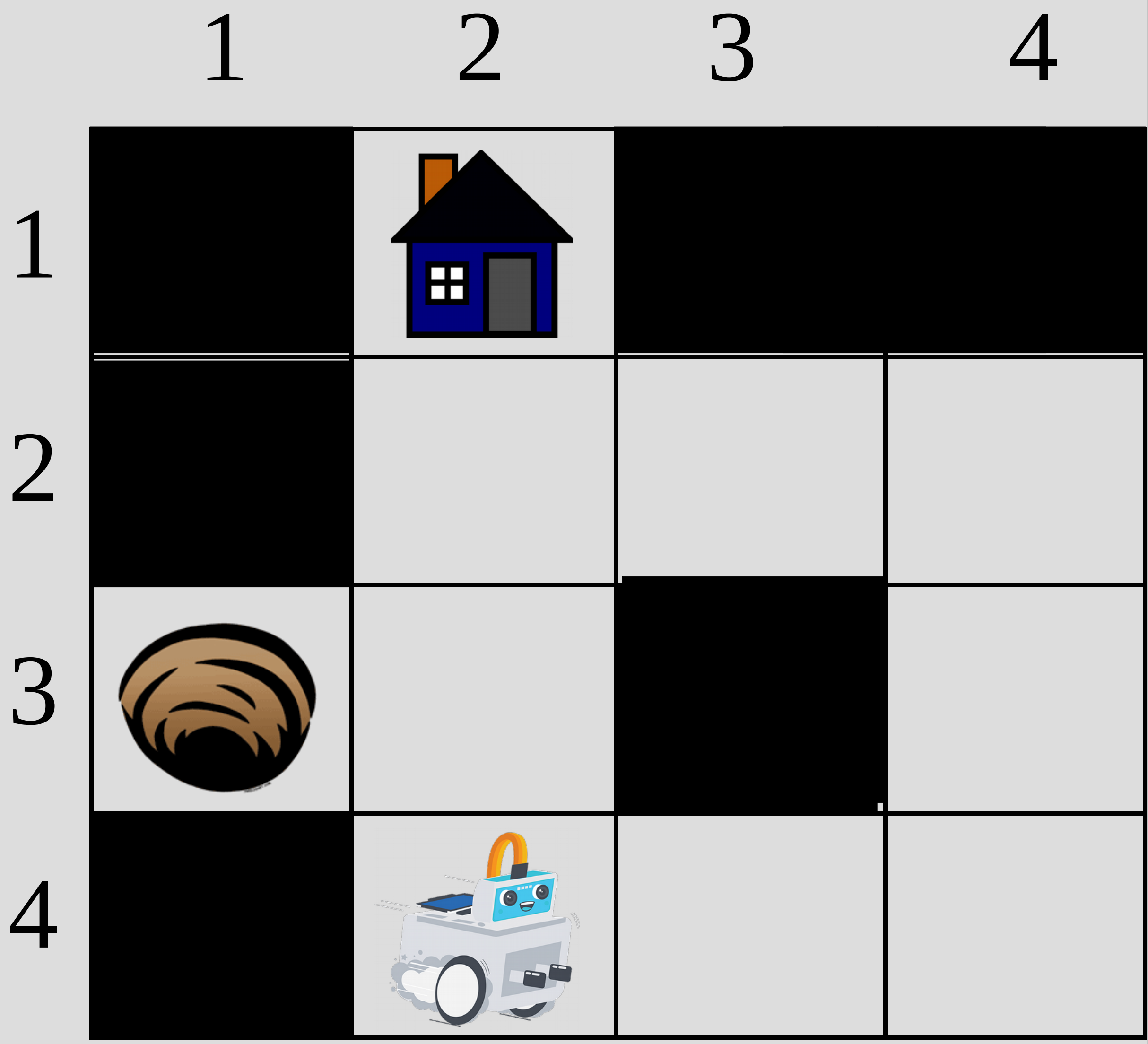
Markov Decision Process

🏠 = +50 (end)

🕳 = -50 (end)

All other = -1 (i.e. -1 for movement)

Goal: maximize score before reaching end

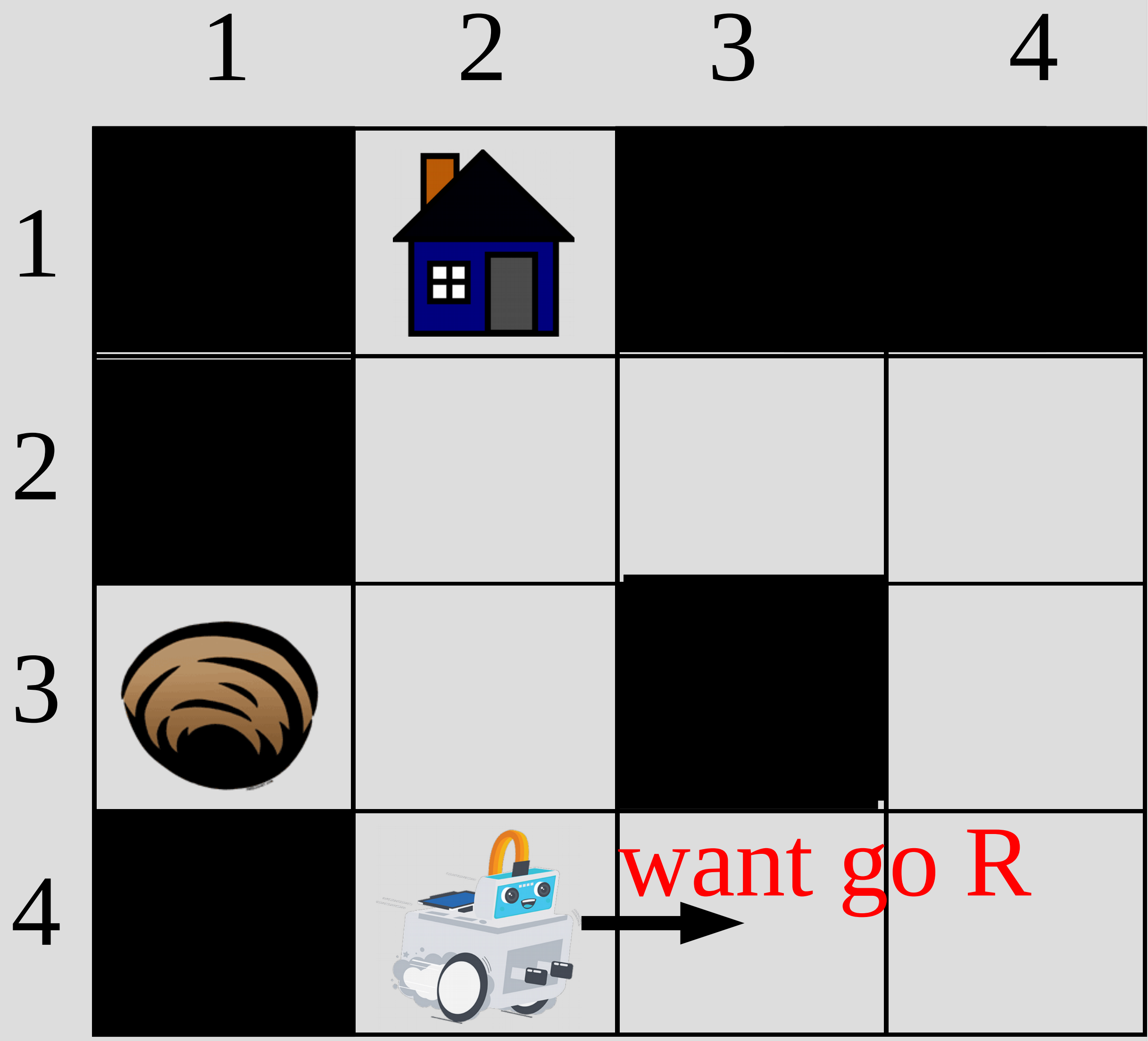


Markov Decision Process

When the robot tries to move, 80% of the time it ends up where it wants to go

10% it will end up 90 degrees off

Wall = no move

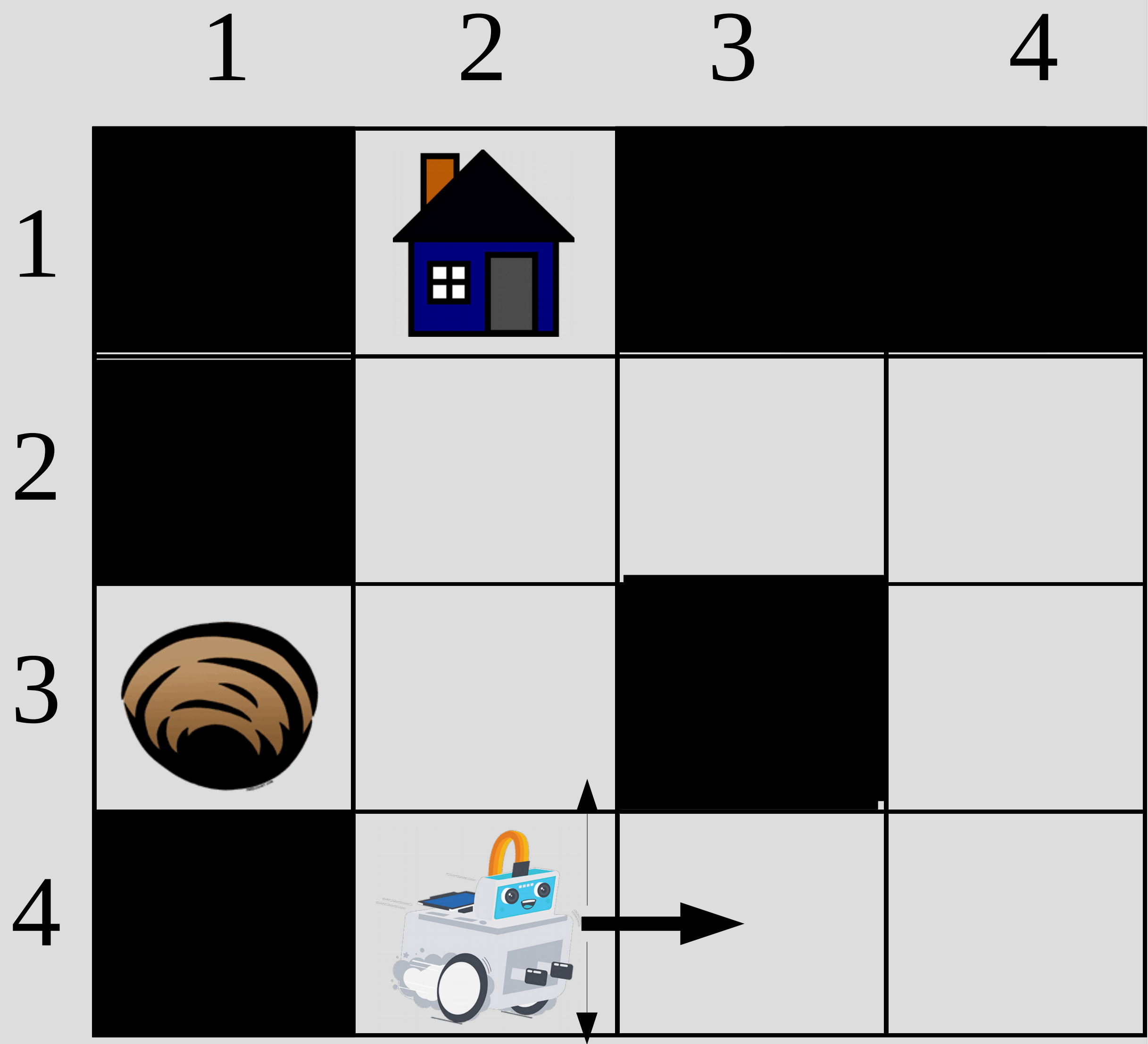


Markov Decision Process

When the robot tries to move, 80% of the time it ends up where it wants to go

10% it will end up 90 degrees off

Wall = no move

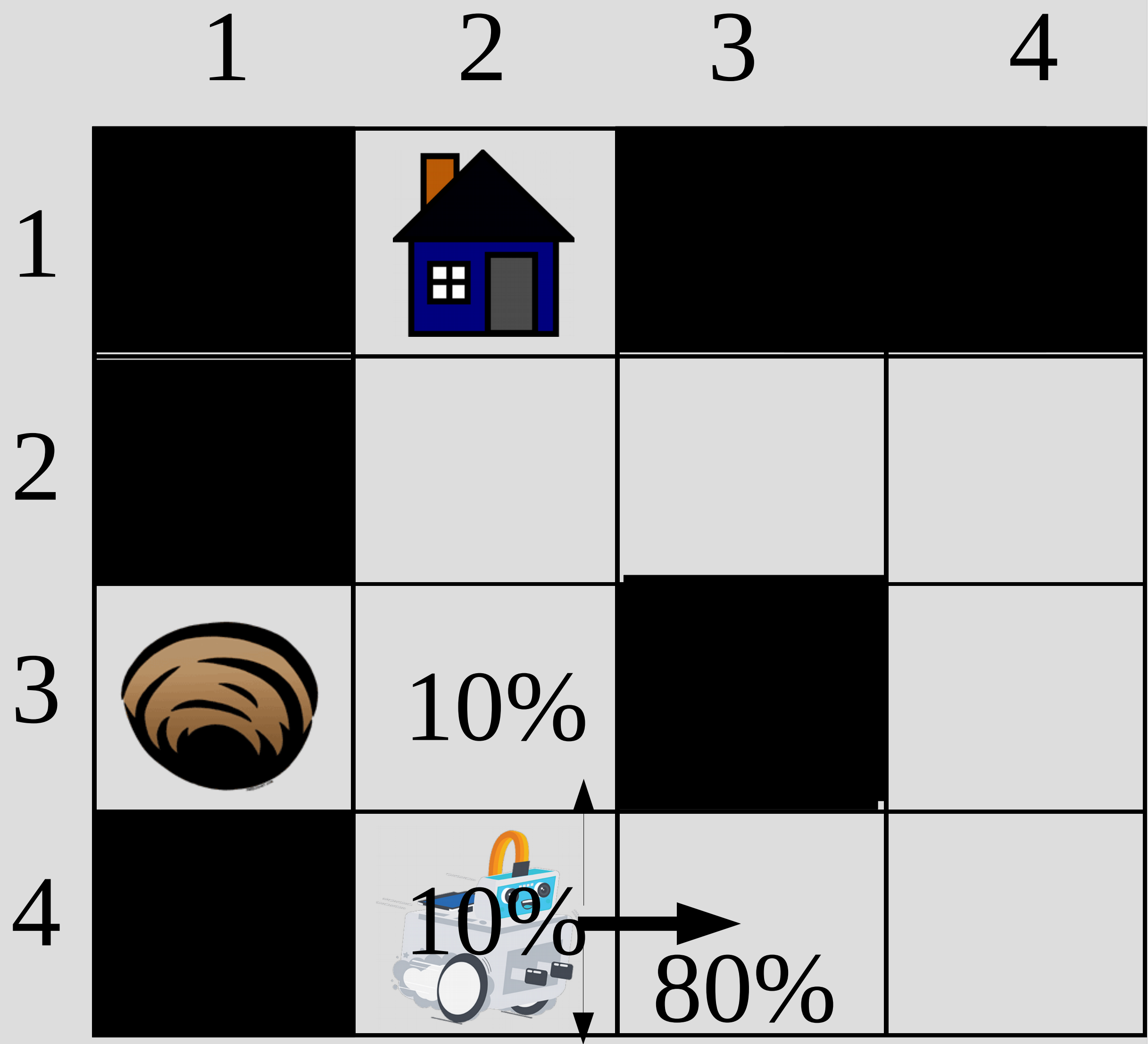


Markov Decision Process

When the robot tries to move, 80% of the time it ends up where it wants to go

10% it will end up 90 degrees off

Wall = no move



Markov Decision Process

Given this setup, we want to find the best sequence of actions that will reach an end with the most utility

Any sequence of actions we call a “policy” and represented as π

We will use a * to represent the “optimal” so π^* would be the optimal/best policy, which is what we want to find

Markov Decision Process

We assume that you get some “reward” for landing in a state after an action, R(s)

In our example, it costs the agent “1” to move, so R(s) = -1 for all states (except the ends)

Since our policy, π, is the sequence of action to go from start to end, we need to evaluate the utility of ending up in a sequence of states

MDP Utilities

In other words, we need to find some number:

$U(s_0, s_1, s_2, s_3, \ldots) = ???$

If the agent started at s_0, then moved to s_1, ...

Consider two sequences of states:

$S = [s_0, s_1, s_2, s_3, \ldots]$, $S' = [s_0', s_1', s_2', s_3', \ldots]$

If you prefer S over S', but $s_0 = s_0'$, what do you think you can conclude?

MDP Utilities

$S = [s_0, s_1, s_2, s_3, ...]$, $S' = [s_0', s_1', s_2', s_3', ...]$
Suppose you prefer S over S' and $s_0 = s_0'$

... If you can then conclude that you would prefer $[s_1, s_2, s_3, ...]$ over $[s_1', s_2', s_3', ...]$

We call this a <u>stationary</u> preference
(and it has some large implications)

MDP Utilities

If you have a stationary preference, then there are only two valid utility functions:

Additive:

$$U(s_0, s_1, s_2, ...) = R(s_0) + R(s_1) + R(s_2) + ...$$

Discounted:

$$U(s_0, s_1, s_2, ...) = R(s_0) + \gamma \cdot R(s_1) + \gamma^2 \cdot R(s_2) + ...$$

... where $0<\gamma<1$

MDP Utilities

We will assume the “discounted” version, as the math is actually easier

However, if you use the “additive” rewards, there are similar results (with assumptions)

If we have:

$$U(s_0, s_1, s_2, ...) = R(s_0) + \gamma \cdot R(s_1) + \gamma^2 \cdot R(s_2) + ...$$

... what does this look like?

MDP Utilities

A geometric series!

$$U(s_0, s_1, s_2, ...) = R(s_0) + \gamma \cdot R(s_1) + \gamma^2 \cdot R(s_2) + ...$$

Let R_{max} be the highest reward of any possible state, then:

$$\begin{aligned} U(s_0, s_1, s_2, ...) &\leq R_{max} + \gamma \cdot R_{max} + \gamma^2 \cdot R_{max} + ... \\ &\leq \sum_{i=0}^{\infty} \gamma^i \cdot R_{max} \\ &\leq R_{max}/(1-\gamma) \end{aligned}$$

Thus utility is always finite (if reward finite)

MDP Utilities

To compare policies, we can calculate them as:

$$U^{\pi}(s_0, s_1, s_2, \ldots) = E[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

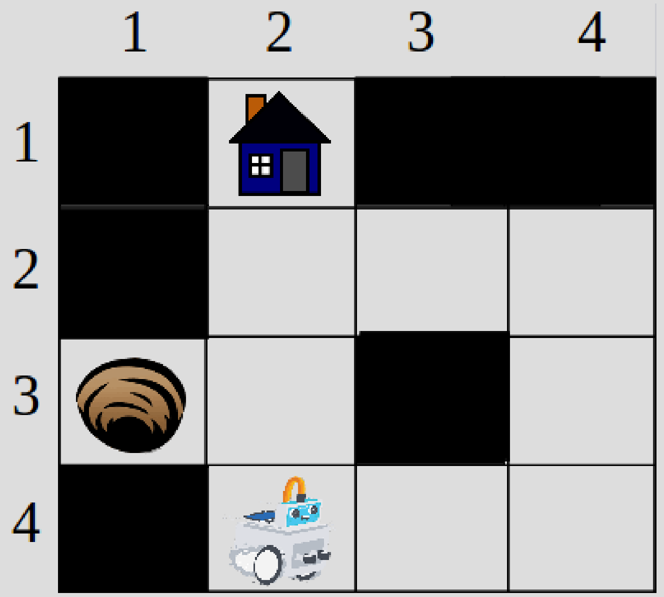
We can use a similar definition as "maximum expected utility" from last time:

$$\pi^*(s) = \underset{a \in Actions}{\arg\max} \sum_{s'} P(s'|s,a) \cdot U(s')$$

best action from state s

80%, 10%, 10%

add over possible states you end up in



MDP Utilities

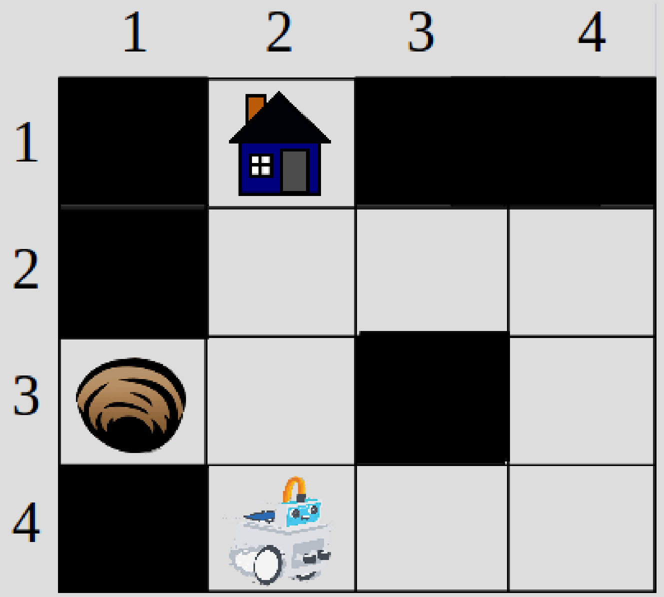
To compare policies, we can calculate them as:

$$U^{\pi}(s_0, s_1, s_2, ...) = E[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

We can use a similar definition as “maximum expected utility” from last time:

$$\pi^*(s) = \arg\max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

... How do you find U(s’)?



Bellman Equations

Turns out, you can represent state utility in terms of other states (Bellman equation):

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

So for example, the utility of (2,2):

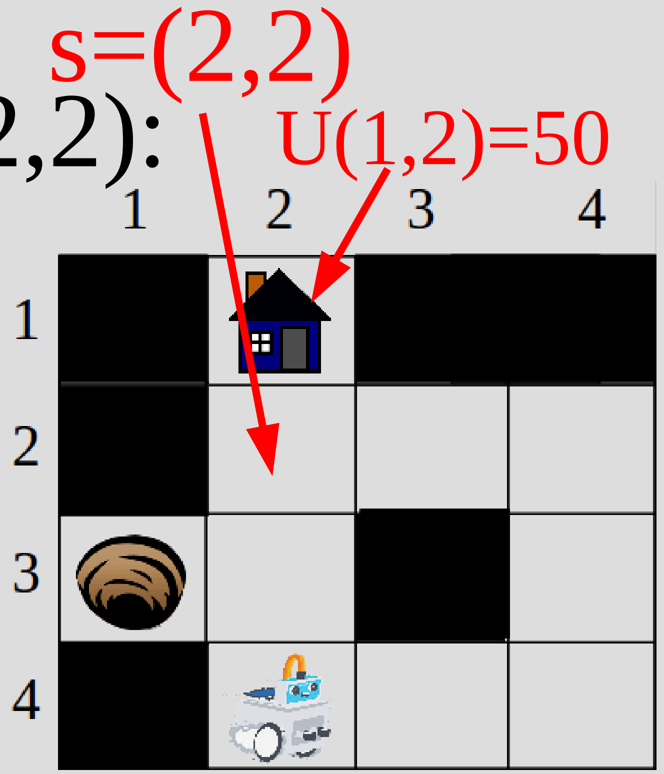
U(2,2) = -1 + γ * max of:

a=Up: $0.8 \cdot 50 + 0.1 \cdot U(2,2) + 0.1 \cdot U(2,3)$

a=D: $0.8 \cdot U(3,2) + 0.1 \cdot U(2,3) + 0.1 \cdot U(2,2)$

a=L: $0.8 \cdot U(2,2) + 0.1 \cdot U(3,2) + 0.1 \cdot 50$

a=R: $0.8 \cdot U(2,3) + 0.1 \cdot 50 + 0.1 \cdot U(3,2)$



Bellman Equations

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

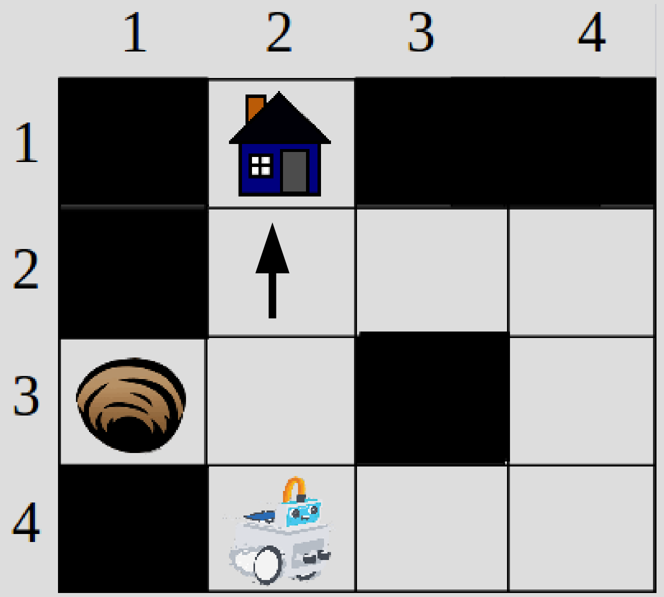
Assuming you should go "up" from (2,2): (let γ=0.9)

$$U(2,2) = -1 + 0.9 \cdot (0.8 \cdot 50 + 0.1 \cdot U(2,2) + 0.1 \cdot U(2,3)$$

Then some algebra:

$$U(2,2) = \frac{35+0.09 \cdot U(2,3)}{0.91}$$

... What is U(2,3) assuming best answer is going "left"?



Bellman Equations

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

$$U(2,2) = \frac{35+0.09 \cdot U(2,3)}{0.91}$$

U(2,3) going “left” is:

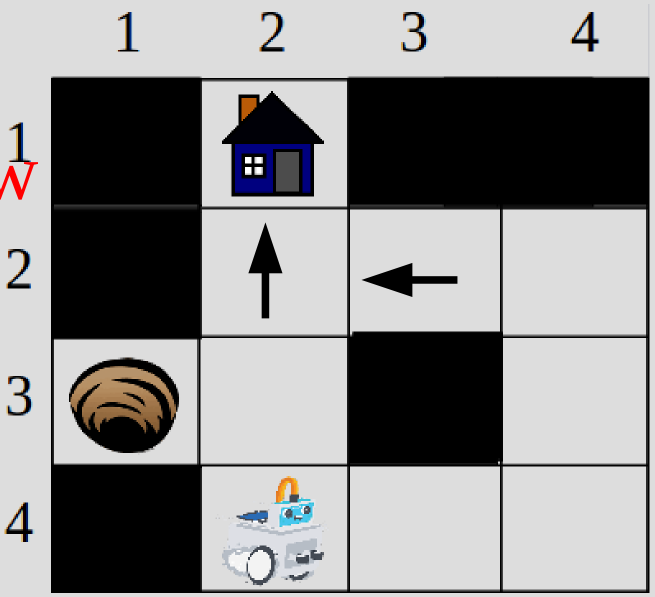
$$U(2,3) = -1 + 0.9 \cdot (0.8 \cdot U(2,2) + 0.1 \cdot U(2,3) + 0.1 \cdot U(2,3)$$

... algebra ...

$$U(2,3) = \frac{-1+0.72 \cdot U(2,2)}{0.82}$$

assumed we knew which actions

Could solve this as sys. linear equations, but we cheated



Value Iteration

This is problematic in general as we have to know the utility of the surrounding states to know our utility

... but these surrounding states need our utility to be know! (Recursive logic...)

We have actually seen something like this before... what was it?

Value Iteration

This is problematic in general as we have to know the utility of the surrounding states to know our utility

... but these surrounding states need our utility to be know! (Recursive logic...)

We have actually seen something like this before... what was it?
Gibbs sampling!

Value Iteration

This will actually be one of our favorite tricks in the course: Both A and B unknown

Step 1: Assume you know A and solve for B
Step 2: Use answer for B to solve for A
... repeat above a lot

So we can actually just assign random utilities and keep using the Bellman equations to get new “estimates” for the states

Value Iteration

Since we can pick any number for utility, let’s assume non-end states are zero at start:

We would then compute U(2,2) as:

$$\mathrm{U}(2{,}2) = -1 + 0.9 \cdot \max \text{ of:}$$

$$0.8 \cdot 50 + 0.1 \cdot 0 + 0.1 \cdot 0 = 40$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$0.8 \cdot 0 + 0.1 \cdot 50 + 0.1 \cdot 0 = 5$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 50 = 5$$

	1	2	3	4
1	■	50	■	■
2	■	0	0	0
3	-50	0	■	0
4	■	0	0	0

So, $\mathrm{U}(2{,}2) = -1 + 0.9 \cdot 40 = 35$

Value Iteration

We would then go through and compute all the rest of the utilities **before** updating them

So for spot (2,3):

$$U(2,3) = -1 + 0.9 \cdot \max:$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$U(2,3) = -1$$

	1	2	3	4
1	■	50	■	■
2	■	0	0	0
3	-50	0	■	0
4	■	0	0	0

Value Iteration

This algorithm is called value iteration as we repeatedly update the utility values

After going through all 8 “unknown” utilities you should get the following:

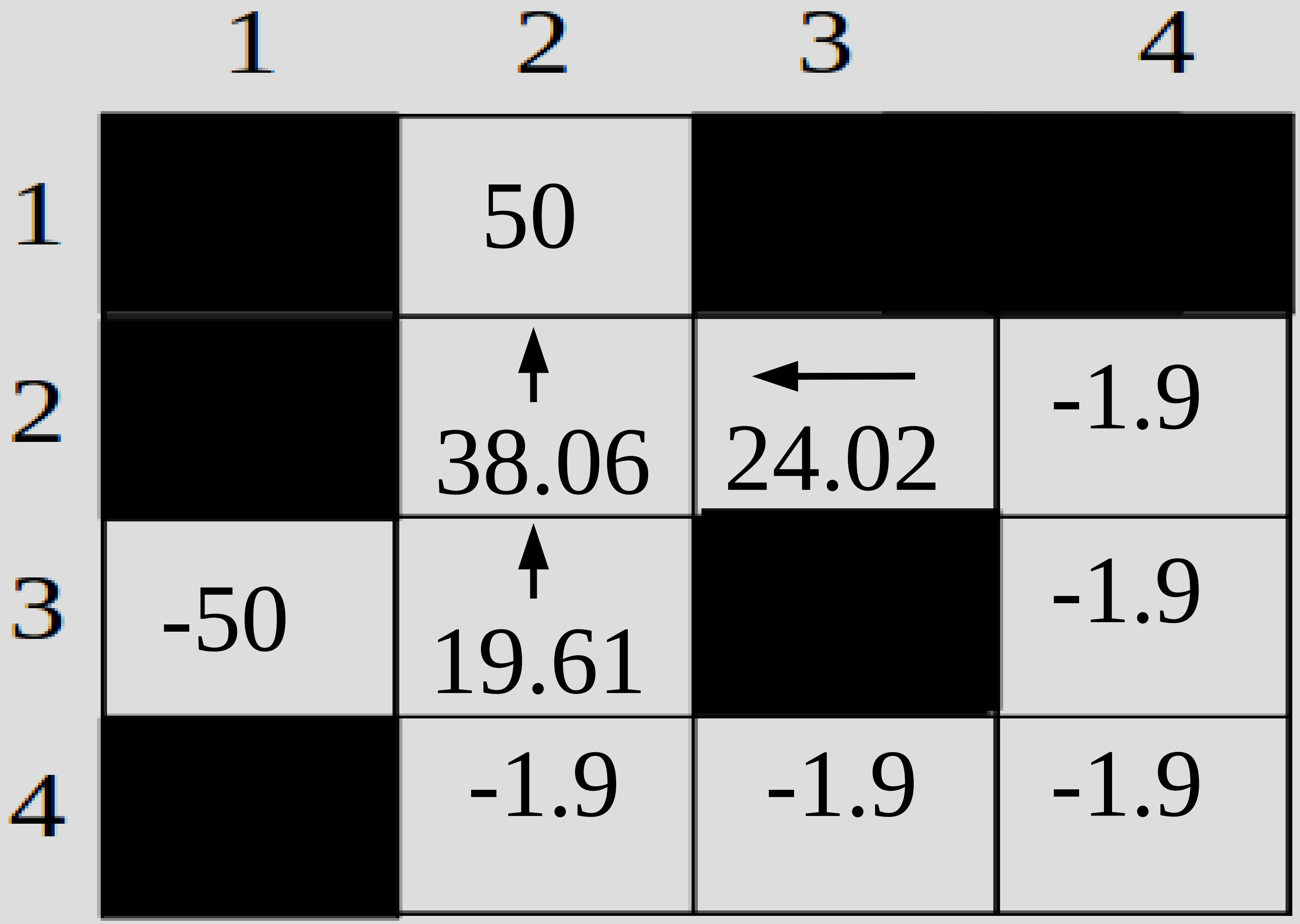
Do one more iteration of all 8 utilities:

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

	1	2	3	4
1	■	50	■	■
2	■	35	-1	-1
3	-50	-1	■	-1
4	■	-1	-1	-1

Value Iteration

You should get on second iteration:



Value Iteration

After doing this a bunch, you should get:

	1	2	3	4
1	■	50	■	■
2	■	↑ 41.99	← 35.65	← 29.55
3	-50	↑ 27.18	■	↑ 24.73
4	■	↑ 22.21	← 18.28	↑ 20.27

Value Iteration Convergence

You simply repeat this process until the numbers "converge" (i.e. stop changing much)

In fact, it is both guaranteed to converge always and within a bounded amount

It has been shown that if you have two sets of utilities U_0 and U_0', then use Bellman eq.s to get U_1 and U_1', then: $||U_0 - U_0'||\gamma \geq ||U_1 - U_1'||$

the ∞-norm (i.e. abs max), $||\ [1,-2] - [2, 4]\ || = 6$

Bellman Equations

This means no matter what two sets of utilities you have, they will become "closer" after applying the Bellman update

This is called a contraction and has a nice property you will always converge to a unique solution (when $\gamma<1$)

We can also notice that if U* are the correct utilities, applying Bellman will not change

Bellman Equations

Thus, if U_i is after applying the Bellman eq. i times: $||U_0 - U^*||\gamma^i \geq ||U_i - U^*||$

But we have a “worst case” utility of:
$U(s_0, s_1, s_2, ...) \leq R_{max}/(1-\gamma)$

Since the difference can at most double this:
$\gamma^i \frac{2 \cdot R_{max}}{1-\gamma} \geq ||U_i - U^*||$

Bellman Equations

If we want to guarantee we are within ε of the optimal solution, we can then find N:

$$\epsilon \geq \gamma^N \cdot \frac{2 \cdot R_{max}}{1-\gamma} \geq ||U_i - U^*||$$

... as each update contracts/shrinks by γ and we start at most $2*R_{max}/(1-\gamma)$ away from opt.

Also do not need to wait for utility to converge as policy just needs to find best action

Bellman Equations

The Bellman equations find some “utility” for each state that you then find best actions

... but our original goal was:

$$U^\pi(s_0, s_1, s_2, ...) = E[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

This is similar to the Bellman equations, but Bellman only look one step ahead... while our goal is start to end

Bellman Equations

This is actually not a problem, as if after doing “i” Bellman updates, you have:

$$||U_i - U^*|| < \epsilon$$

... then you are guaranteed (worse case) to be within a a bound of the optimal policy:

$$||U^{\pi_i} - U^*|| < \frac{2 \cdot \epsilon \cdot \gamma}{1 - \gamma}$$

We can the above the “policy loss”