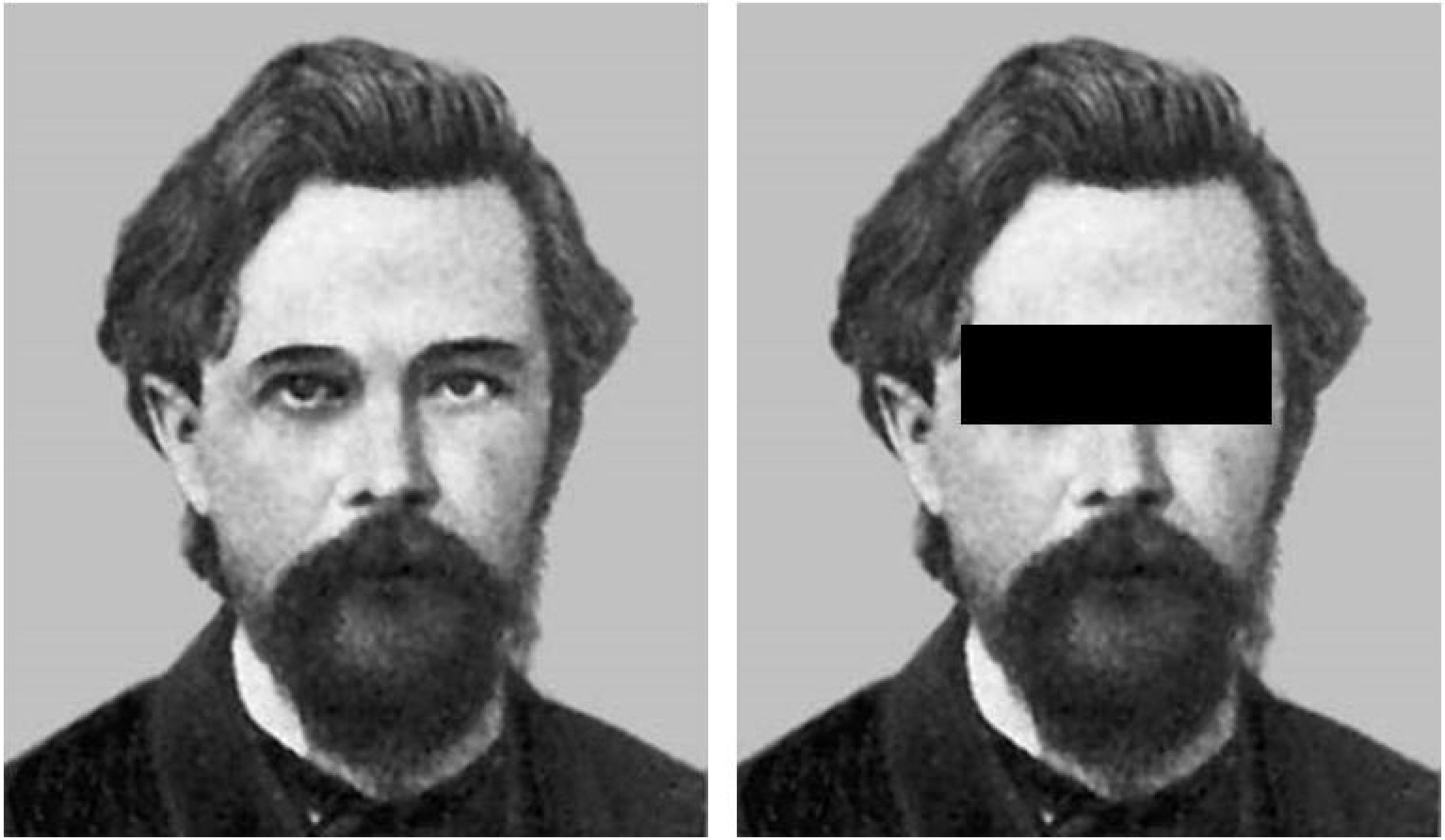
Multi-variable optimization

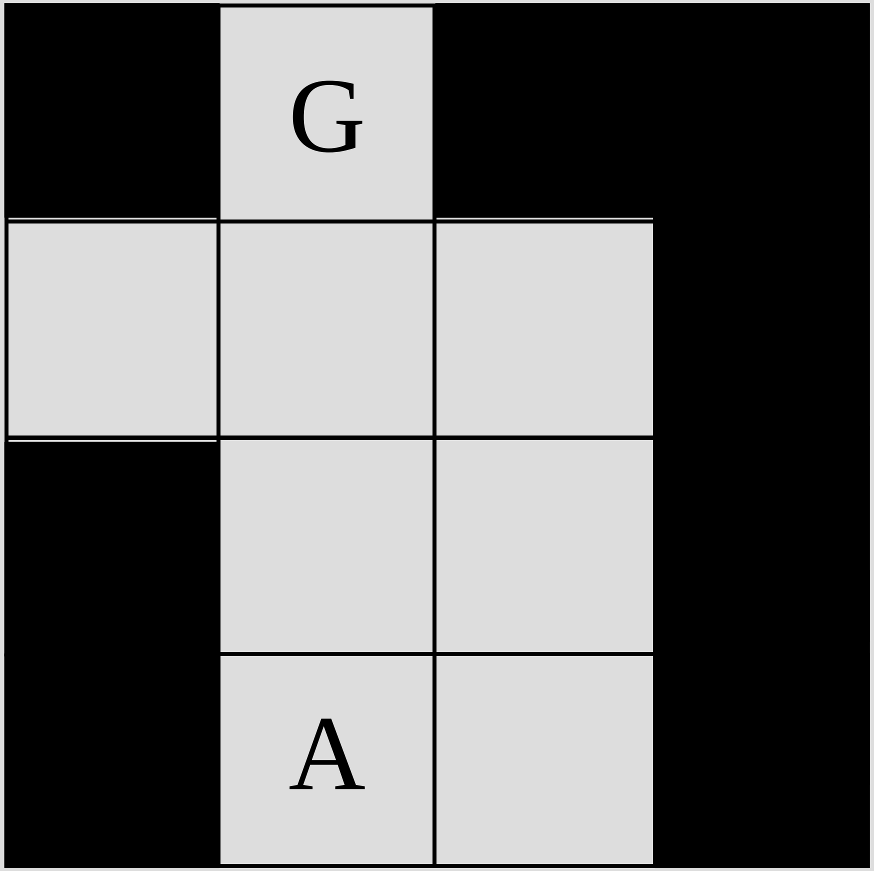


Markov | Hidden Markov

Planning

Actions: Agent face N, S, E, W

Agent movement:
80% forward
10% left
10% right
(e.g. agent wants to go E from current loc 80% goes E, 10% N 10% nowhere)



Planning

I assumed you know how to get to the goal as fast as possible, but how?

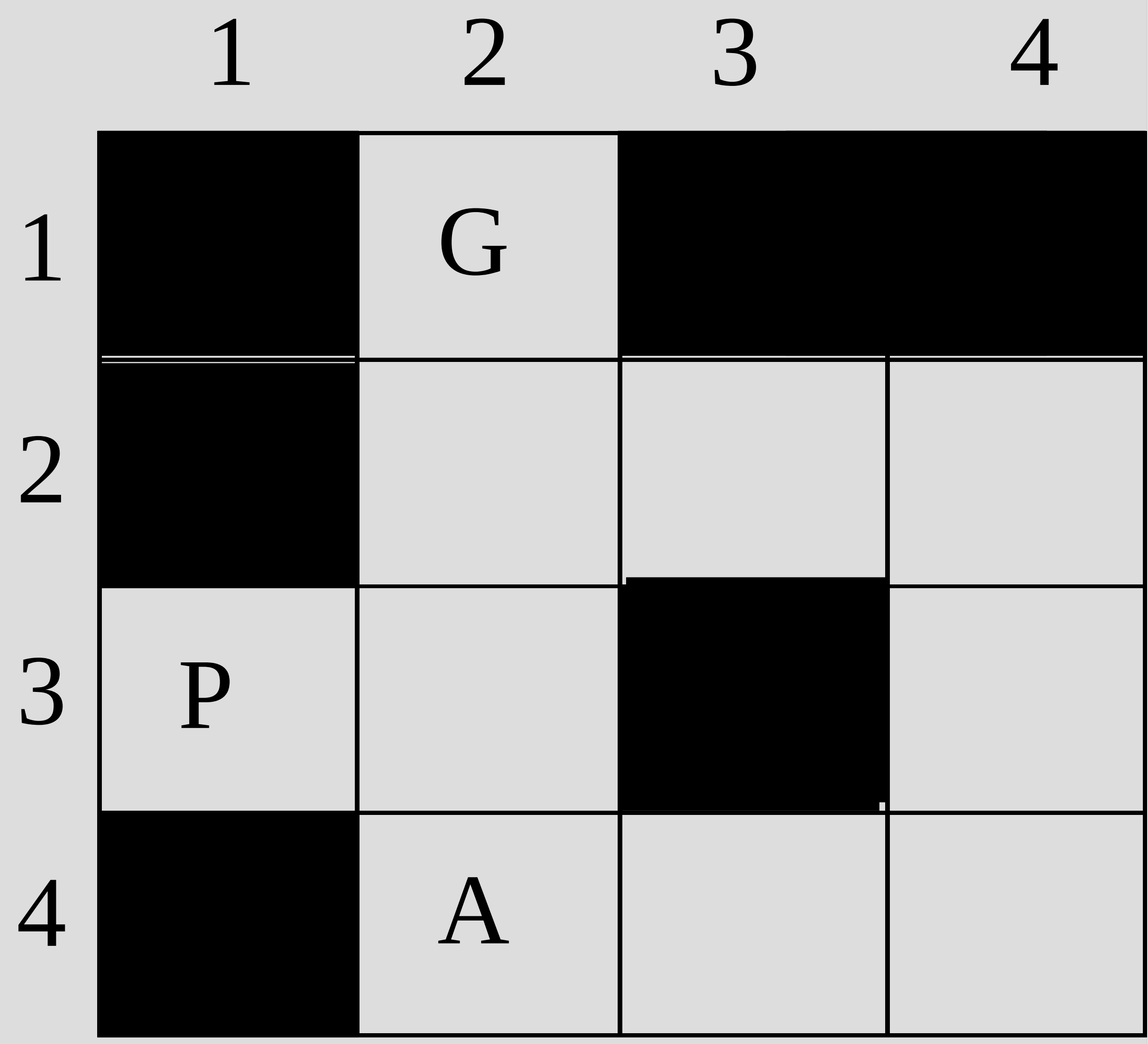
Formally, we need to assign costs to each action (or state)

We will assume moving has a cost of 1 (though we will see how to generalize this)

Planning

G = +50 (end)
P = -50 (end)
All other = -1
(i.e. -1 for movement)

Goal: maximize score before reaching end



Planning

What is the cost of going to another state?

Planning

What is the cost of going to another state?

Let's start a bit easier...

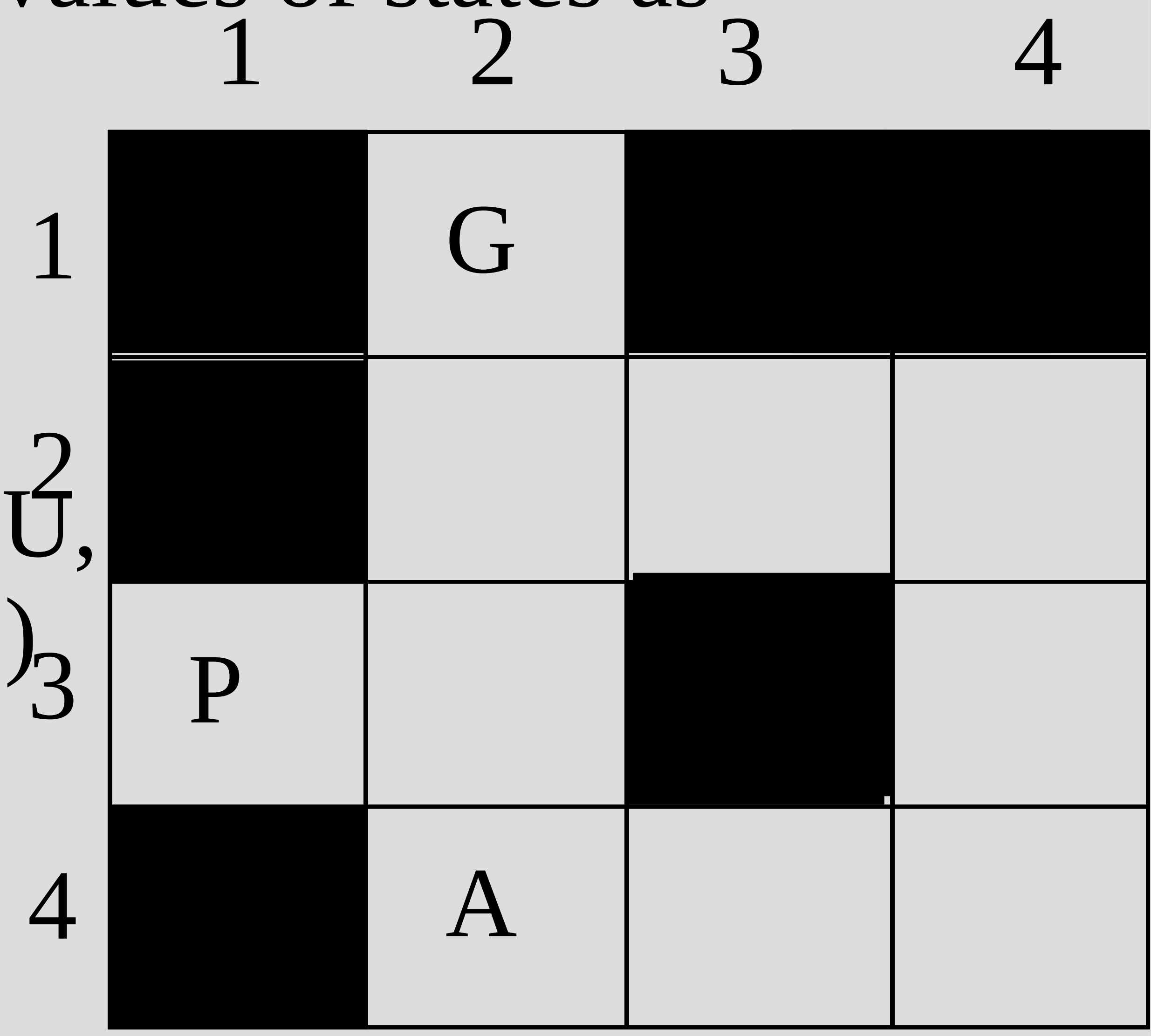
Assume the agent always moves in the direction that it wants

We can then find the “best action” starting from the goal and working backwards

Policy iteration

We will frame the values of states as relationships to each other

Value (2,2):
argmax(-1+γ*(Go U, Go D, Go L,Go R))
=-1+γ*(Go U)
=-1+γ*50



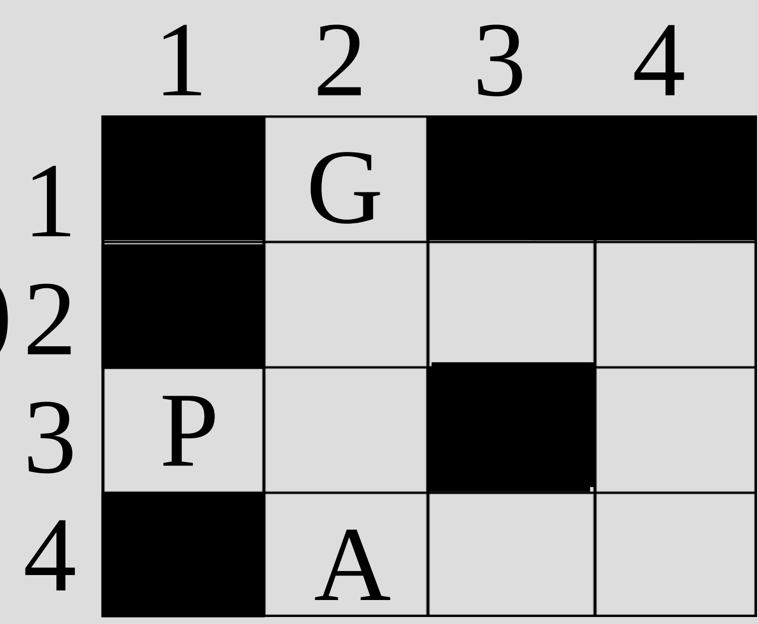
Policy iteration

Now that we know the value of (2,2), we can find the values of (2,3) and (3,2) (assuming we know how to find the best action)

However, if we re-introduce the random movement:

Value(2,2)=argmax(-1+ γ*(Go U, Go D, Go L, Go R))

= -1 + γ*(E(Go U))



Expected value

The expected value of a random variable (i.e. values with associated probabilities) is:

$$\sum_{\text{all value-probability pairs}} \text{probability} \cdot \text{value}$$

For example: Let's flip a fair coin. If it is heads, I win $10. If it is tails, I lose $5.

Random variable X = (p(heads)=0.5 : 10)
(p(tails)=0.5 : -5)

E(X) = 0.5*10 + 0.5*-5 = 2.5

Expected value

Example 2: A fair dice...

probability : value

Random variable X = (p(roll 1)=1/6 : 1)

(p(roll 2)=1/6 : 2)

(p(roll 3)=1/6 : 3)

(p(roll 4)=1/6 : 4)

(p(roll 5)=1/6 : 5)

(p(roll 6)=1/6 : 6)

E(X) = 1/6 * 1 + 1/6 * 2 + 1/6 * 3 + 1/6 * 4 + 1/6 * 6 = 3.5

Policy iteration

$V(2,2)=\text{argmax}(1+\gamma*(\text{Go U},\text{Go D},\text{Go L},\text{Go R}))$

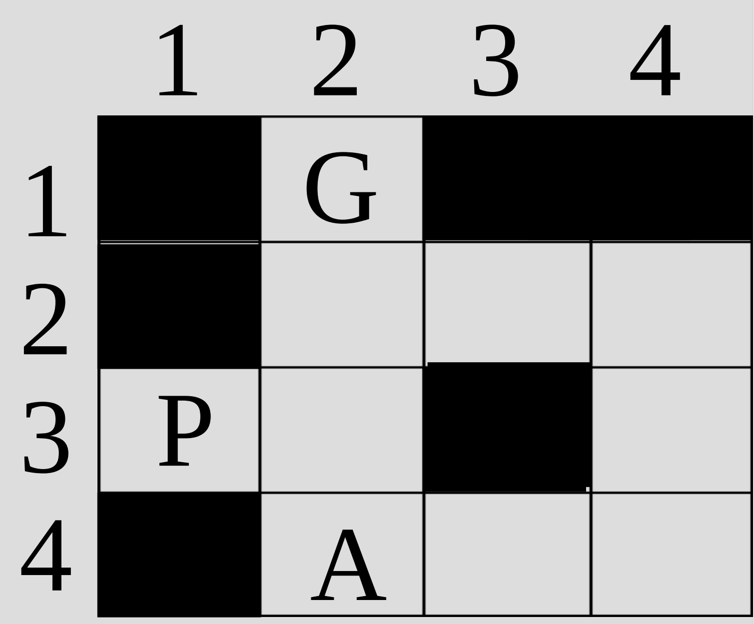
$=-1+\gamma*(E(\text{Go U}))$

$=-1+\gamma*(0.8*V(1,2) + 0.1*V(2,2) + 0.1*V(2,3))$

$=-1+\gamma*(0.8*50 + 0.1*V(2,2) + 0.1*V(2,3))$

But wait... value of (2,2) depends on value of (2,3)

Value (2,3) depends on value (2,2)... (system of lin. eq.)

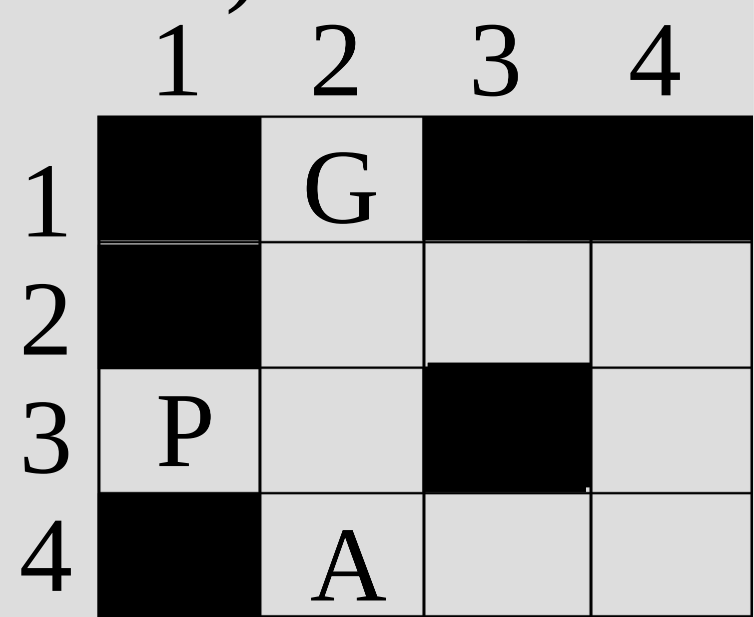


Policy iteration

However, we have been assuming we know what the best action is (finding the max)

Finding the best action is easy if we know the values of each square (but we don't)

Finding the values of each square is easy if we know the best actions (but we don't)



Policy iteration

This type of problem happens a lot:

- If you knew A, you could solve for B
- If you knew B, you could solve for A
- Yet you know neither A or B

Solution: Initialize A to guess (or random)

1. Solve for B with fixing A
2. Solve for A with fixing B
3. Repeat above 2 until convergence

Policy iteration

We call this method <u>policy iteration</u>

Initialize the values in grid with with deterministic movement

	50		
	49	48	47
-50	48		46
	47	44	45

Then we find best action for each square, we use this equation:

$$\underset{a \in \text{actions}}{\operatorname{argmax}} \sum_{s' \text{ from } s} P(s,a,s') \cdot (R_a(s,s') + \gamma V(s'))$$

(called Bellman equation)

Policy iteration

We call this method policy iteration

Initialize the values in grid with with deterministic movement

	50		
	49	48	47
-50	48		46
	47	44	45

Then we find best action for each square, we use this equation:

value of state going to

$$\underset{a \in \text{actions}}{\operatorname{argmax}} \sum_{s' \text{ from } s} P(s,a,s') \cdot \left(\boxed{R_a(s,s')} + \gamma \boxed{V(s')}\right)$$

(called Bellman equation)

move cost=-1

Find best action

$$\underset{a \in \text{actions}}{\operatorname{argmax}} \sum_{s' \text{ from } s} P(s,a,s') \cdot (R_a(s,s') + \gamma V(s'))$$

Consider the agent's starting square (the 47 on bottom row)

	50		
	49	48	47
-50	48		46
	47	44	45

Find best action (above eq.):

V(2,4) = argmax(Go U, D, L, R)

=argmax(-1 + γ*(0.8*[U] + 0.1*[L] + 0.1*[R]),

-1 + γ*(0.8*[D] + 0.1*[R] + 0.1*[L]),

-1 + γ*(0.8*[L] + 0.1*[D] + 0.1*[U]),

-1 + γ*(0.8*[R] + 0.1*[U] + 0.1*[D]))

Find best action

	50		
	49	48	47
-50	48		46
	47	44	45

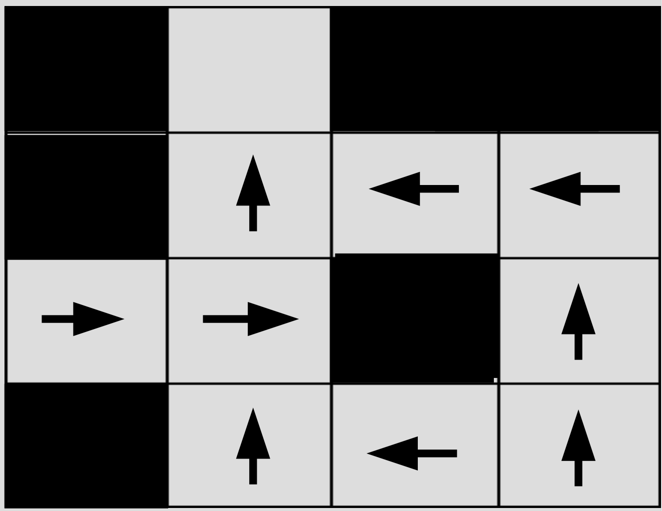
From the 47 (agent start):
[U] = 48, [L] = 47 = [D],
[R] = 44, let $\gamma=1$ (typically <1)

$$\begin{aligned}\text{argmax}(&-1 + (0.8*48 + 0.1*47 + 0.1*44),\\ &-1 + (0.8*47 + 0.1*44 + 0.1*47),\\ &-1 + (0.8*47 + 0.1*47 + 0.1*48),\\ &-1 + (0.8*44 + 0.1*48 + 0.1*47))\end{aligned}$$

=argmax(46.5, 45.7, 46.1, 43.7)
=Go U

Find values

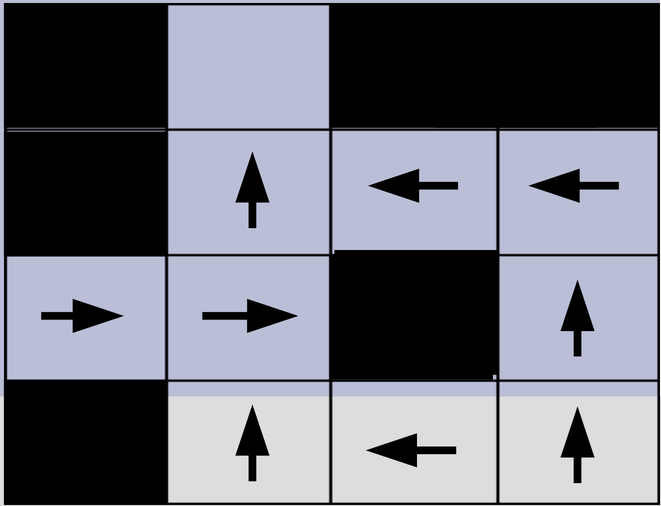
We repeat this process for every
square and get a "best action"
grid



We then use the Bellman eq.
to get system of linear equations
(each state is 1 unknown value with 1 equation)

(see next slide)

Find values



$V(2,1)= +50$ (goal)

$V(2,2)=-1+.8*V(1,2)+ 0.1*V(2,2)+0.1*V(2,3)$

$V(2,3)=-1+0.8*V(2,2)+0.1*V(2,3)+0.1*V(2,3)$

$V(2,4)=-1+0.8*V(2,3)+0.1*V(3,4)+0.1*V(2,4)$

$V(3,1)= -50$ (pit)

$V(3,2)=-1+0.8*V(3,2)+0.1*V(2,2)+0.1*V(4,2)$

$V(3,4)=-1+0.8*V(2,4)+0.1*V(3,4)+0.1*V(3,4)$

$V(4,2)=-1+0.8*V(3,2)+0.1*V(4,2)+0.1*V(4,3)$

$V(4,3)=-1+0.8*V(4,2)+0.1*V(4,3)+0.1*V(4,3)$

$V(4,4)=-1+0.8*V(3,4)+0.1*V(4,3)+0.1*V(4,4)$

Find values

Solving that mess gives you these new values:

	50		
	48.59	47.34	45.93
-50	37.18		44.68
	35.78	34.53	42.44

At this point, you would again find the best move for the values above and repeat until the actions do not change

In-class activity

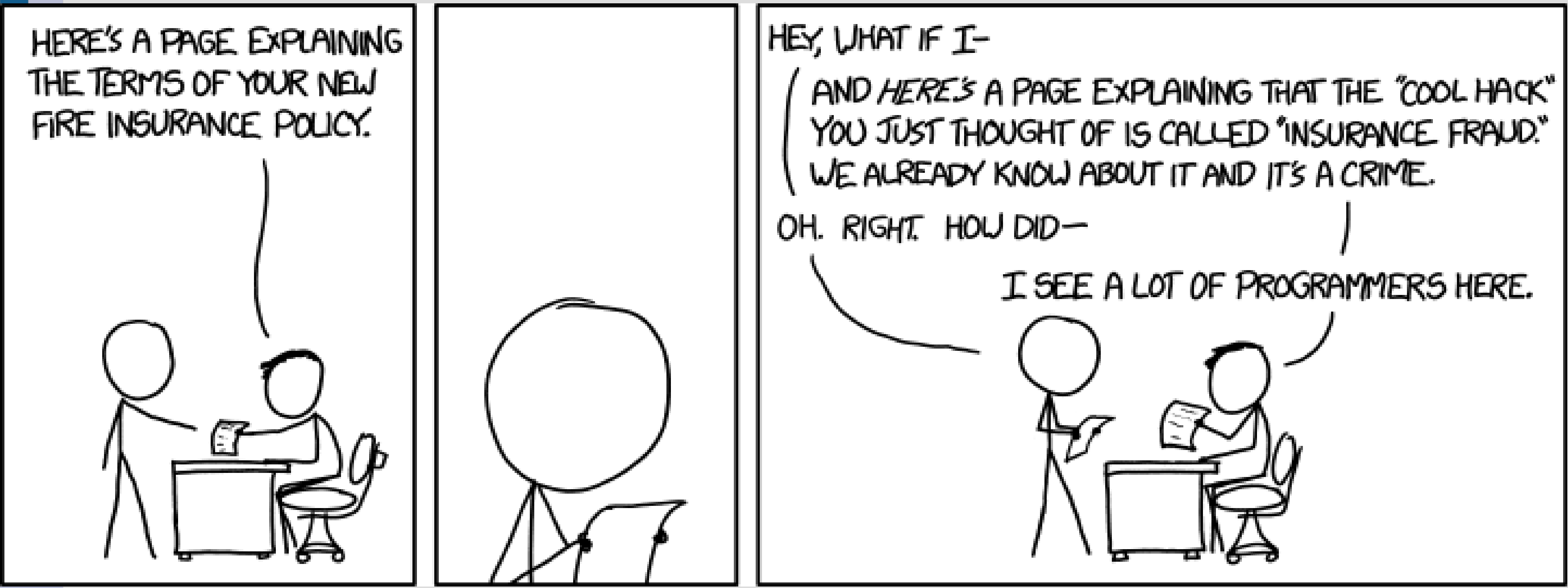
$$\operatorname*{argmax}_{a \in \text{actions}} \sum_{s' \text{ from } s} P(s,a,s') \cdot (R_a(s,s') + \gamma V(s'))$$

1. Find the best actions for these values
2. If any actions changed, setup sys. lin. eq.
   (otherwise you know best paths)

	50		
	48.59	47.34	45.93
-50	37.18		44.68
	35.78	34.53	42.44

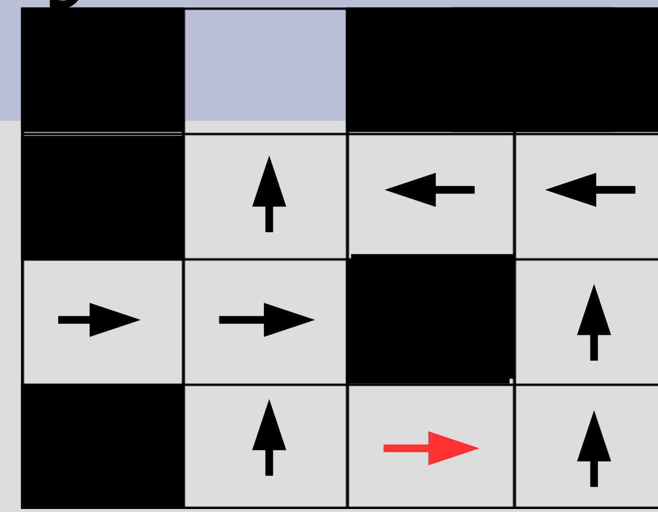
In-class activity

DON'T CHEAT AND LOOK AT ANSWERS BELOW!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!



In-class activity

1.



2.

	50		
	48.59	47.34	45.93
-50	37.93		44.68
	37.28	42.03	43.28

In-class activity

After 1 more system of linear equations, the actions stabilize and we find that we should go around the long way to the goal

(i.e. pit is too dangerous)

The starting node will have a value of 40.6526, so it will take approximately 9.34743 steps to reach the goal (optimally)