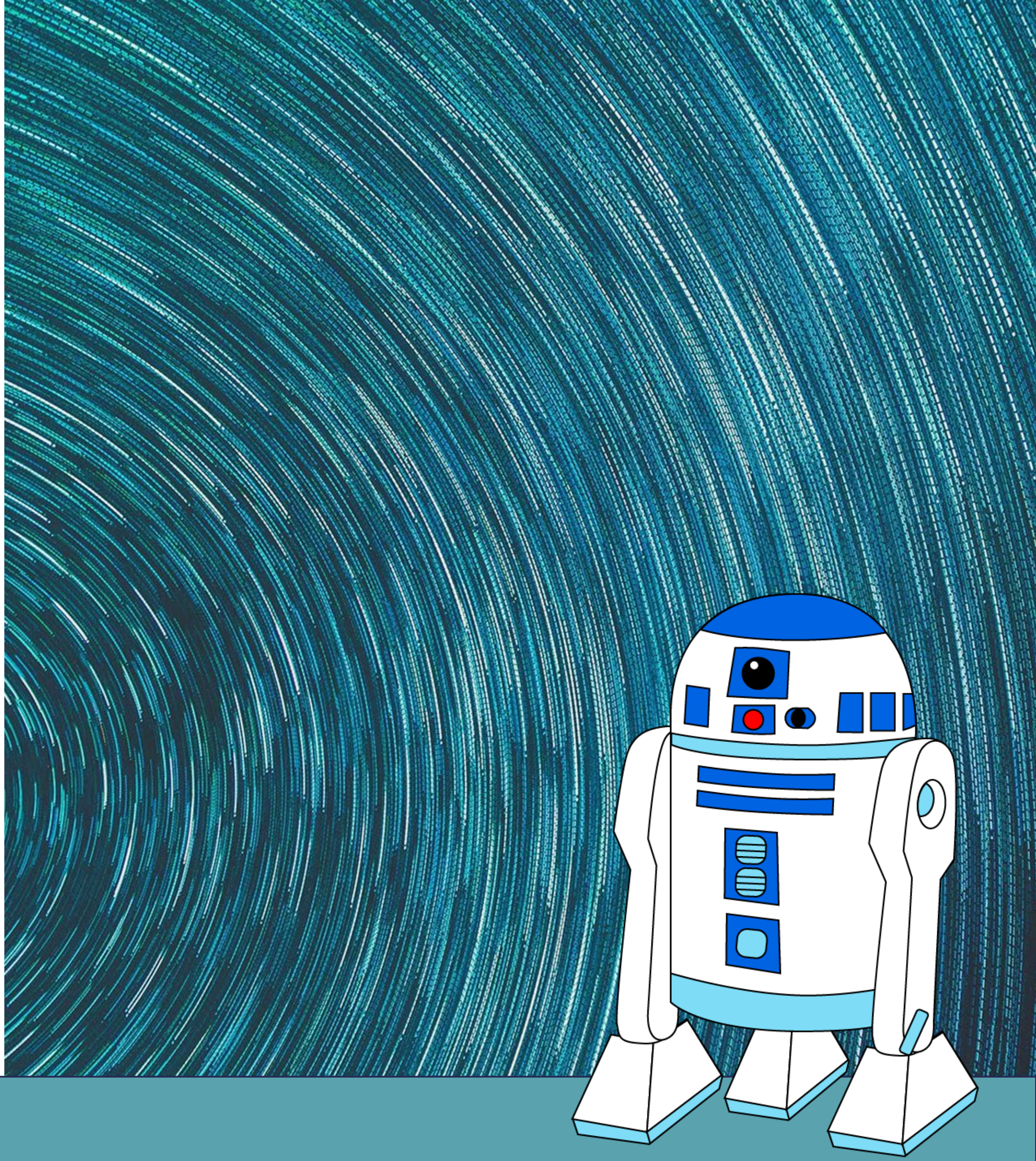
CIS 521:
ARTIFICIAL INTELLIGENCE

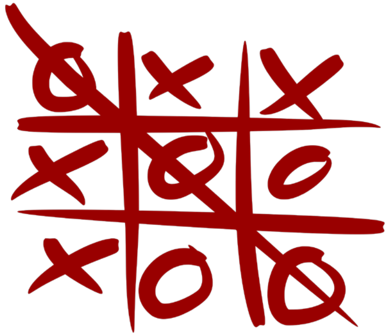
Expectimax and Utilities

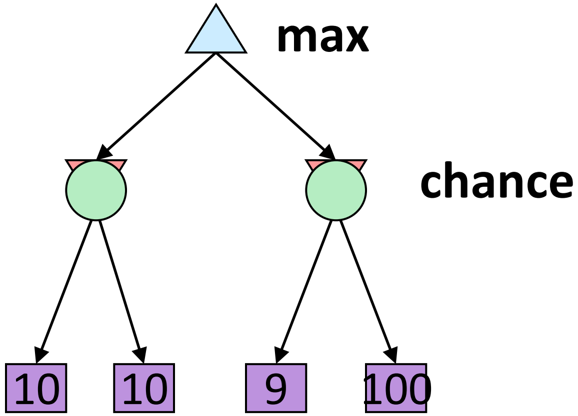
Harry Smith

Many of today's slides are courtesy of Dan Klein and
Pieter Abbeel of University of California, Berkeley



Uncertain Outcomes





Idea: Uncertain outcomes controlled by chance, not an adversary!

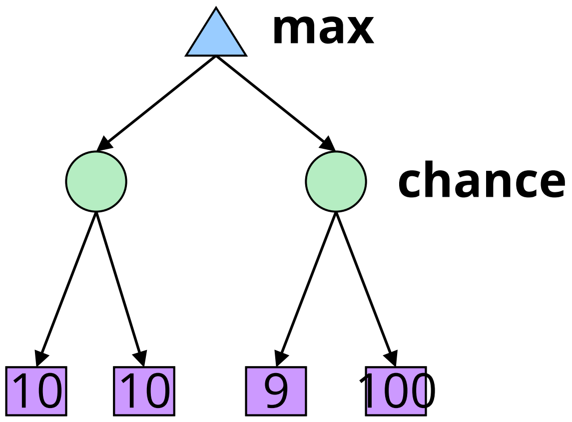
Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the opponent isn't optimal
  - Actions can fail: when moving a robot, wheels might slip

- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children

- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes



Expectimax Pseudocode

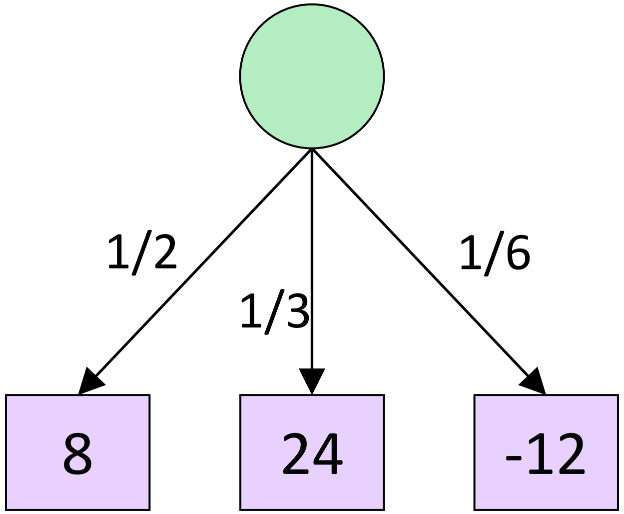
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

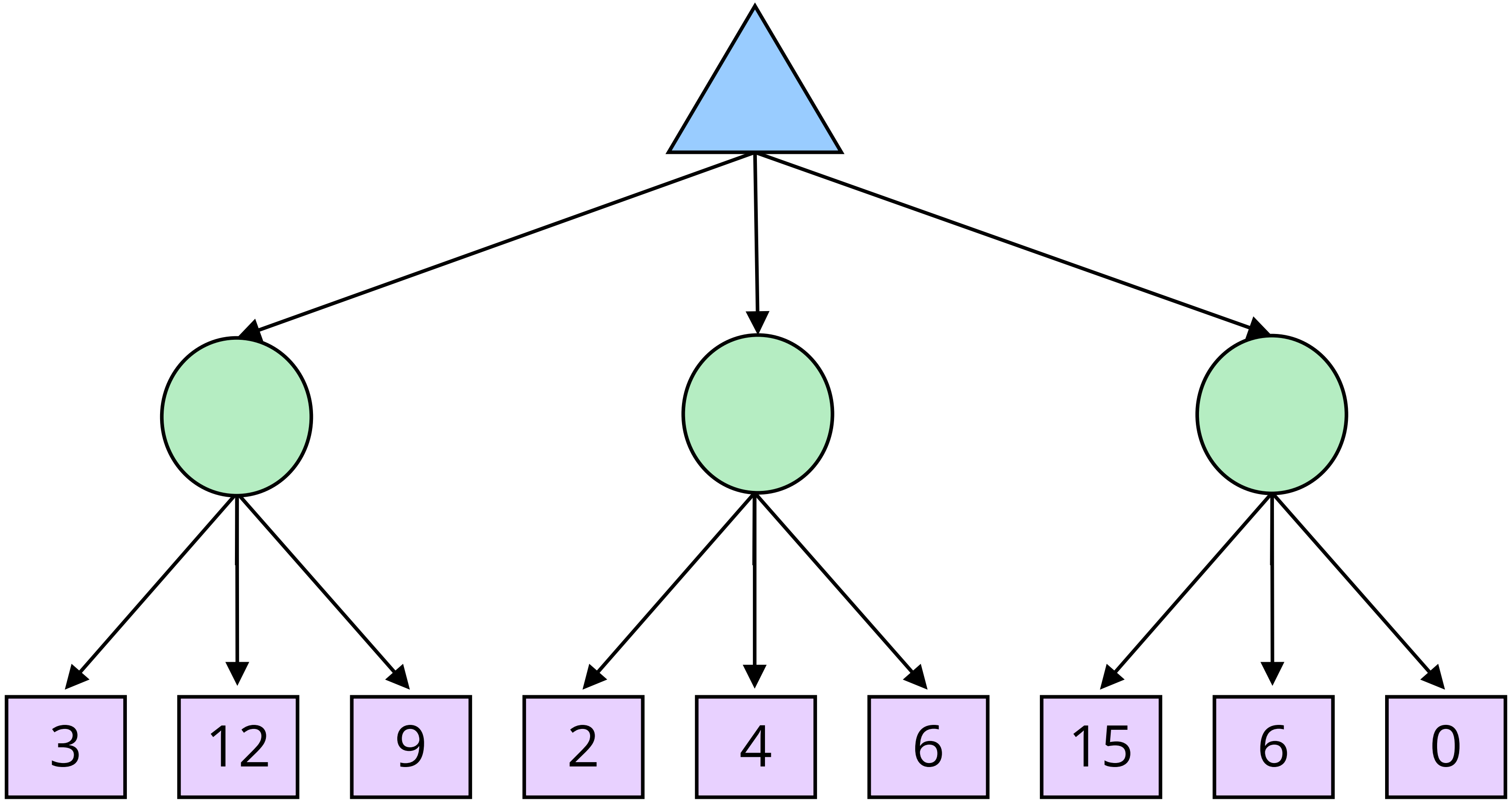
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

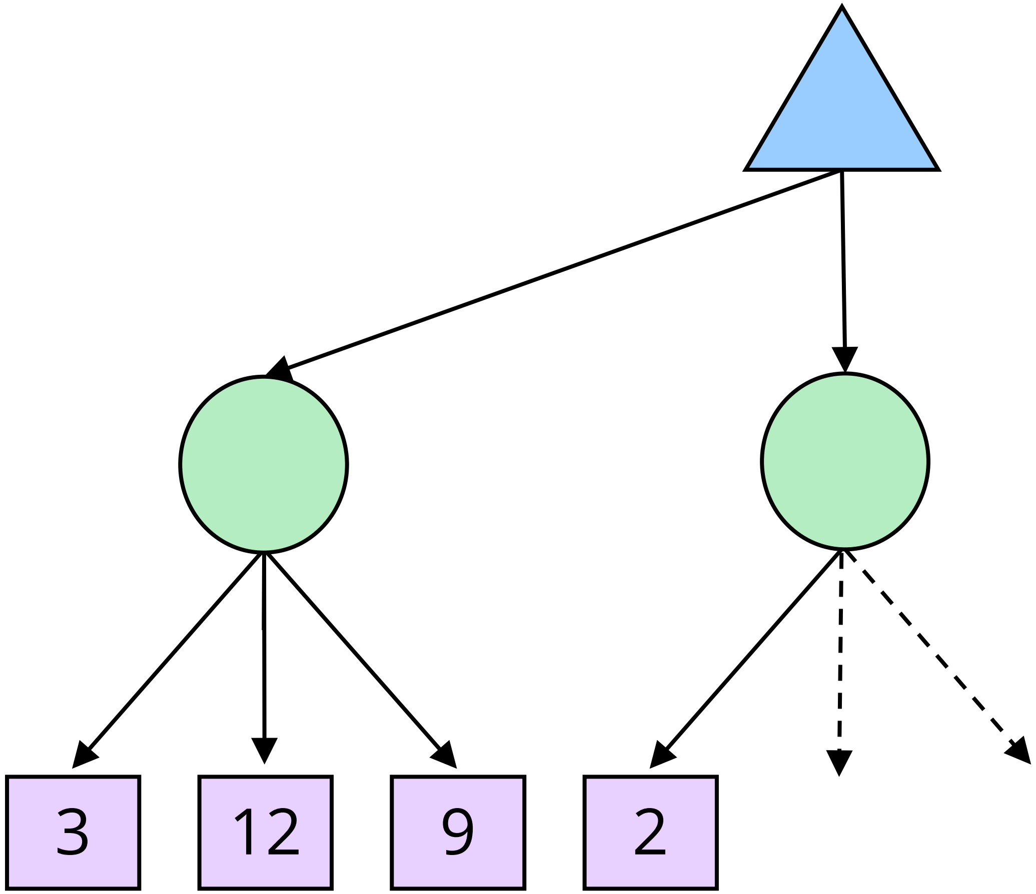
Expectimax Pseudocode

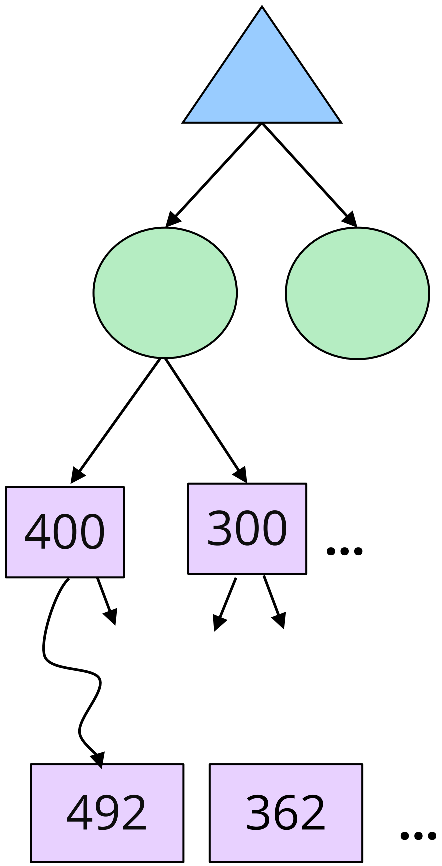
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



$$v = \frac{1}{2} \cdot (8) + \frac{1}{3} \cdot (24) + \frac{1}{6} \cdot (-12)$$

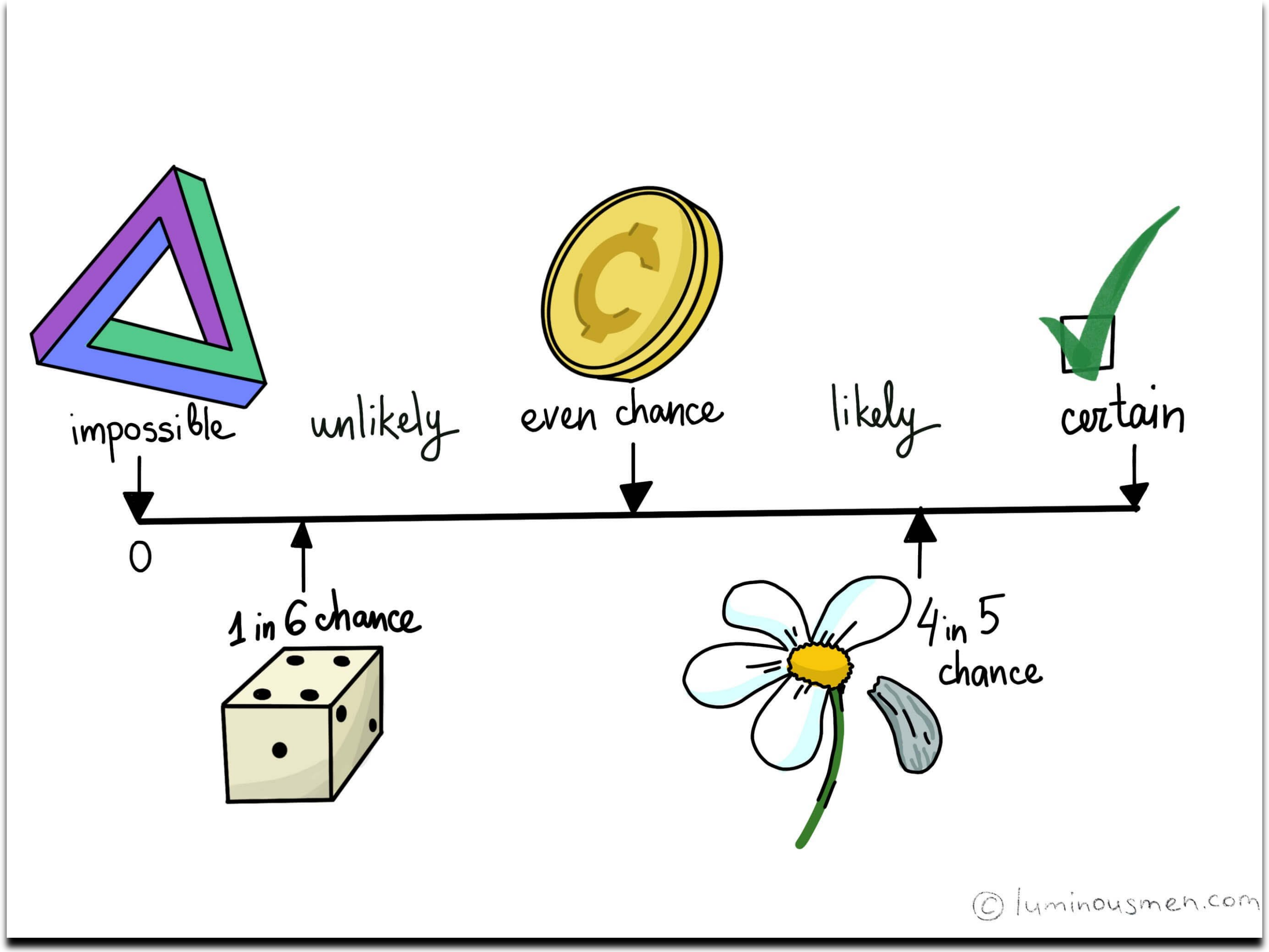

3
12
9
2
4
6
15
6
0


3
12
9
2



Estimate of true expectimax value (which would require a lot of work to compute)

Probabilities



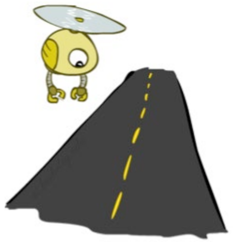
Probabilities

- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

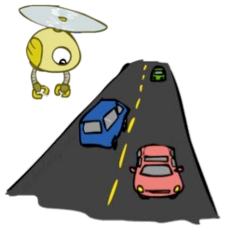
- Example: Traffic on freeway
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}
  - Distribution: $P(T = \text{none}) = 0.25$, $P(T = \text{light}) = 0.50$, $P(T = \text{heavy}) = 0.25$

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one

- As we get more evidence, probabilities may change:
  - $P(T = \text{heavy}) = 0.25$, $P(T = \text{heavy} \mid \text{Hour} = \text{8am}) = 0.60$
  - We'll talk about methods for reasoning and updating probabilities later



0.25



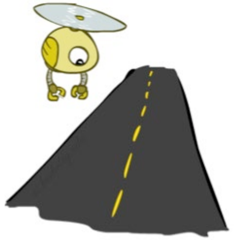
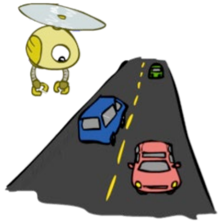
0.50

0.25

Probabilities

- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes
- Example: How long to get to the airport?

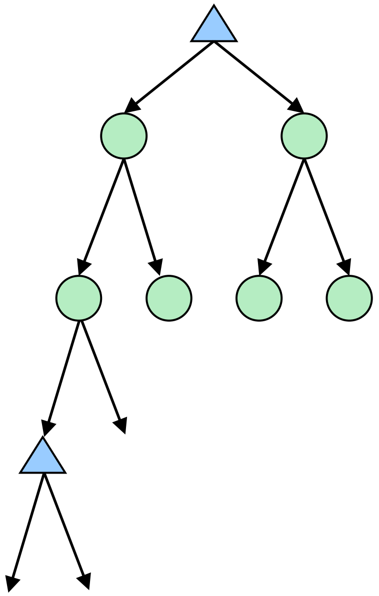
Time:	20 min	+	30 min	+	60 min		35 min
	x		x		x		
Probability:	0.25		0.50		0.25		


What Probabilities to Use?

- **In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state**
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!

- **For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes**



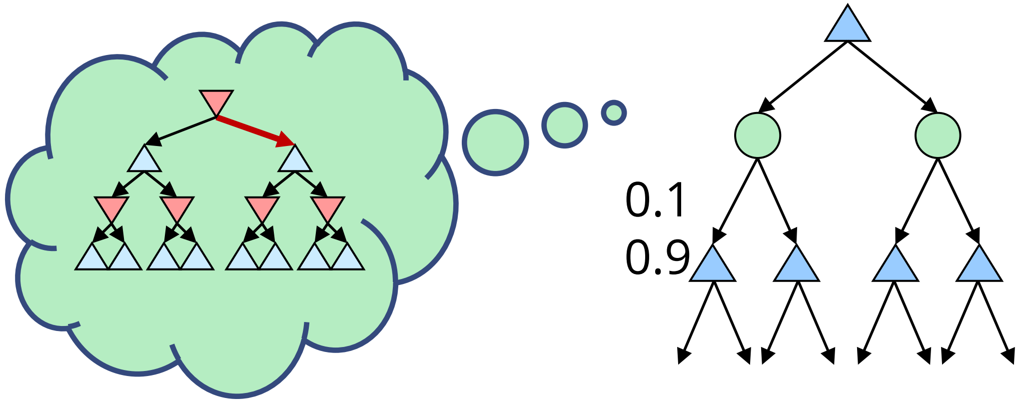
Having a probabilistic belief about another agent's action does not mean that the agent is flipping any coins!

- **Objectivist / frequentist answer:**
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - Assertion about how future experiments will go (in the limit)
  - New evidence changes the *reference class*
  - Makes one think of *inherently random* events, like rolling dice

- **Subjectivist / Bayesian answer:**
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. agent's belief how an opponent will behave, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

Quiz: Informed Probabilities

- Let's say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search should you use?



- Answer: Expectimax!
  - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
  - This kind of thing gets very slow very quickly
  - Even worse if you have to simulate your opponent simulating you…
  - … except for minimax, which has the nice property that it all collapses into one game tree

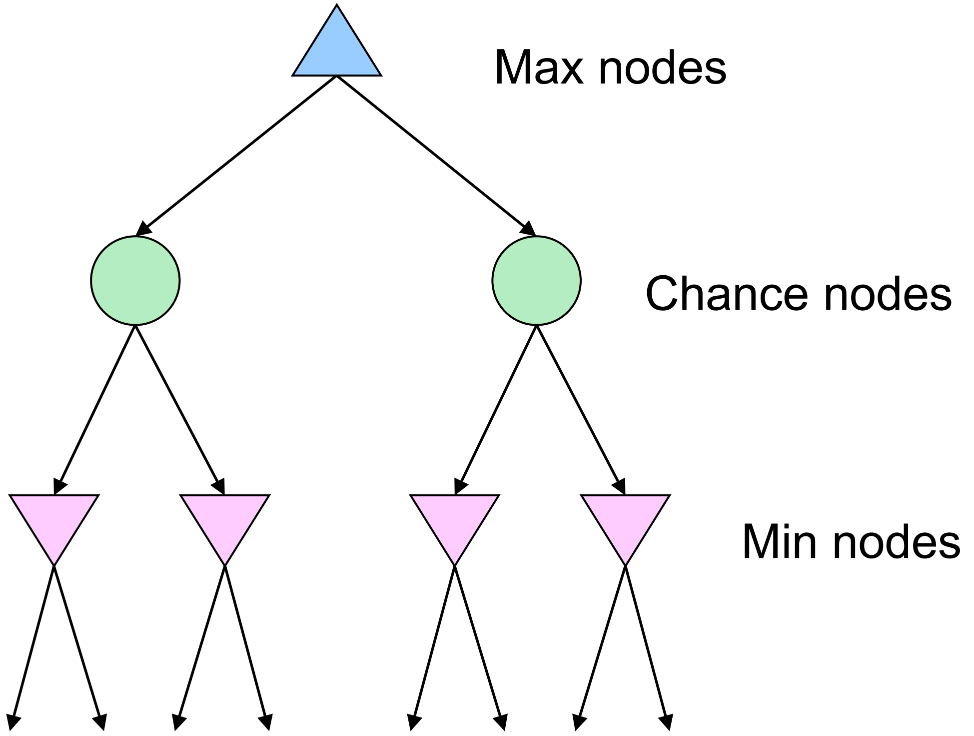
- **Dice rolls increase *b*: 21 possible rolls with 2 dice**
  - Backgammon ≈ 20 legal moves
  - Depth 2 → $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- **As depth increases, probability of reaching a given search node shrinks**
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…

- **Historic AI: TDGammon uses depth-2 search + very good evaluation fuanction + reinforcement learning → world-champion level play**

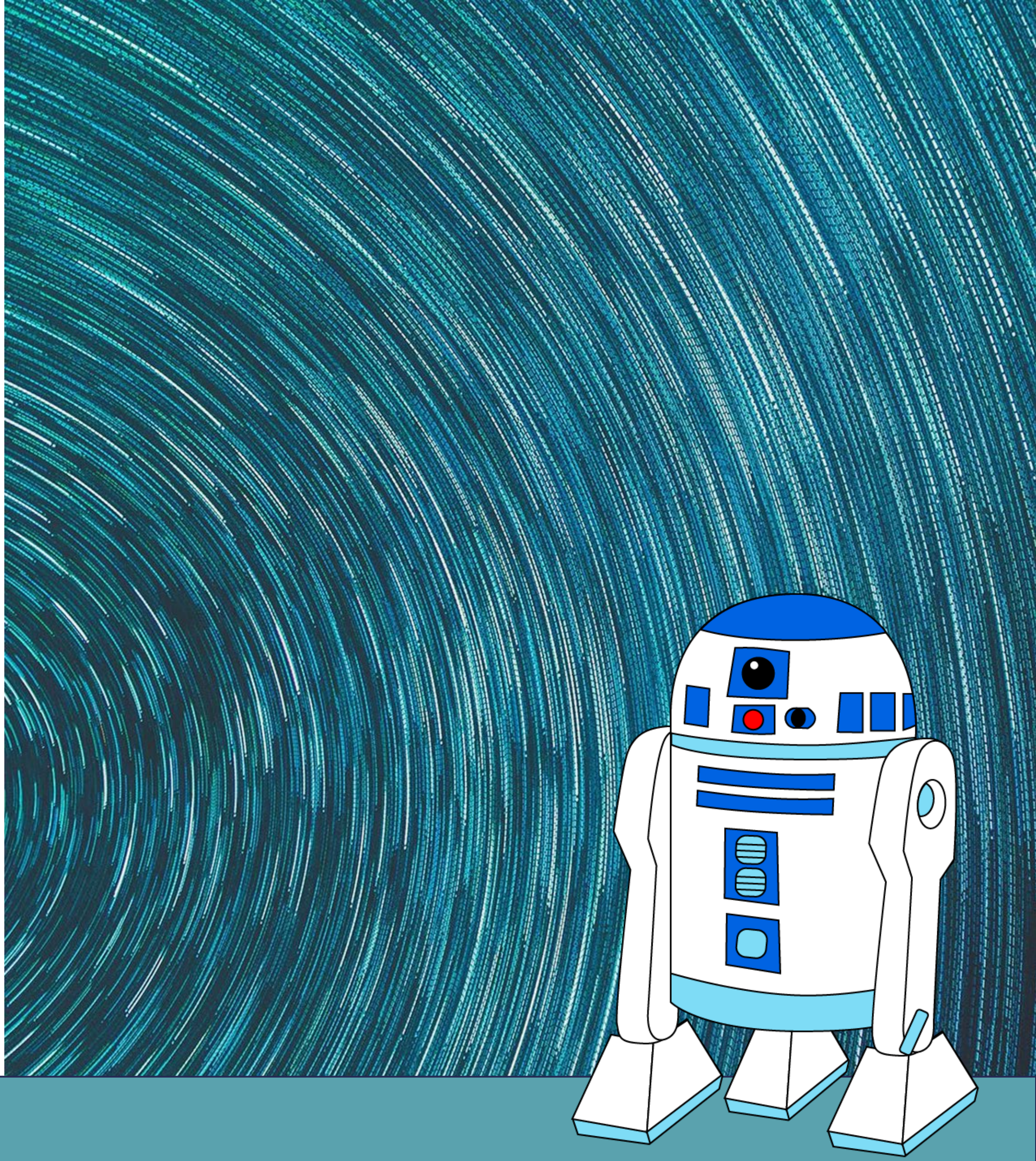
- **1st AI world champion in any game!**

- **E.g. Backgammon**
- **Expectiminimax**
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



CIS 421/521:
ARTIFICIAL INTELLIGENCE

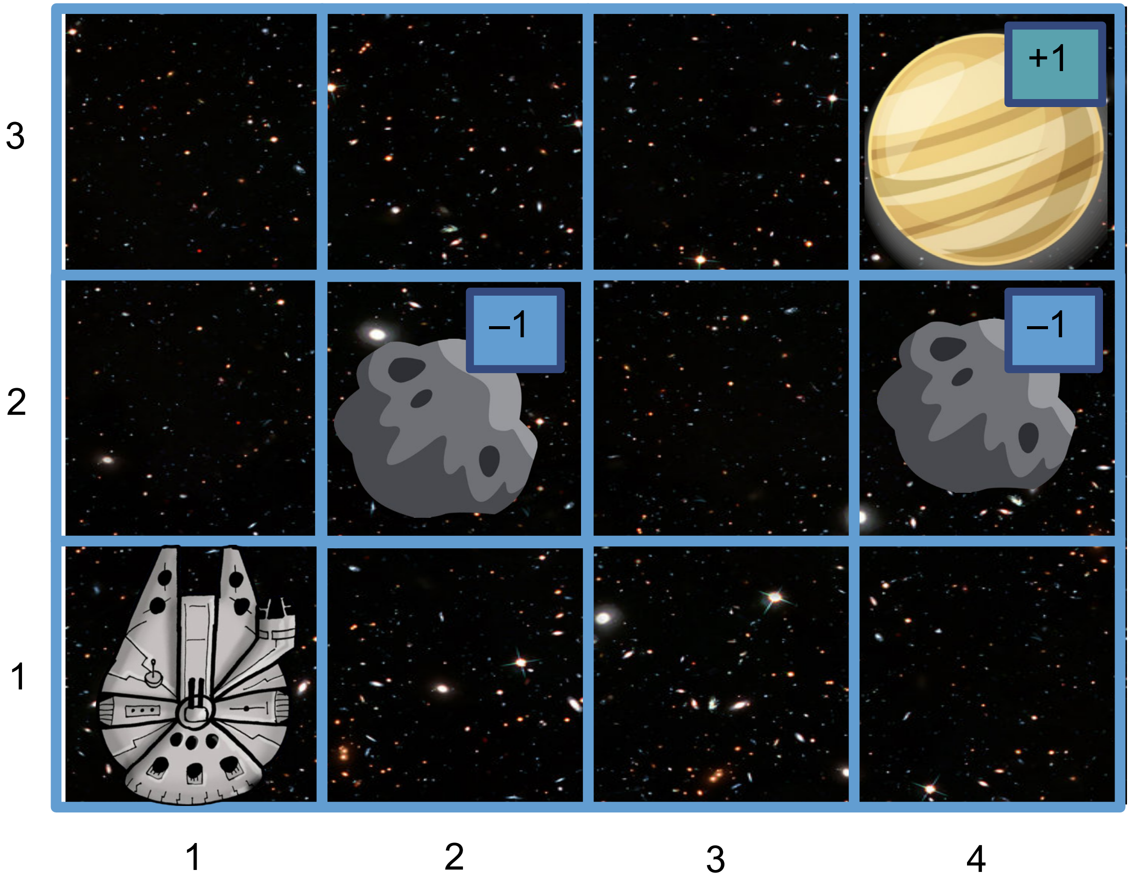
Markov Decision Processes



Penn Engineering
UNIVERSITY of PENNSYLVANIA

Navigating an Asteroid Field

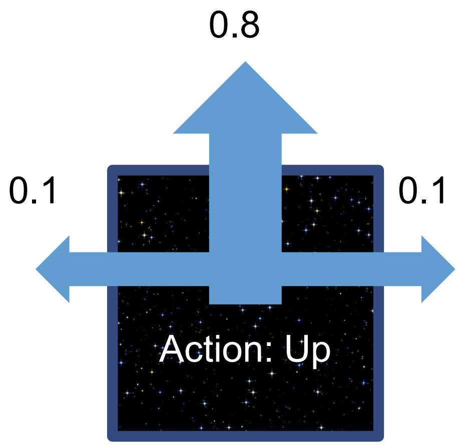
- Suppose we have a **fully-observable** 4x3 environment with goal states.
- The millennium falcon begins in the start state and **picks an action at each time step**.
- Actions: *Up, Down, Left, Right*
- The game **terminates when it reaches a goal state** (+1 or -1).
- If the environment were **deterministic**, the solution would be easy:
    - [*Up, Up, Right, Right, Right*]

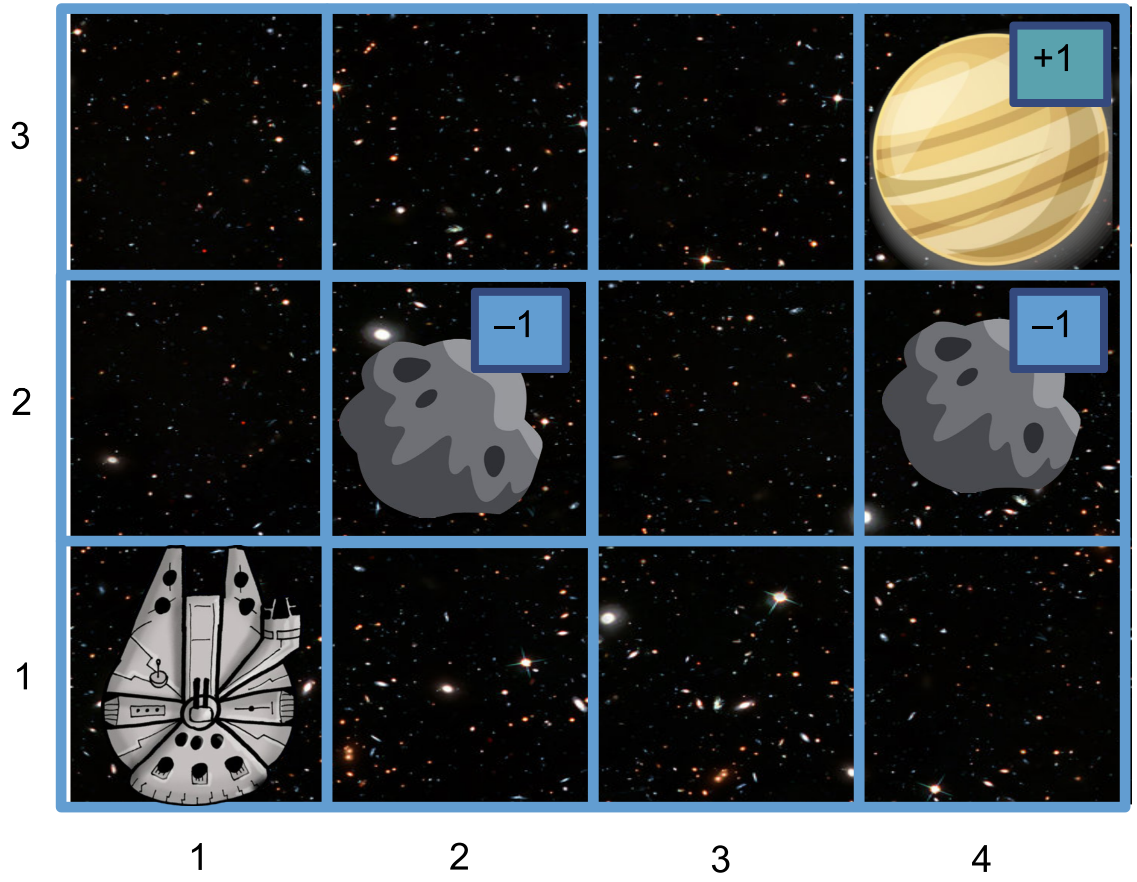


Navigating an Asteroid Field

- Instead of making the environment deterministic, we will make it **stochastic**.
- If the Falcon selects the action *Up* then it only moves up 80% of the time.
- 10% of the time the weird gravity fields cause it to veer off to the left or right.

Transition Model:

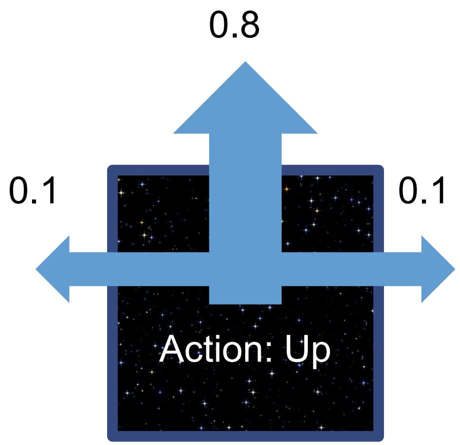


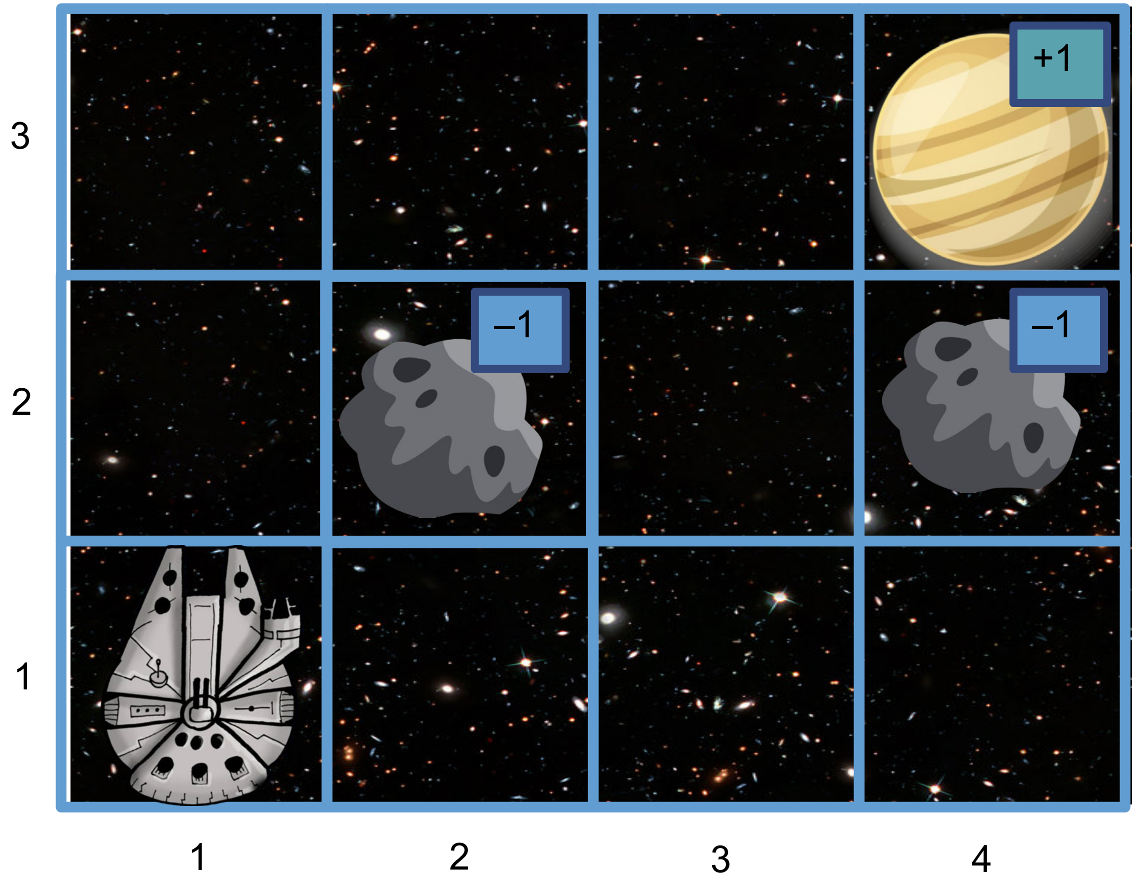


Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

Transition Model:



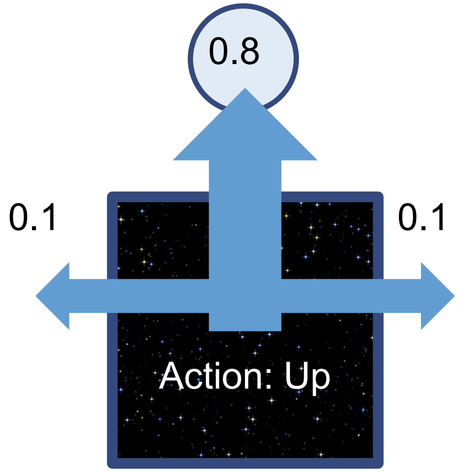


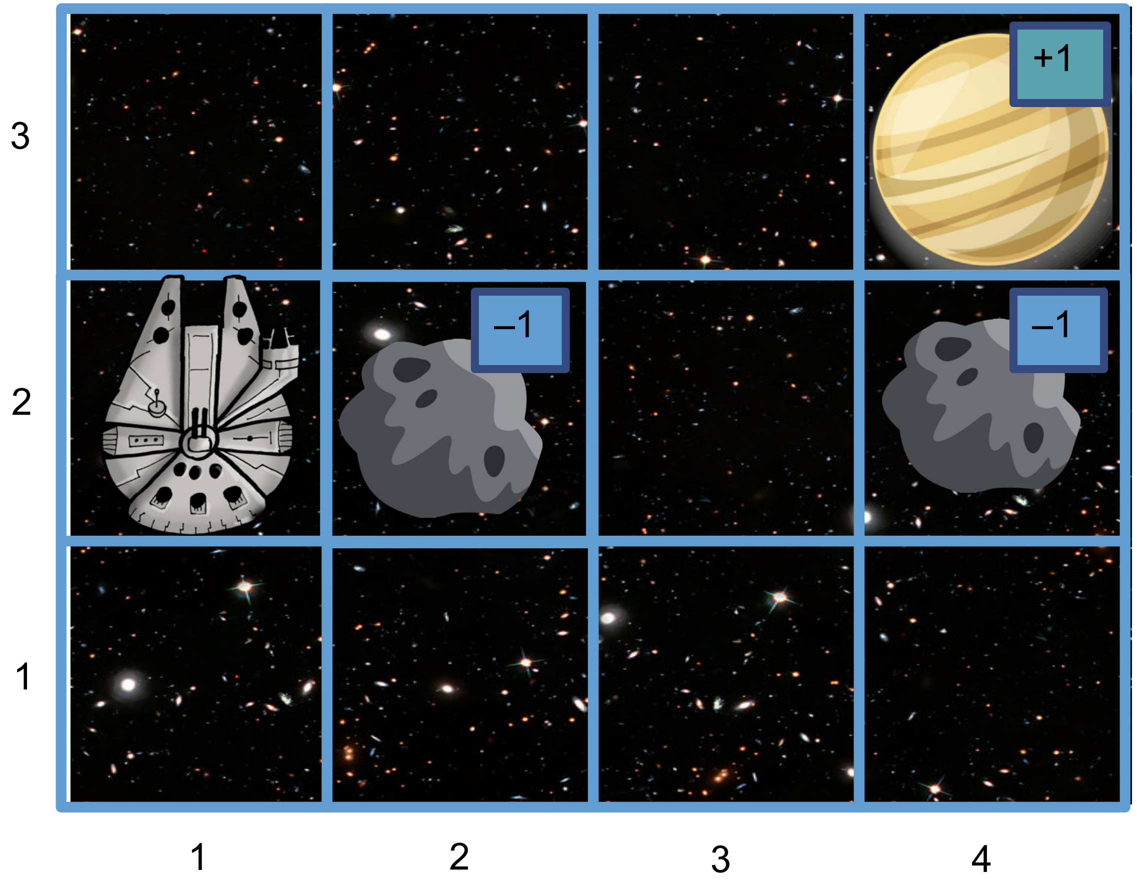
Navigating an Asteroid Field

- For action sequence
    - [*Up*, *Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.8

Transition Model:



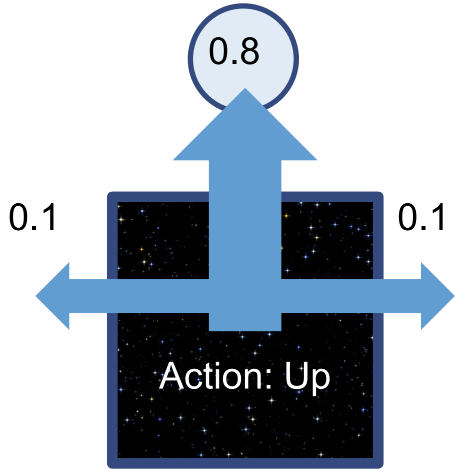


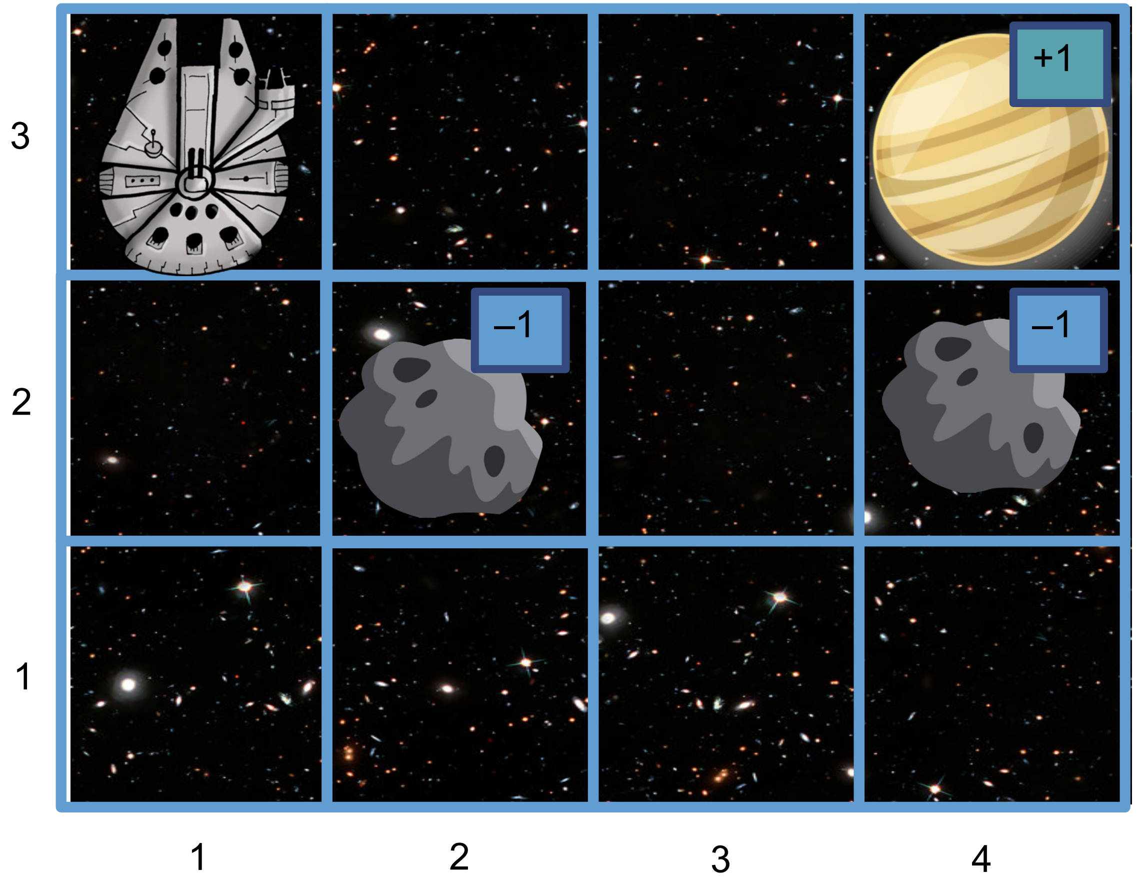
Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.8 * 0.8

Transition Model:



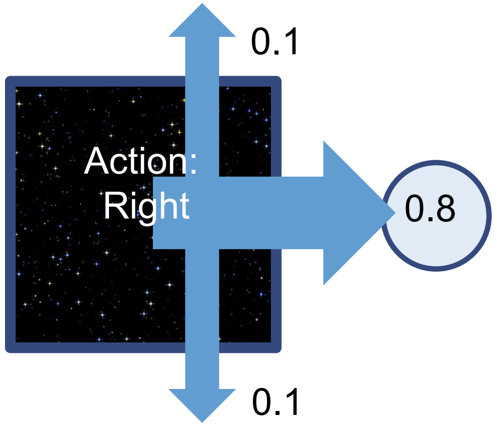


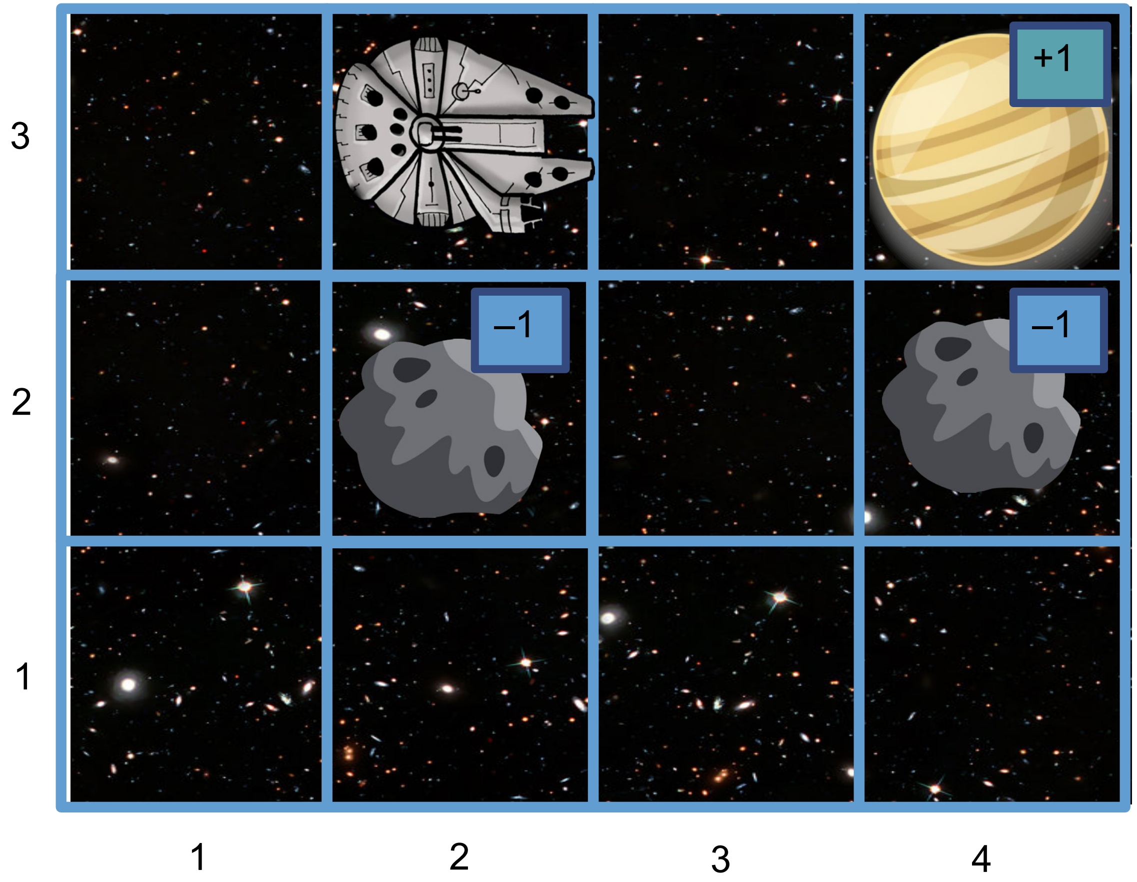
Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.8 * 0.8 * 0.8

Transition Model:



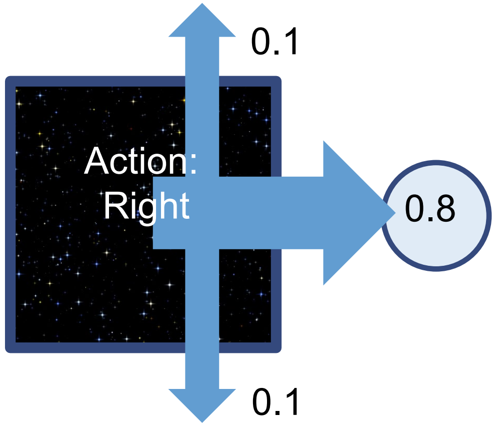


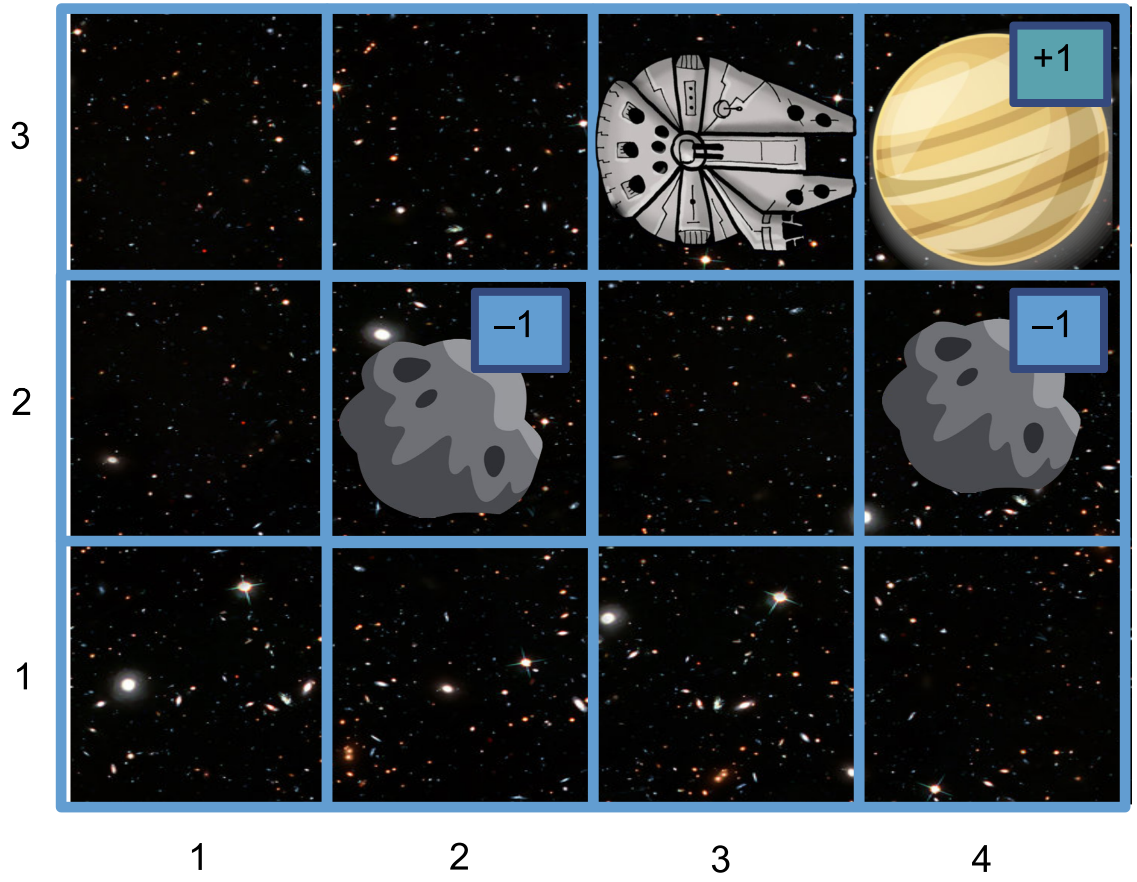
Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.8 * 0.8 * 0.8 * 0.8

Transition Model:





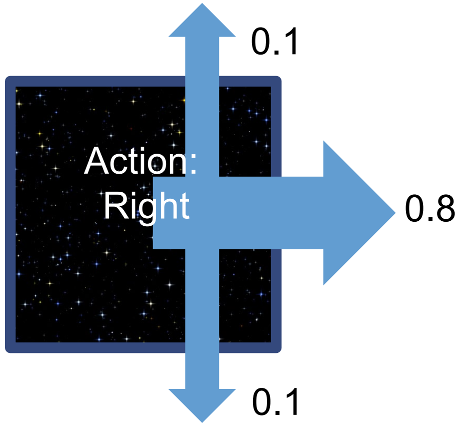
Navigating an Asteroid Field

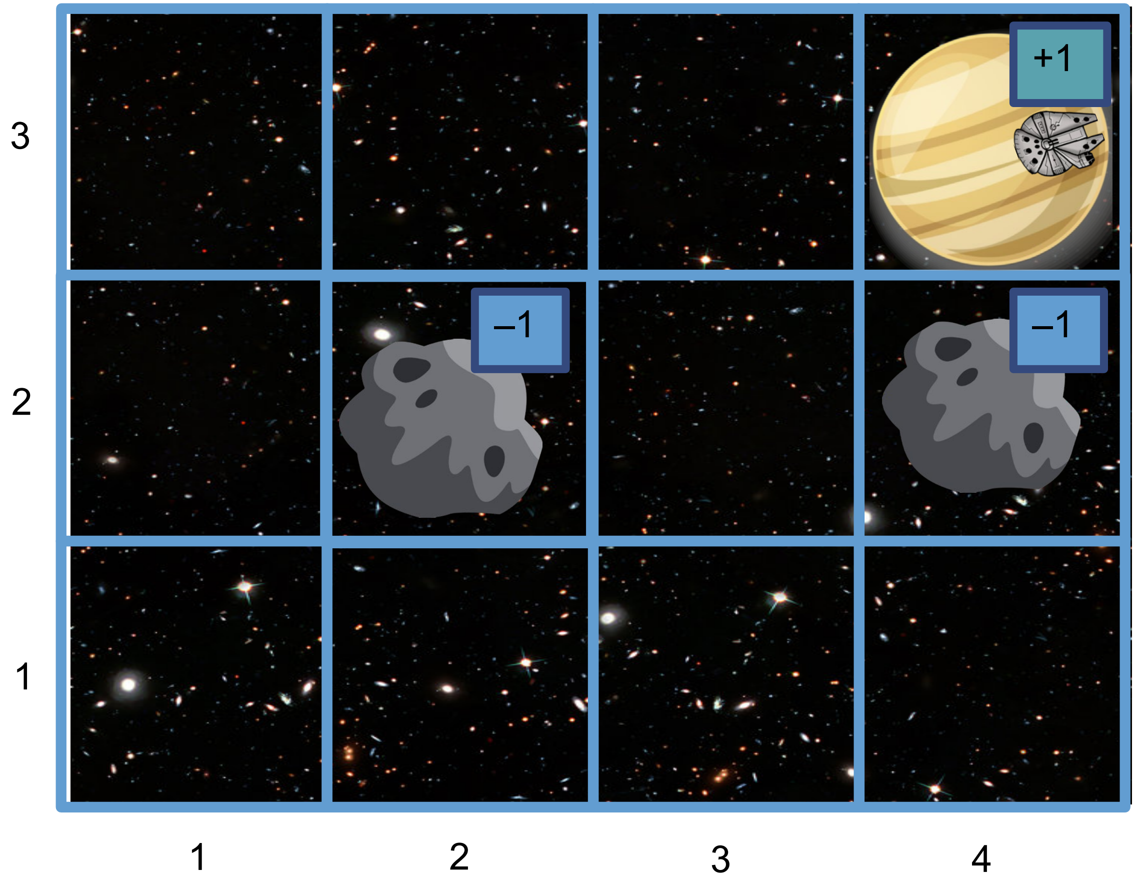
- For action sequence
    - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.8 * 0.8 * 0.8 * 0.8 * 0.8

= 0.32768

Transition Model:

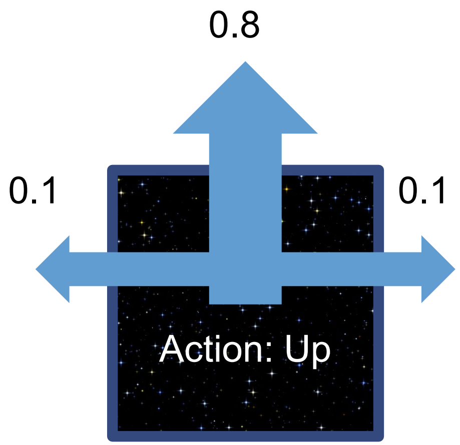


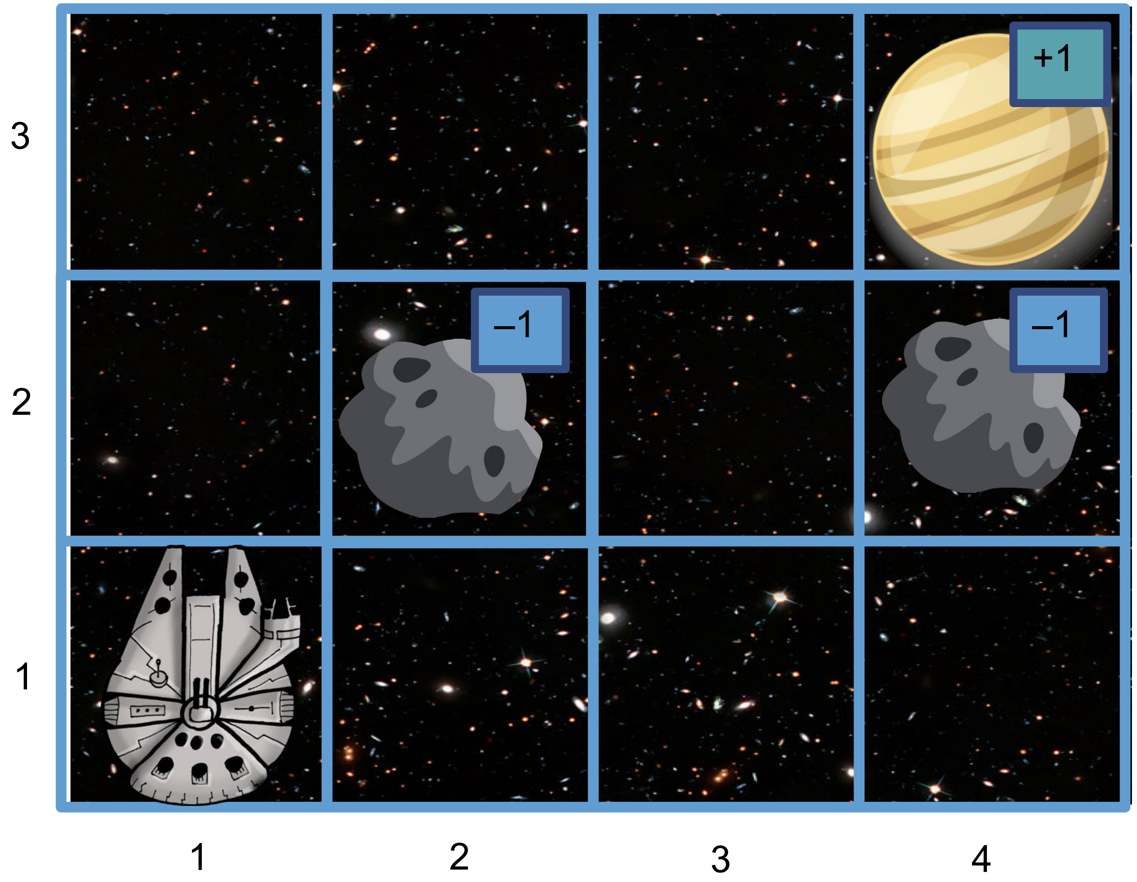


Navigating an Asteroid Field

- For action sequence
    - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

Transition Model:



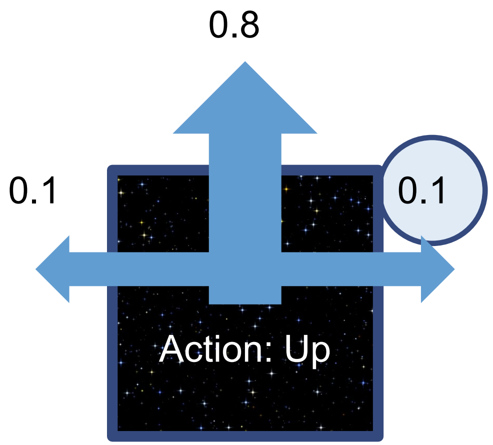


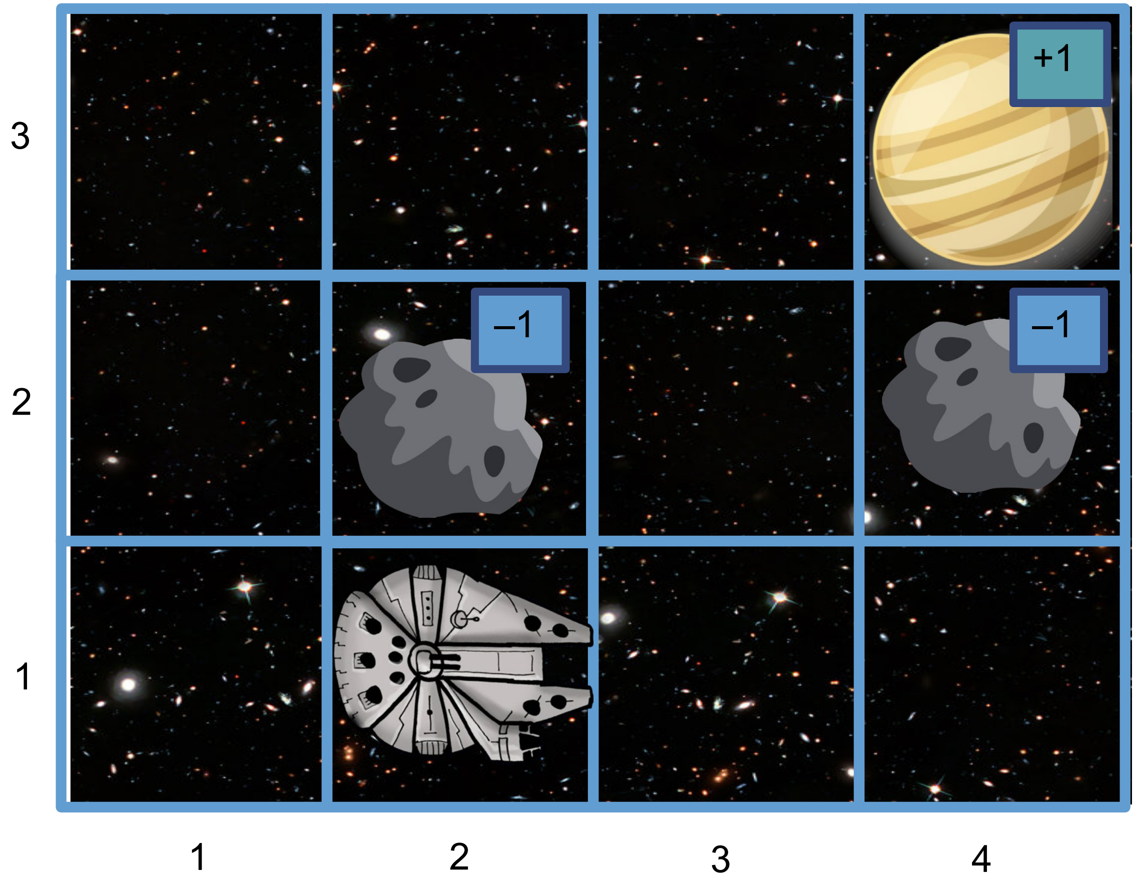
Navigating an Asteroid Field

- For action sequence
  - *[Up, Up, Right, Right, Right]*,
- what's the probability that the millennium falcon reaches the intended goal?

0.1

Transition Model:



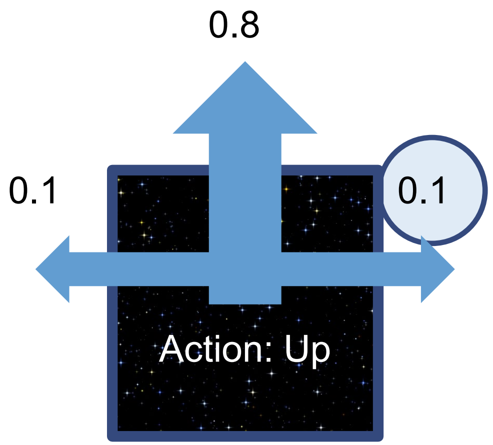


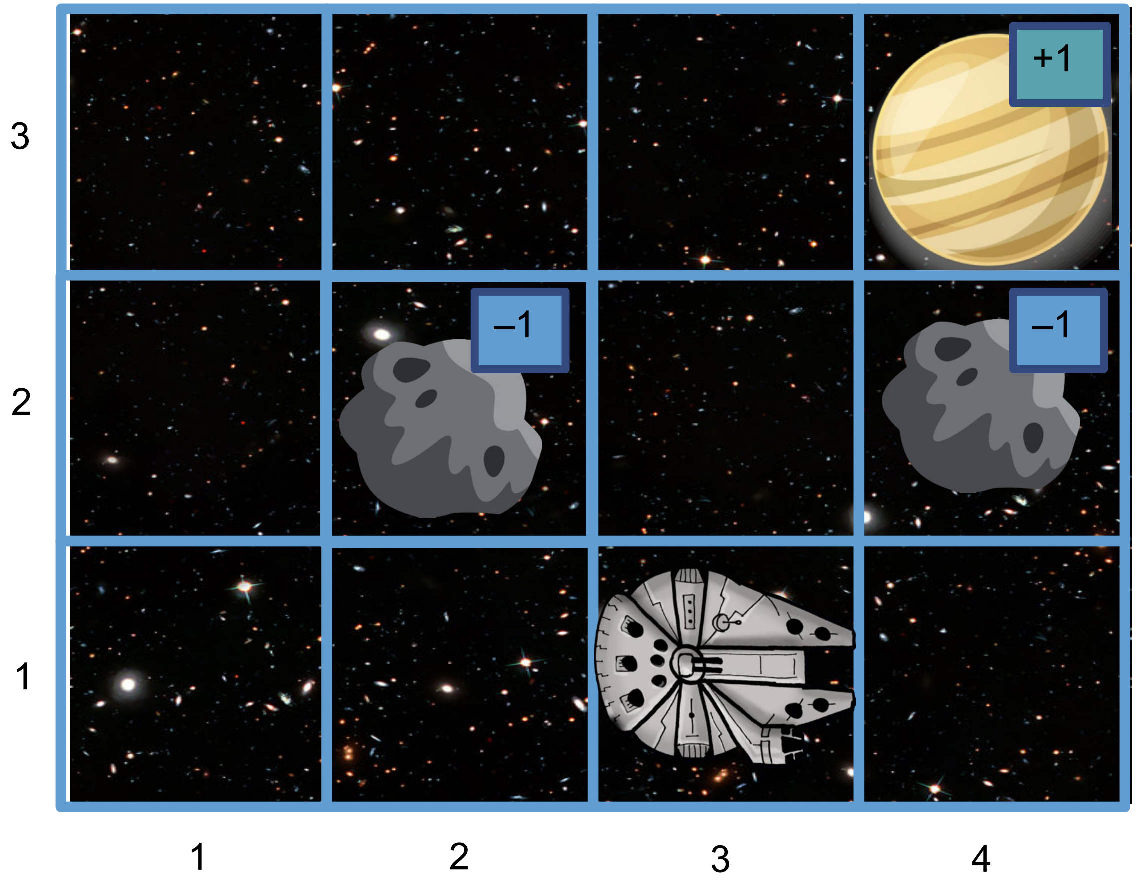
Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.1 * 0.1

Transition Model:



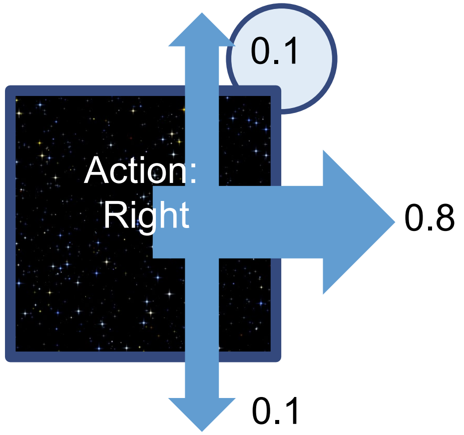


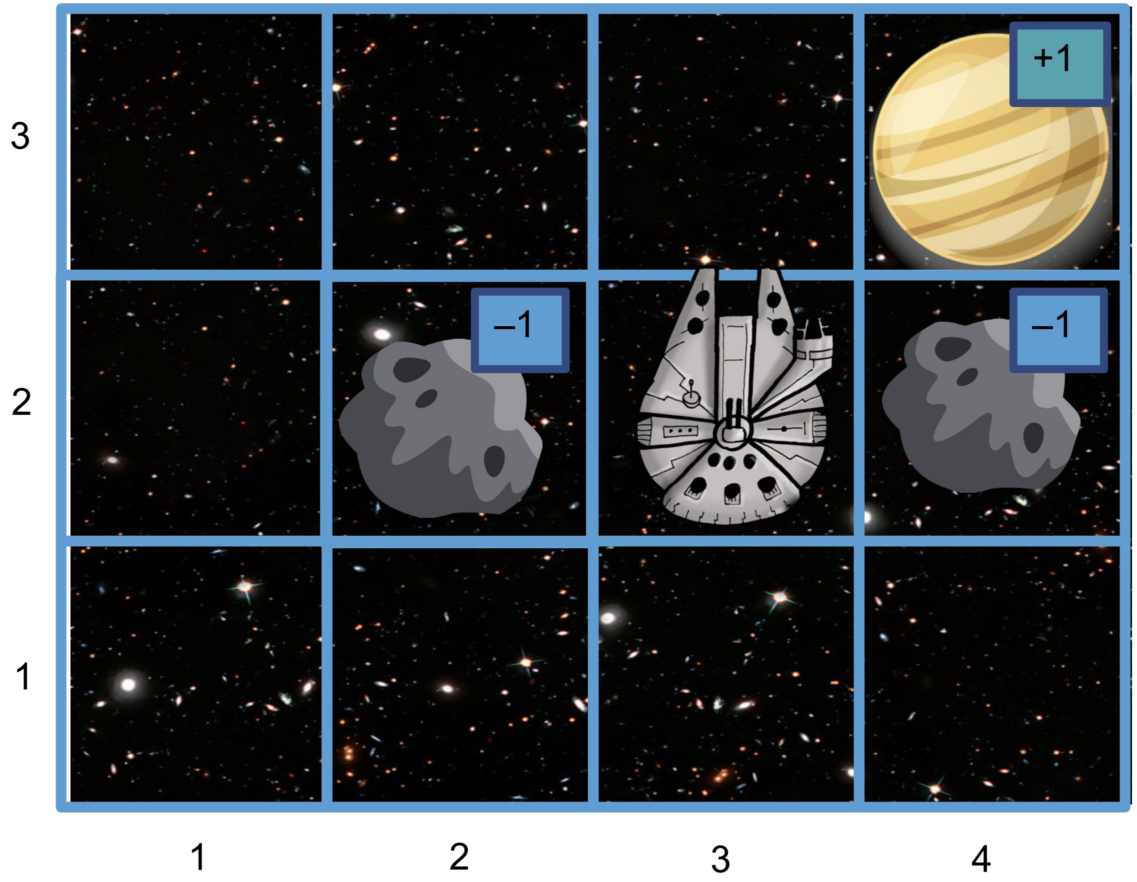
Navigating an Asteroid Field

- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what’s the probability that the millennium falcon reaches the intended goal?

0.1 * 0.1 * 0.1

Transition Model:



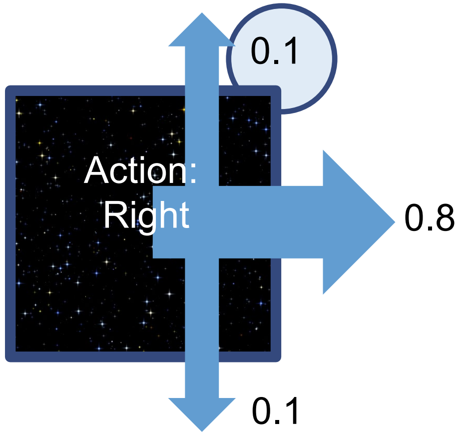


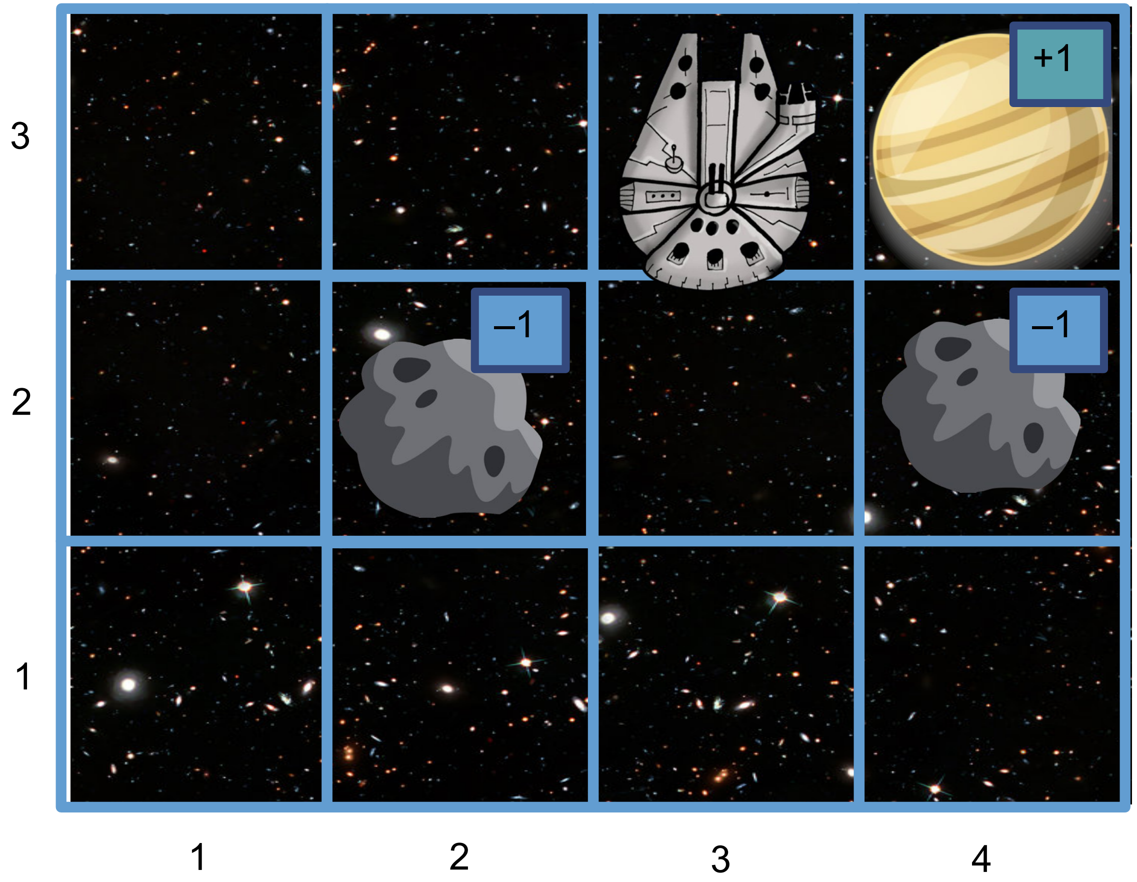
Navigating an Asteroid Field

- For action sequence
    - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.1 * 0.1 * 0.1 * 0.1

Transition Model:





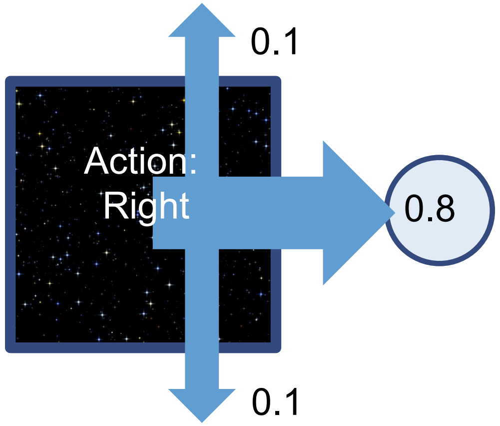
Navigating an Asteroid Field

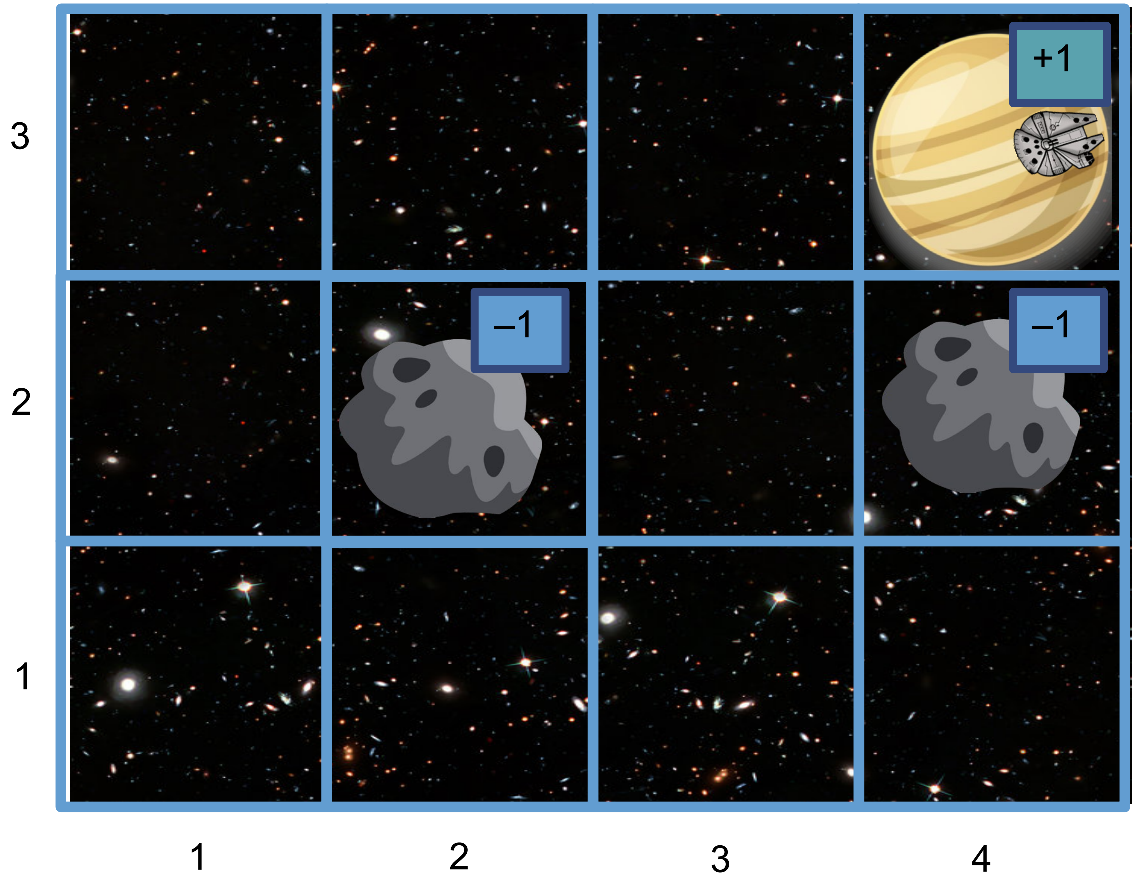
- For action sequence
  - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

0.1 * 0.1 * 0.1 * 0.1 * 0.8

= 0.00008

Transition Model:





Navigating an Asteroid Field

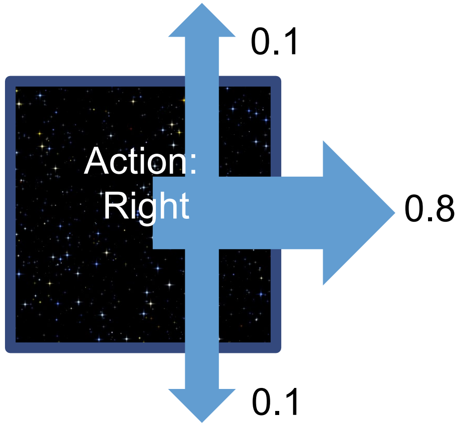
- For action sequence
    - [*Up, Up, Right, Right, Right*],
- what's the probability that the millennium falcon reaches the intended goal?

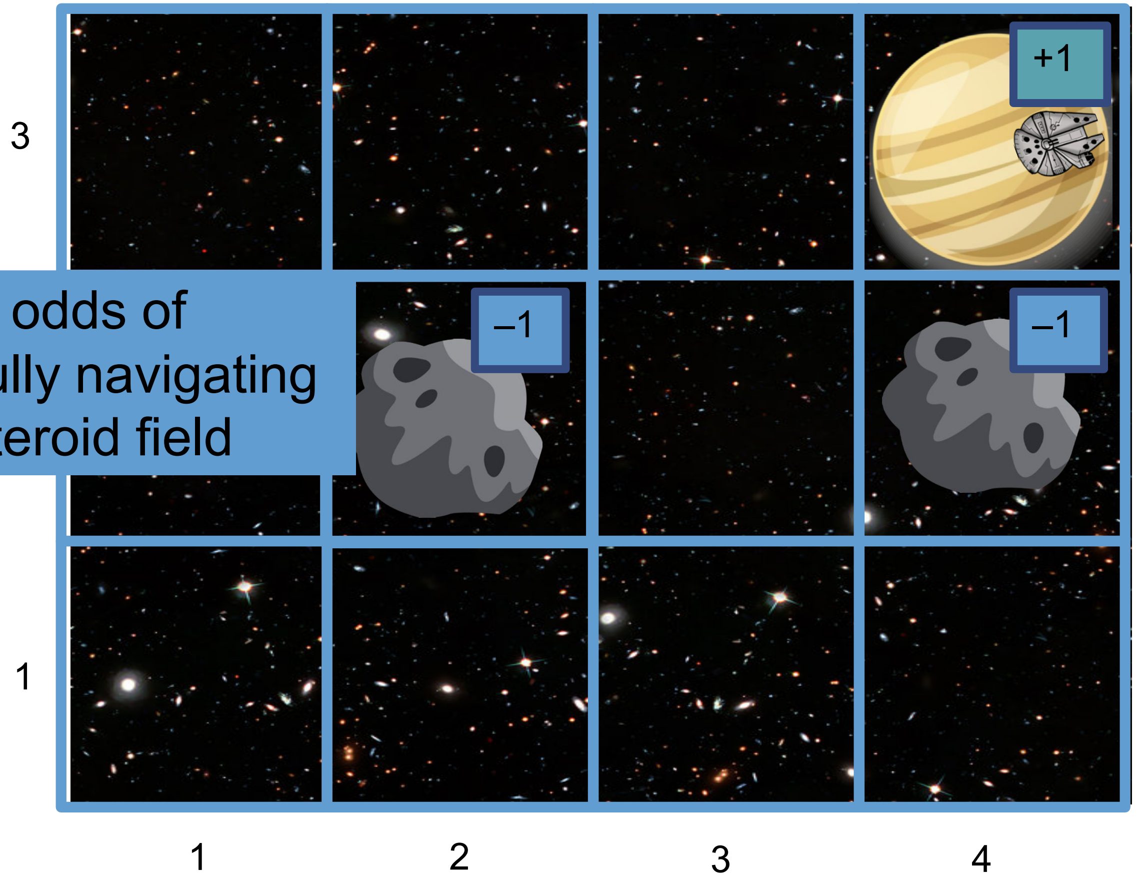
0.32768 + 0.00008

= 0.32776

The odds of successfully navigating an asteroid field

Transition Model:





Stochastic Transition Model

- In our search algorithms so far, the transition model was deterministic and described the outcome of each action in each state.
- The transition function is sometimes written as T(s, a, s'), or explicitly as a probability:

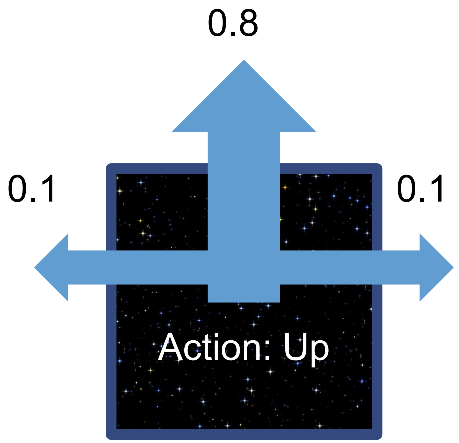
The probability of

arriving in state **s'**

$$p(s' \mid s, a)$$

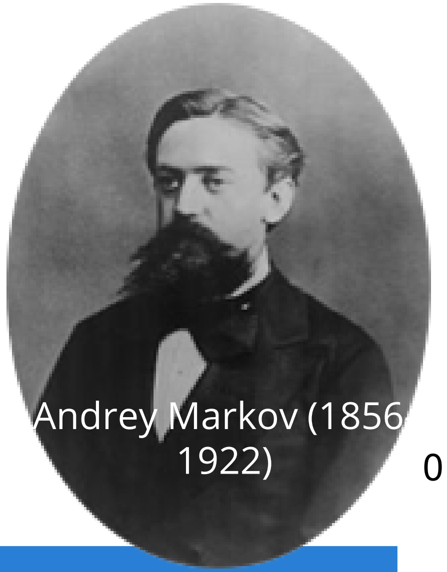
given that

we are in state **s** and we selected action **a**



Stochastic Transition Model

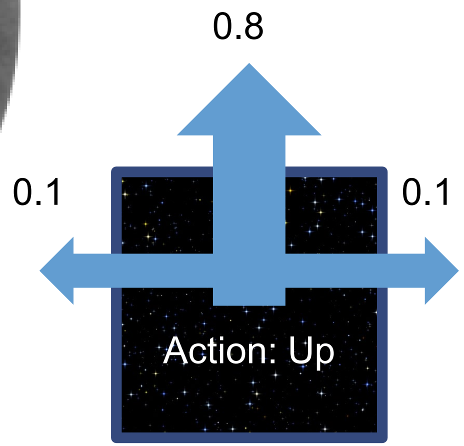
- In our search algorithms so far, the transition model was deterministic and described the outcome of each action in each state.
- The transition function is sometimes written as T(s, a, s’), or explicitly as a probability:



Andrey Markov (1856-1922)

$$p(\, s' \mid s, a \,)$$

Transitions are **Markovian**: the probability of arriving in s’ only depends on s and not the history of earlier states.



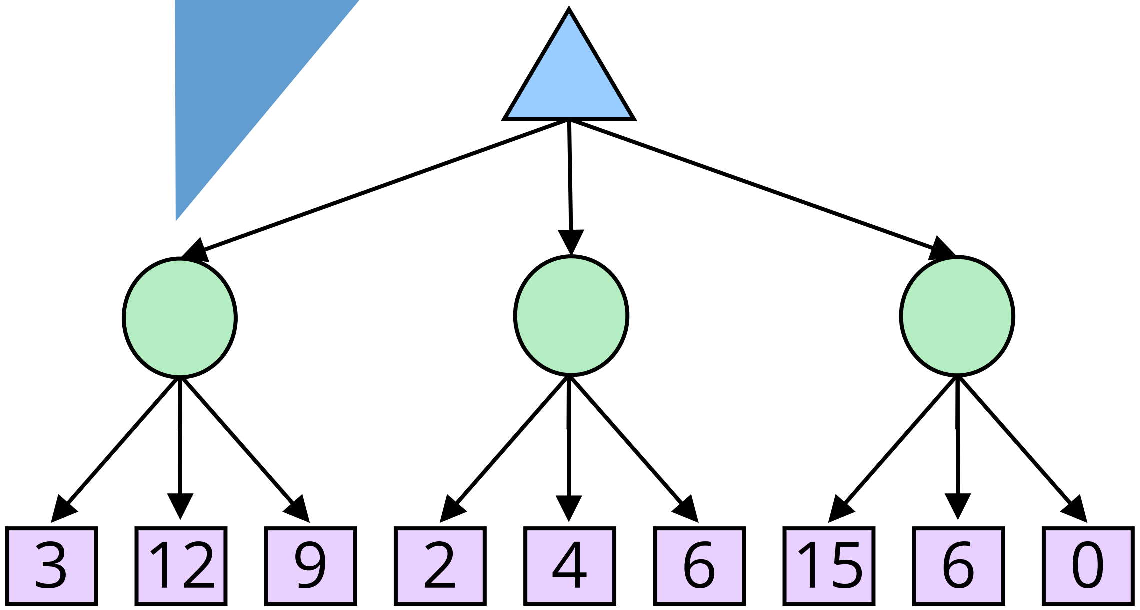
Reward function

- We will specify a **utility or reward function** for the agent.
- The “rewards“ can be **positive** or **negative** but are bounded by some maximum value.
- Because the decision process is **sequential,** we must specify the utility function on a sequence of states and actions.
- Instead of only giving a reward at the goal states, the agent can **receive a reward at each time step**, based on its transition from **s** to **s’** via action **a**.
- This is defined by a reward function
- R(s, a, s’)
- *For example, we could give the Millennium Falcon a small negative reward of -0.04 for every transition except for entering the terminal states (+1 for entering the planet’s orbit or -1 for smashing into an asteroid).*
- The **rewards are additive**, so if the Millennium Falcon takes 4 steps before entering the planet's orbit, it gets *-0.04* + *-0.04* + *-0.04* + *-0.04* + *1* = *0.84* for that solution.

Markov Decision Pr

Expectimax node: outcome is uncertain. In expectimax search we calculate their expected utilities.

- A **Markov decision process** or **MDP** is
  - a **sequential** decision problem
  - for a **fully observable** environment
  - with a **stochastic** transition model
  - that has **additive rewards**



- **MDPs are non-deterministic search problems.** One way of solving them is via **expectimax** search.

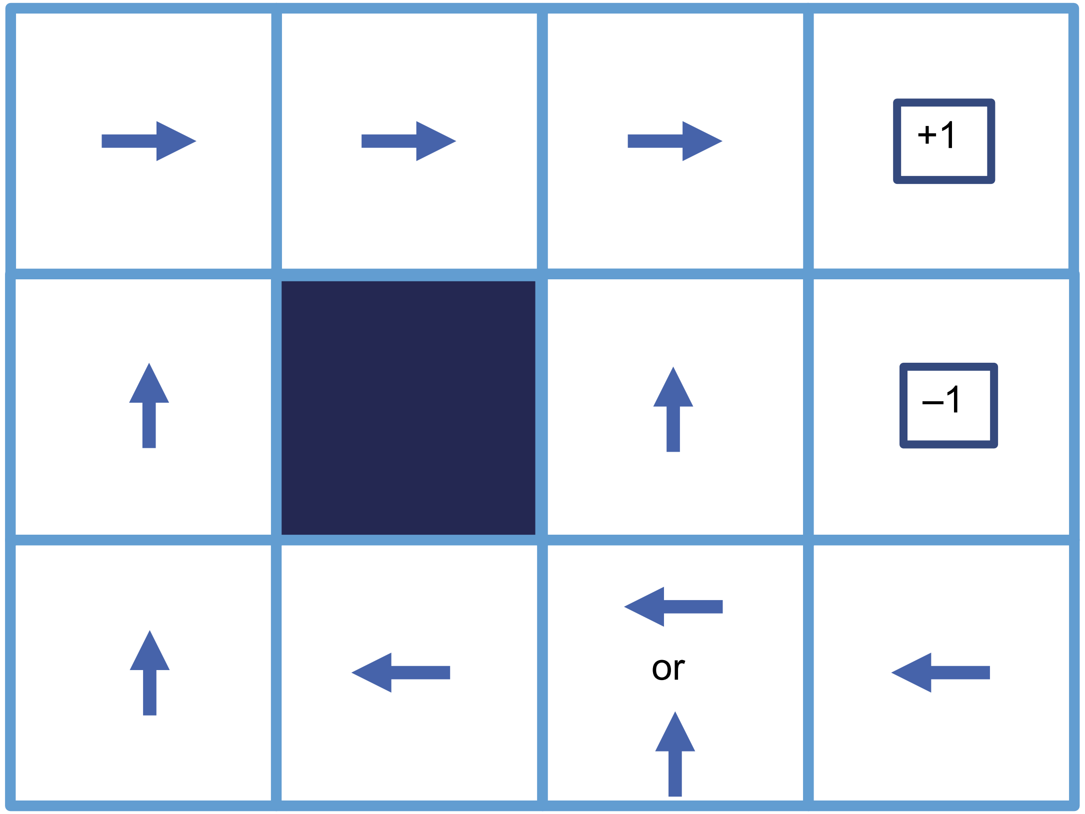
Markov Decision Process

- To find a solution to an MDP, you need to define the following things:
- **A set of states** $s \in S$
- **A set of actions** $a \in A$
- A transition function **T(s, a, s')**
  - Probability that executing action **a** in **s** will lead to **s' P(s' | s, a)**
  - The probability is called **the model**
- A reward function **R(s, a, s')**
  - Sometimes just R(s) or R(s')
- An **initial state** s_0
- Optionally, one or more **terminal states**

Solution == Policy

- In search problems a solution was a sequence of action that corresponded to the shortest path.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$

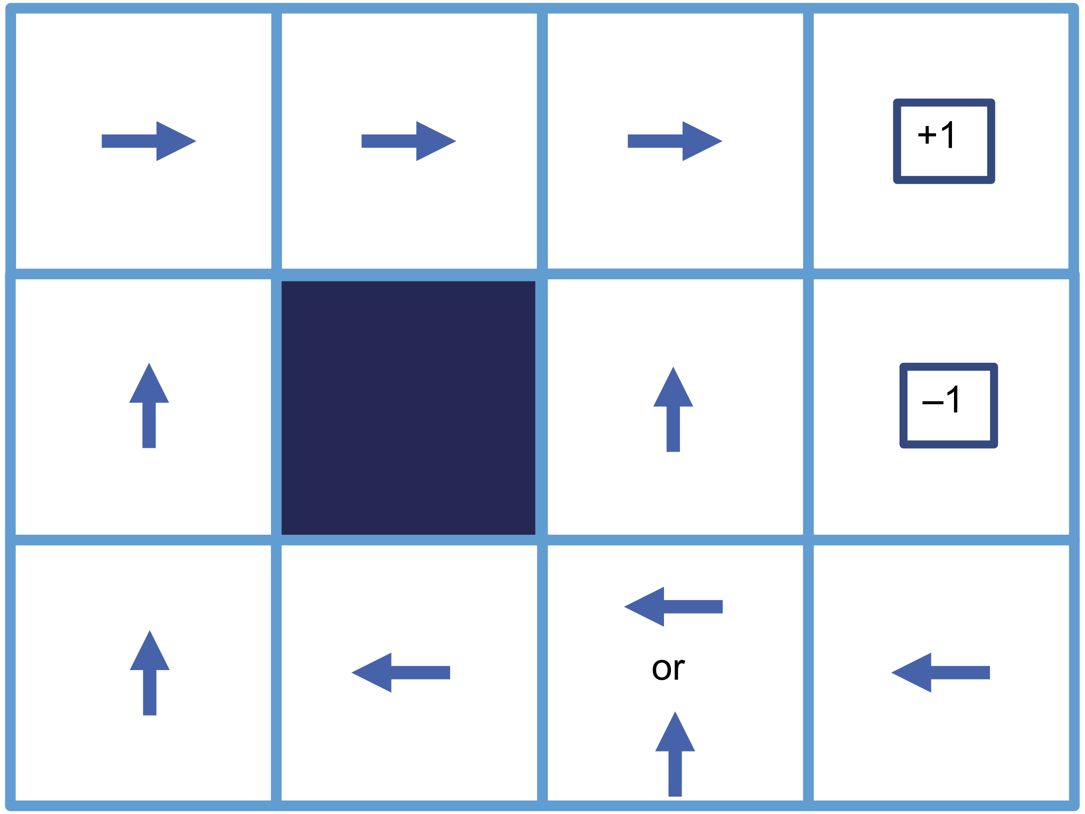
Policy **π** tells the agent what action to take at state s.



This is an example policy for a grid world

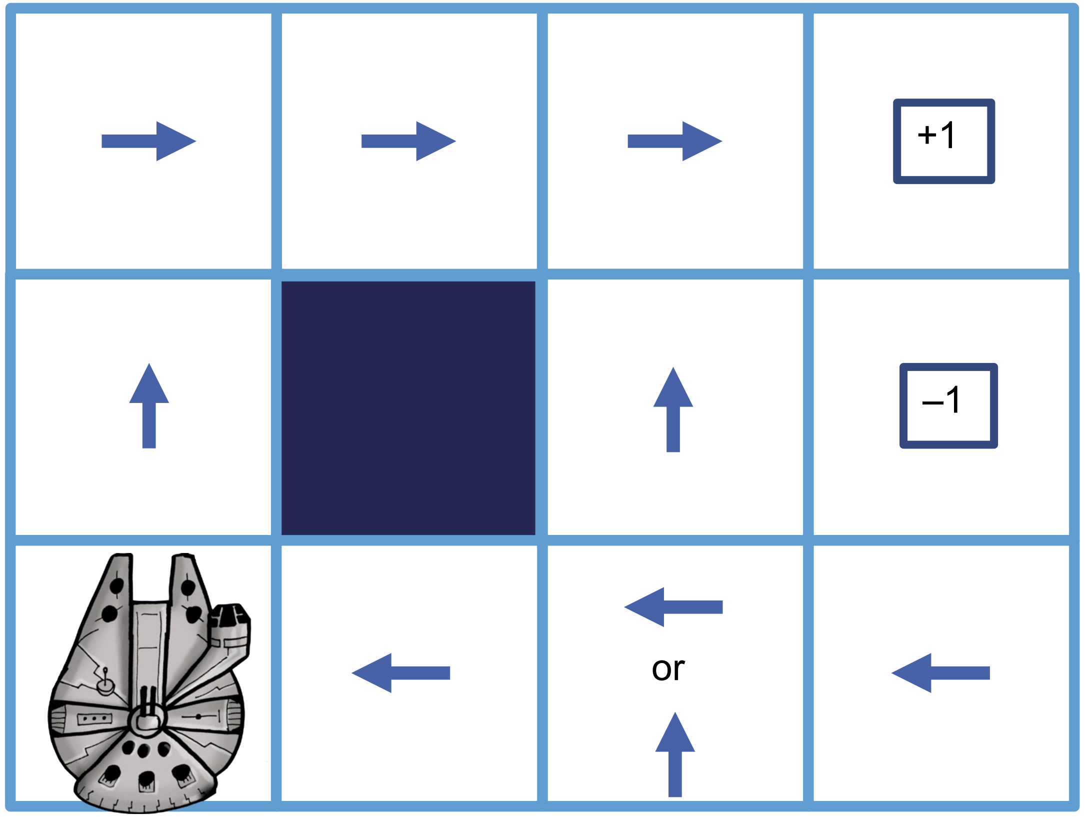
Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$



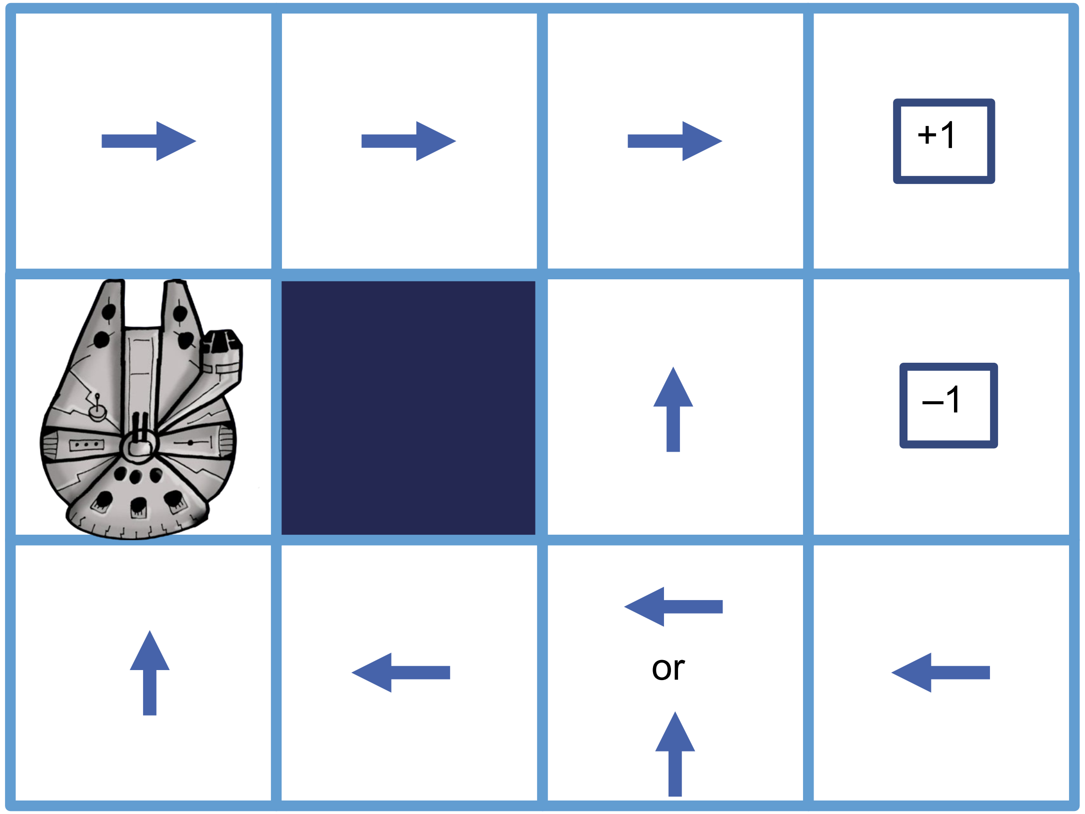
Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$



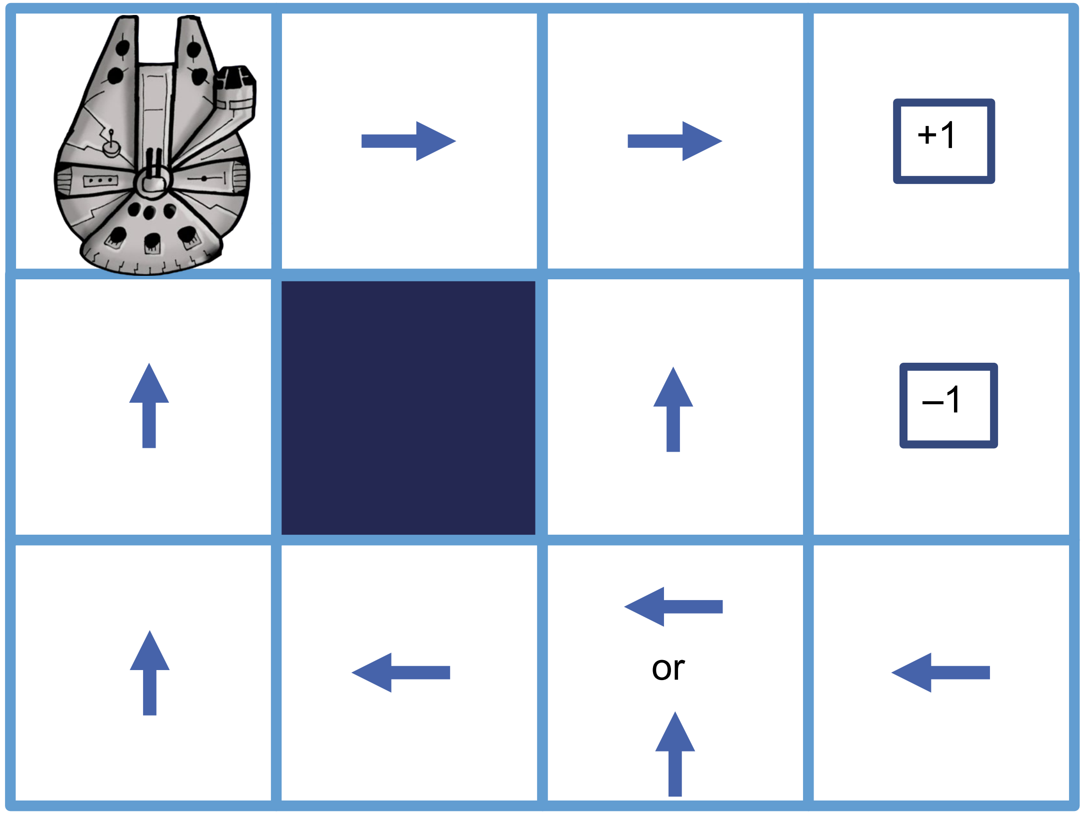
Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$



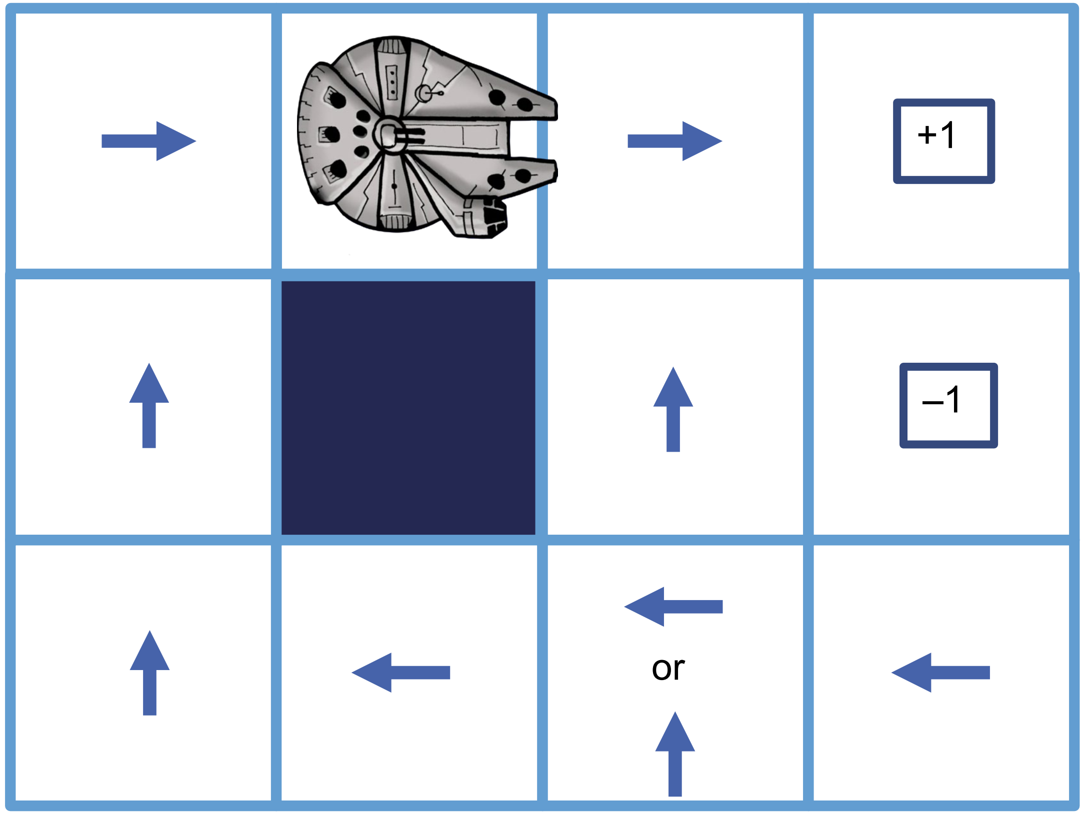
Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
    - $\boldsymbol{\pi}(s) = a$



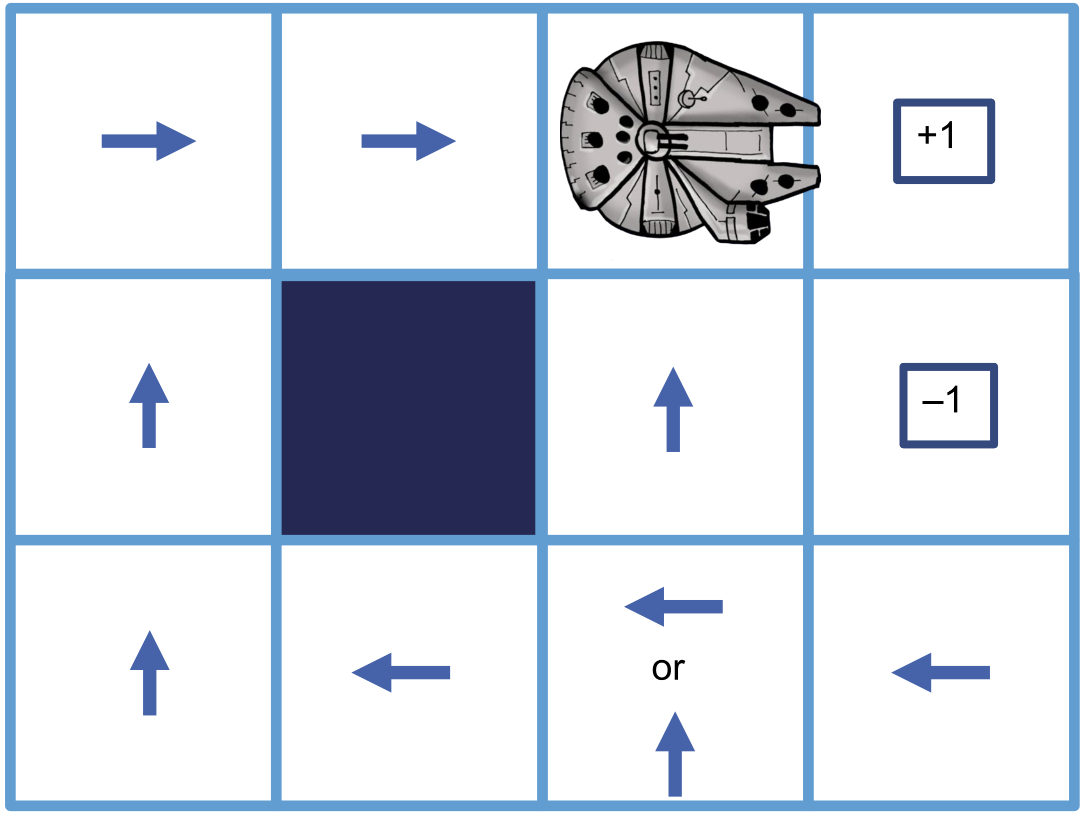
Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$



Solution == Policy

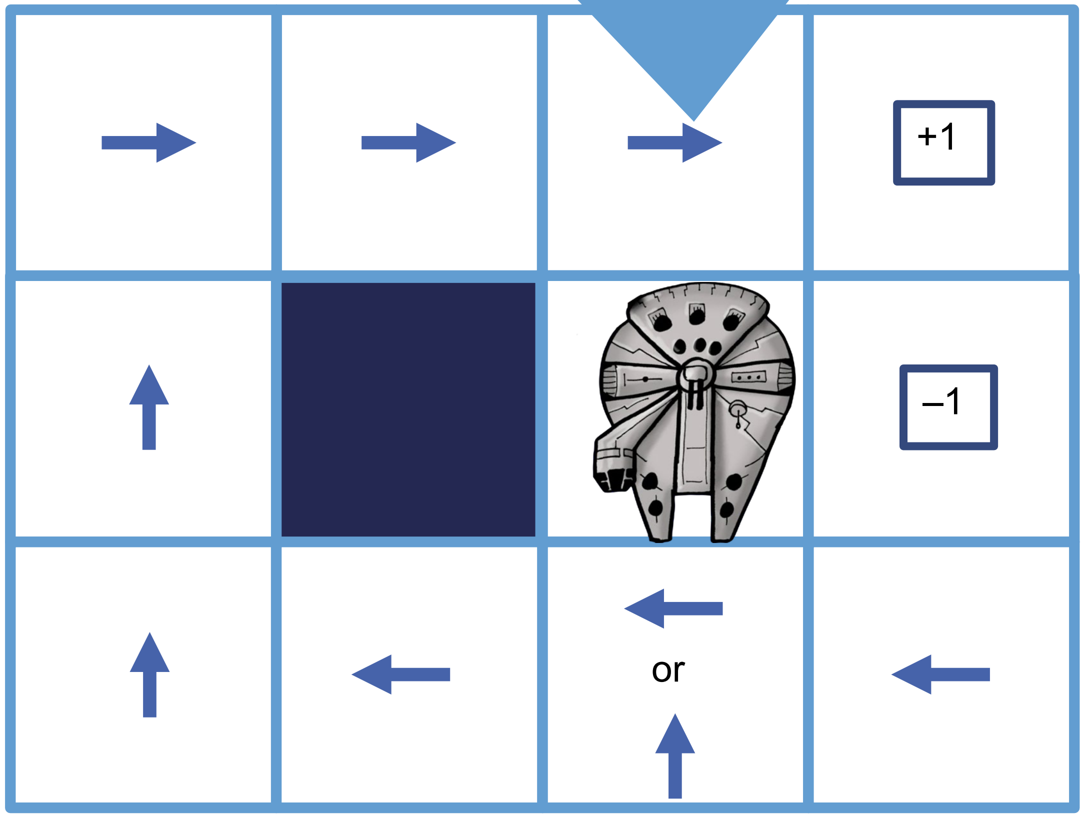
- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$



Solution == Policy

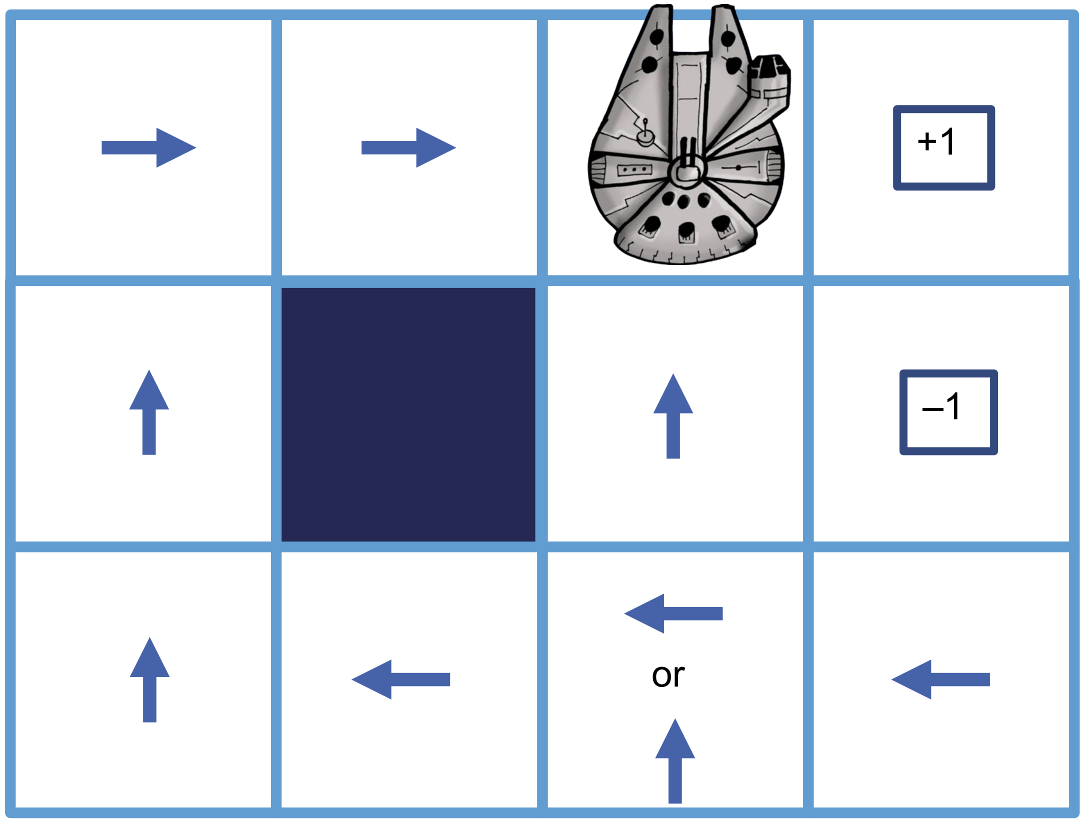
- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$

Even though the policy told me to go right here, there's no guarantee that me picking the action Right will result in me moving right. It's stochastic!



Solution == Policy

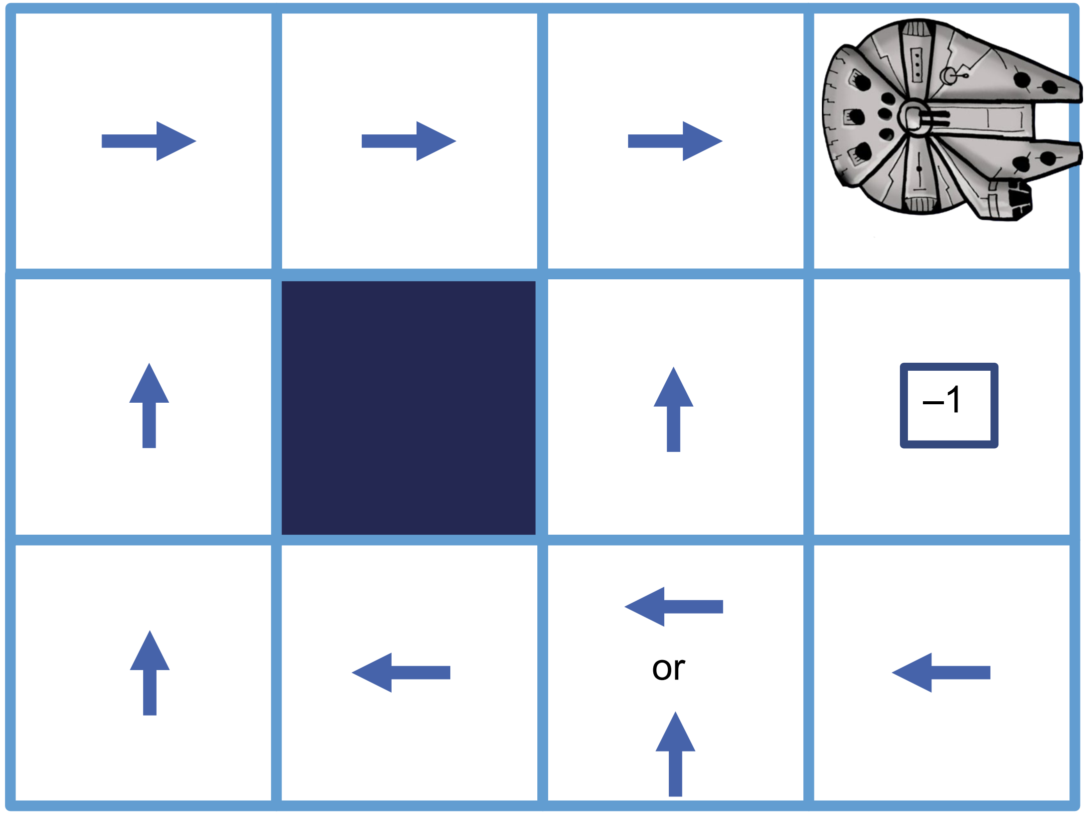
- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
    - $\boldsymbol{\pi}(s) = a$



Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
  - $\boldsymbol{\pi}(s) = a$

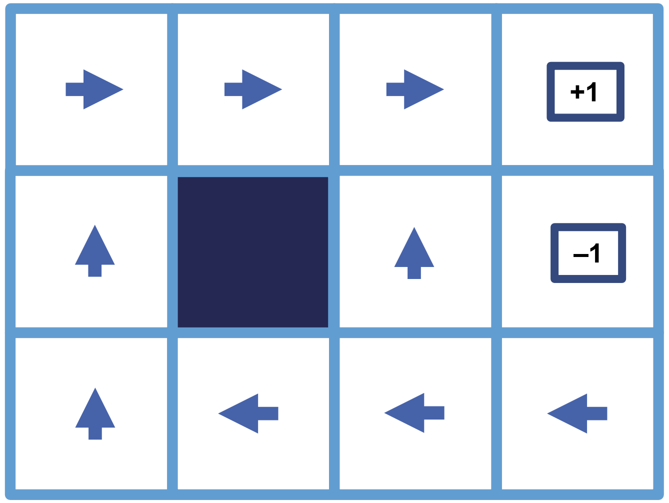
We will use $\boldsymbol{\pi}^*$ to denote the **optimal policy**.



Policies and Rewards

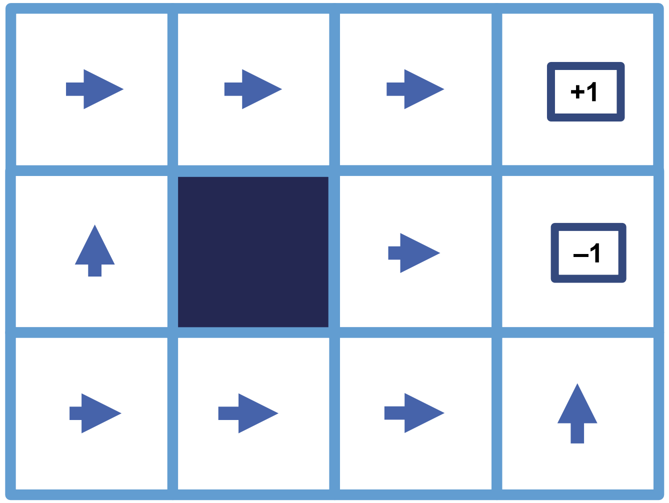
- Even if the **same policy** is executed multiple times by the agent, this may lead to different sequence of states and actions (**environment history**), and thus a **different score** under the reward function.
- Therefore we need to compute the **expected utility** of all the possible paths generated by a policy.

r = -0.04

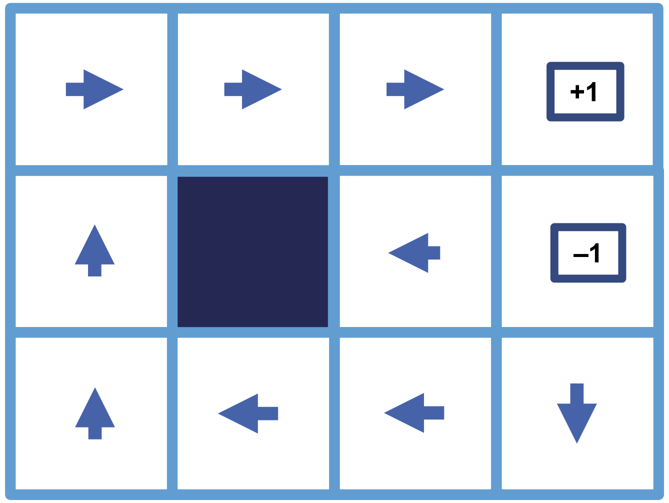


r is the reward per action

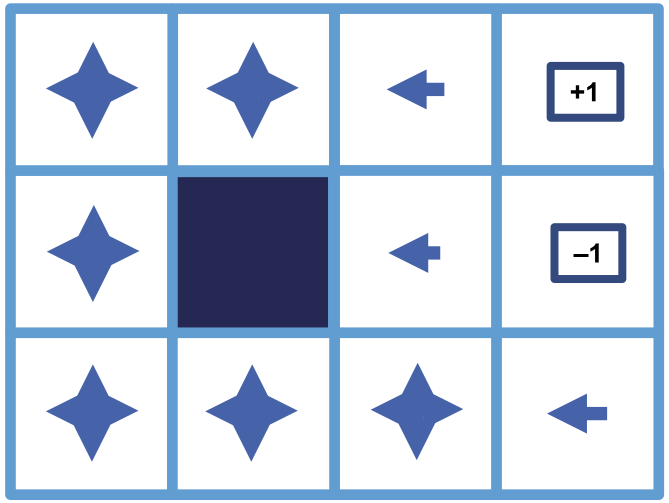
r < -1.65



-0.03 < r < 0



r > 0



Sequences of Rewards

- The performance of an agent in an MDP is the sum of the rewards for the transitions it takes.

  - $U_h([s_0, a_0, s_1, a_1 \ldots, s_n])$

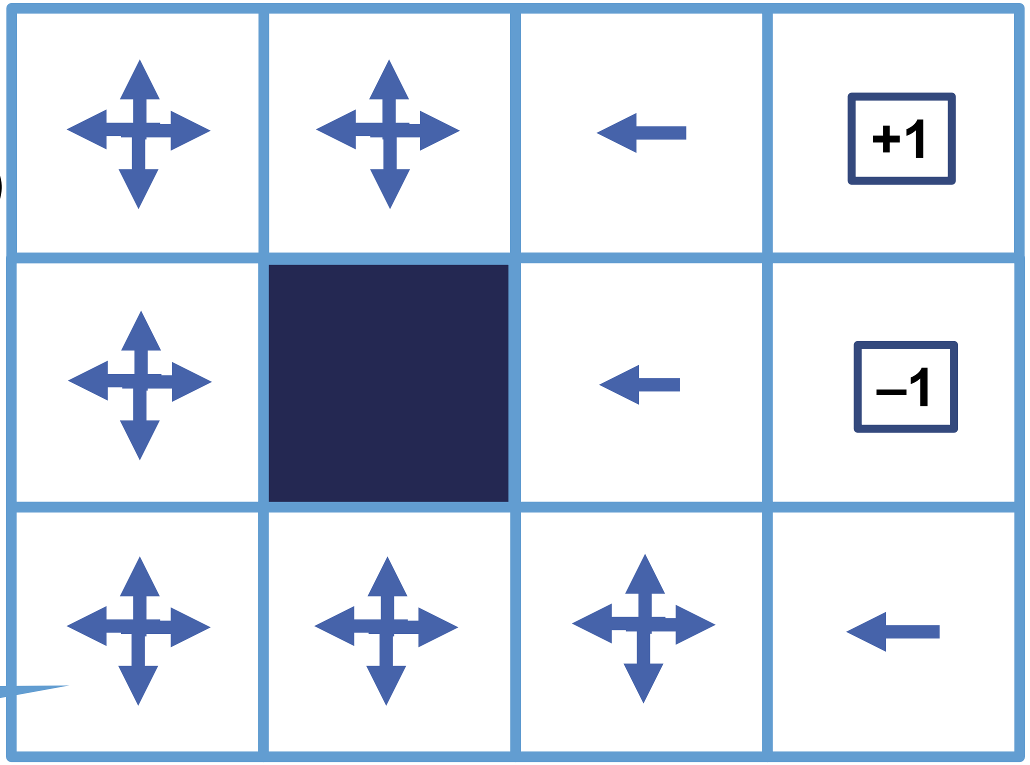
$= R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + \ldots + R(s_{n-1}, a_{n-1}, s_n)$

Utility function on an environment history.

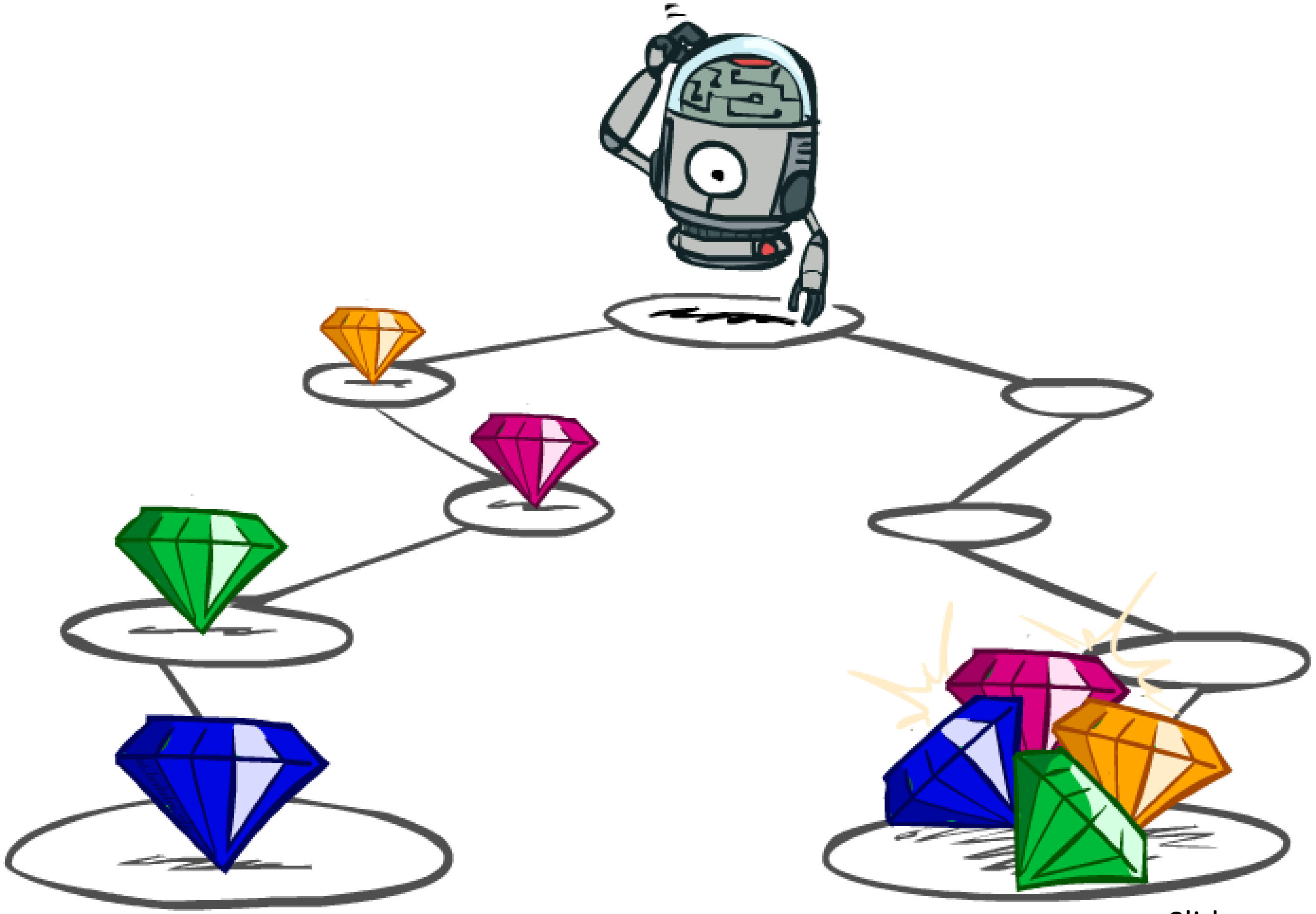
Sequence of states and actions

Bounce around forever, and avoid the exits …
infinite rewards!!

r > 0



Utilities of Sequences



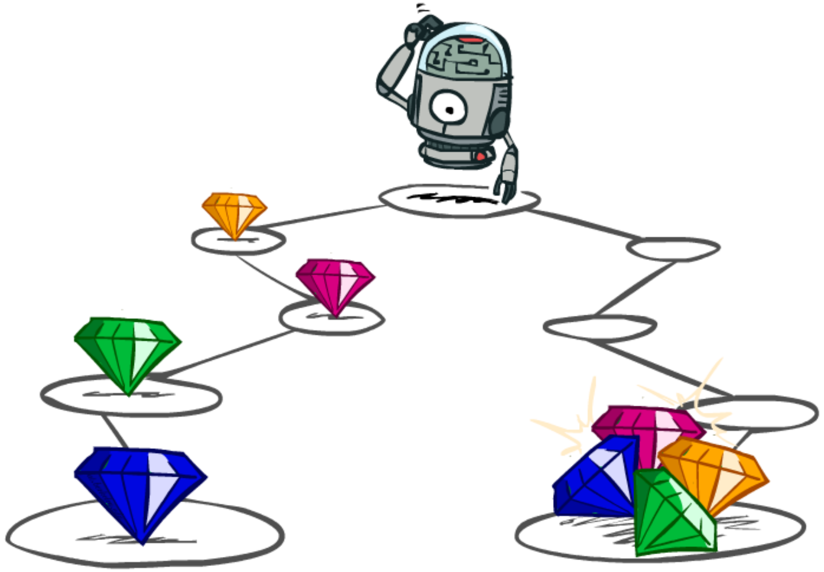
Slides courtesy of Dan Klein and Pieter Abbeel
University of California, Berkeley

Utilities of Sequences

- What preferences should an agent have over reward sequences?

- More or less? [1, 2, 2] or [2, 3, 4]

- Now or later? [0, 0, 1] or [1, 0, 0]



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

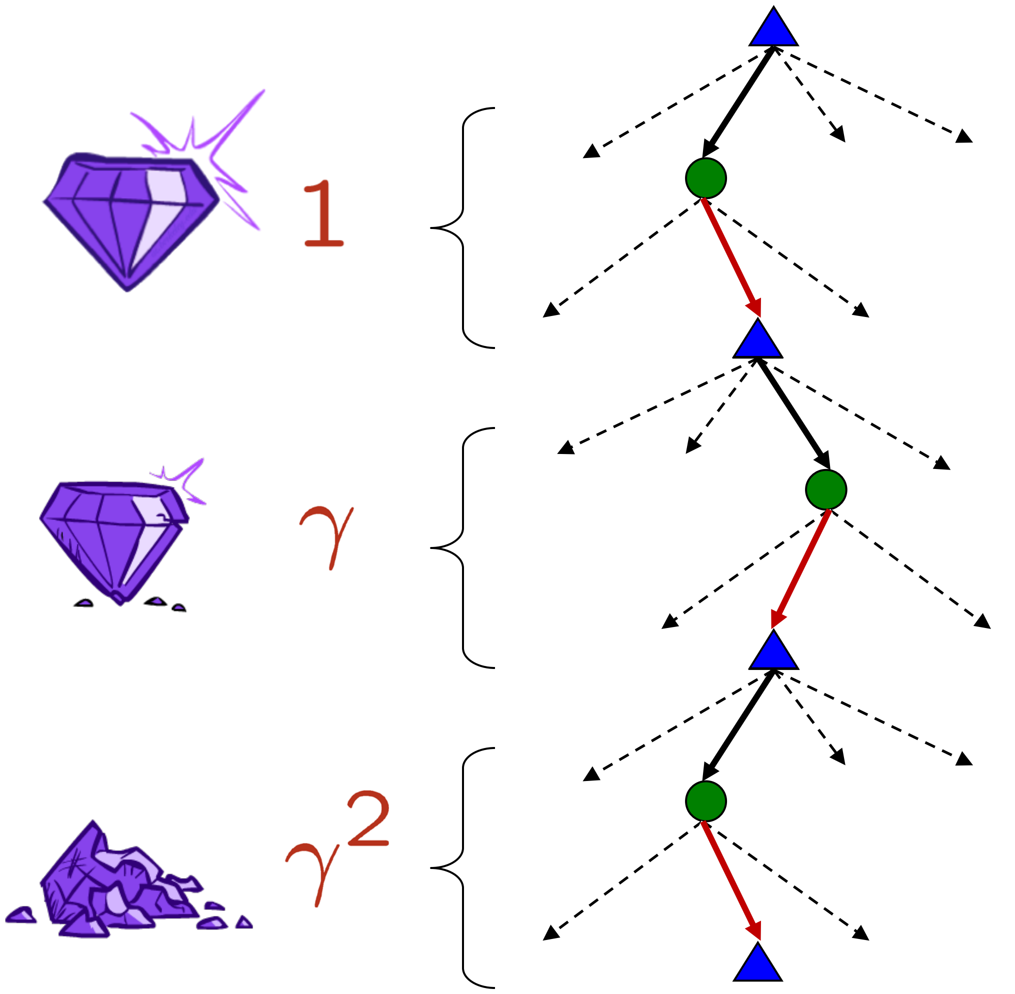
Worth Next Step

γ^2

Worth In Two Steps

Discounting

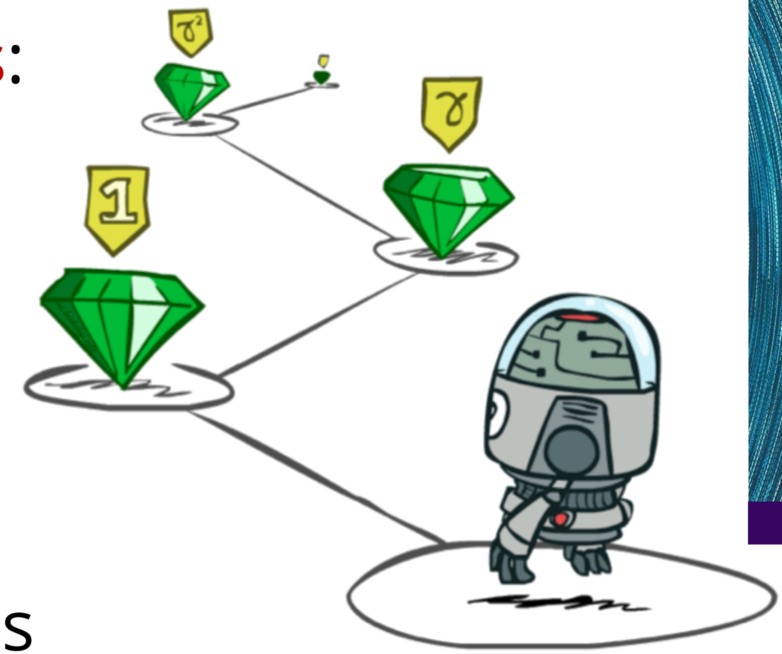
- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - $U([1,2,3]) = 0.5^0*1 + 0.5^1*2 + 0.5^2*3$
    $= 1*1 + 0.5*2 + 0.25*3$
  - $U([1,2,3]) < U([3,2,1])$



Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

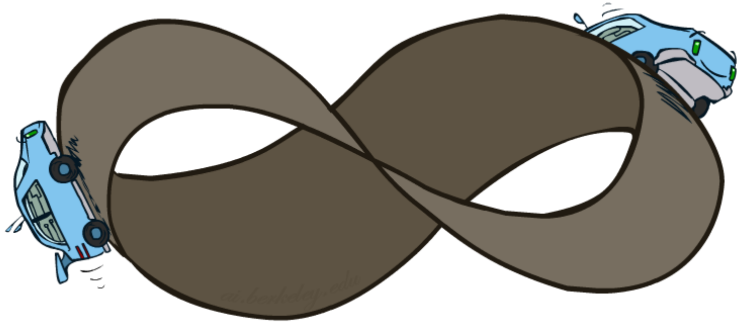


- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

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 $0 < \gamma < 1$

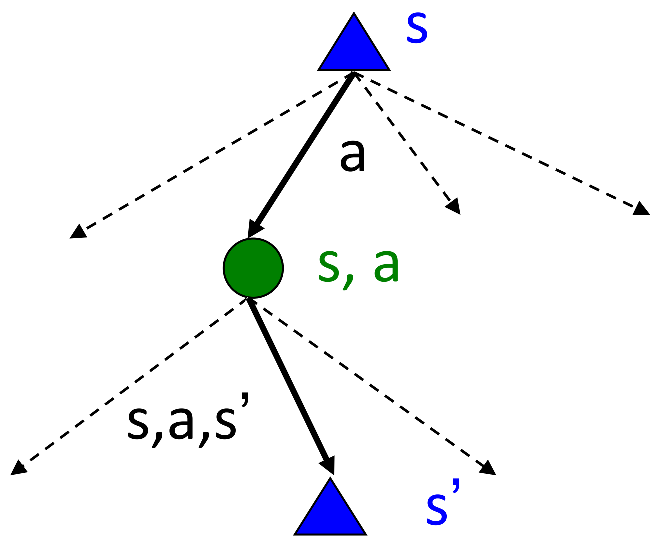
$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

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

  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Recap: Defining MDPs

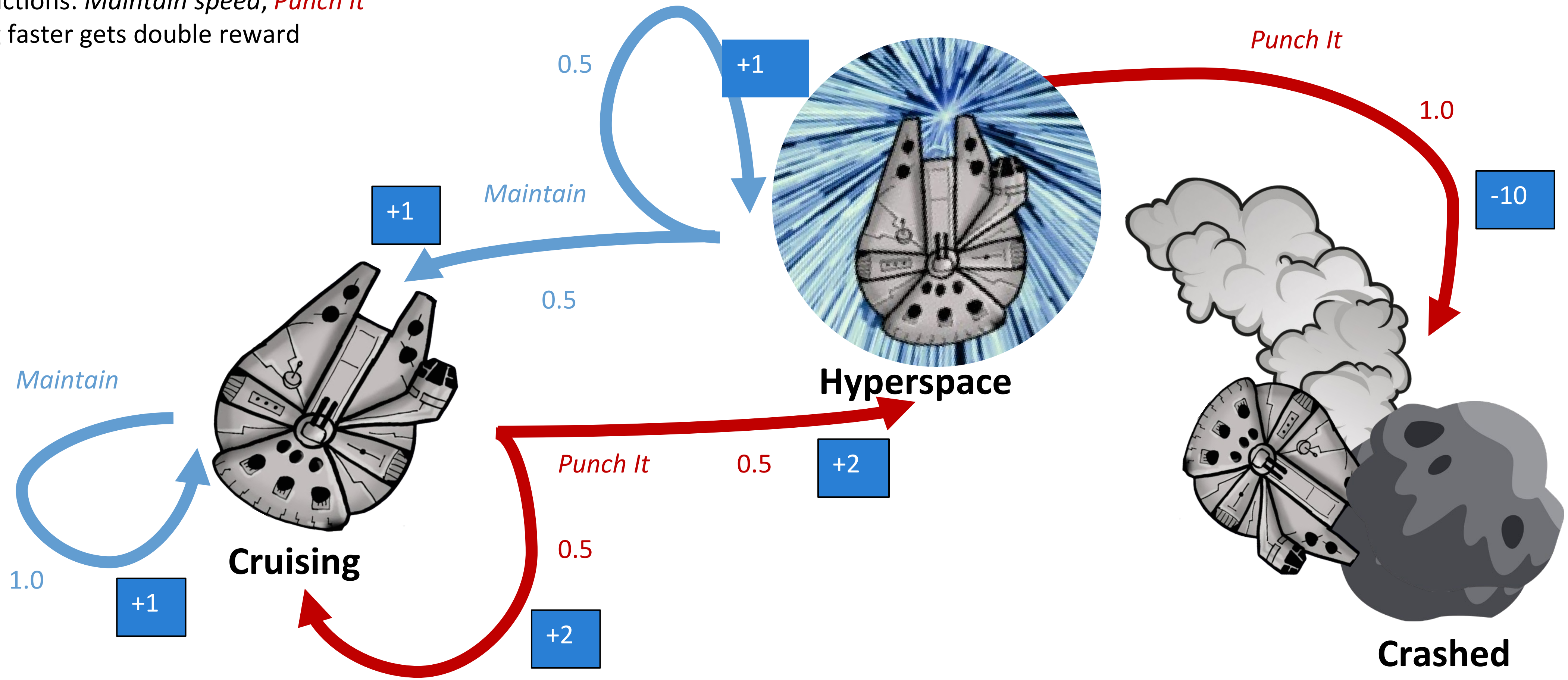
- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

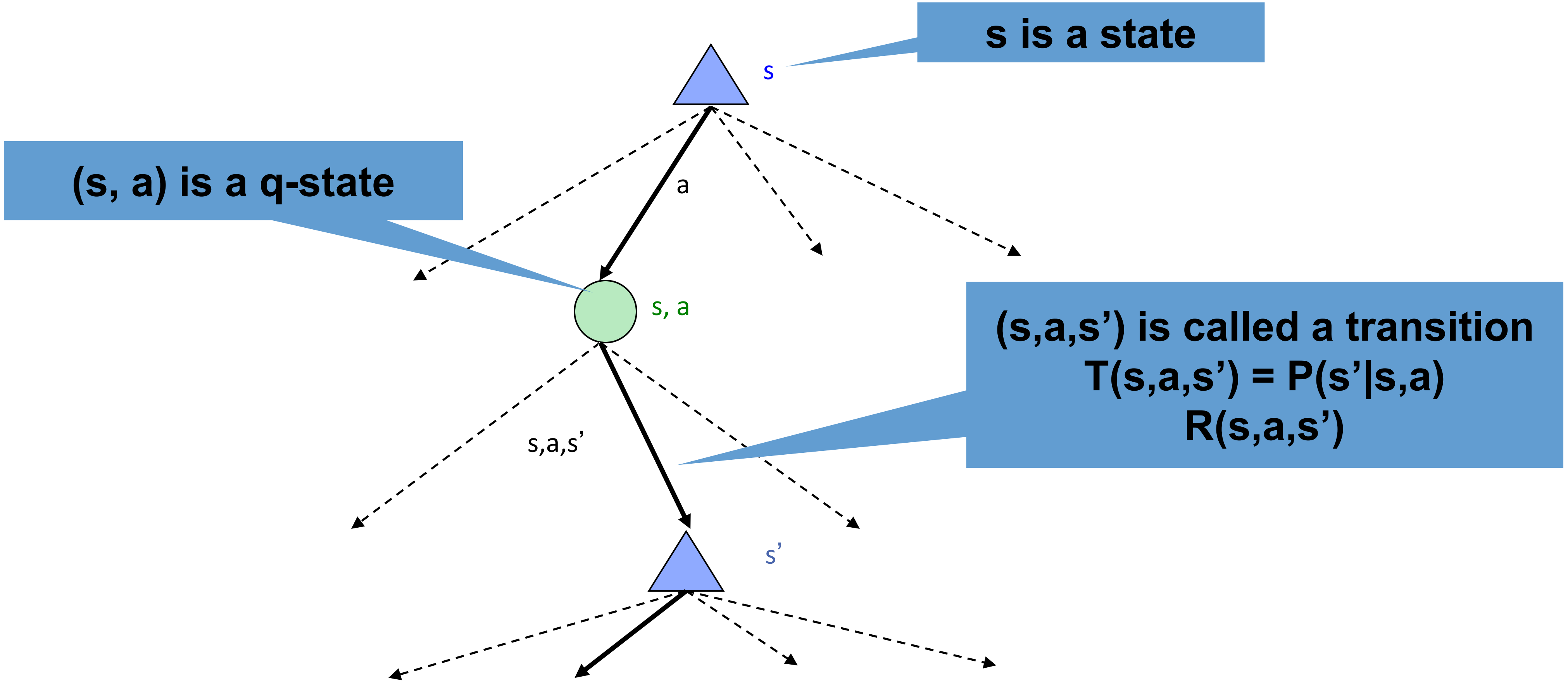
Example Hyperdrive MDP

The Millennium Falcon needs to travel far far away, quickly
Three states: Cruising, Hyperspace, Crashed
Two actions: *Maintain speed*, *Punch it*
Going faster gets double reward

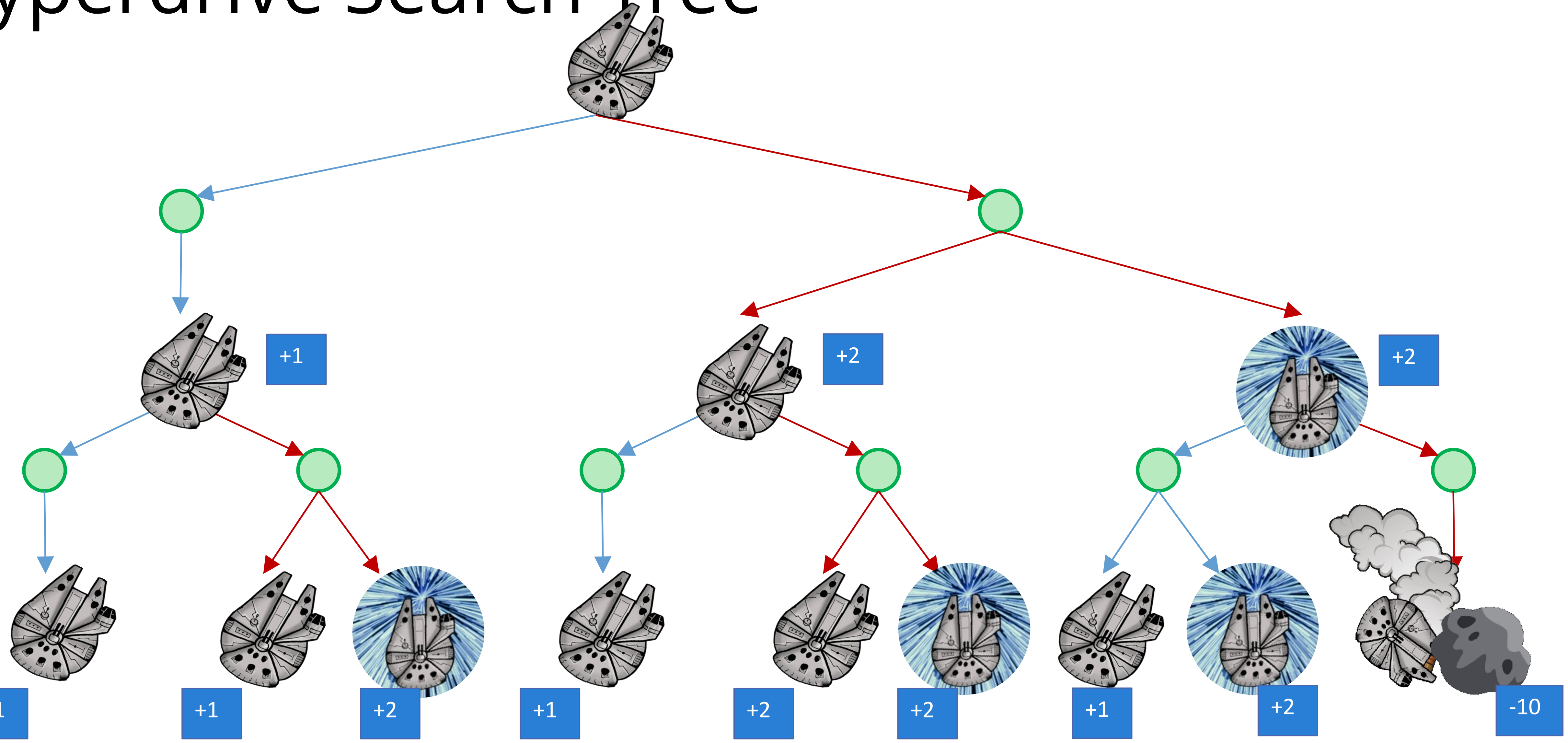


MDP Search Trees

- Each MDP state projects an expectimax-like search tree



Hyperdrive Search Tree



Optimal Quantities

- The value (utility) of a state s:

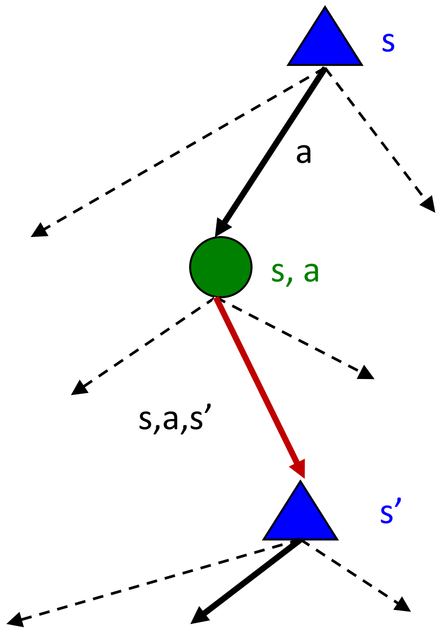
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s



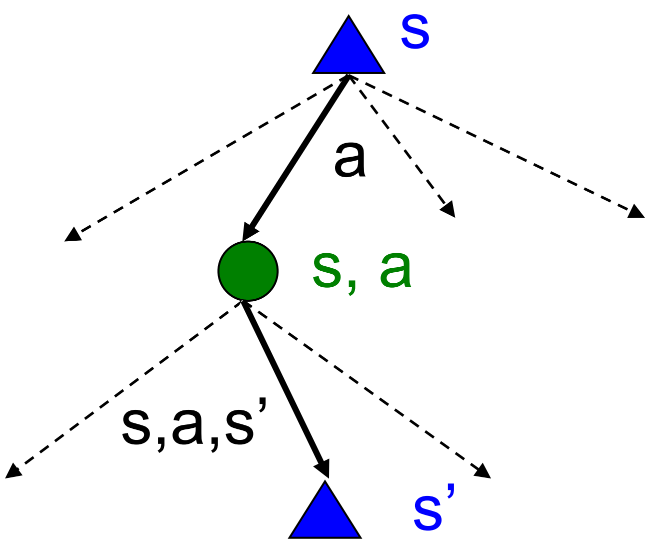
s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!



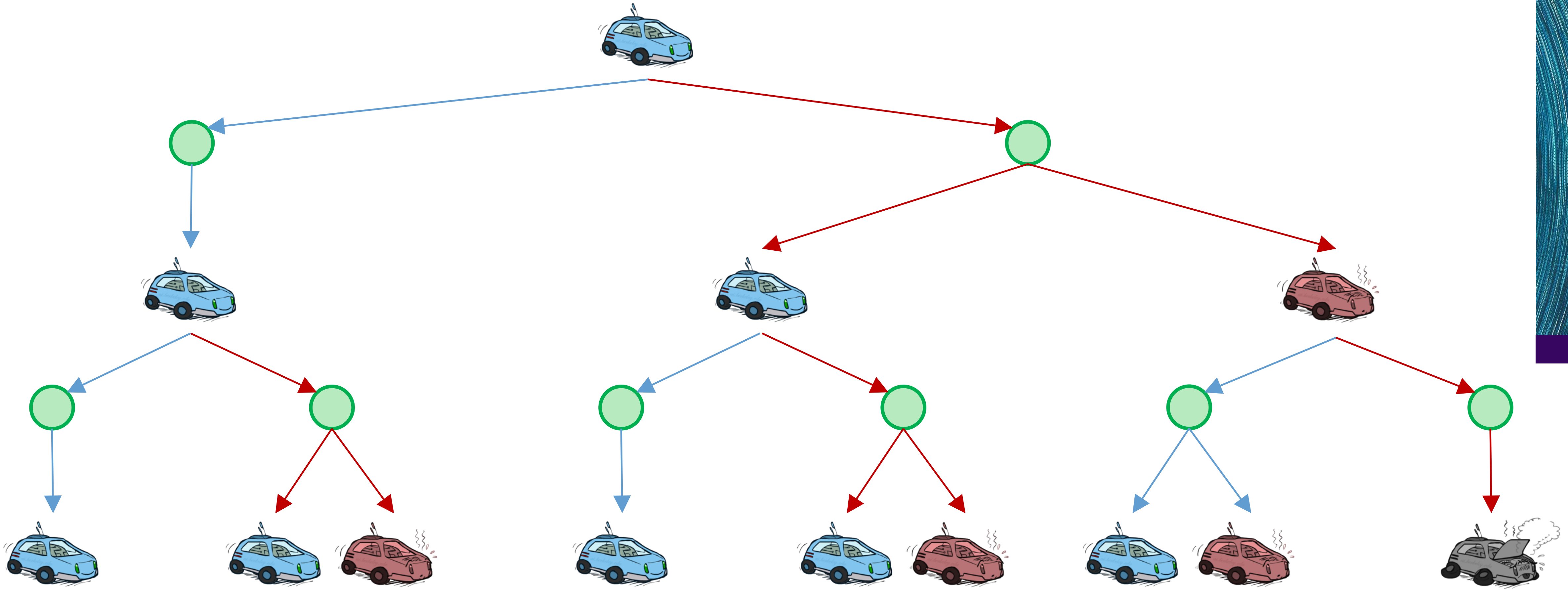
- Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a)$$

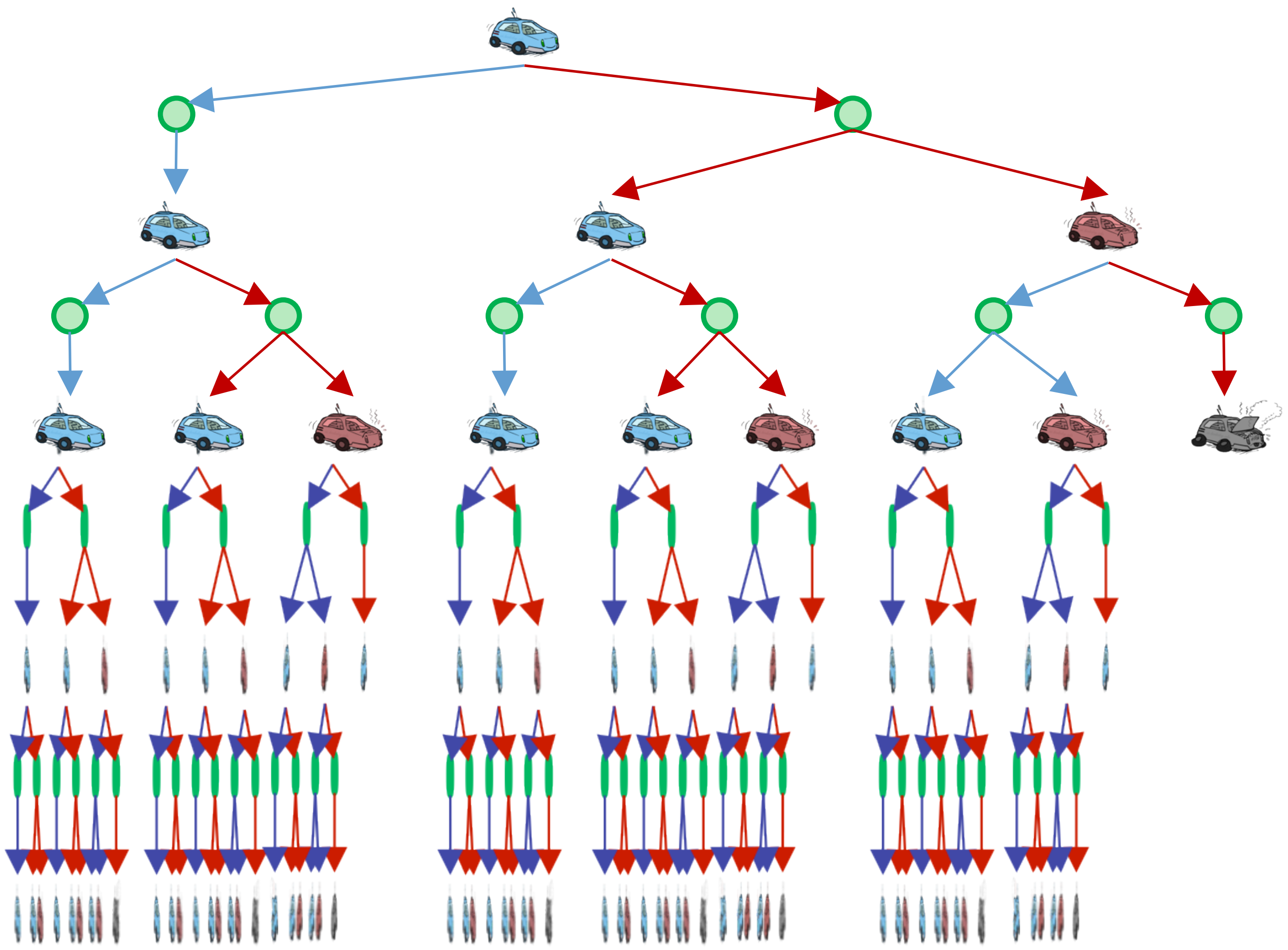
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Racing Search Tree

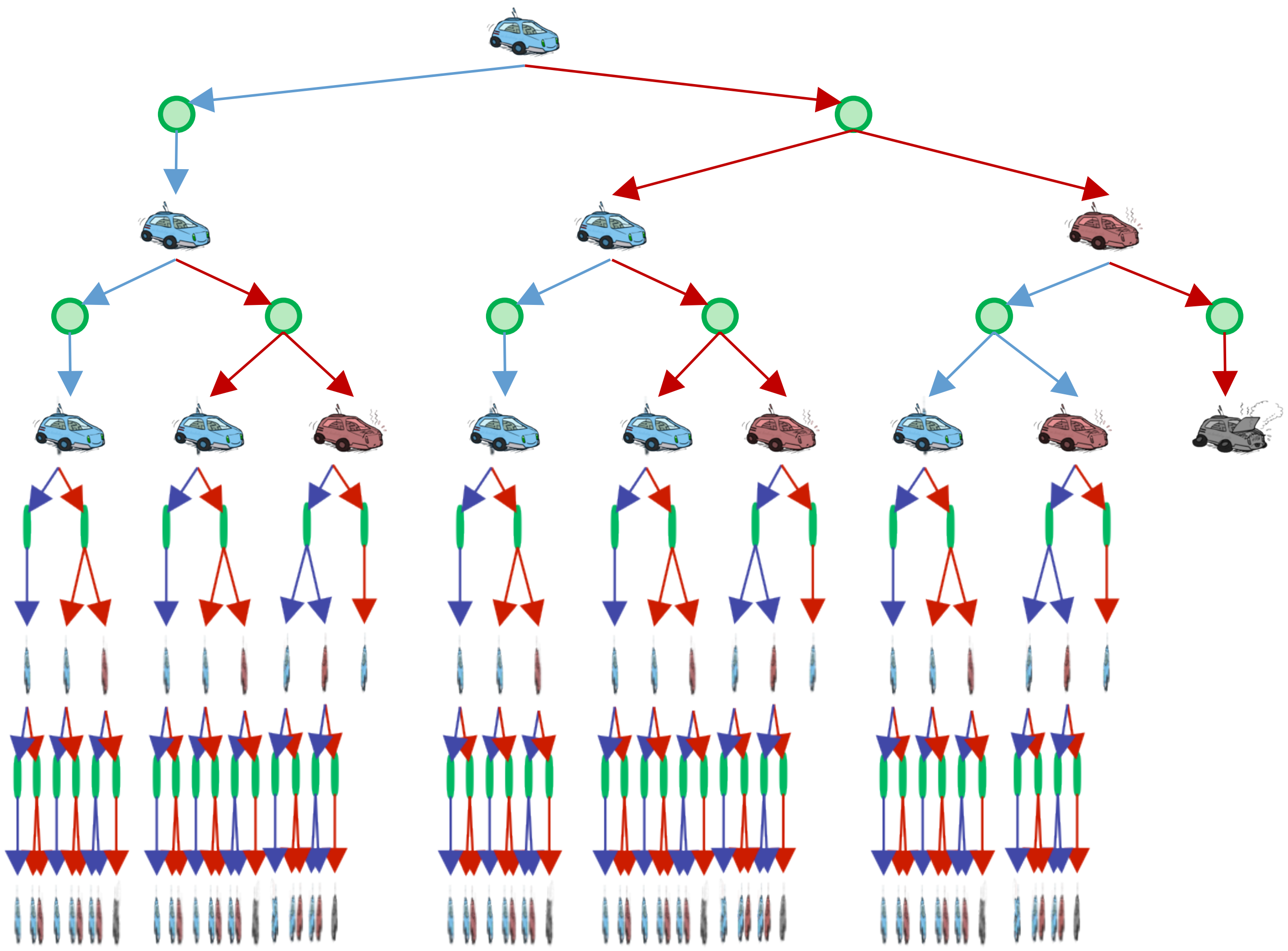


Racing Search Tree



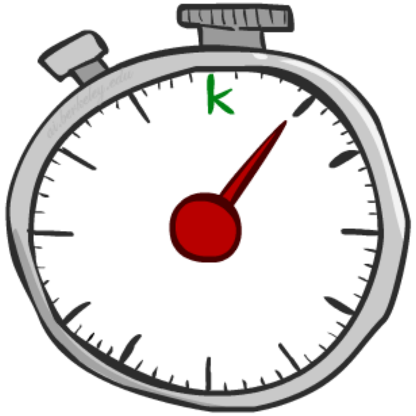
Racing Search Tree

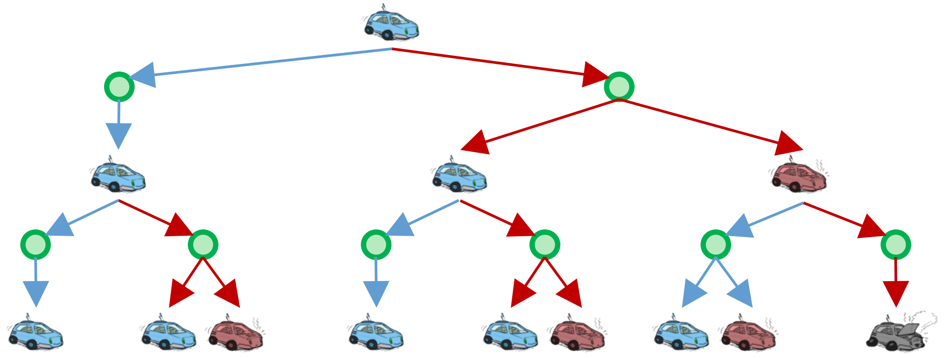
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$

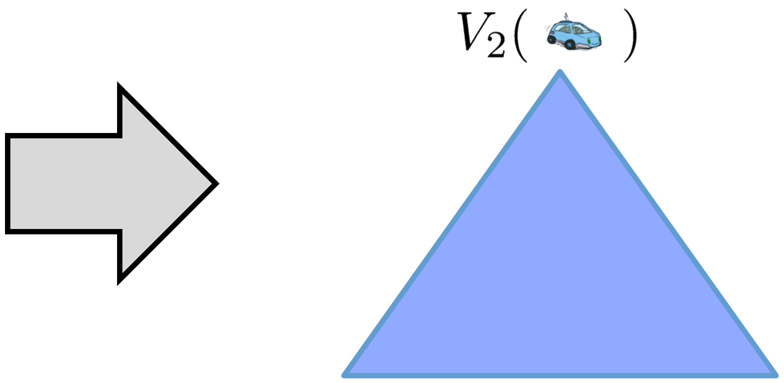


Time-Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s







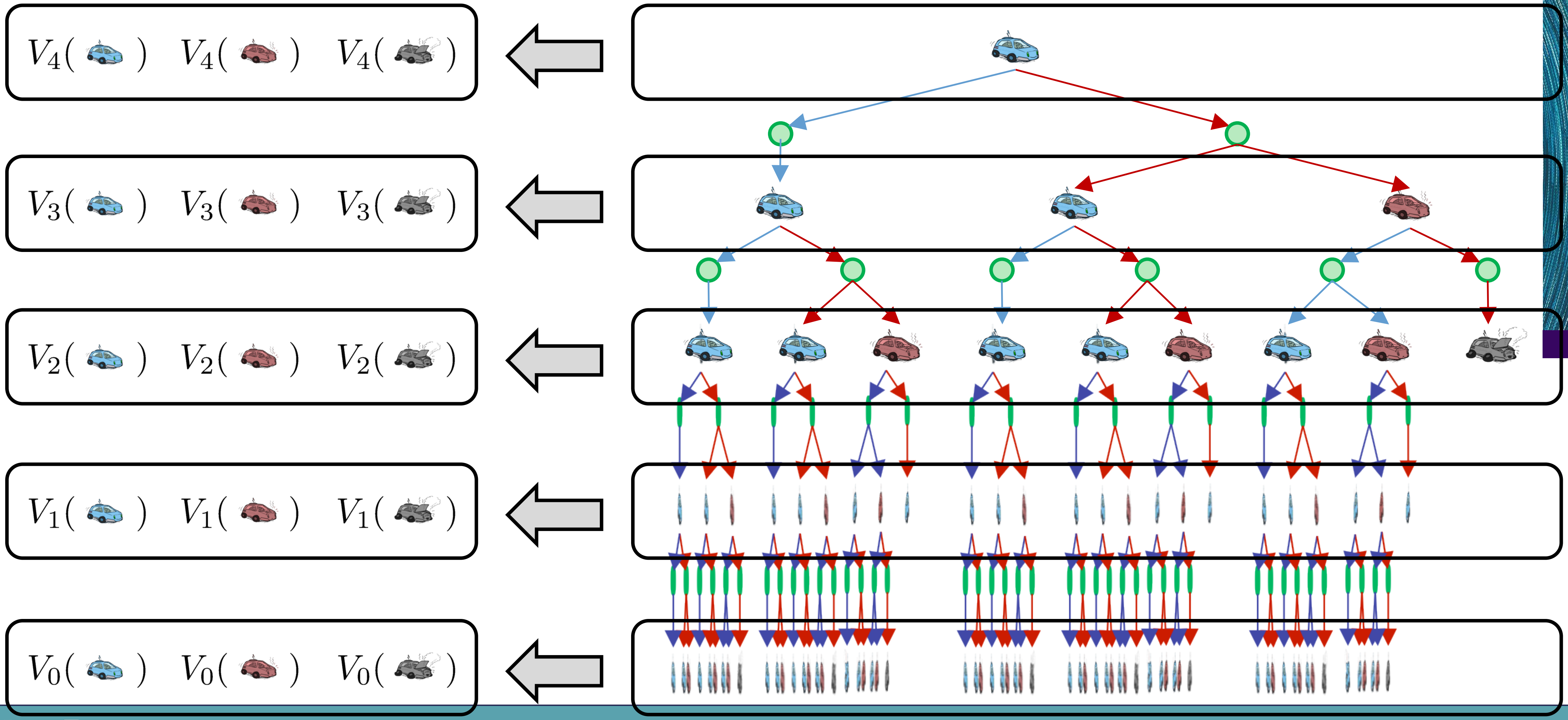
Reminders

- 21 days until the American election. I voted. Did you?
- Deadline to register to vote in PA is **Monday, Oct 19.**

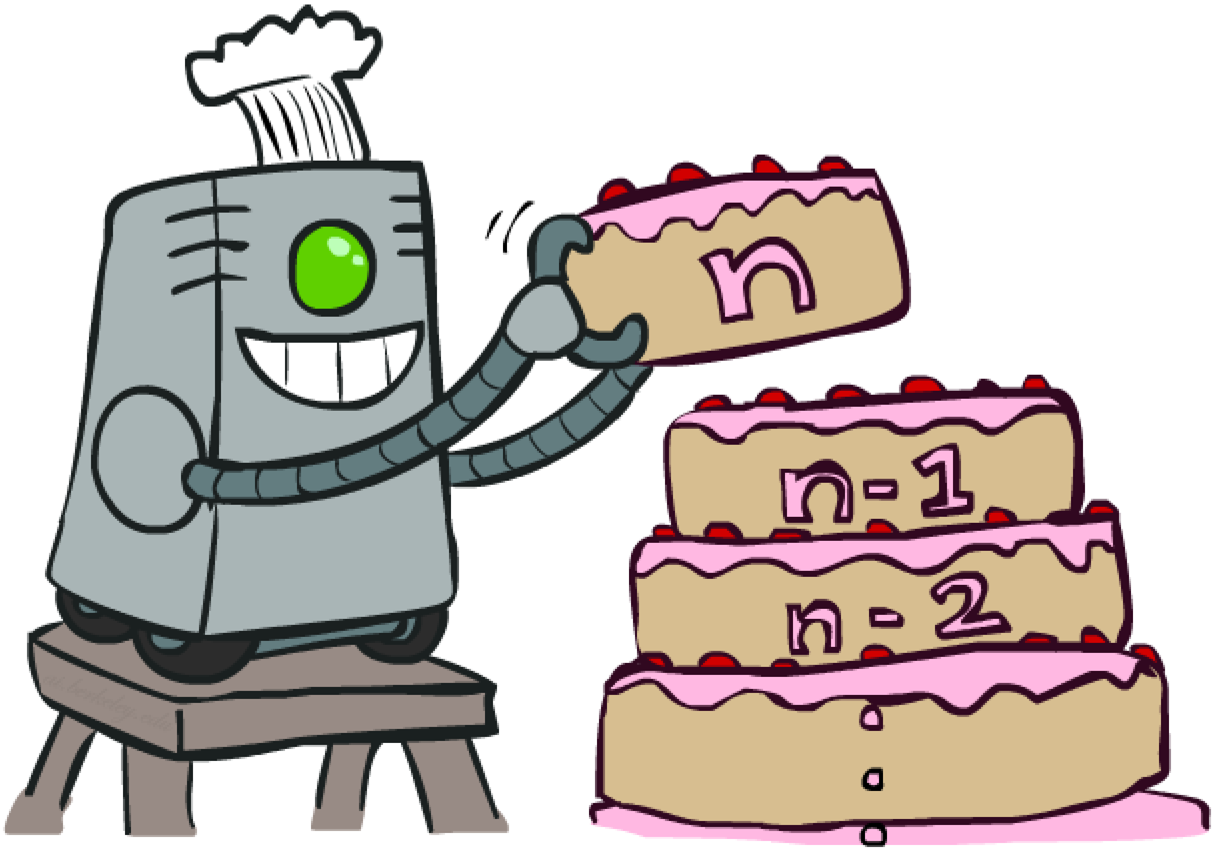
- HW4 due tonight at 11:59pm Eastern.
- Quiz 5 on Adversarial Search is due tomorrow.
- HW5 has been released. It will be due on Tuesday Oct 20.
- No lecture on Thursday.

- Midterm details:
- * No HW from Oct 20-27.
  * Tues Oct 20: Practice midterm released (for credit)
  * Saturday Oct 24: Practice midterm is due.
  * Midterm available Monday Oct 26 and Tuesday Oct 27.
  * 3 hour block. Open book, open notes, no collaboration.

Computing Time-Limited Values



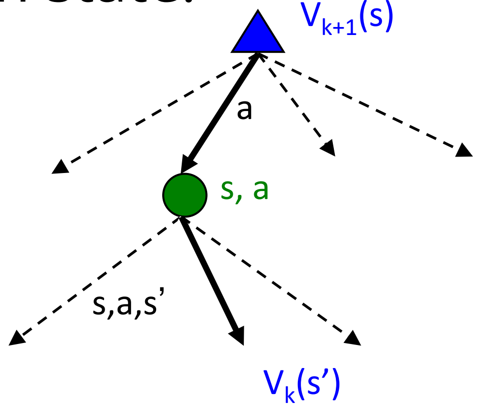
Value Iteration



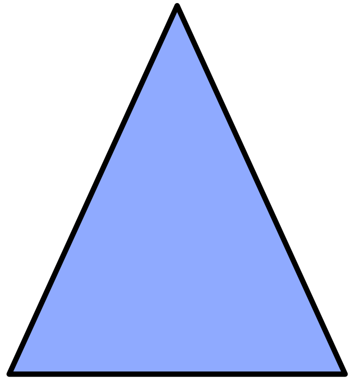
Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

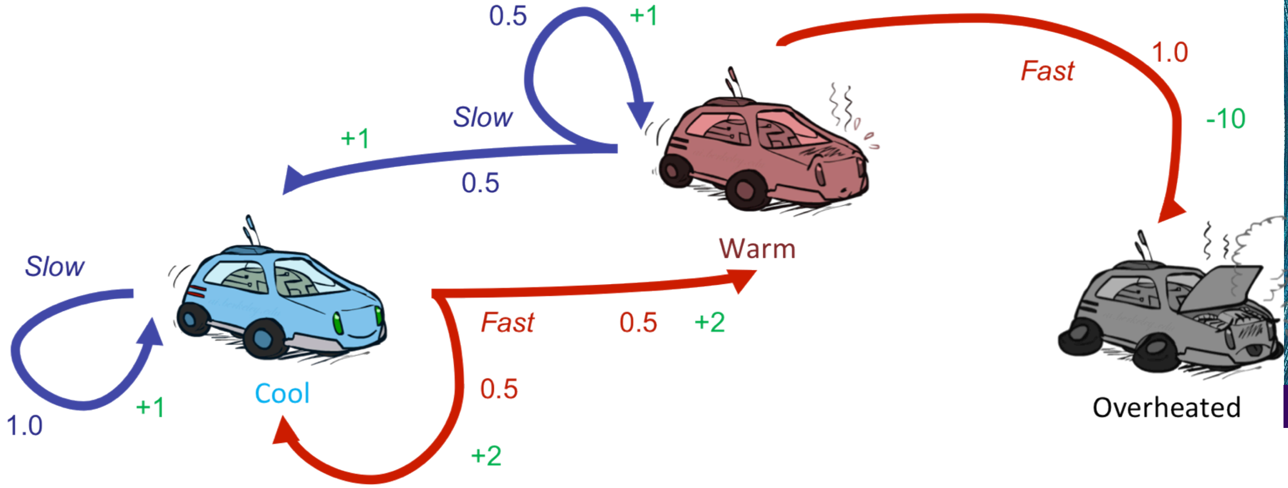


Example: Value Iteration

V_2

V_1

V_0

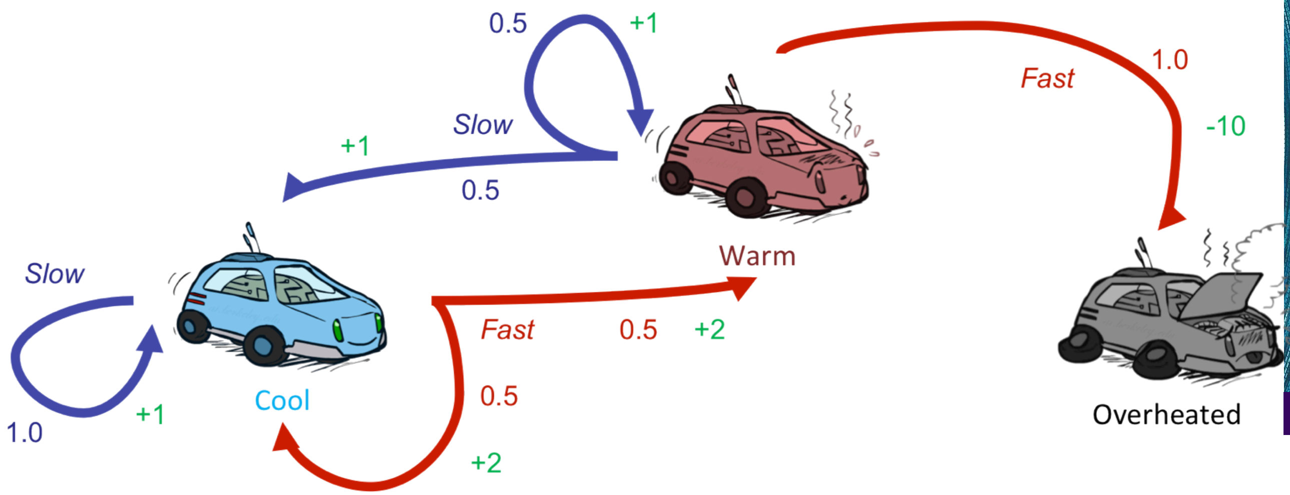


Assume no discount!

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Example: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

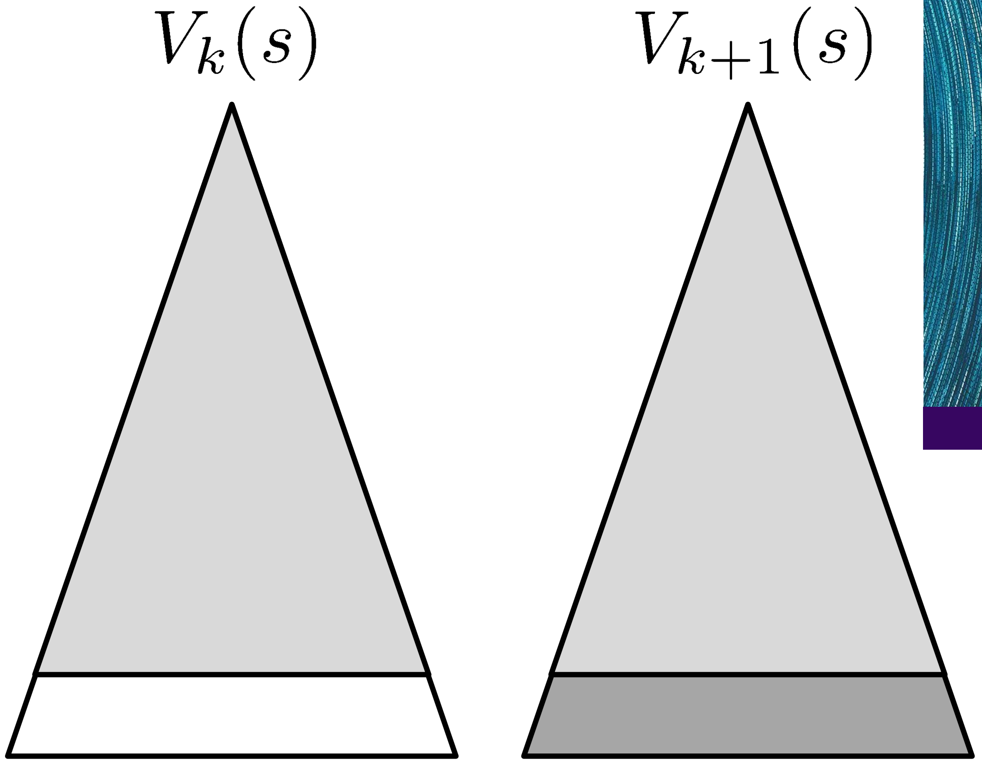
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Convergence*

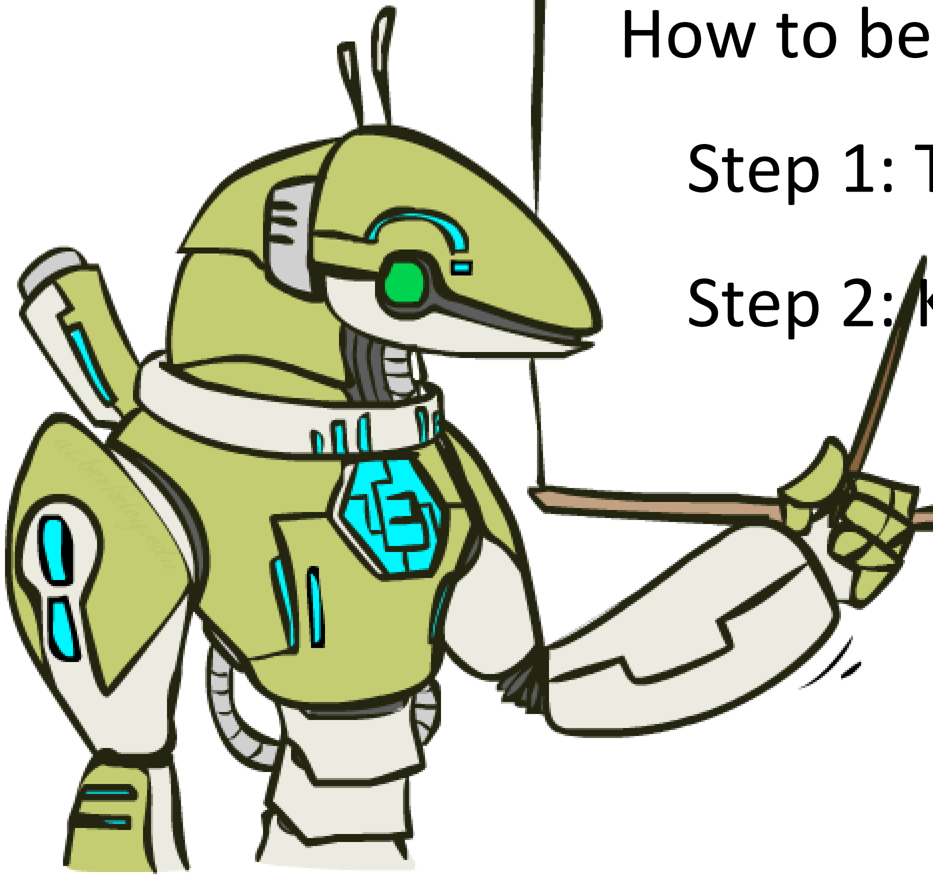
- How do we know the V_k vectors are going to converge?

- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values

- Case 2: If the discount is less than 1
  - Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most γ^k max|R| different
  - So as k increases, the values converge



The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

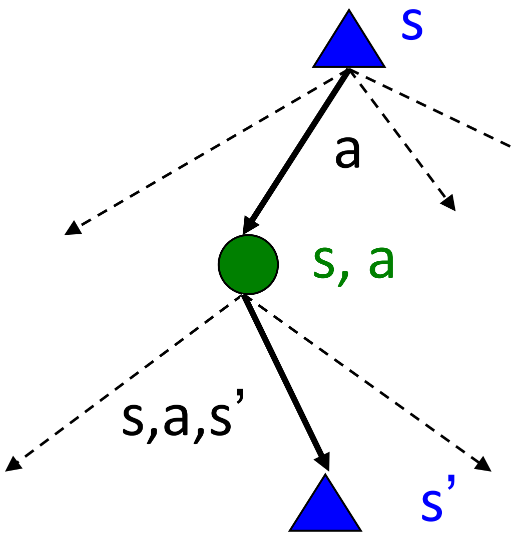
The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s,a)$$

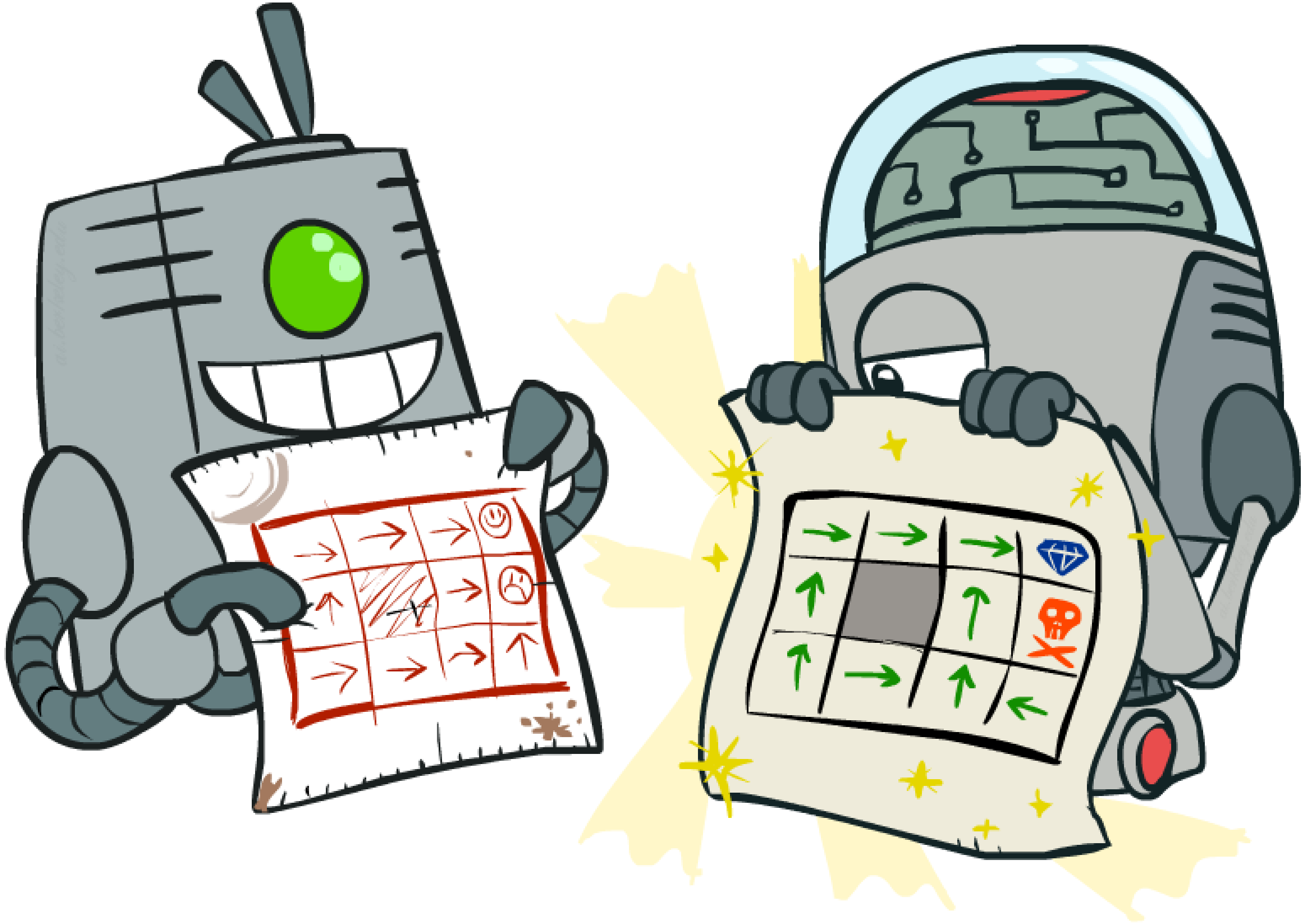
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

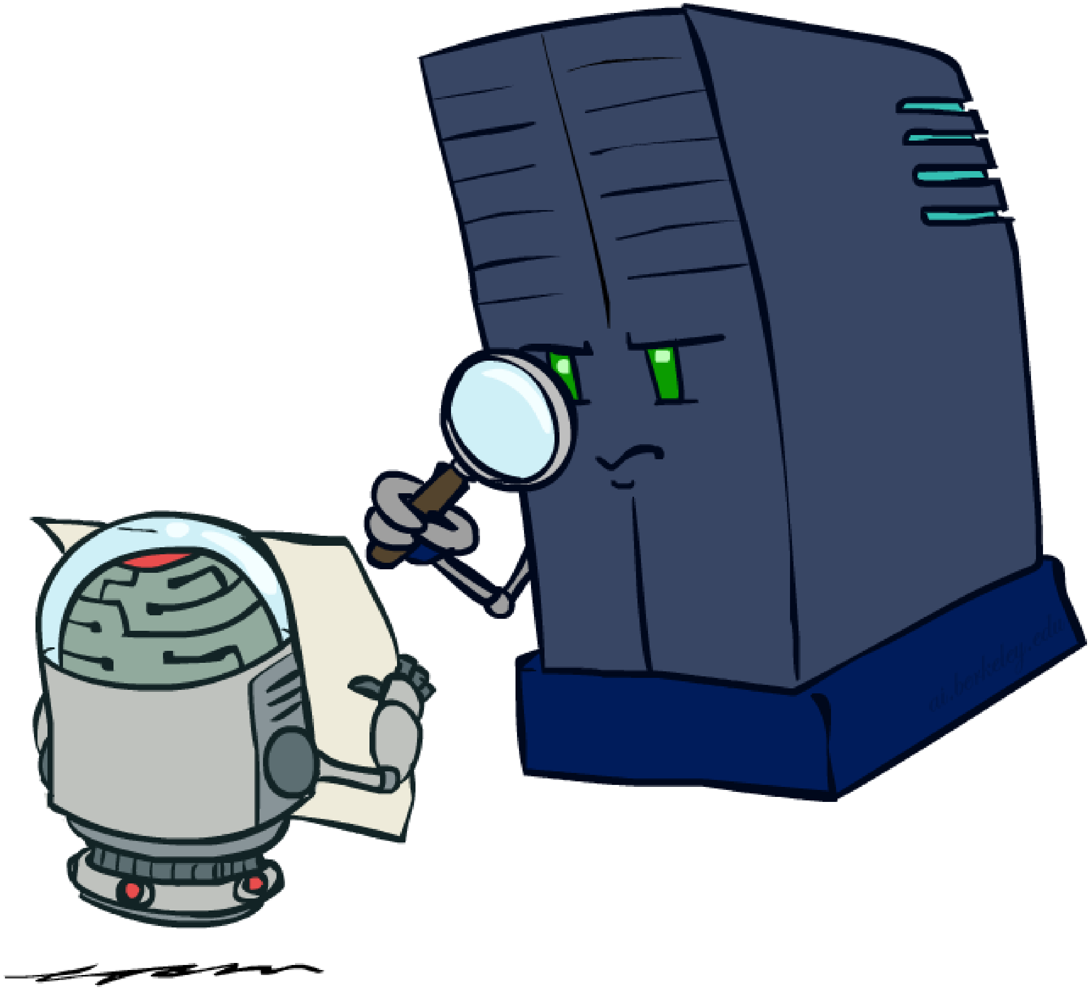


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

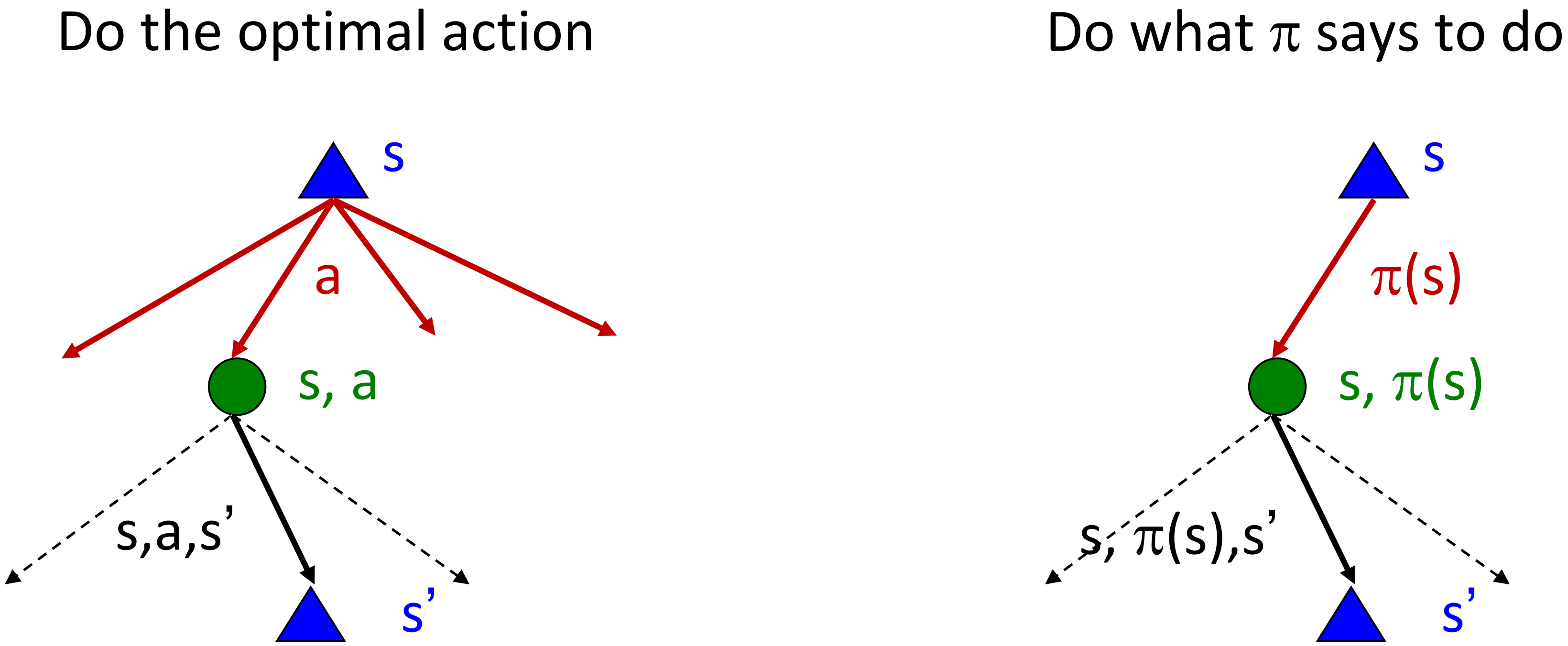
Policy Methods



Policy Evaluation



Fixed Policies



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

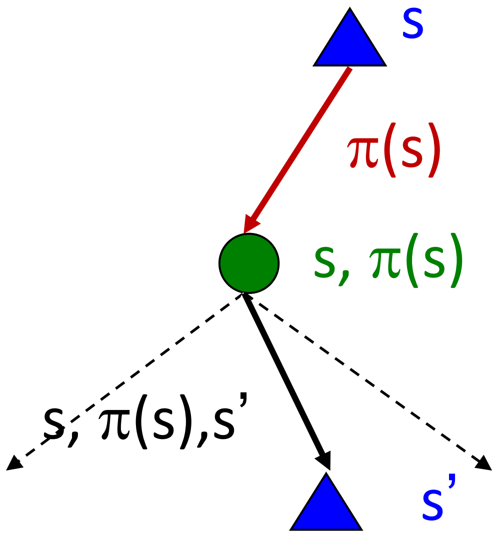
Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy π:
  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

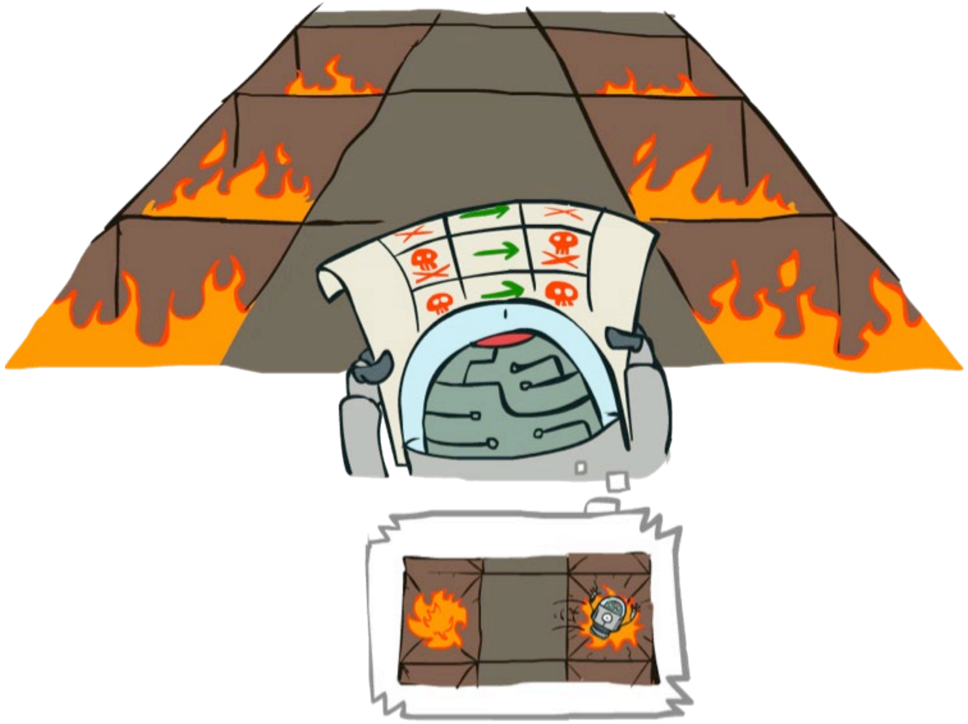
- Recursive relation (one-step look-ahead / Bellman equation):

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

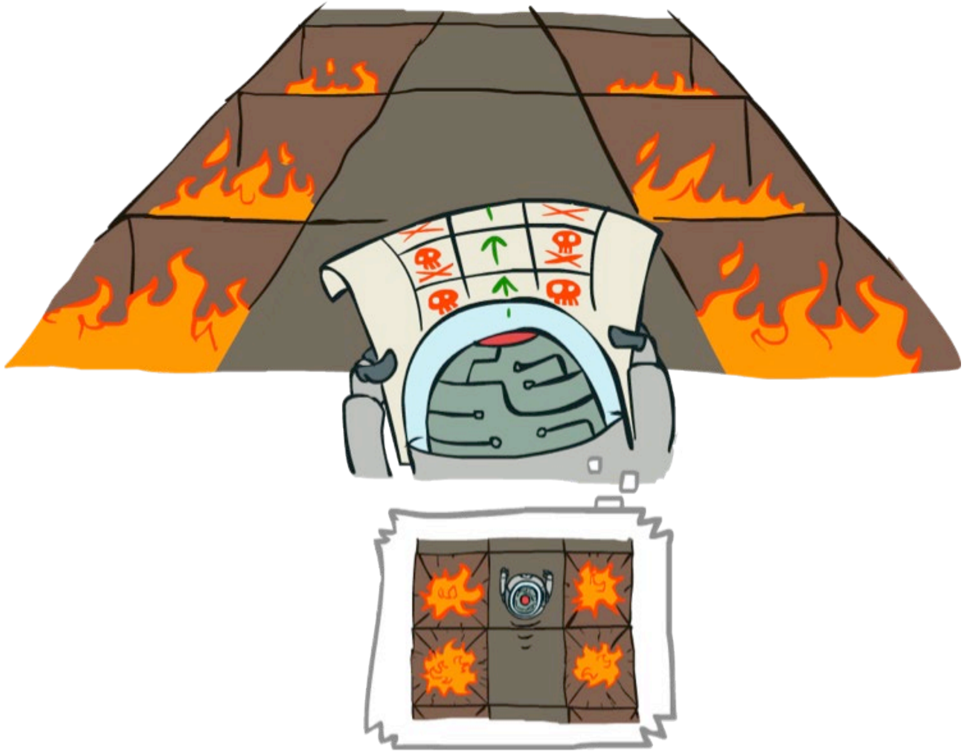


Example: Policy Evaluation

Always Go Right



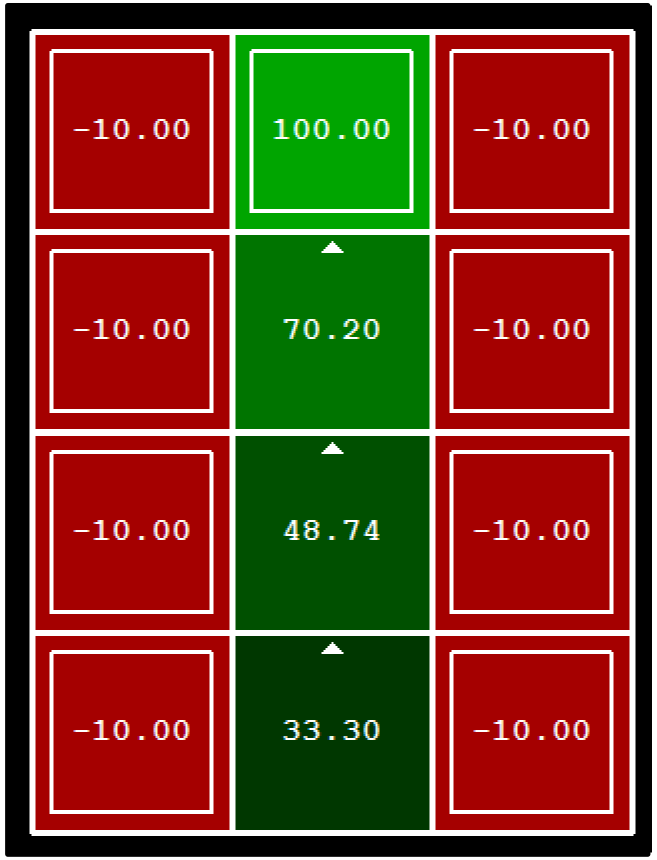
Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward



Policy Evaluation

- How do we calculate the V's for a fixed policy π?

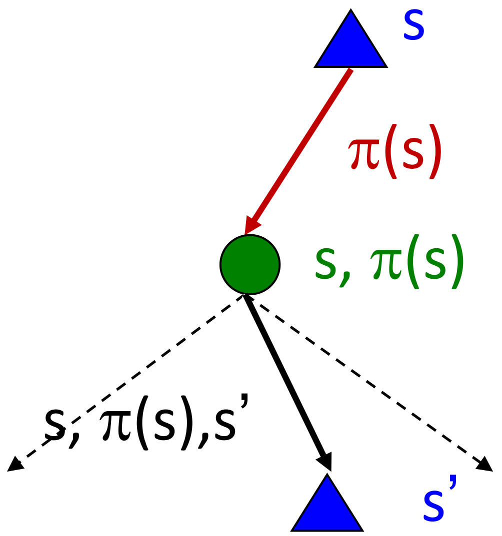
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$

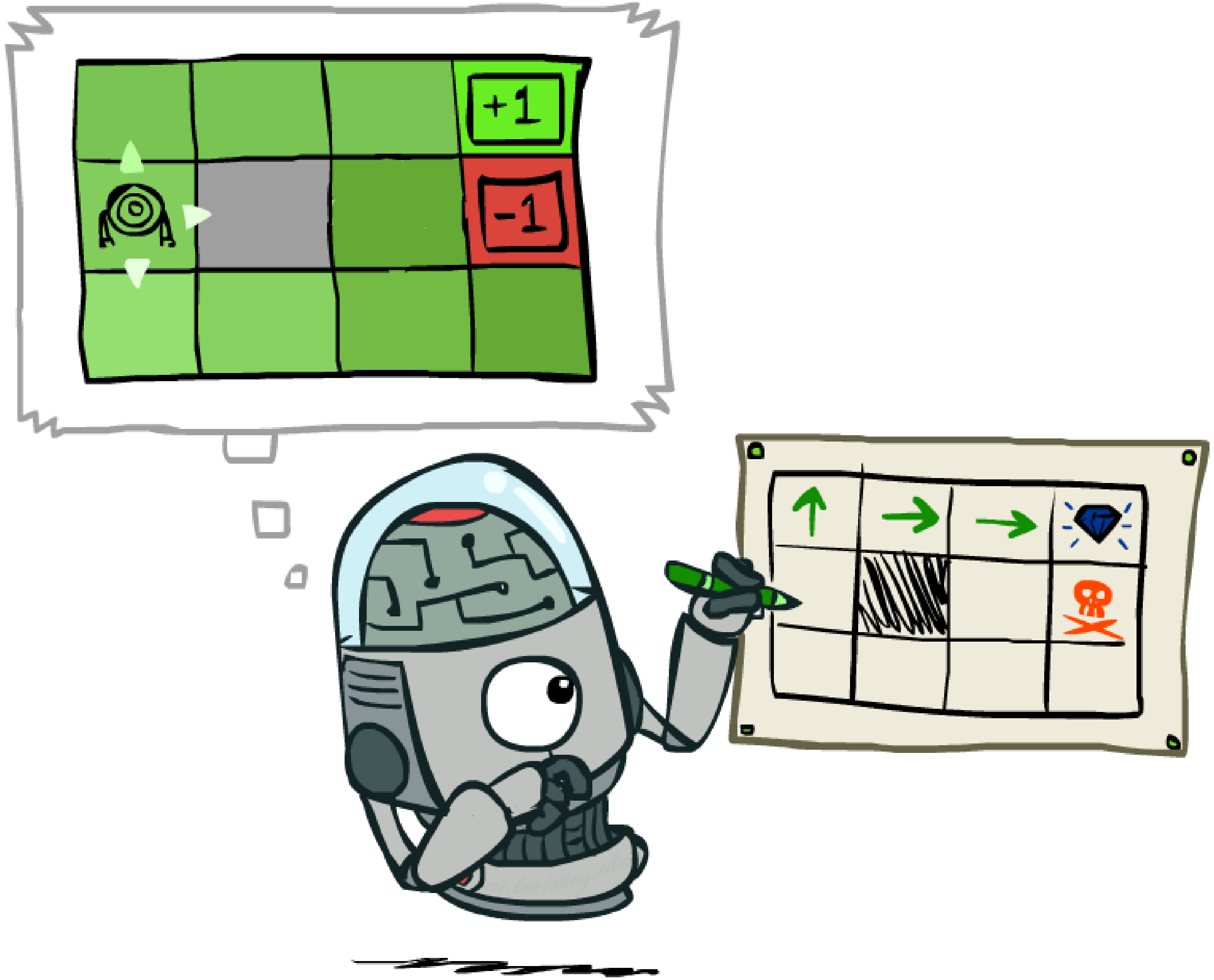
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$

- Efficiency: $O(S^2)$ per iteration

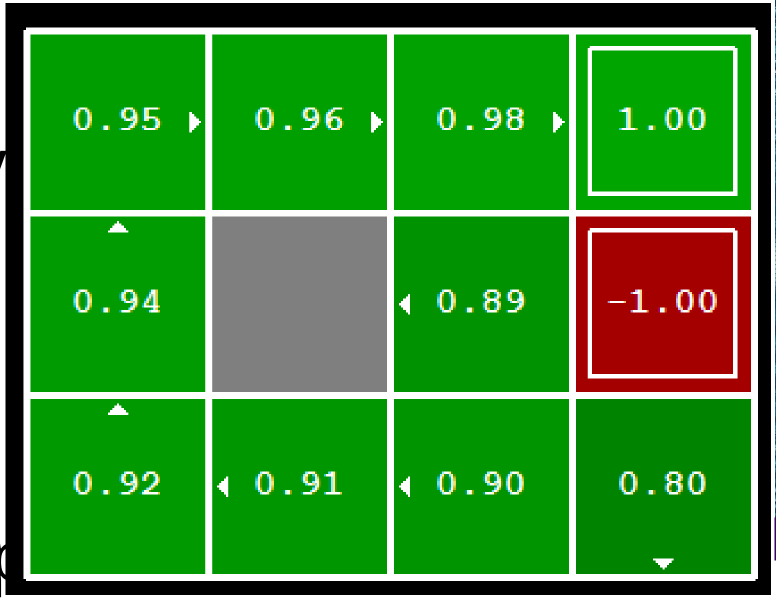
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)



Policy Extraction



Computing Actions from Values



- Let's imagine we have the optimal values V

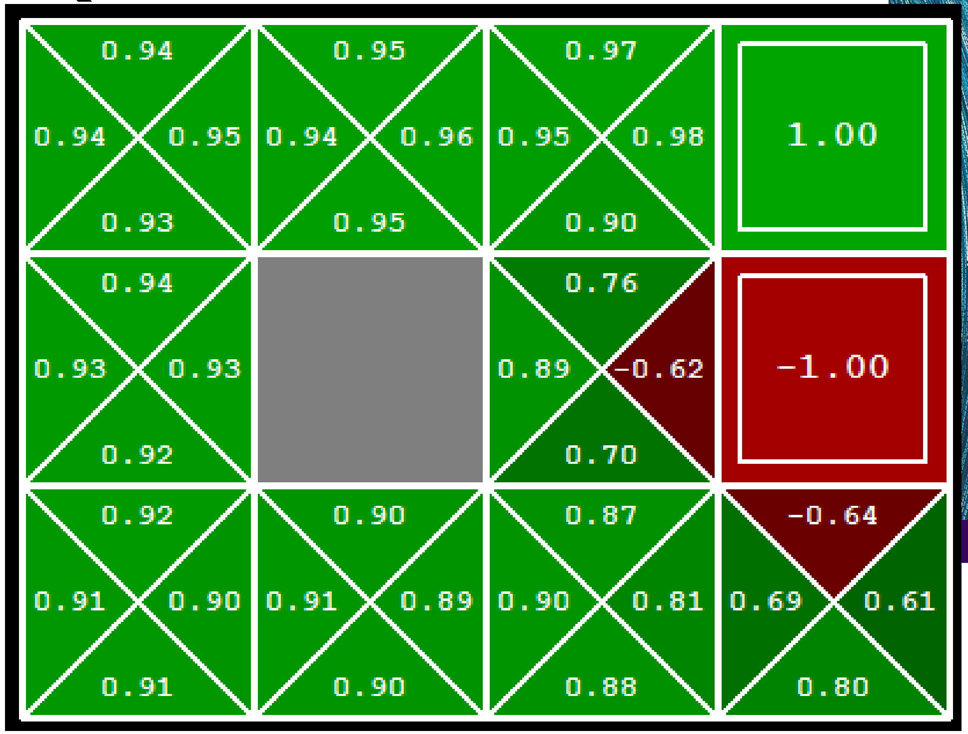
- How should we act?
  - It's not obvious!

- We need to do a mini-expectimax (one step

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

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

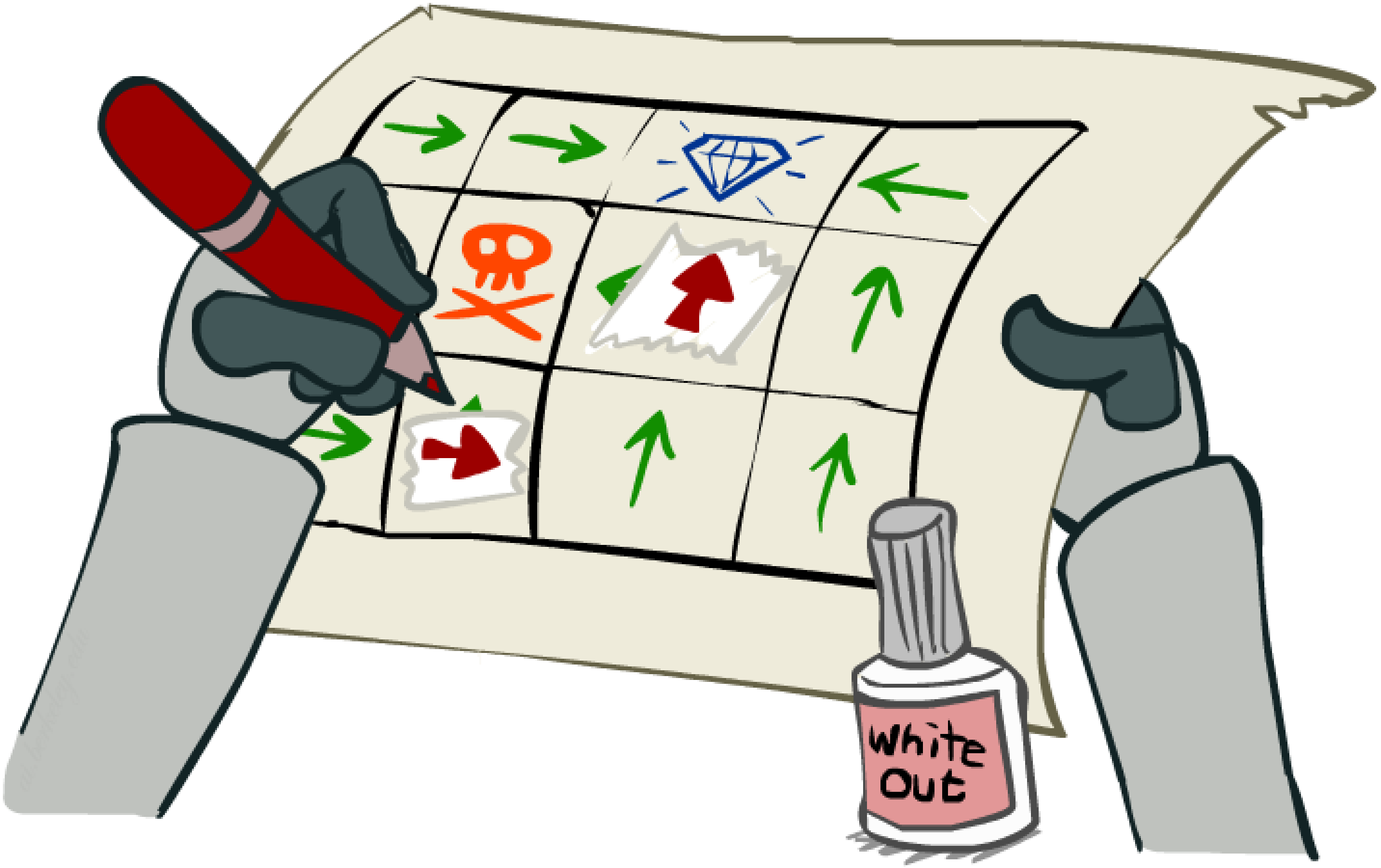


- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

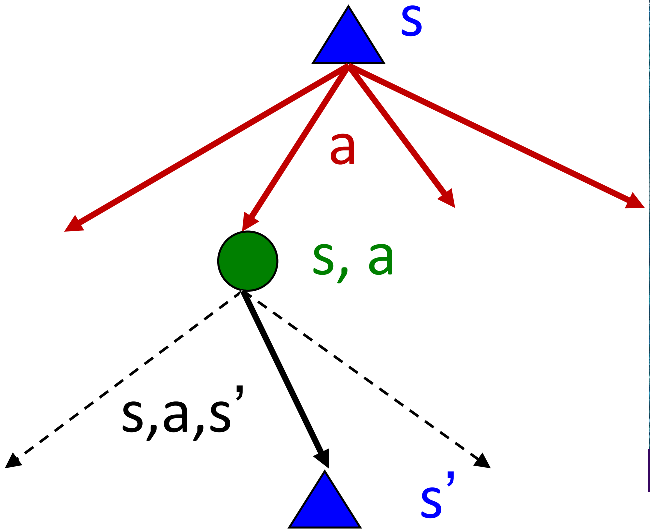
Policy Iteration



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



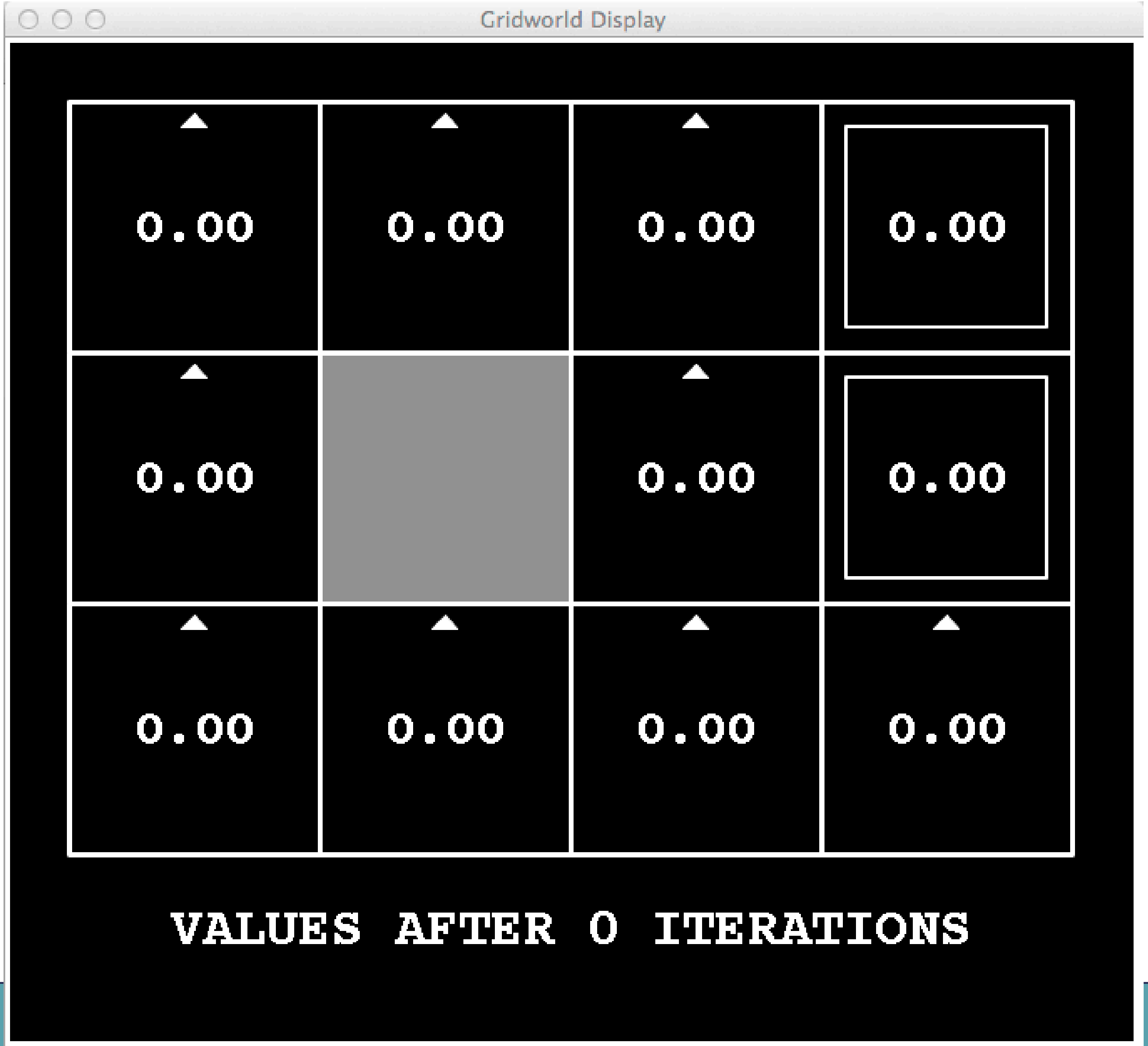
- Problem 1: It's slow – O(S^2A) per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

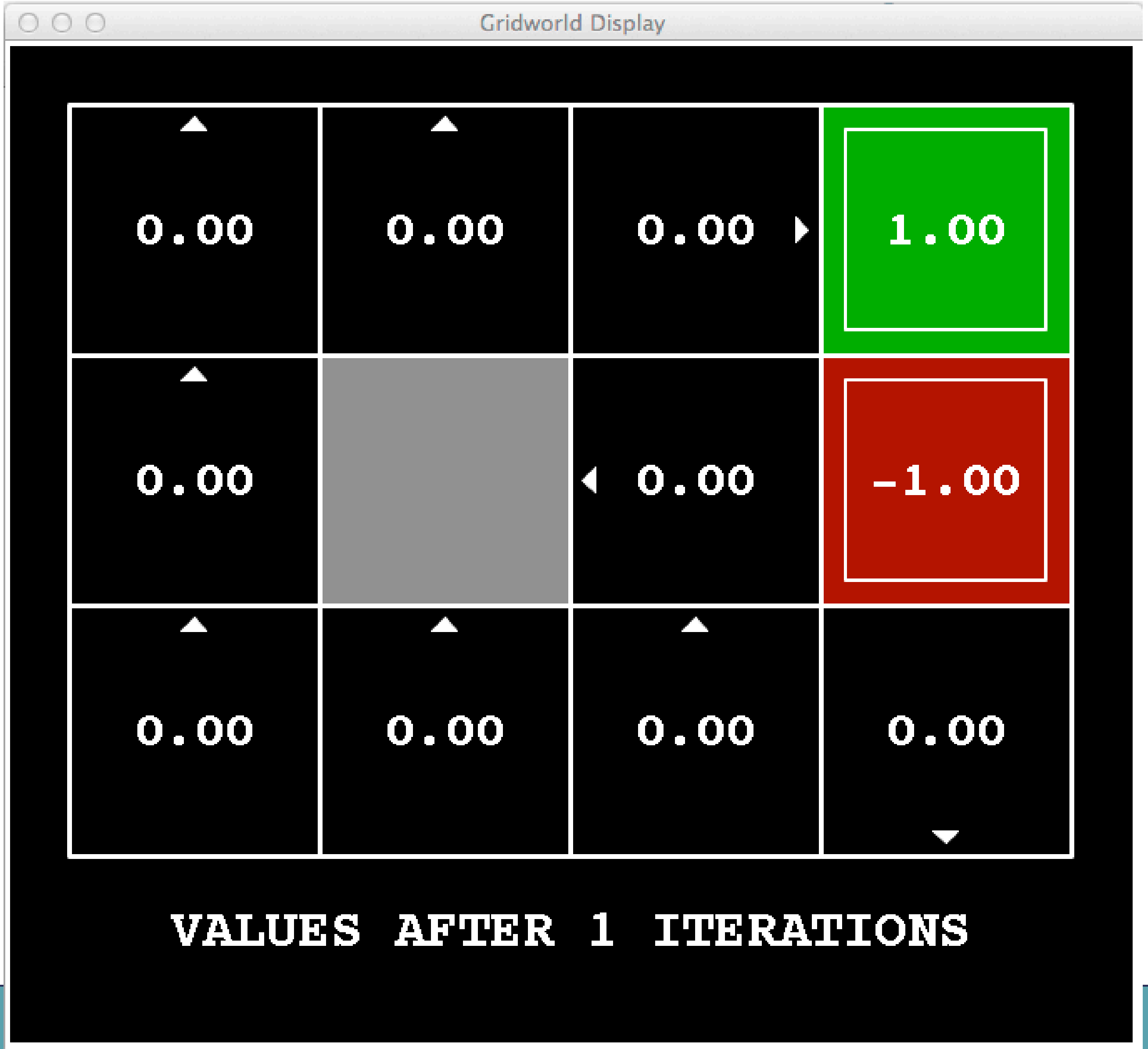
[Demo: value iteration (L9D2)]

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

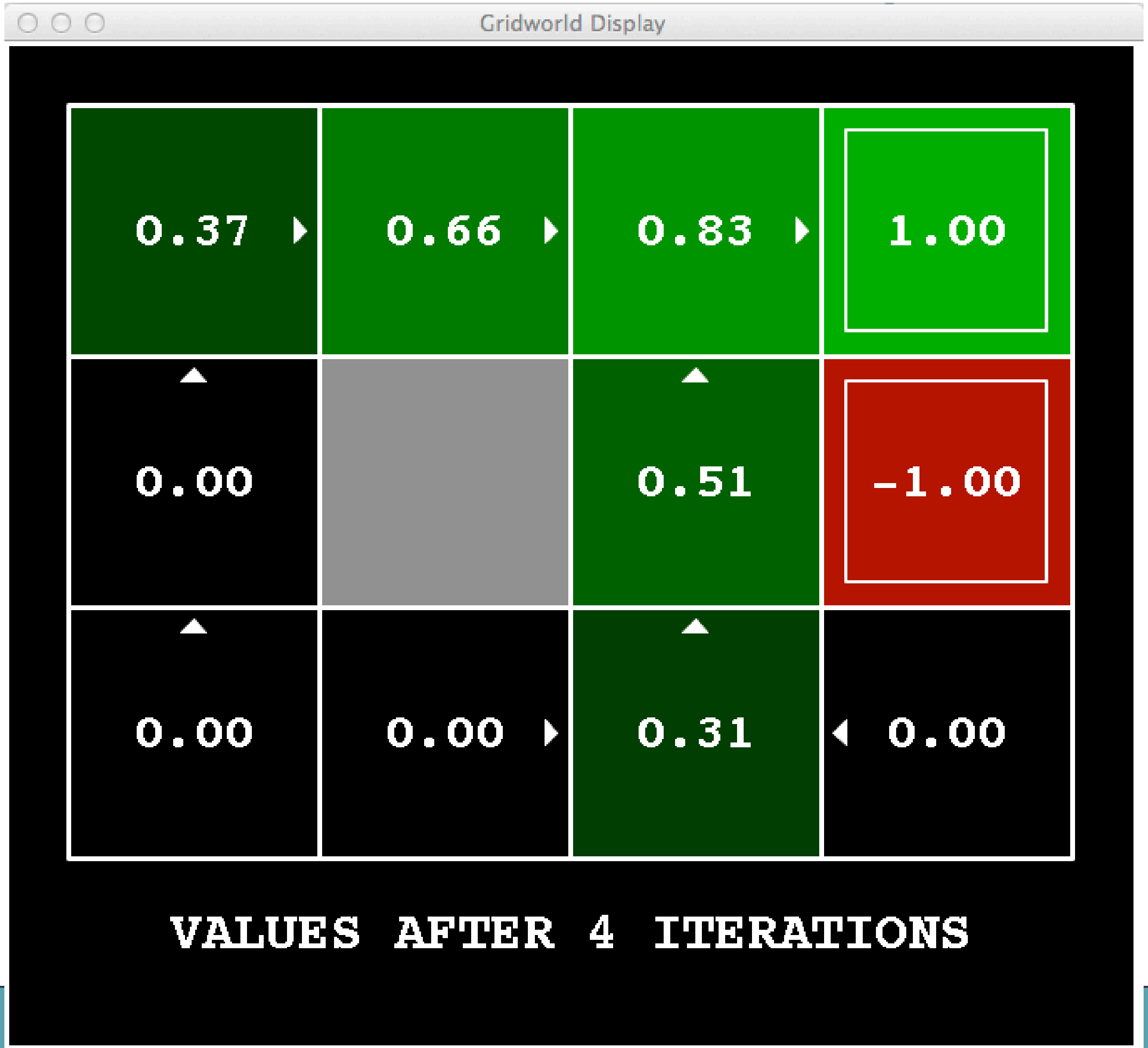
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

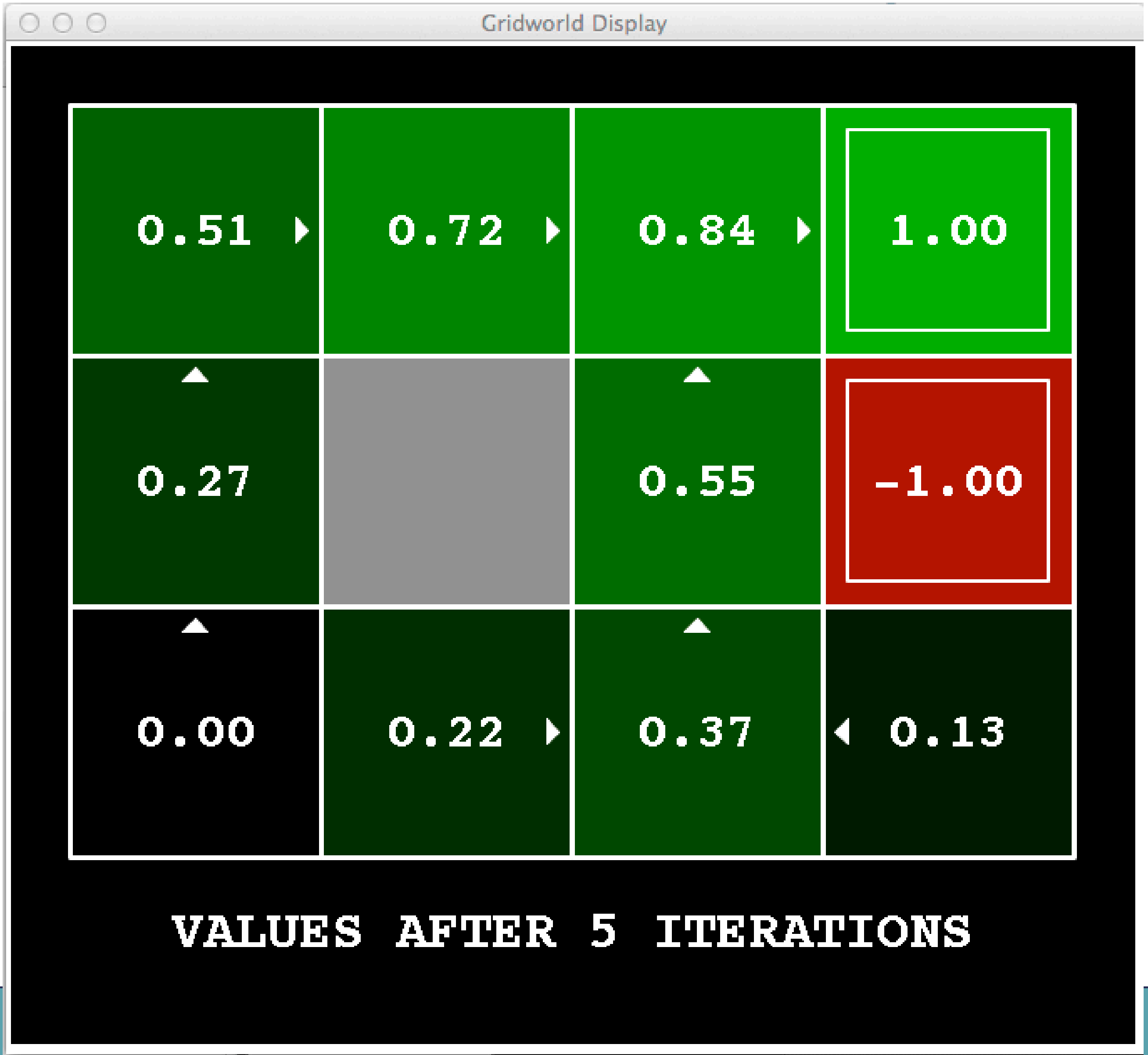
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



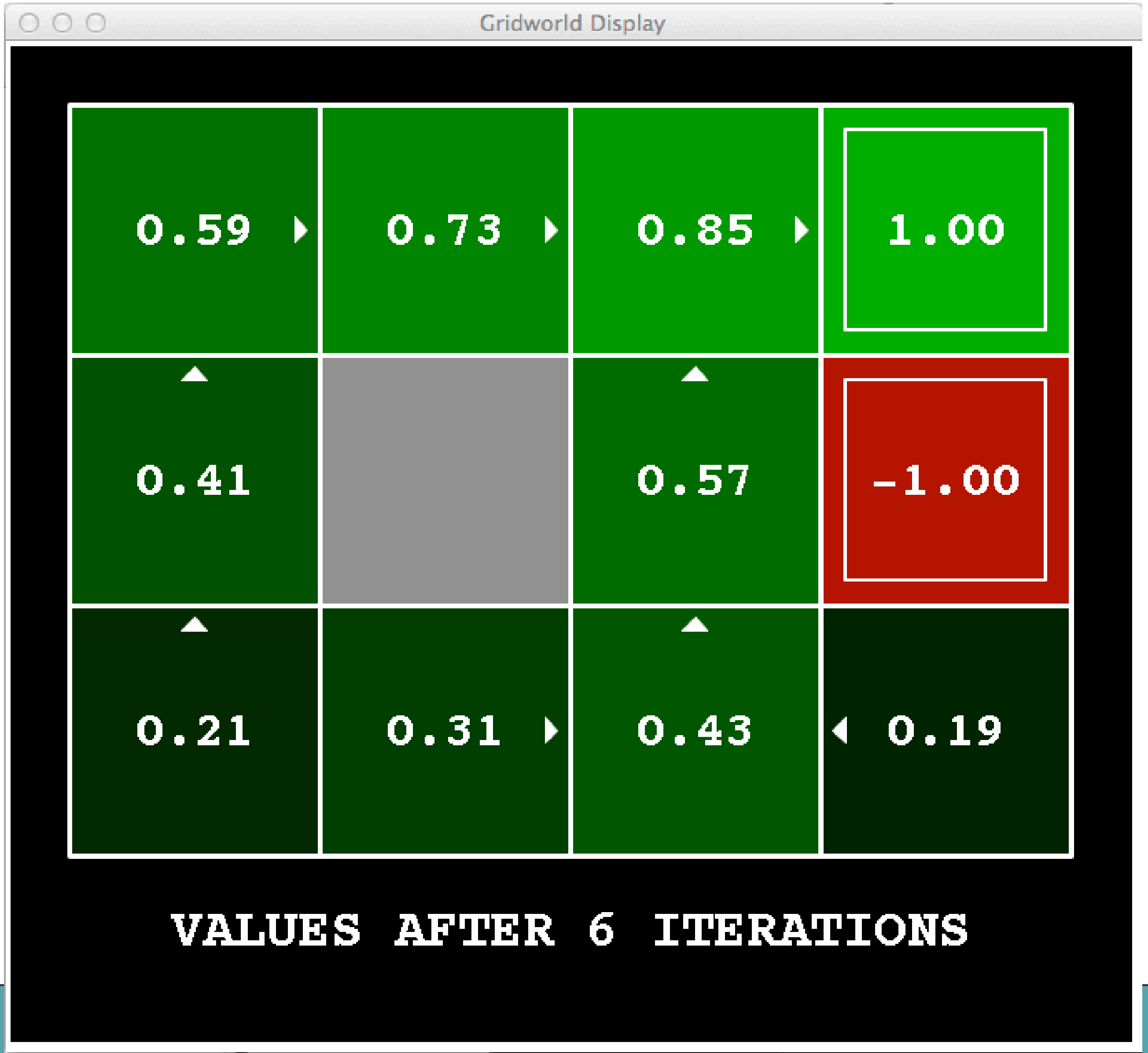
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



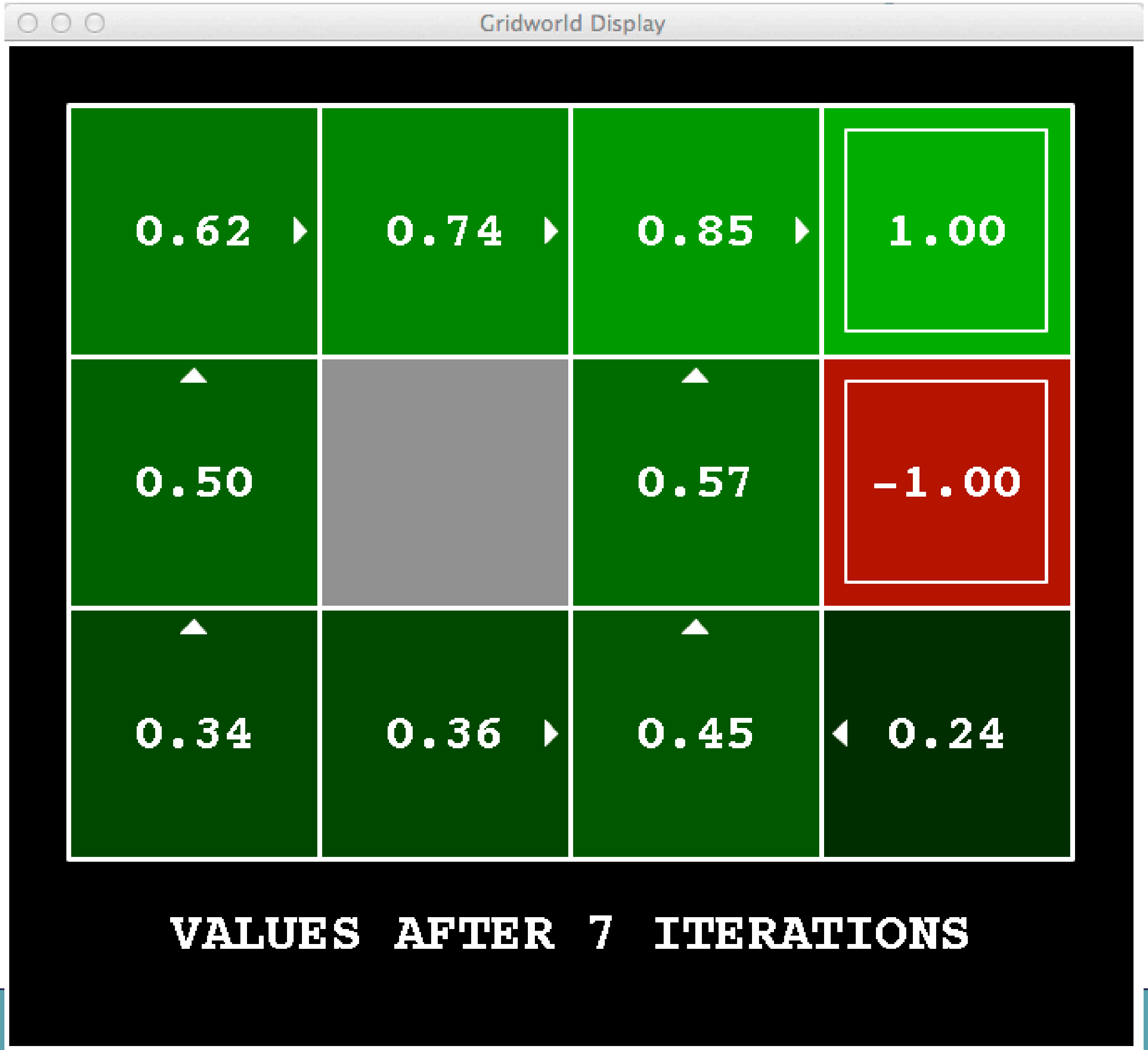
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



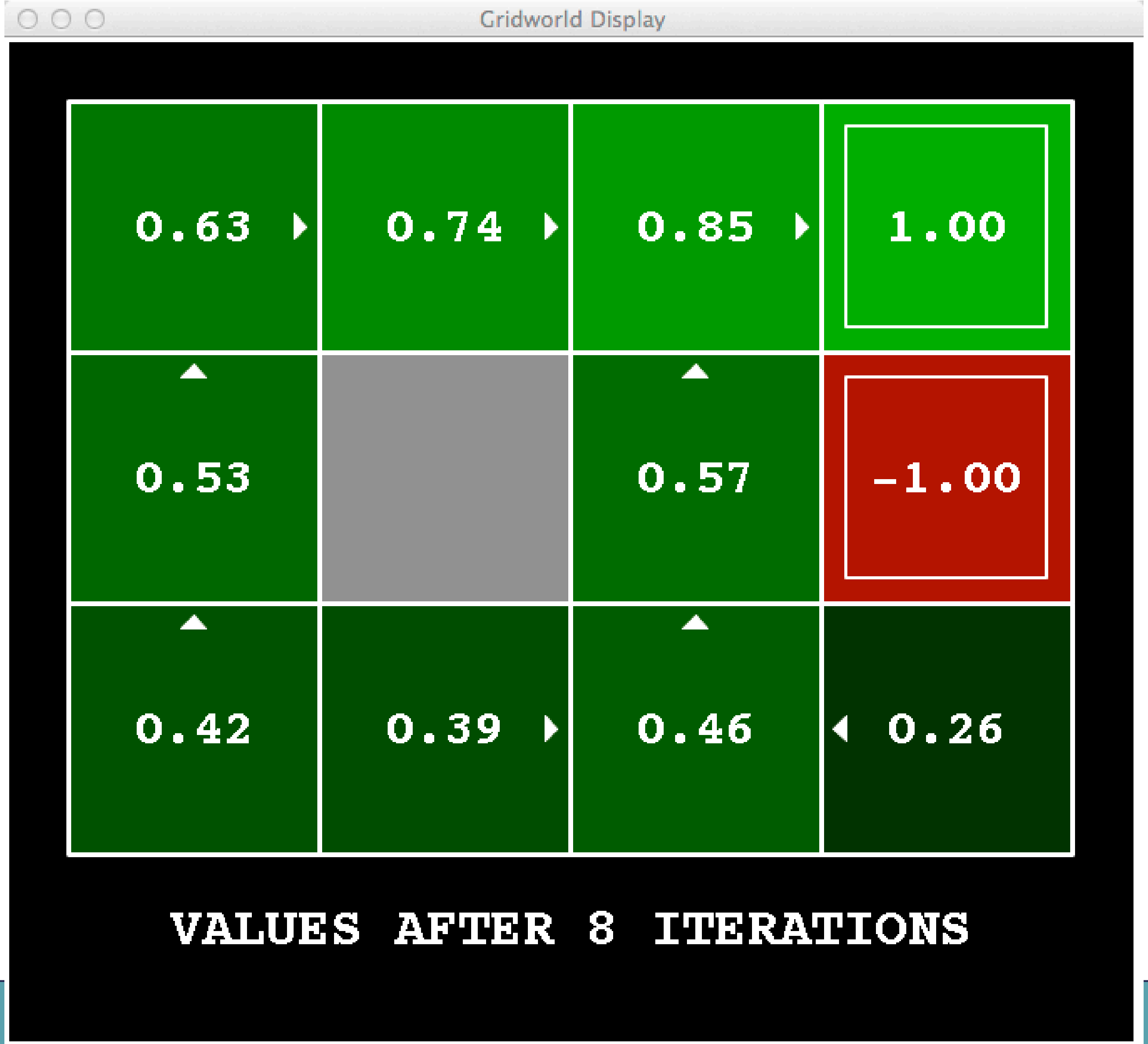
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



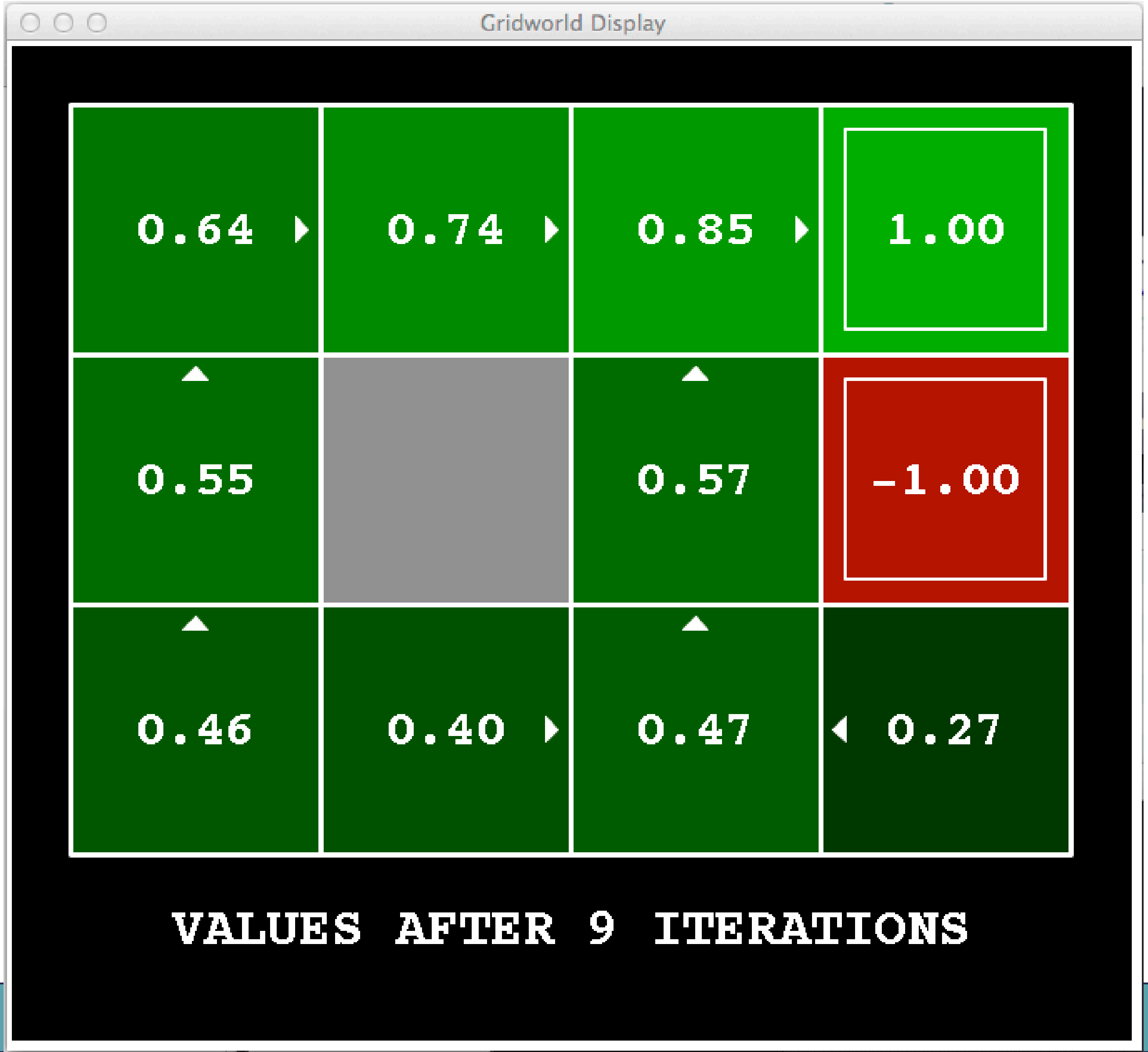
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



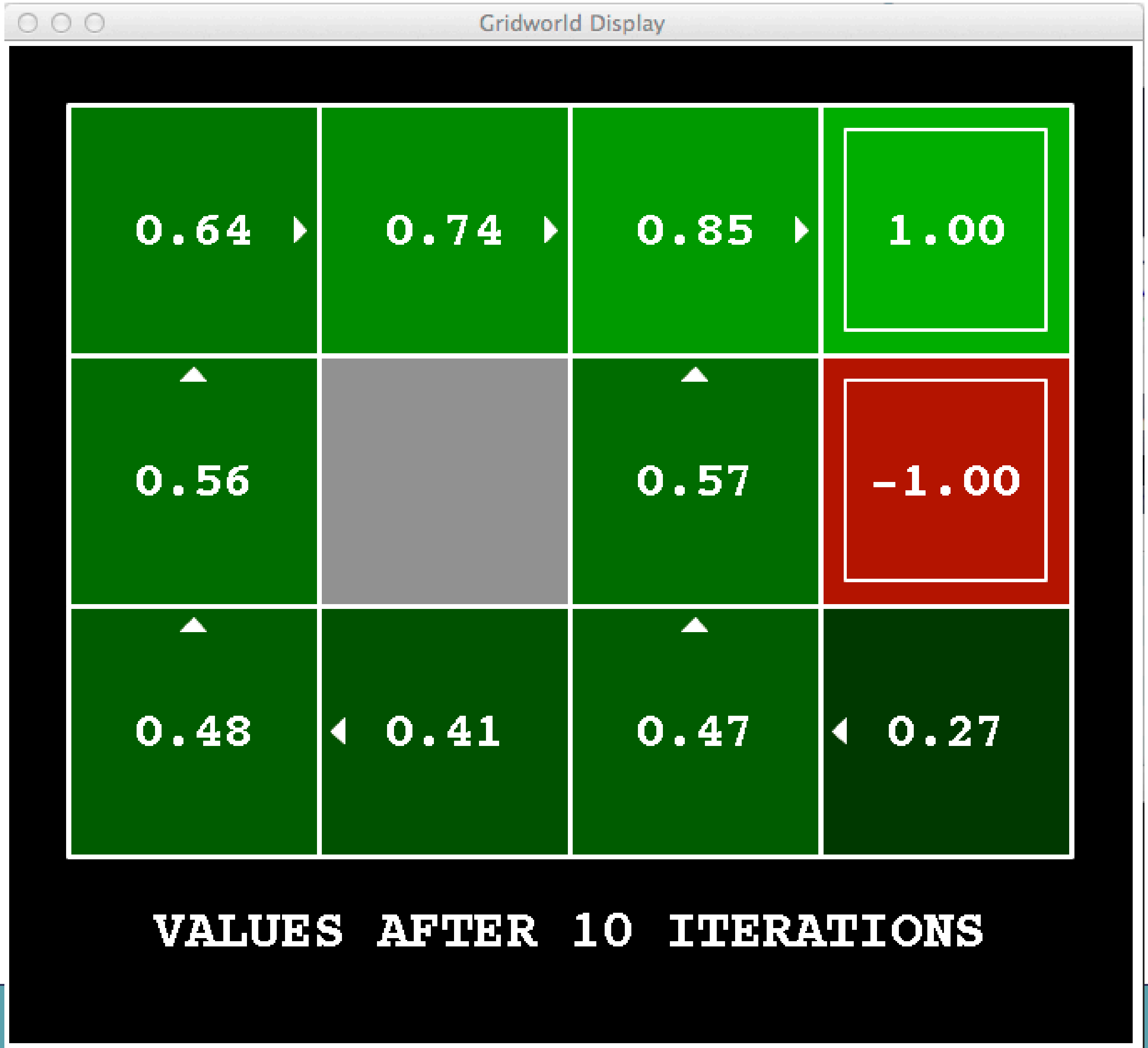
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



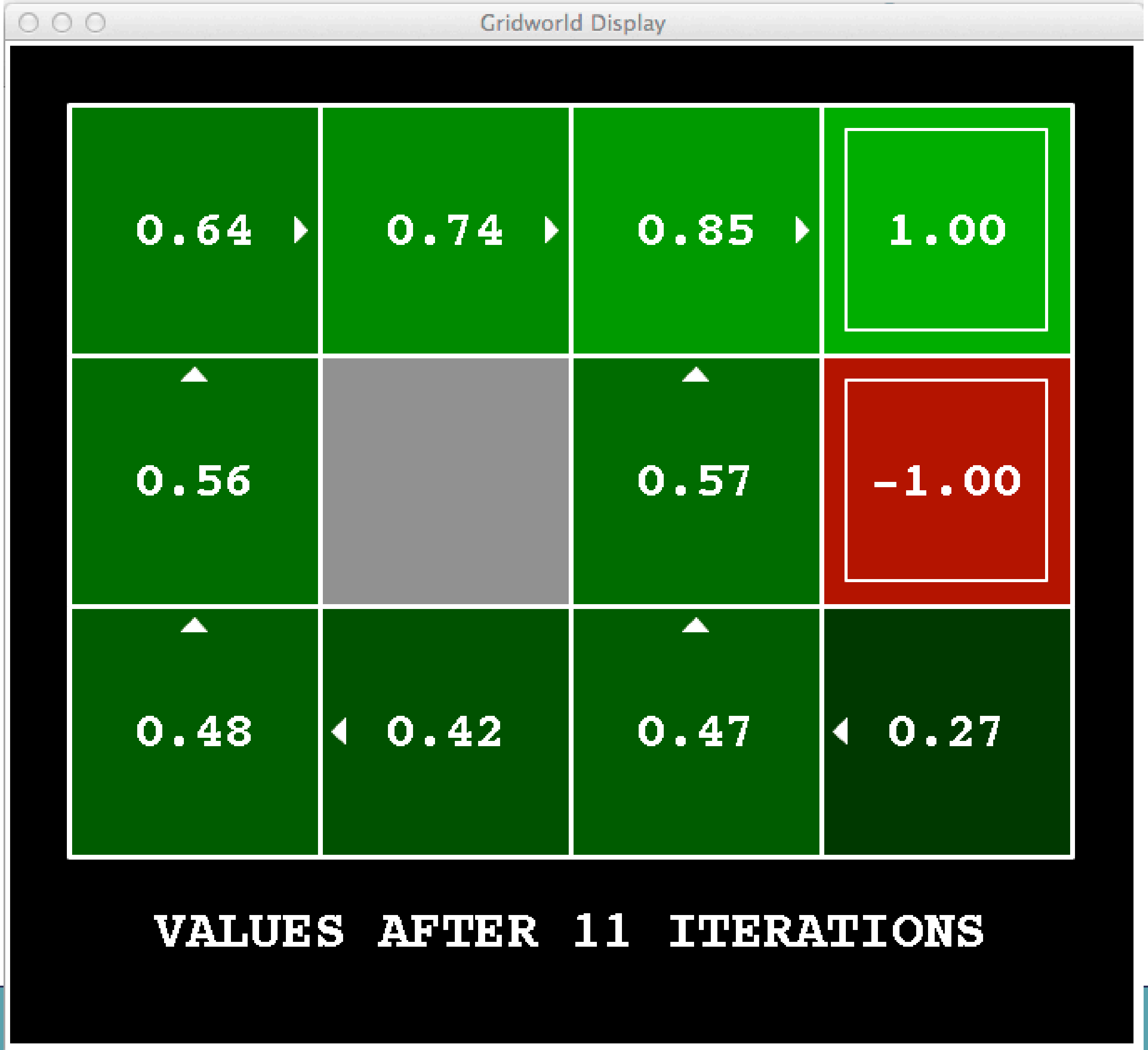
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



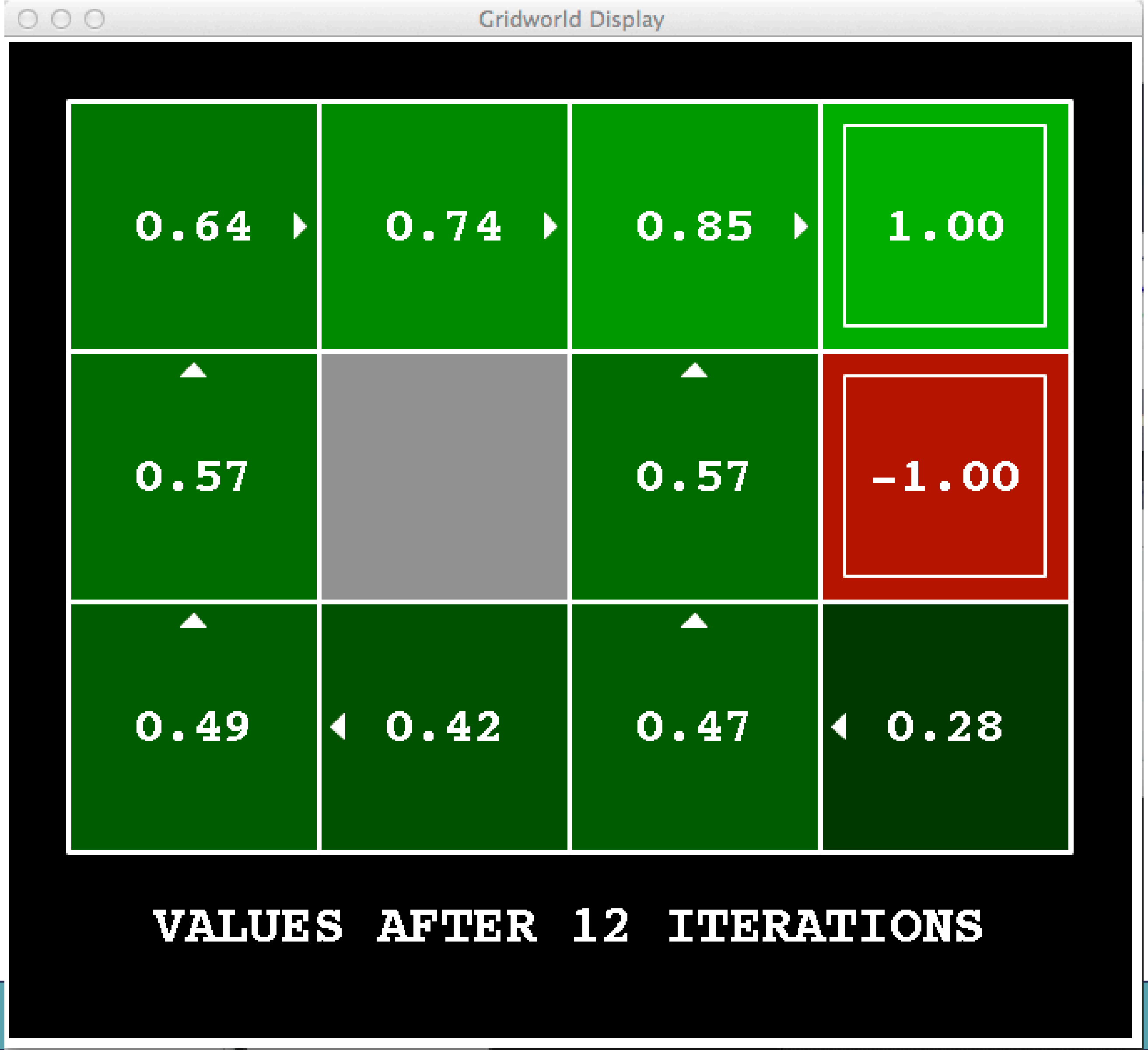
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



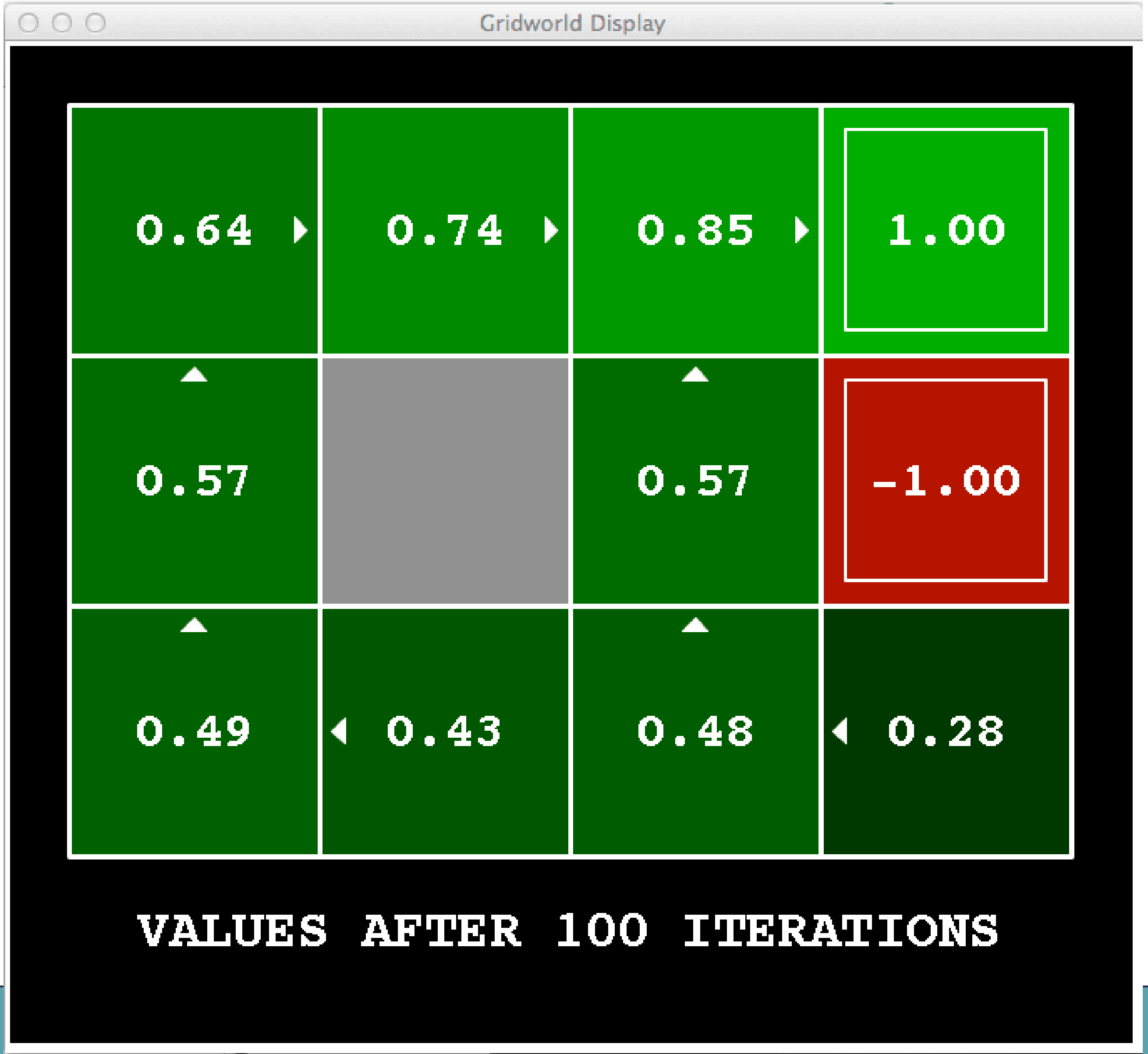
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

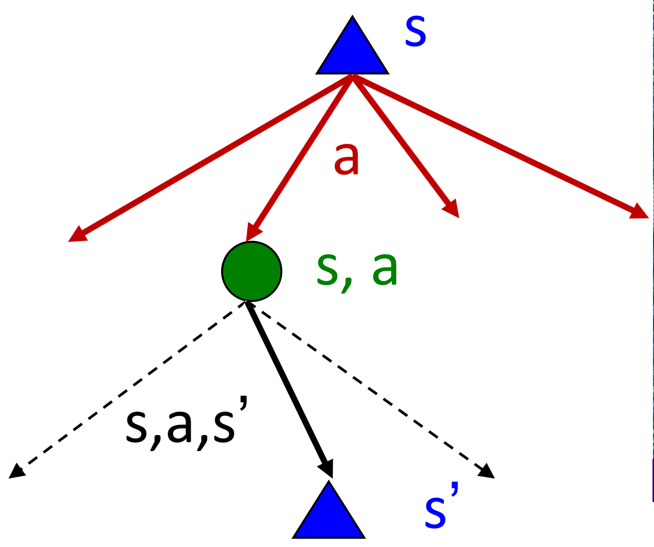


Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$



- Problem 1: It's slow – O(S^2A) per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - It

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

- Improvement: For fixed values, get a better policy using policy extra
  - One-step look-ahead:

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

Comparison

- 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)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

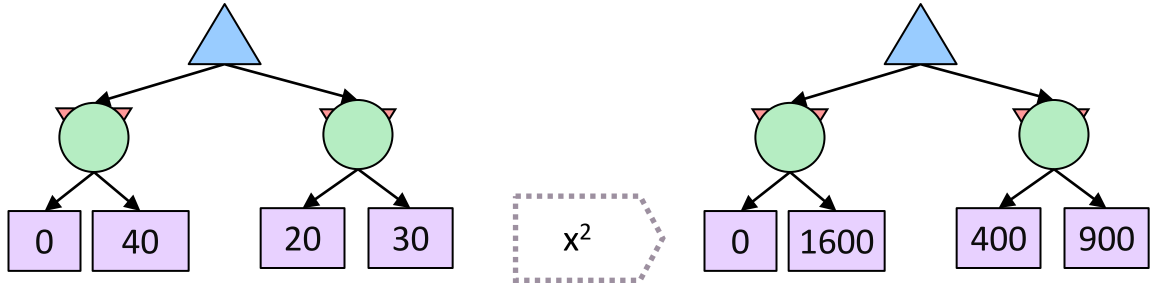
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility:
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge
- Questions:
  - Where do utilities come from?
  - How do we know such utilities even exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can't be described by utilities?

What Utilities to Use?



- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations

- For average-case expectimax reasoning, we need *magnitudes* to be meaningful

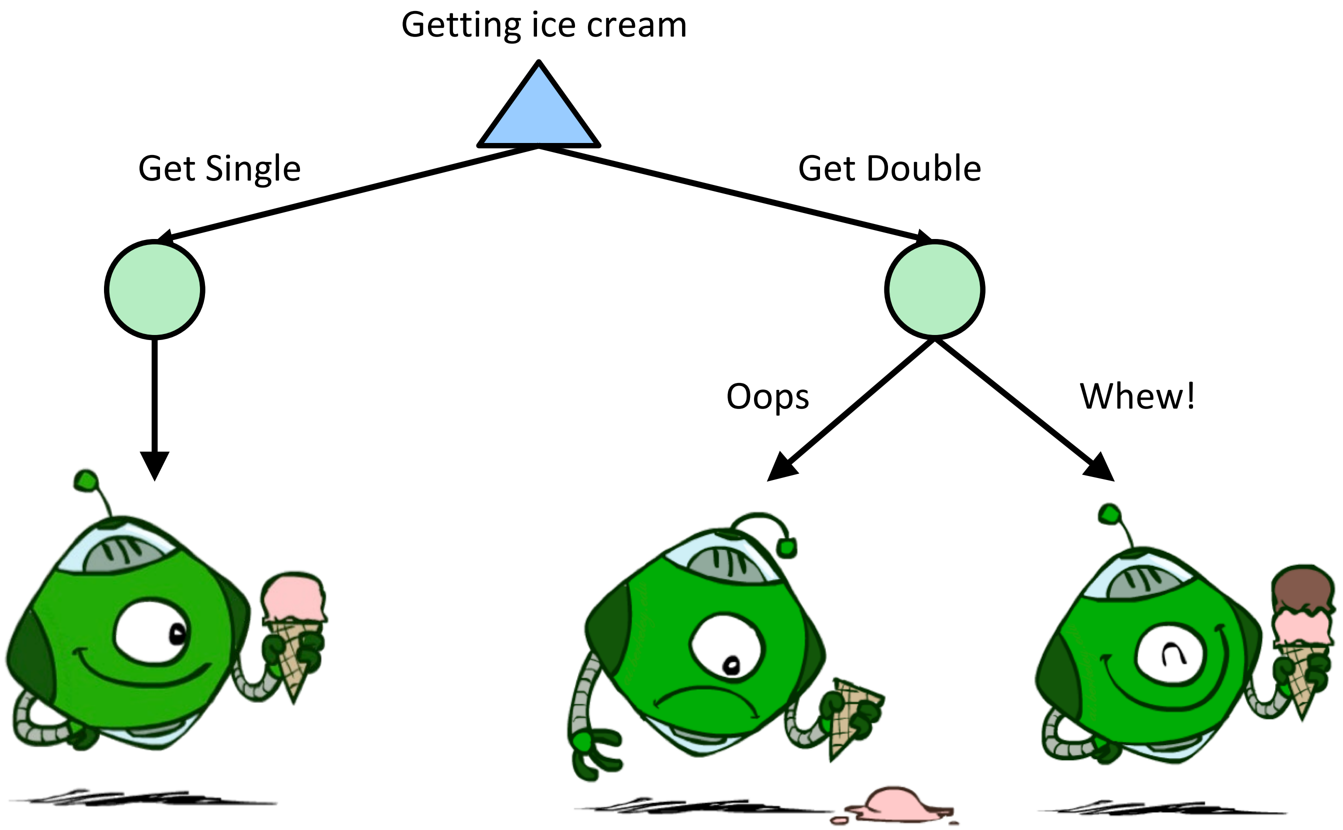
Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences

- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any "rational" preferences can be summarized as a utility function

- We hard-wire utilities and let behaviors emerge
  - Why don't we let agents pick utilities?
  - Why don't we prescribe behaviors?

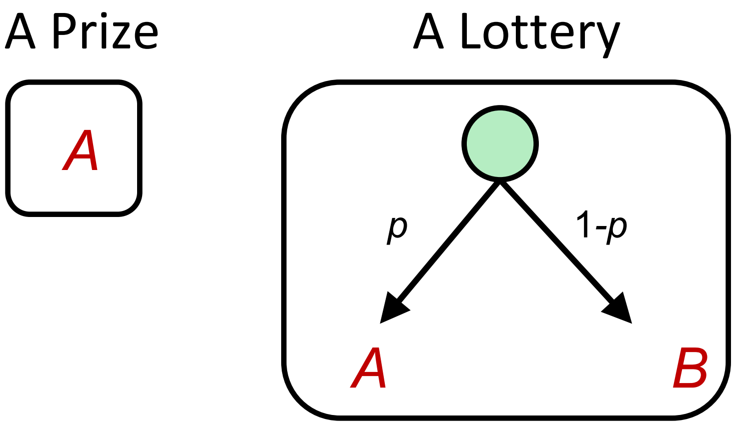
Utilities: Uncertain Outcomes



Preferences

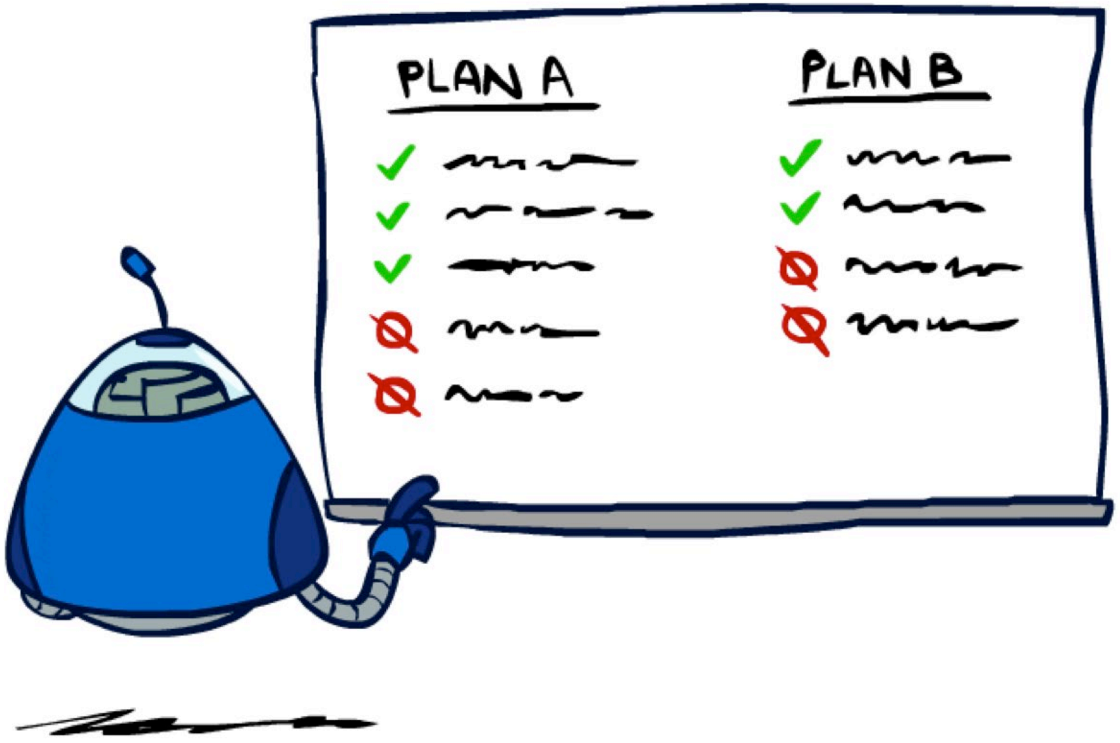
- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes

    $$L = [p, A; \ (1-p), B]$$



- Notation:
  - Preference: $A \succ B$
  - Indifference: $A \sim B$

Rationality

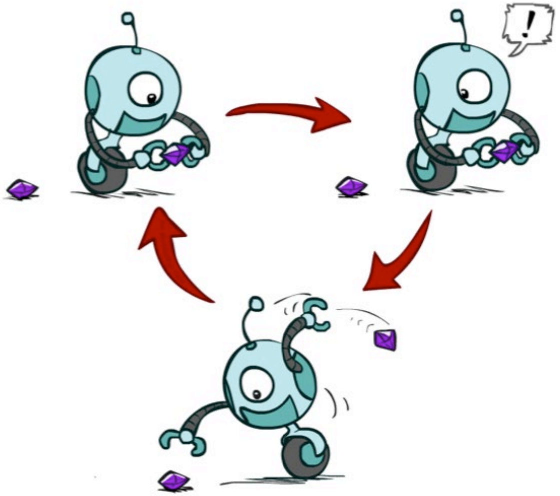


Rational Preferences

- We want some constraints on preferences before we call them rational, such as:

Axiom of Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- For example: an agent with intransitive preferences can be induced to give away all of its money
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C



Rational Preferences

The Axioms of Rationality

Orderability

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

Transitivity

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Continuity

$$A \succ B \succ C \Rightarrow \exists p \ [p, A; \ 1-p, C] \sim B$$

Substitutability

$$A \sim B \Rightarrow [p, A; \ 1-p, C] \sim [p, B; 1-p, C]$$

Monotonicity

$$A \succ B \Rightarrow$$
$$(p \geq q \Leftrightarrow [p, A; \ 1-p, B] \succeq [q, A; \ 1-q, B])$$

Theorem: Rational preferences imply behavior describable as maximization of expected utility

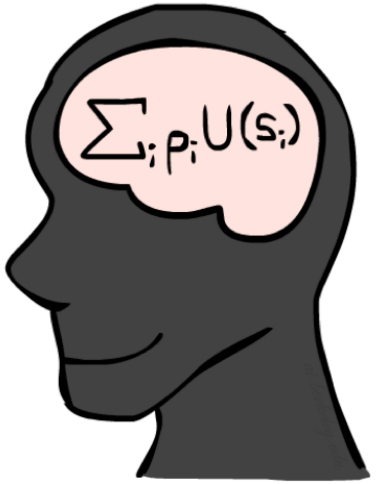
MEU Principle

- Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \;\Leftrightarrow\; A \succeq B$$

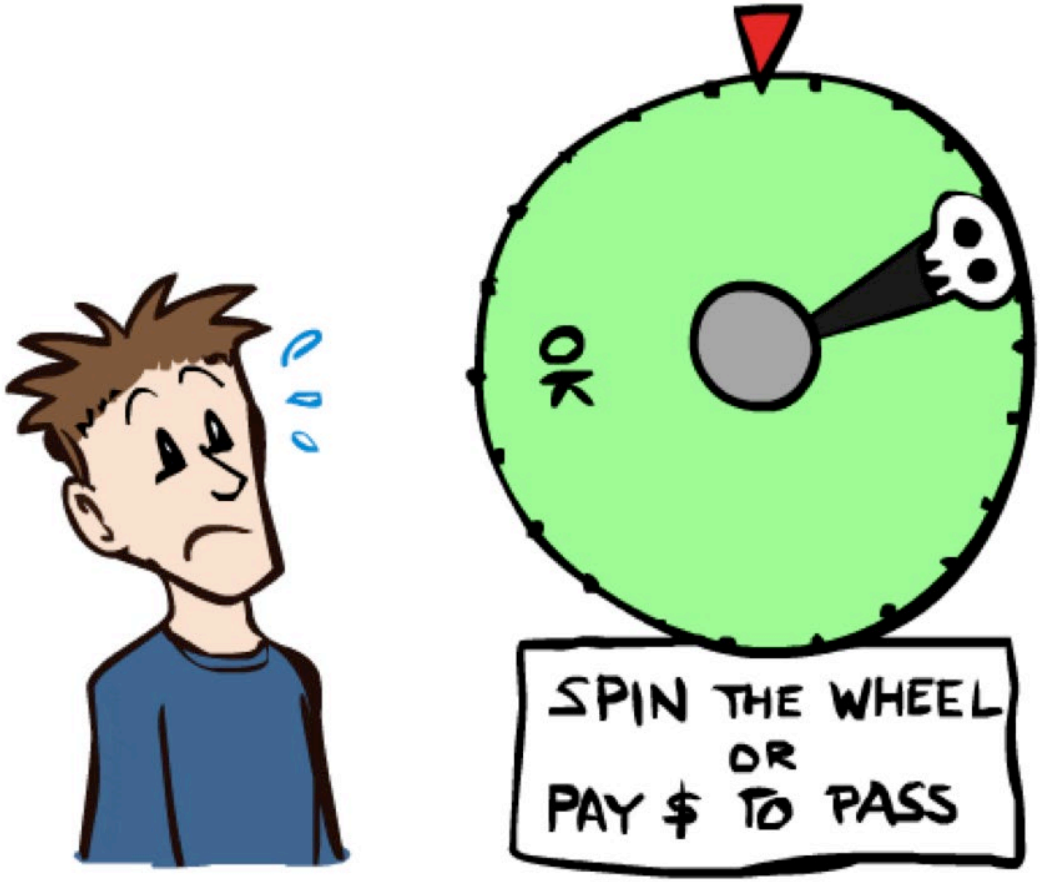
$$U([p_1, S_1;\; \ldots \;;\; p_n, S_n]) = \sum_i p_i U(S_i)$$

  - I.e. values assigned by U preserve preferences of both prizes and lotteries!



- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner

Human Utilities



Utility Scales

- Normalized utilities: $u_+ = 1.0$, $u_- = 0.0$
- Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc.
- QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

- With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

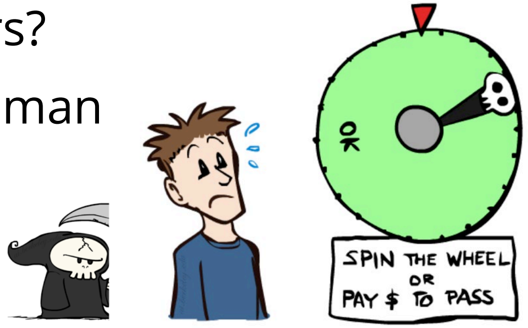
Micromort examples

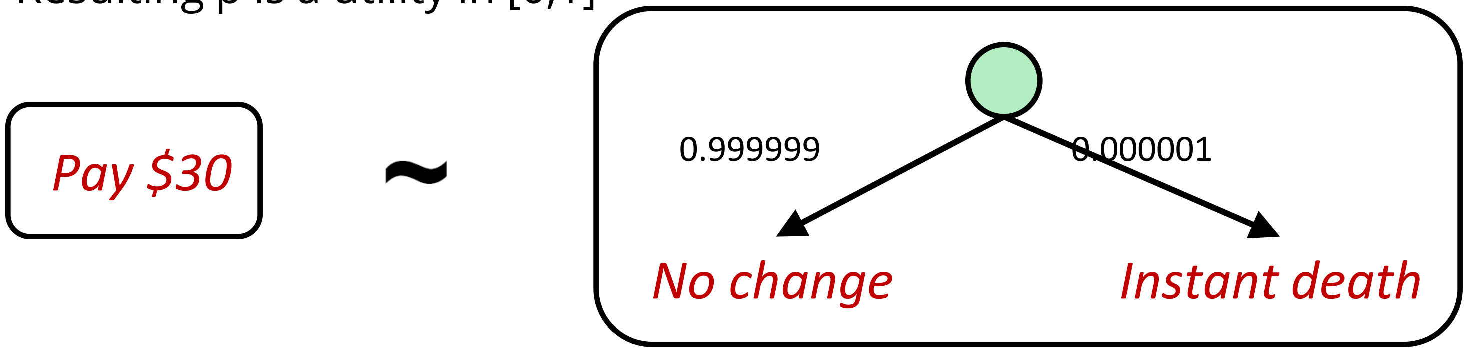
Death from	Micromorts per exposure
Scuba diving	5 per dive
Skydiving	7 per jump
Base-jumping	430 per jump
Climbing Mt. Everest	38,000 per ascent

1 Micromort	
Train travel	6000 miles
Jet	1000 miles
Car	230 miles
Walking	17 miles
Bicycle	10 miles
Motorbike	6 miles

Human Utilities

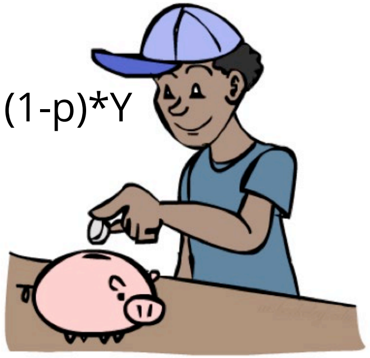
- Utilities map states to real numbers. Which numbers?
- Standard approach to assessment (elicitation) of human utilities:
  - Compare a prize A to a standard lottery L_p between
    - "best possible prize" u_+ with probability p
    - "worst possible catastrophe" u_- with probability 1-p
  - Adjust lottery probability p until indifference: $A \sim L_p$
  - Resulting p is a utility in [0,1]

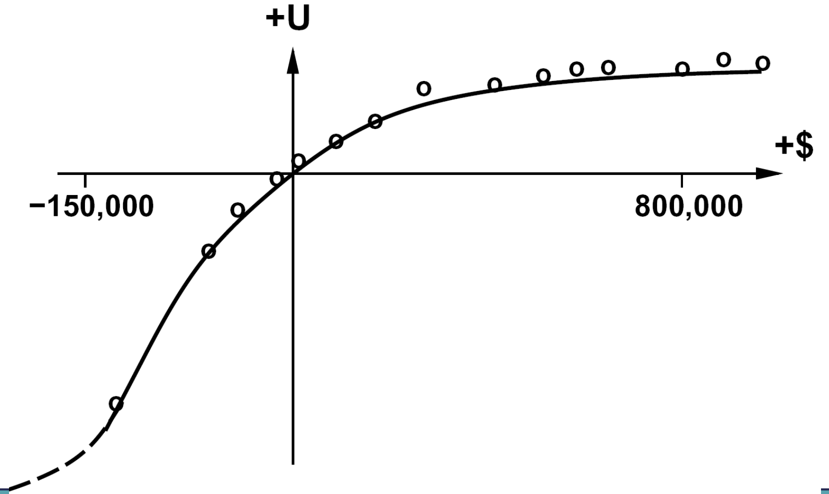




Money

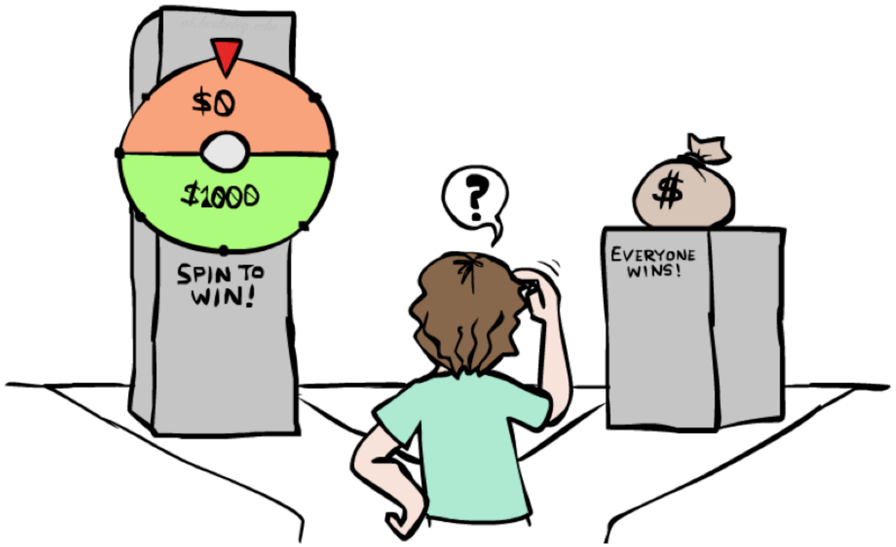
- Money <u>does not</u> behave as a utility function, but we can talk about the utility of having money (or being in debt)
- Given a lottery L = [p, $X; (1-p), $Y]
  - The expected monetary value EMV(L) is p*X + (1-p)*Y
  - U(L) = p*U($X) + (1-p)*U($Y)
  - Typically, U(L) < U(EMV(L))
  - In this sense, people are risk-averse
  - When deep in debt, people are risk-prone





Example: Insurance

- Consider the lottery [0.5, $1000; 0.5, $0]
  - What is its expected monetary value? ($500)
  - What is its certainty equivalent?
    - Monetary value acceptable in lieu of lottery
    - $400 for most people
  - Difference of $100 is the insurance premium
    - There's an insurance industry because people will pay to reduce their risk
    - If everyone were risk-neutral, no insurance needed!
  - It's win-win: you'd rather have the $400 and the insurance company would rather have the lottery (their utility curve is linear and they have many lotteries)



Example: Human Rationality?

- Famous example of Allais (1953)
  - A: [0.8, $4k;   0.2, $0]
  - B: [1.0, $3k;   0.0, $0]
  - C: [0.2, $4k;   0.8, $0]
  - D: [0.25, $3k;   0.75, $0]
- Most people prefer B > A, C > D
- But if U($0) = 0, then
  - B > A $\Rightarrow$ U($3k) > 0.8 U($4k)
  - C > D $\Rightarrow$ 0.8 U($4k) > U($3k)