Markov Decision Processes

a.k.a. Probabilistic Long-term Decision Making

Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada
Oct. 22, 2024

Slide Credits:Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).
Pearson 2021

Midterm: Next Tuesday (25%)

It will take place in our normal lecture theatre.
Arrive at 10:30am, we will have a short lecture.

The midterm will take place from **11:00am-12:15pm** (75min).
There will be short answer questions and long answer questions. It will cover all content presented so far, not including today's lecture on MDPs.

Bring your SFU ID and a pencil.

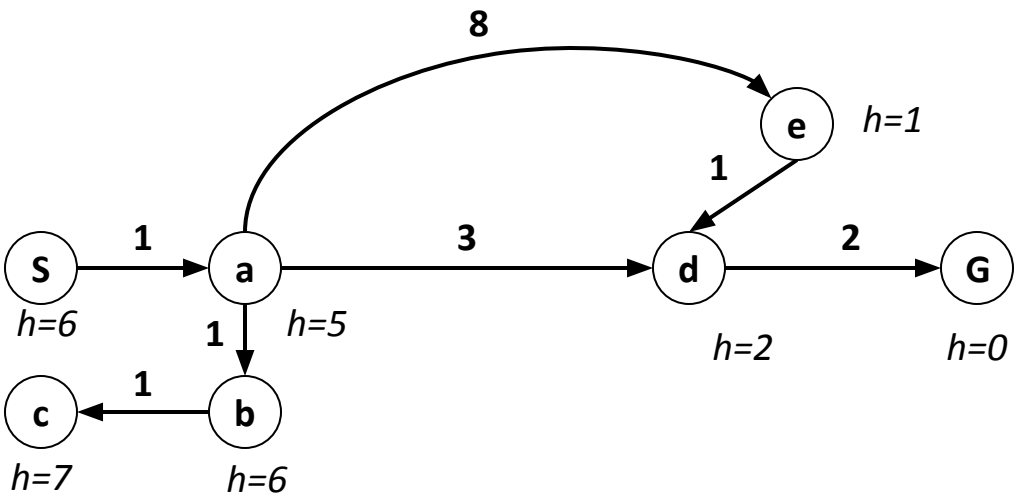
No coding questions.
No calculators are needed

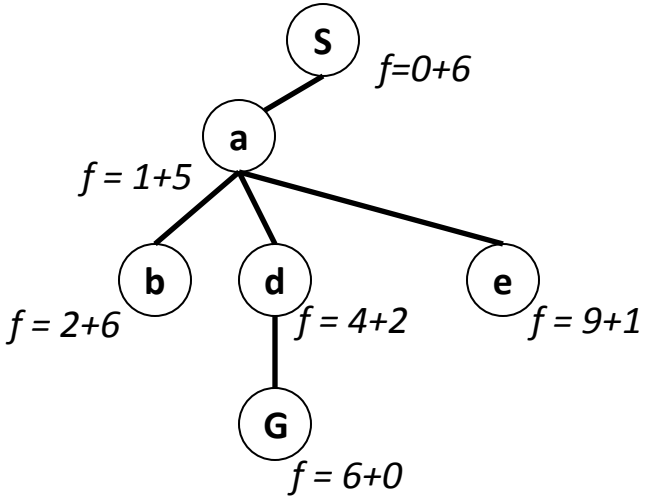
How to prepare:

- Review the slides, weekly activities and assignments.

Recap: A* Search

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)





- A* Search orders by the sum: f(n) = g(n) + h(n)

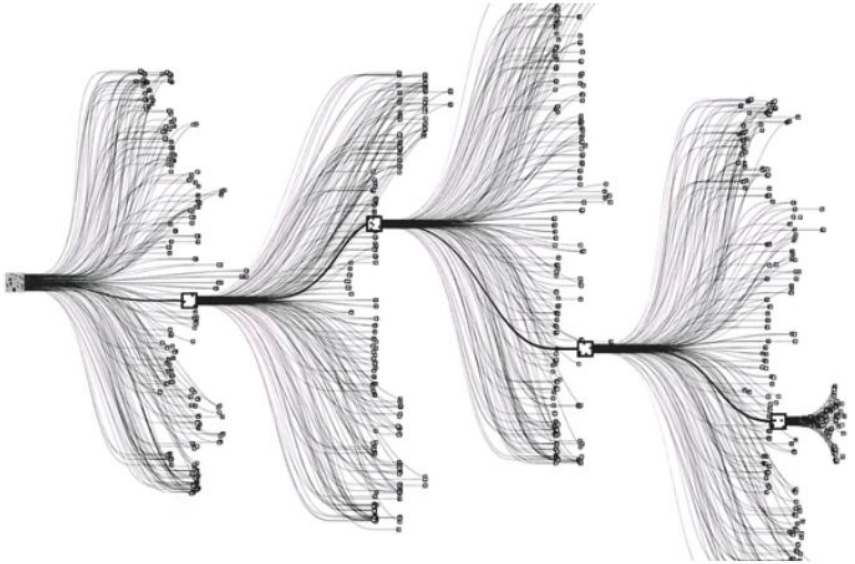
Example: Teg Grenager

State-based models

Key idea: Model the state of the world and transitions between states, triggered by actions

Tasks that require forethought, *e.g. playing chess, planning a big trip, collecting dots on a map*

Solutions are **procedural**, step by step sequences of actions.



Applications:

- Games: Chess, Go, Pac-Man, Starcraft, etc.
- Robotics: motion planning

Beyond reflex

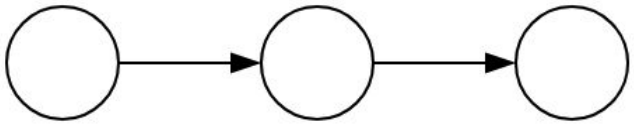
Classifier (reflex-based models):

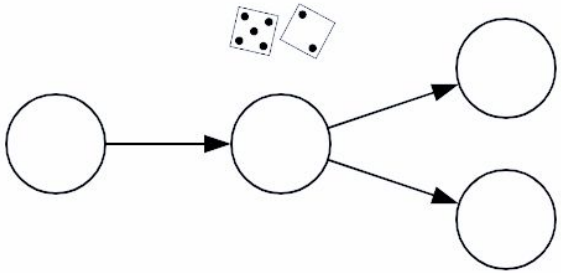
Search problem (state-based models):

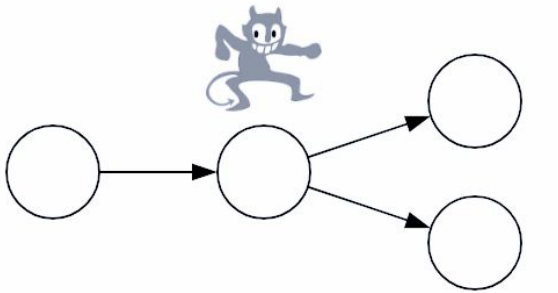
Key: need to consider future consequences of an action!

Credit: Stanford CS 221, used with permission

State-based models





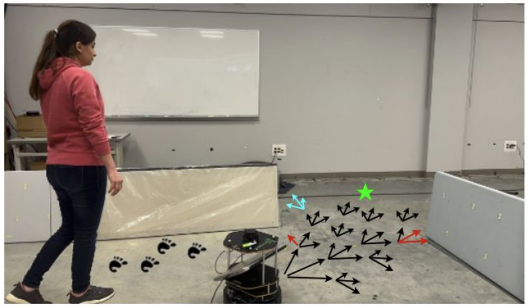


Search problems: when you have an environment with no uncertainty, ie. perfect information. But realistic settings are more complex

Markov decision processes (MDPs) handle situations with randomness, e.g. Blackjack

Game playing handles tasks where there is interaction with another agent. Adversarial games assume an opponent, e.g. Chess

Motivation

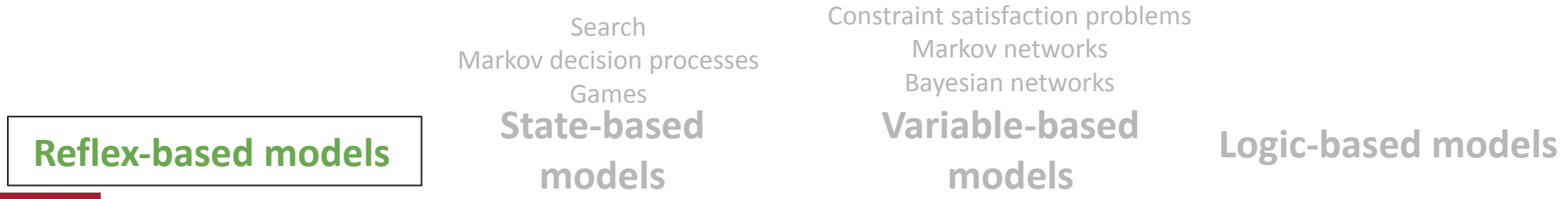


Robot Motion Planning
Actions: straight, turn left 45°, turn right 45°
Goal: robot stays in front of the human

Any time the agent tries to take an action, it may not succeed in doing so.

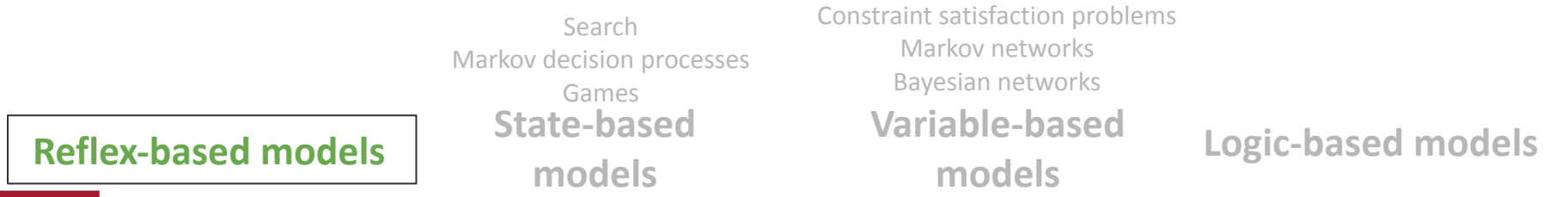
Course Overview

Week 1 : Getting to know you
Week 2 : Introduction to Artificial Intelligence
Week 3: Machine Learning I: Basic Supervised Models (Classification)
Week 4: Machine Learning II: Supervised Regression, Classification and Gradient Descent, K-Means
Week 5: Machine Learning III: Neural Networks and Backpropagation
Week 6 : Search
Week 7 : Markov Decision Processes
Week 8 : Midterm
Week 9 : Reinforcement Learning
Week 10 : Games
Week 11 : Hidden Markov Models and Bayesian Networks
Week 12 : Constraint Satisfaction Problems
Week 13 : Ethics and Explainability

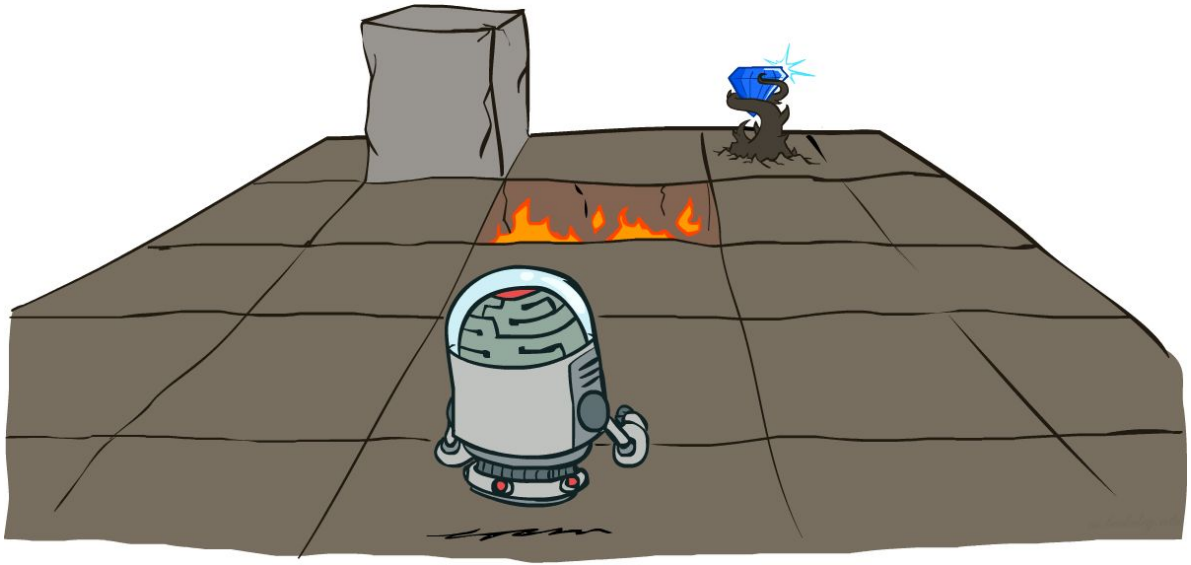


Readings

- Chapter 2 - Intelligent Agents
- Chapter 16.1 - Combining Beliefs and Desires under Uncertainty
- Chapter 17.1 - Sequential Decision Problems
- Chapter 17.1.1 - Utilities over Time
- Chapter 17.1.2 - Optimal Policies and the Utilities of States

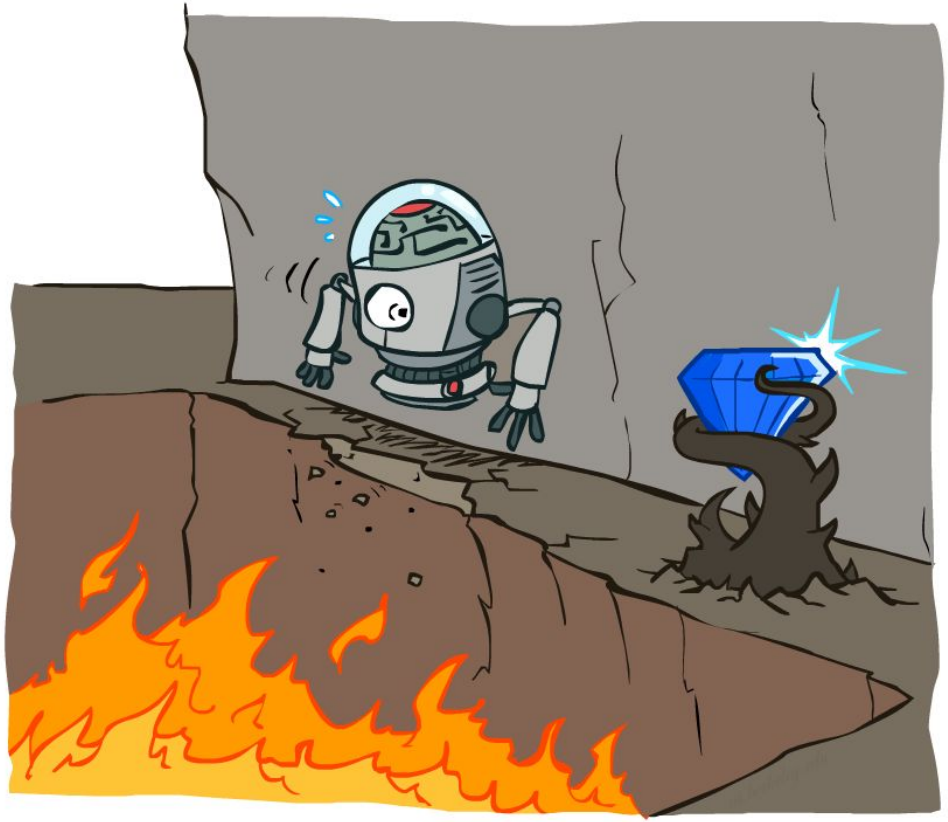


Markov Decision Processes



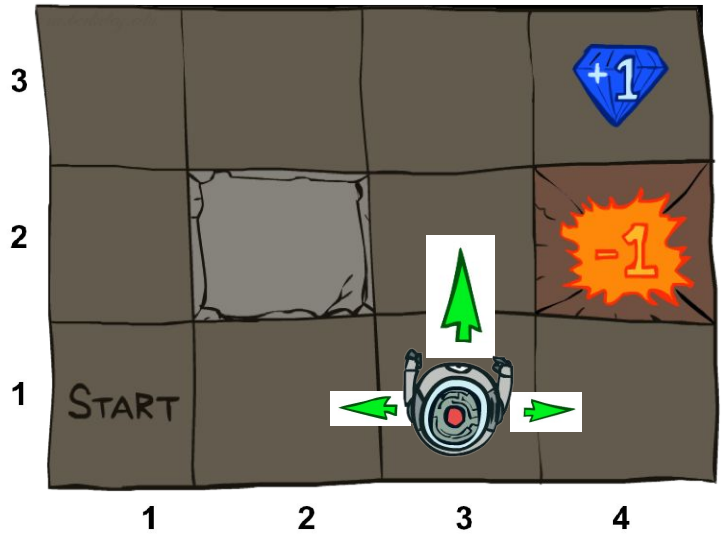
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Actions + Search + Probabilities + Time



Example: Grid World

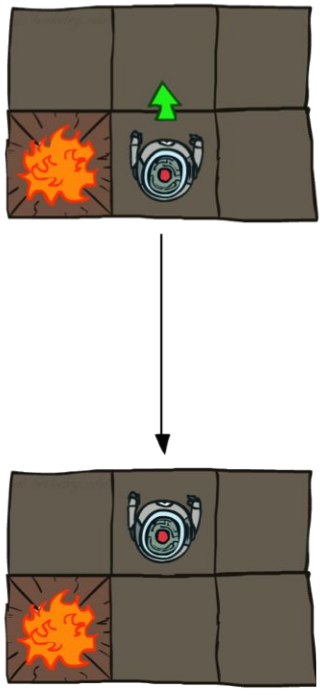
- Consider a maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have taken, the agent stays put
- The agent receives rewards each time step
  - Small "staying alive" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards



With no noisy actions, you could move to the left square and then back and accumulate an infinite amount of rewards. However, noisy actions may cause the robot to fall in the hole or end up in another unintended state.

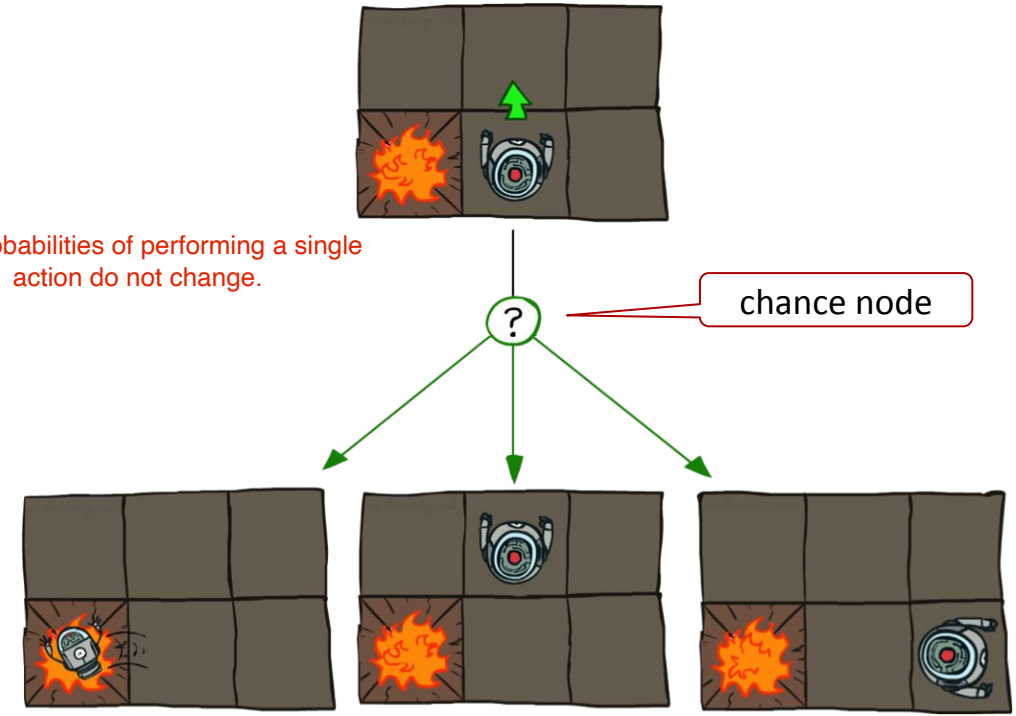
Grid World Actions

Deterministic Grid World



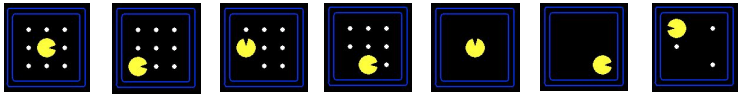
Stochastic Grid World

The probabilities of performing a single action do not change.

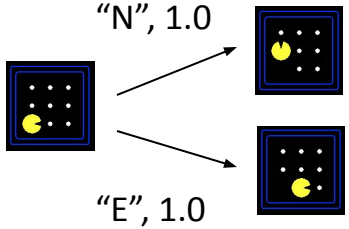


Recall: Search Problems

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)



  - A start state and a goal test (am I done?)

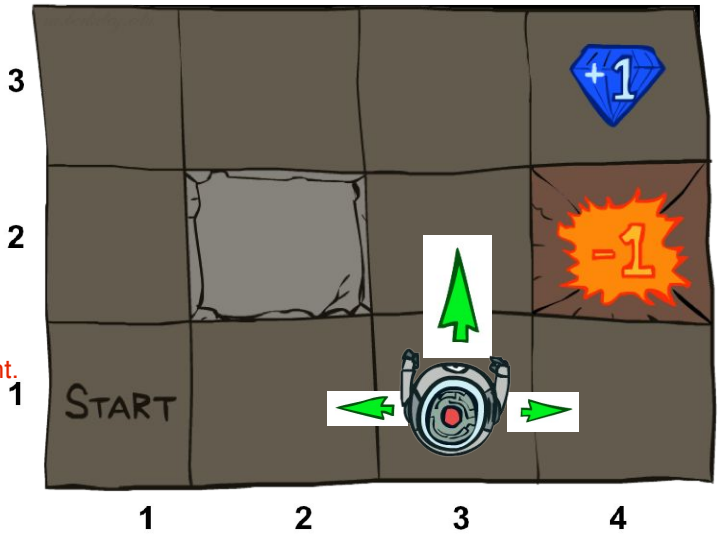
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function 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')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state

Instead of successor function (action, cost)

Are the actions independent from the state?
Depending on my state, the action taken is different.



- MDPs are non-deterministic search problems

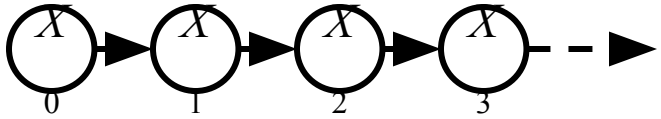
[Demo – gridworld manual intro (L8D1)]

What is Markov about MDPs?

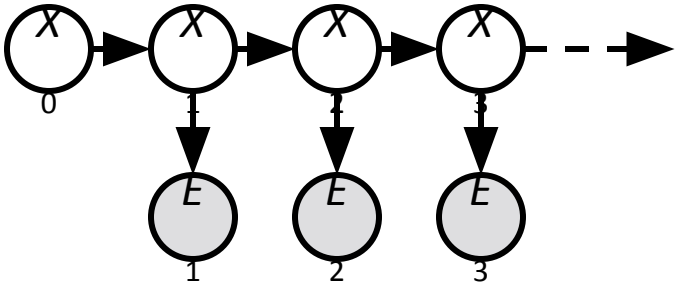
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means action outcomes depend only on the current state
- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

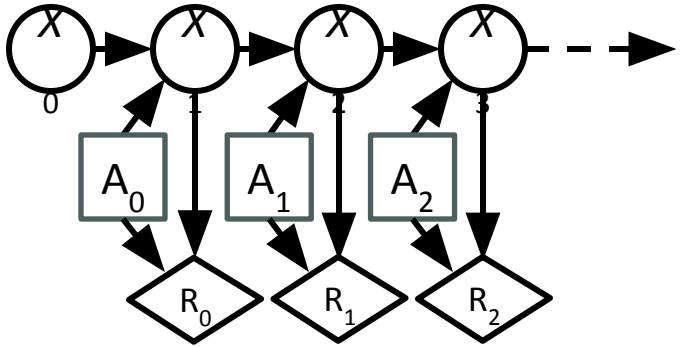
“Markov” as in Markov Chains?



Markov Chain



Hidden Markov Model



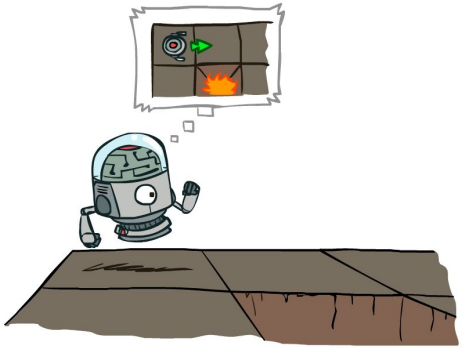
Markov Decision Process

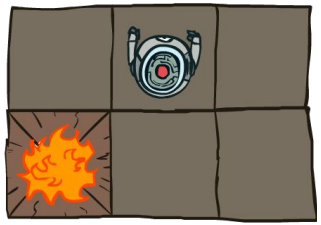
???

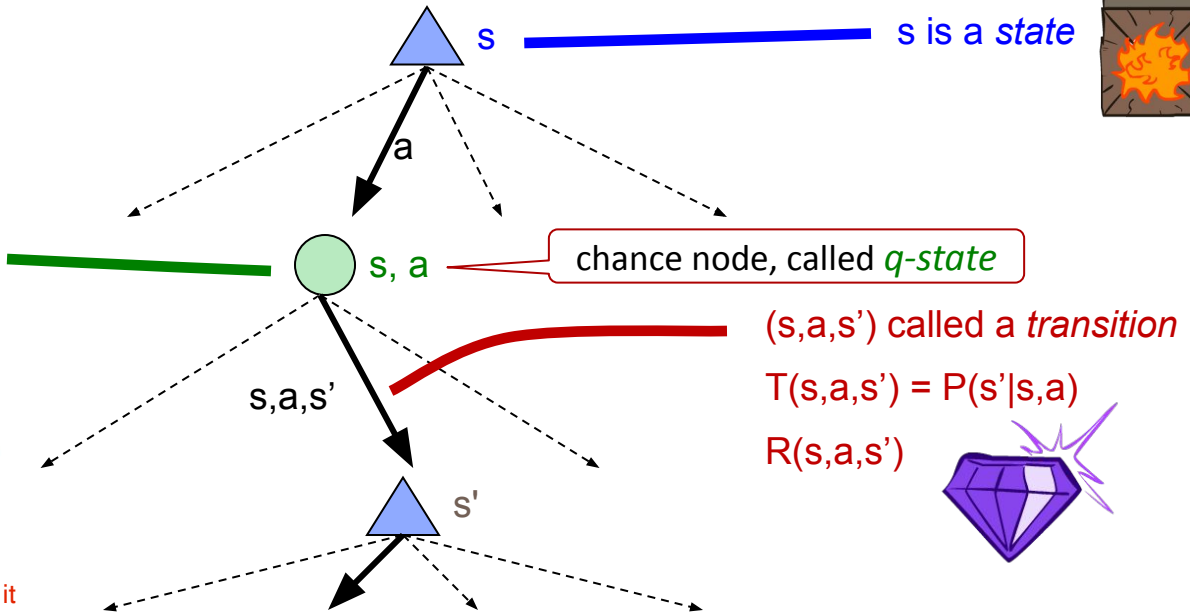
Partially Observable
Markov Decision Process

MDP Search Trees

- Each MDP state projects a search tree

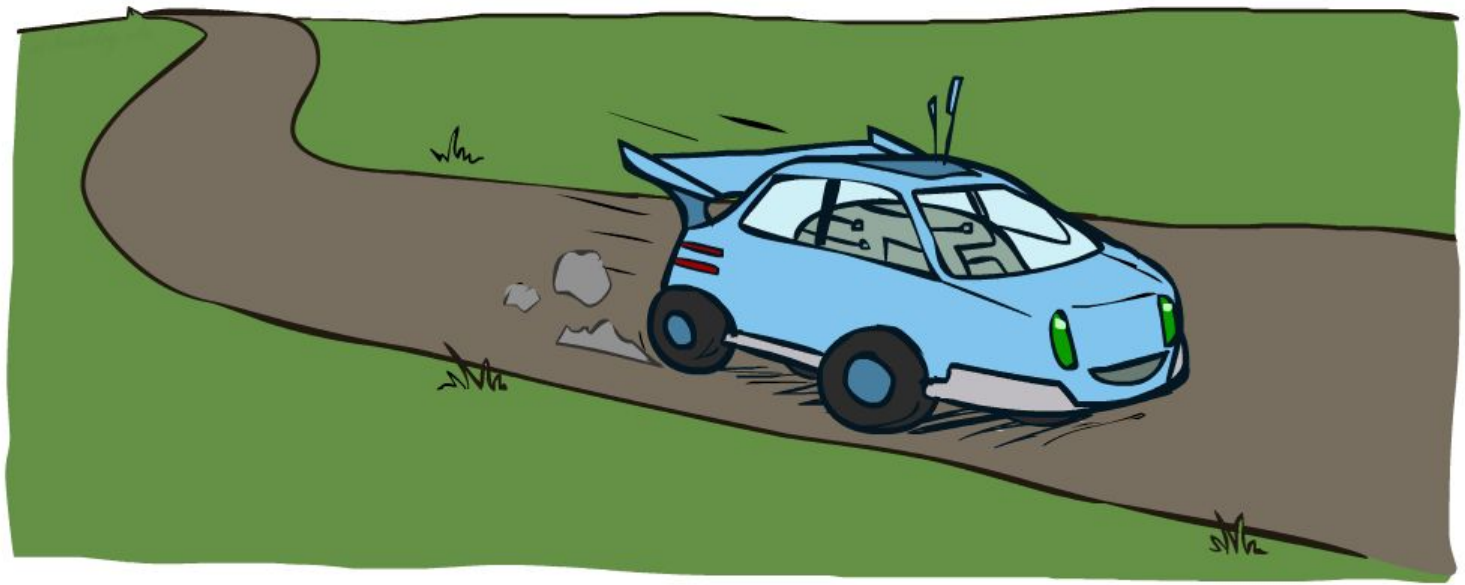






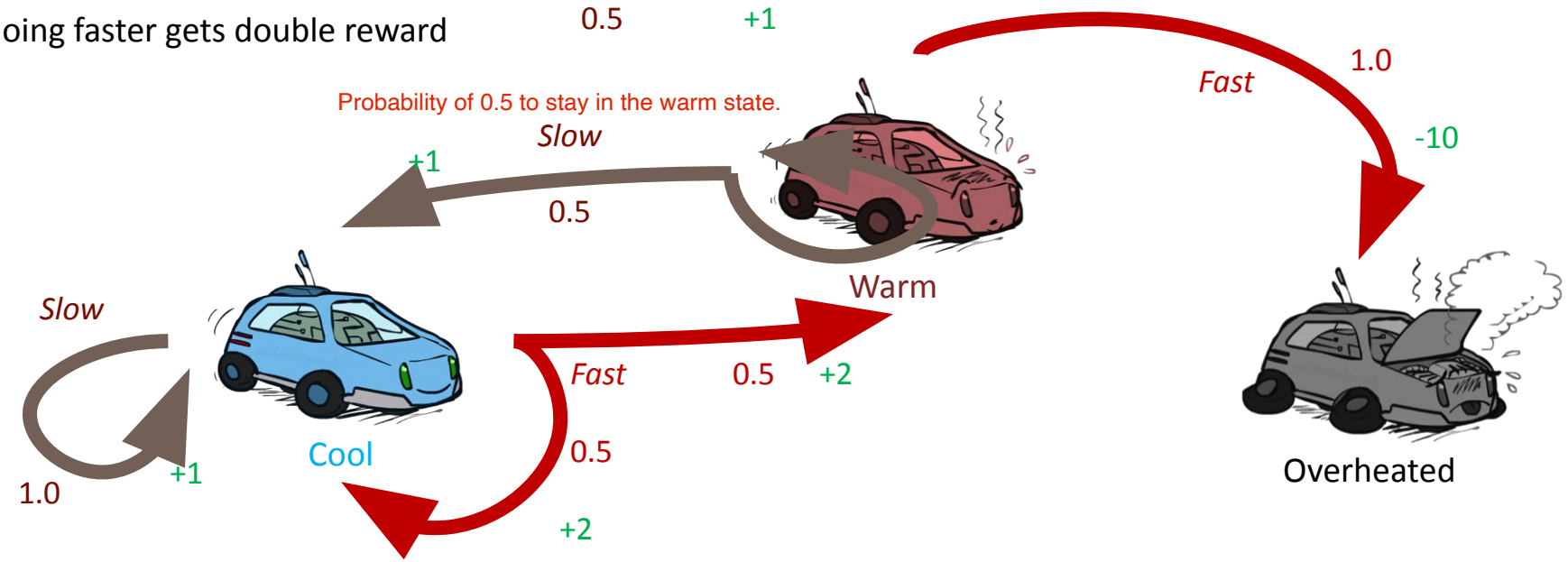
This line has a probability and a reward associated with it

Example: Racing



Example: Racing

- A robot car wants to travel far, quickly
- Three **states**: Cool, Warm, Overheated
- Two **actions**: *Slow*, *Fast*
- Going faster gets double reward

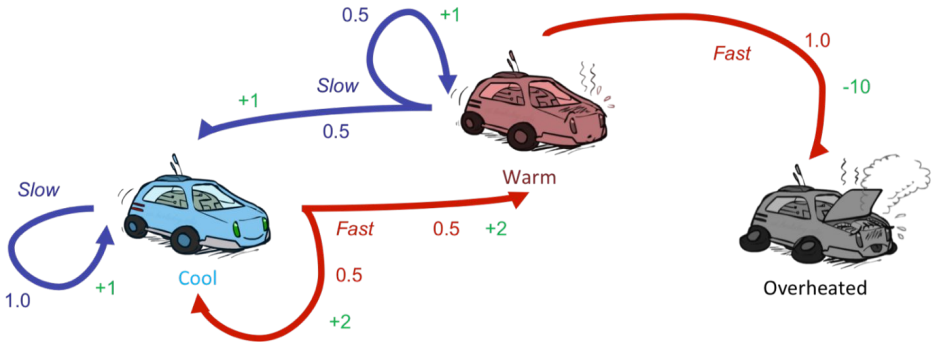


Example: Racing

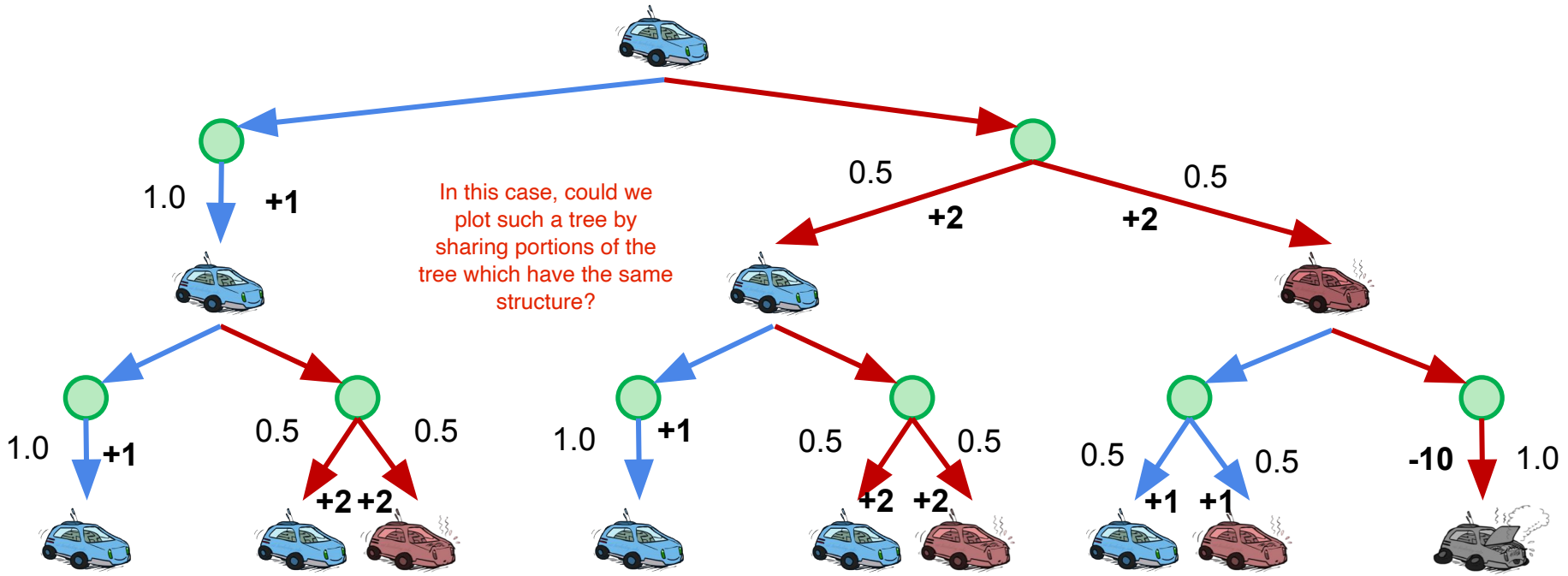
Transition probability

Reward

s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	–10
	(end)		1.0	0



Racing Search Tree: Transition Probabilities and **Rewards**



Relationship between Search Tree and MDP Search Tree

The main difference with MDPs:

- Transition probabilities instead of successor function
- Instead of minimizing cost, we want to maximize rewards (negate one to get the other)

The **utility** is the **sum of rewards** received.

Similar to overall cost in basic weighted Search

Similar to forward cost

Expected utility of an **action a** is the average of the outcome utilities weighted by the probability of the outcome.

Expected utility also looks at the actions that follows it

$$EU(a) = \sum_{s'} P(\text{RESULT}(a) = s')\, U(s')\,.$$

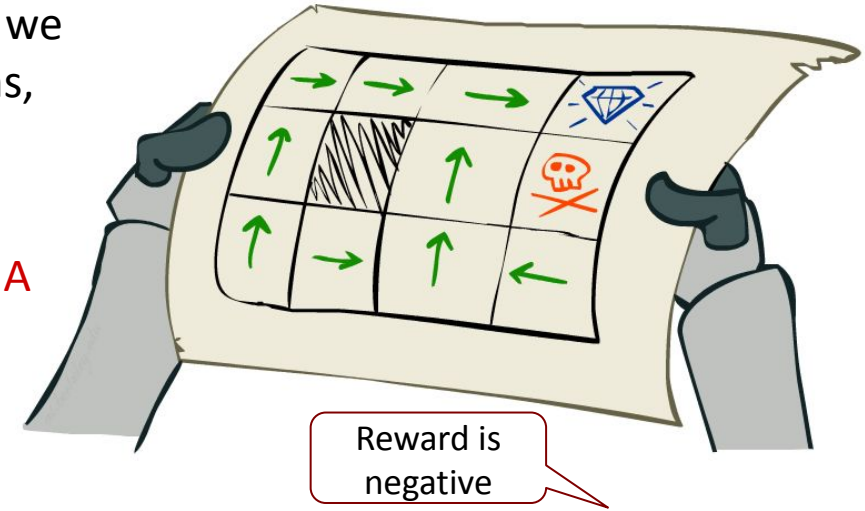
Policies

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

Want to generate the actions needed for each state to obtain the end goal.

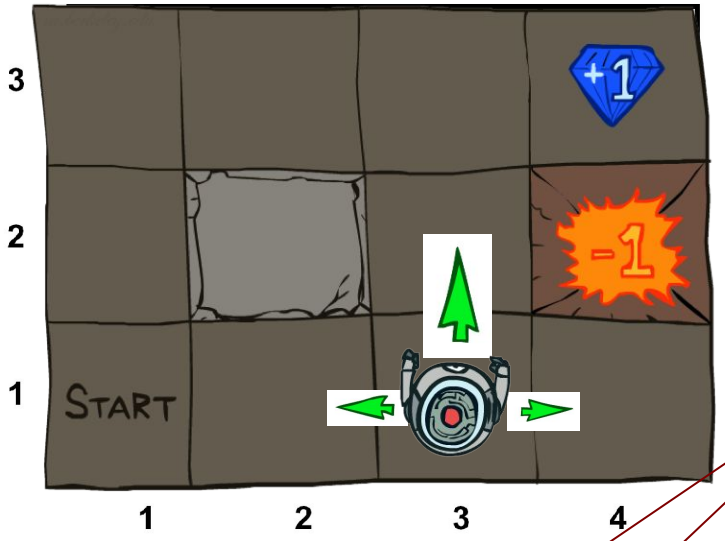
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A **policy π** is a mapping of an **action** for each **state**
  - An **optimal policy π*** is one that **maximizes expected utility** if followed

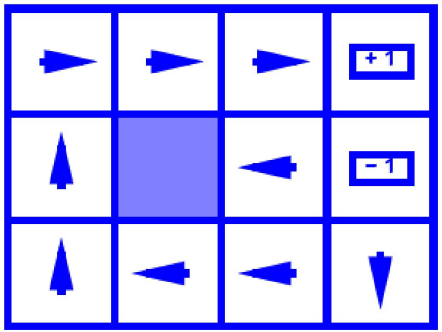
A policy gives you a scenarios to use in each state, but is not always guaranteed to succeed.
A plan will always work.



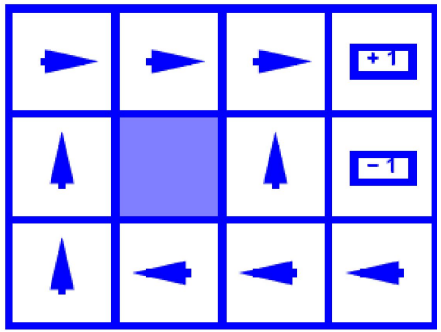
Optimal policy if R(s, a, s') = -0.03 for all non-terminal states

Optimal Policies

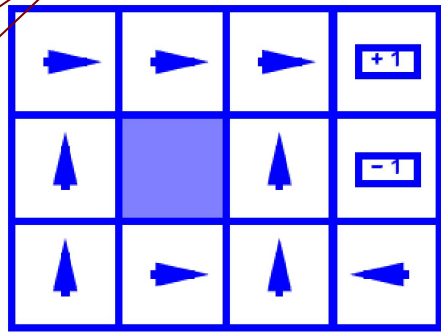




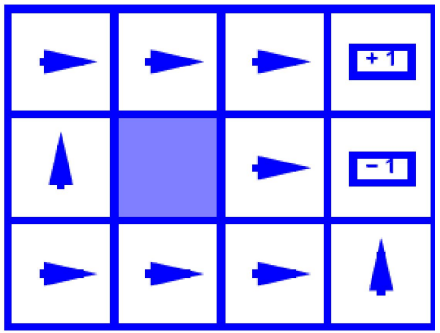
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4

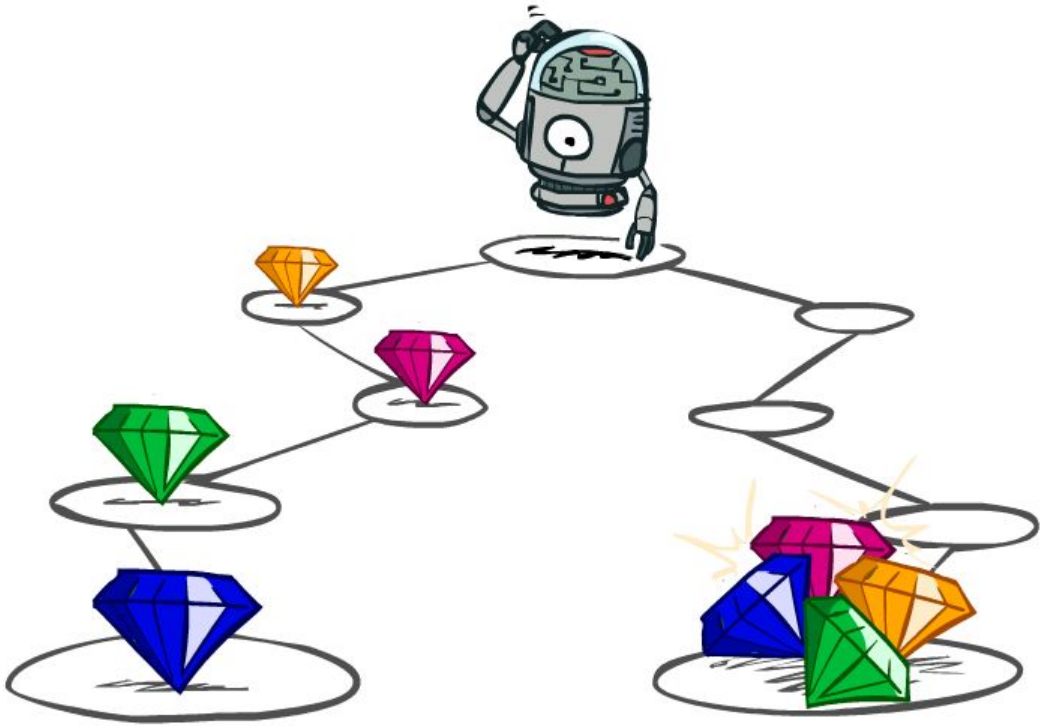


R(s) = -2.0

"Staying alive" reward

We cannot normally change the reward.
We can only affect the policy.

Utilities of Sequences



Utilities of Sequences

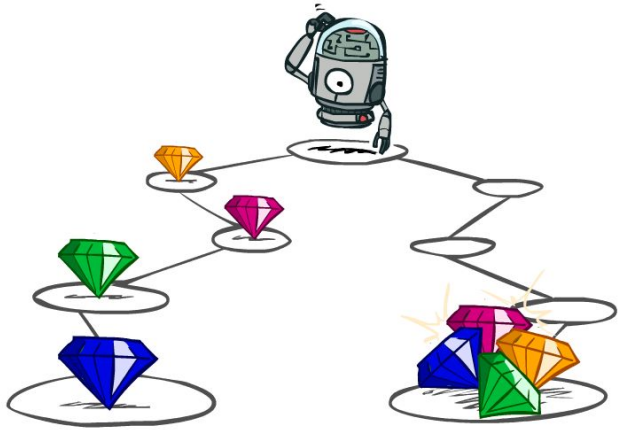
What preferences should an agent have over reward sequences?

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

You want to maximize the reward

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

Either is fine.
The stochasticacy (is this right?) of the system is



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**

Gamma is a value in [0, 1]

1

Worth Now

γ

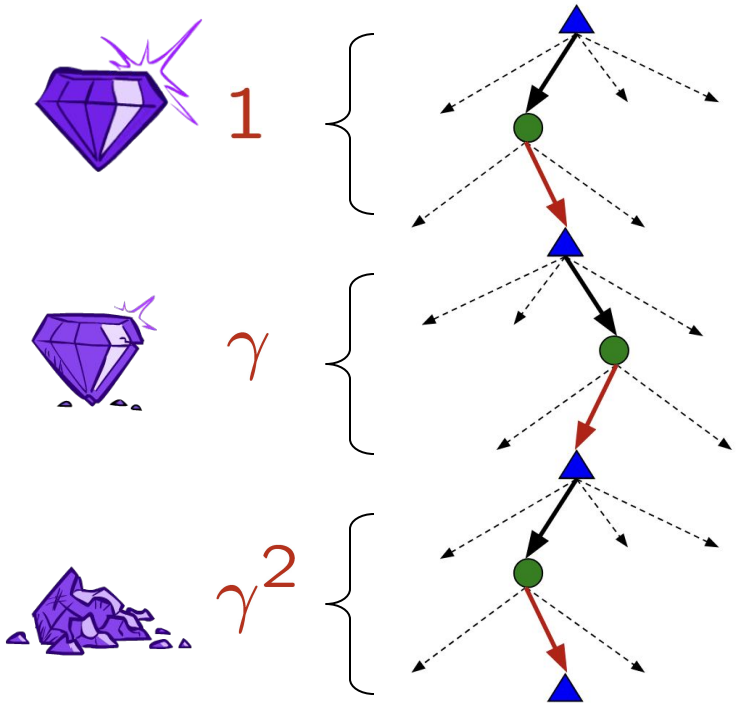
Worth Next Step

γ^2

Worth In Two Steps

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



Stationary Preferences

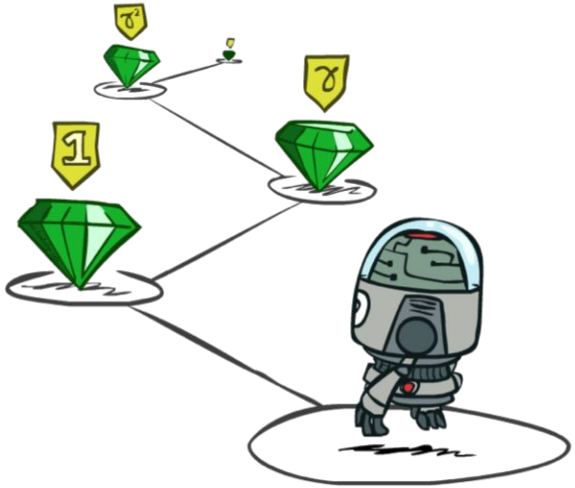
- Theorem: if we assume stationary preferences:

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

$$\Updownarrow$$

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

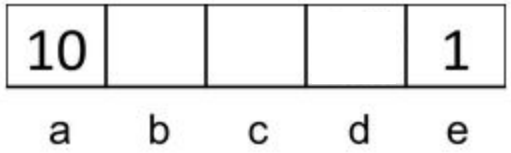
Dynamic preferences result in the preferences changing, altering the paths chosen.



- 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: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

Quiz: Discounting

- Given:



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

b, c, d have a reward of zero

- Quiz 1: For γ = 1, what is the optimal policy?

Go to the left

If at c, we move to 10

If in d, we move to 1

- Quiz 2: For γ = 0.1, what is the optimal policy?

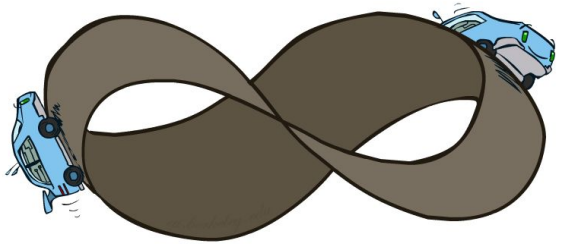
- Quiz 3: For which γ are West and East equally good when in state d?

Remember, gamma is in [0, 1]

+1/sqrt(10) = lambda

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution options:
  - **Finite horizon**: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives non-stationary policies (π depends on time left)
  - **Discounting**: use $0 < \gamma < 1$



What is R_max?
It is the maximum reward that can be obtained

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

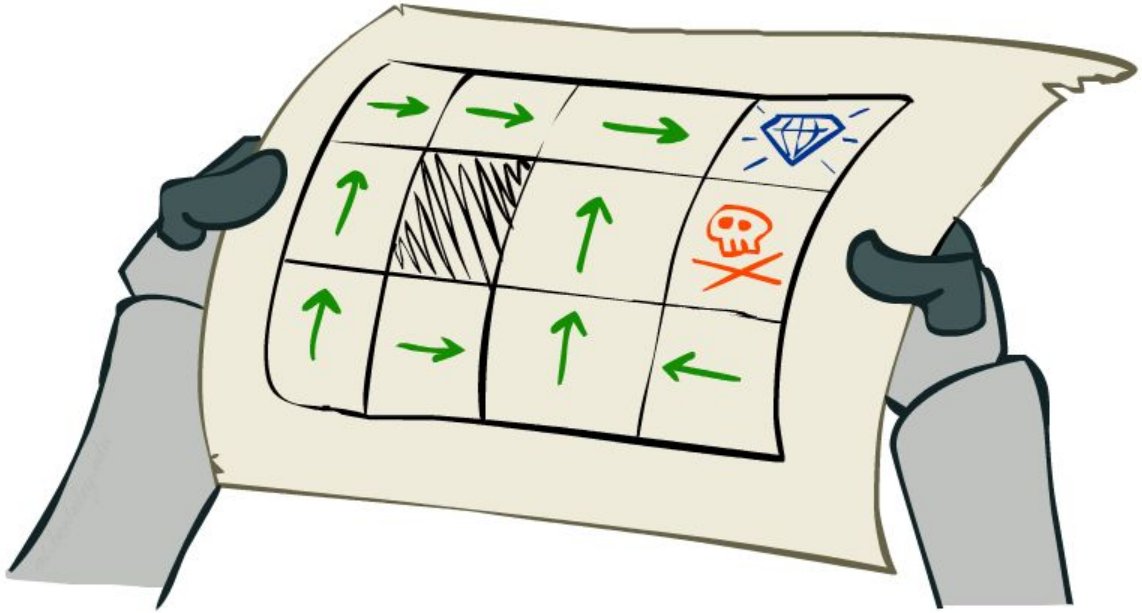
Utility will be bounded by this sum of an infinite geometric series

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

Solving MDPs



Optimal Quantities

$$V^*(s) = \text{max Q*(s, a) over all a}$$

- The **value** (utility) of a **state** s:
  $V^*(s)$ = expected utility starting in s and acting optimally

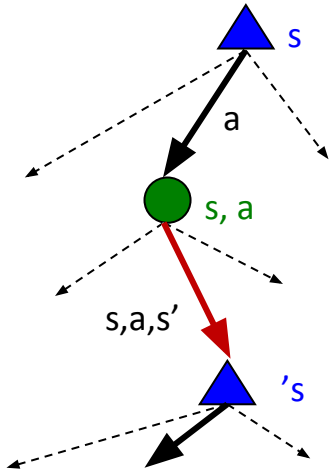
The q-state incorporates the probability

- 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

Ask Angelica to clarify whether the action has occurred

- The optimal policy:
  $\pi^*(s)$ = optimal action from state s

$$\pi^*(s) = \text{arg max Q*(s, a) over all a}$$



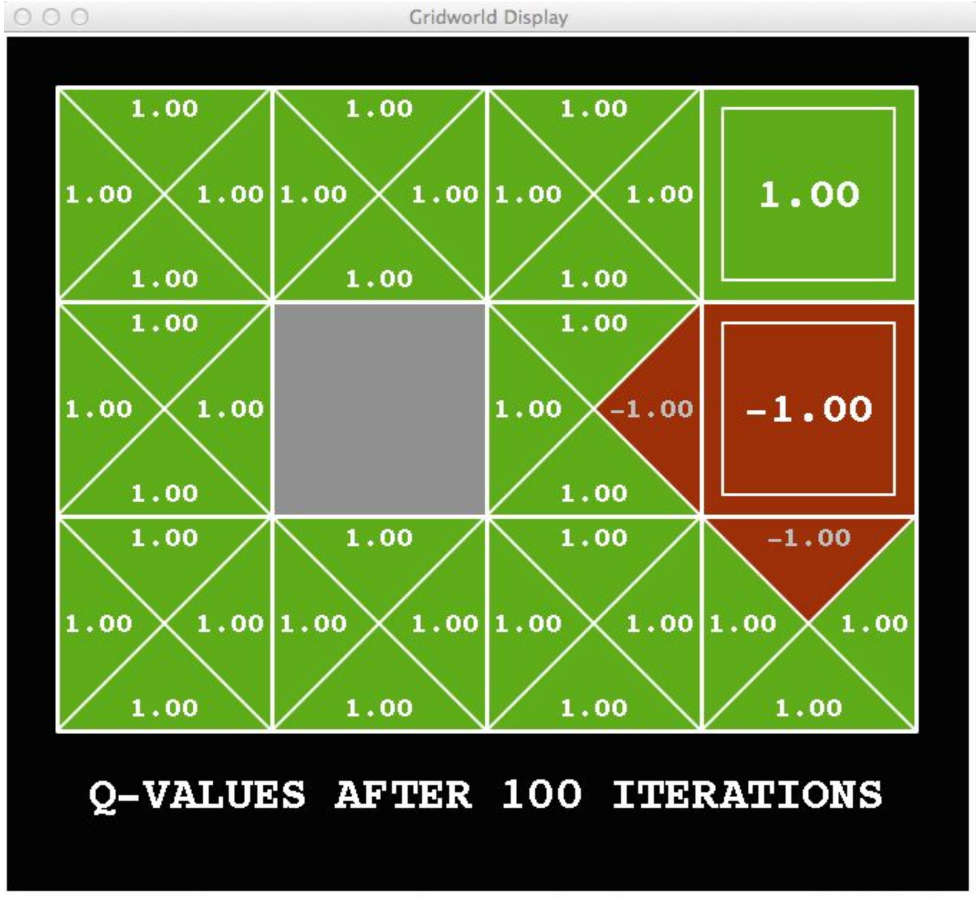
s is a *state*

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

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

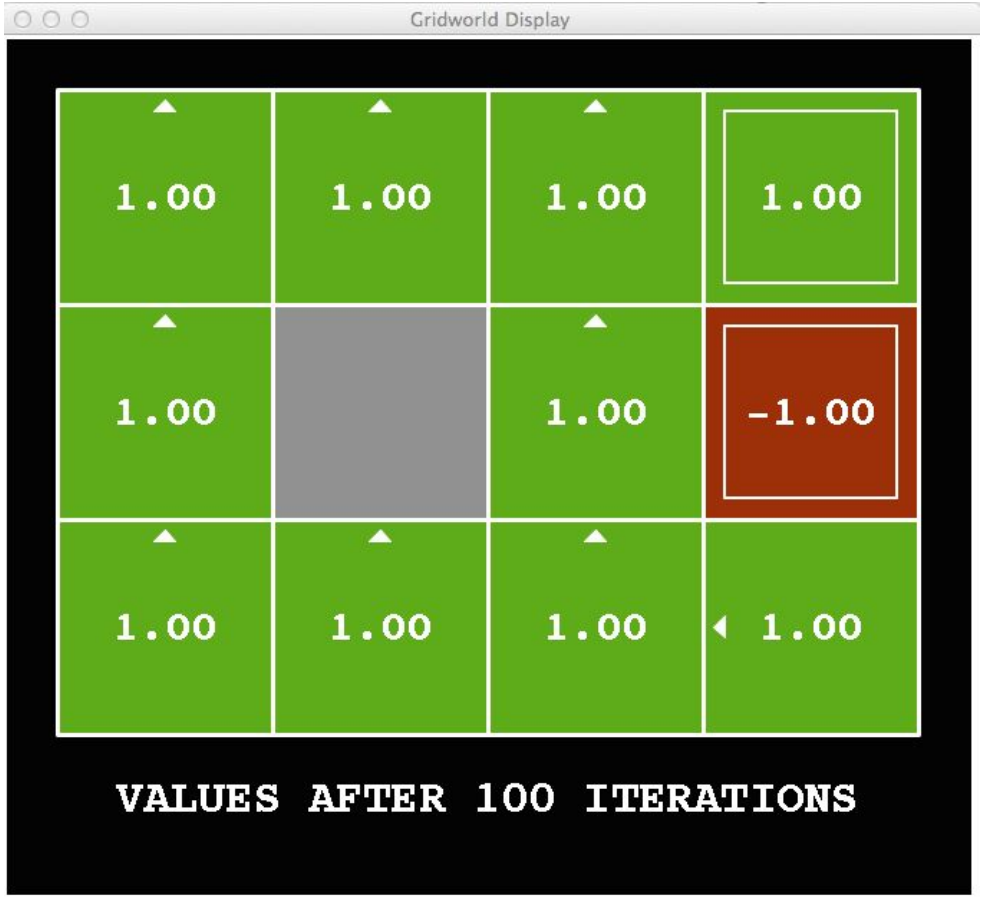
Gridworld **Q** Values

Only way to obtain -1 is by moving to that square.



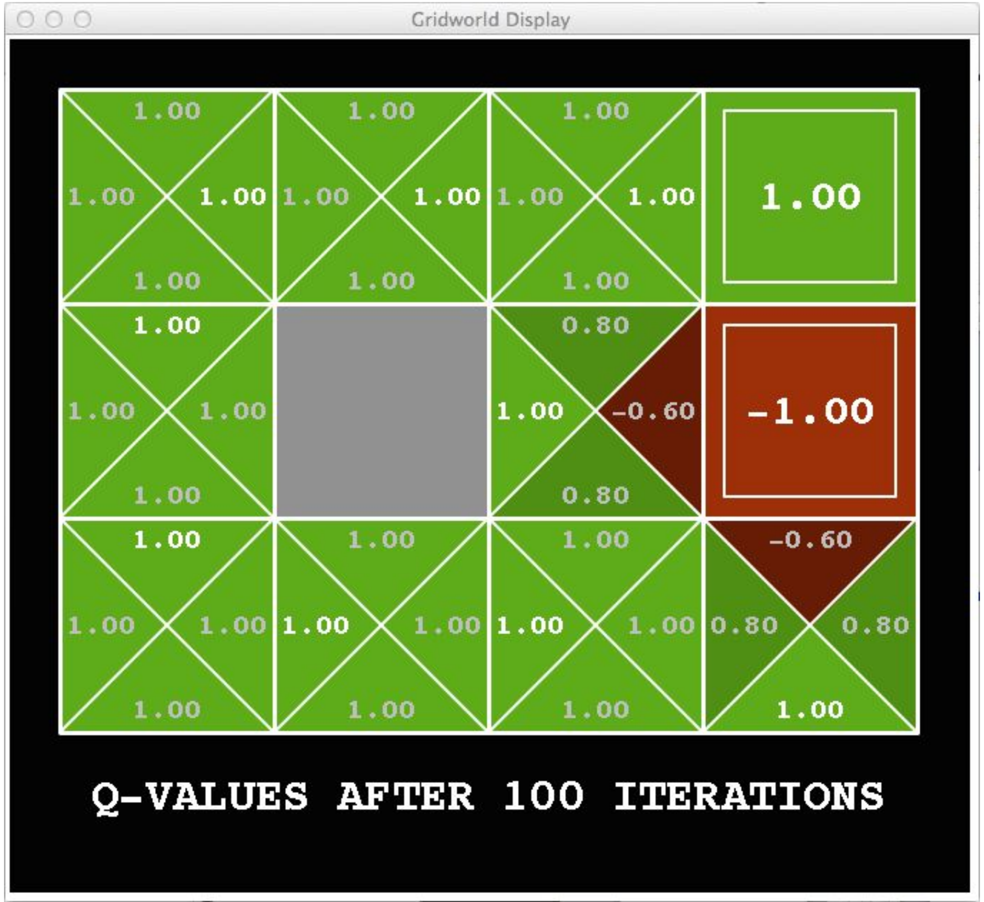
Noise = 0
Discount = 1
Staying alive reward = 0

Gridworld **V** Values



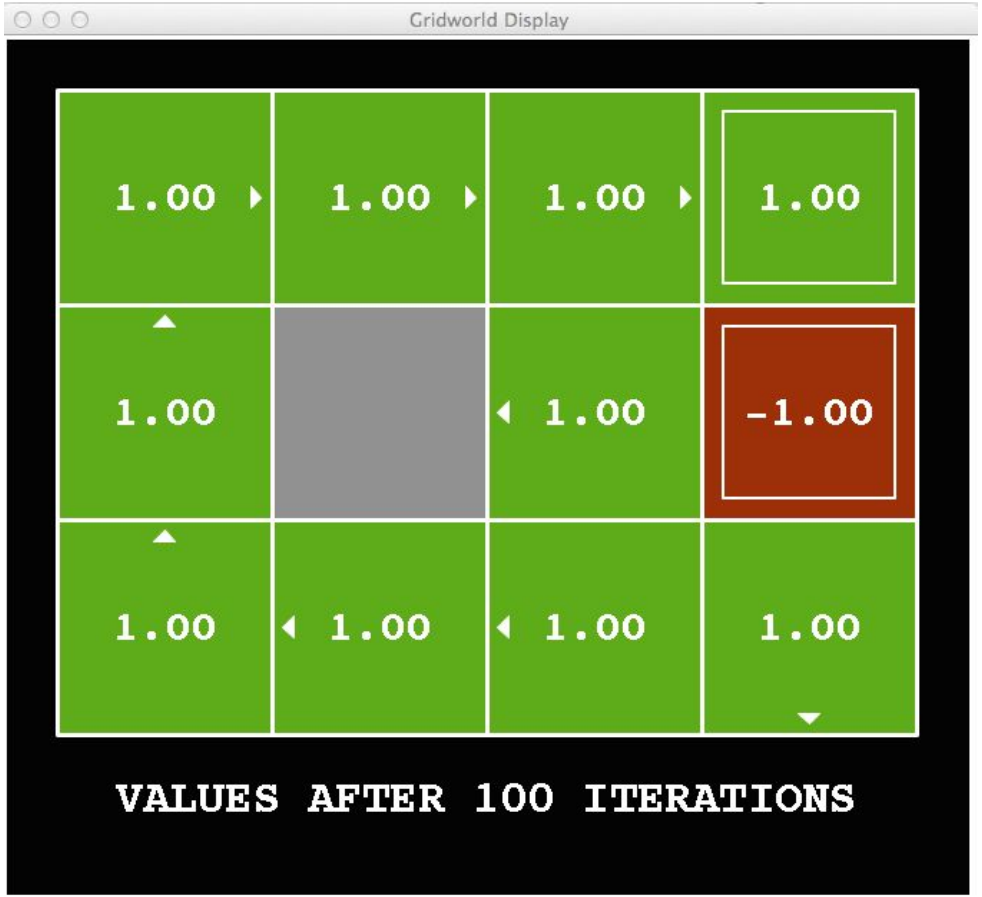
Noise = 0
Discount = 1
Staying alive reward = 0

Gridworld Q Values



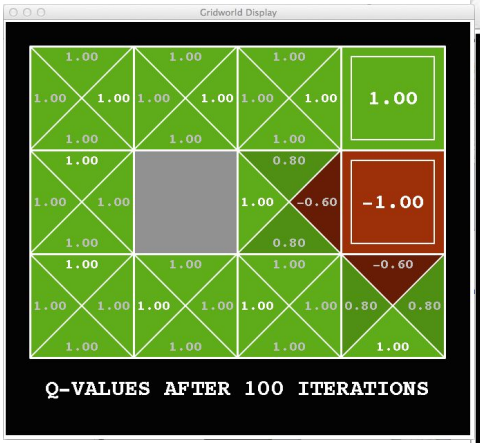
Noise = 0.2
Discount = 1
Staying alive reward = 0

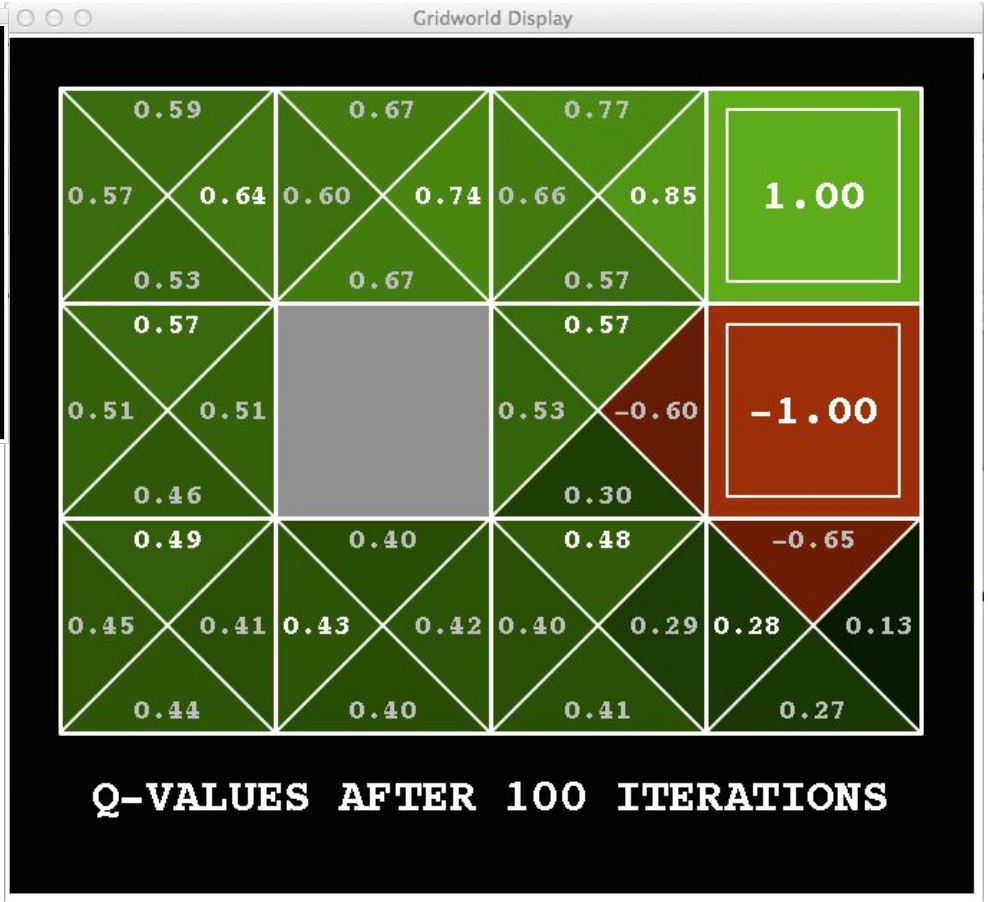
Gridworld V Values



Noise = 0.2
Discount = 1
Staying alive reward = 0

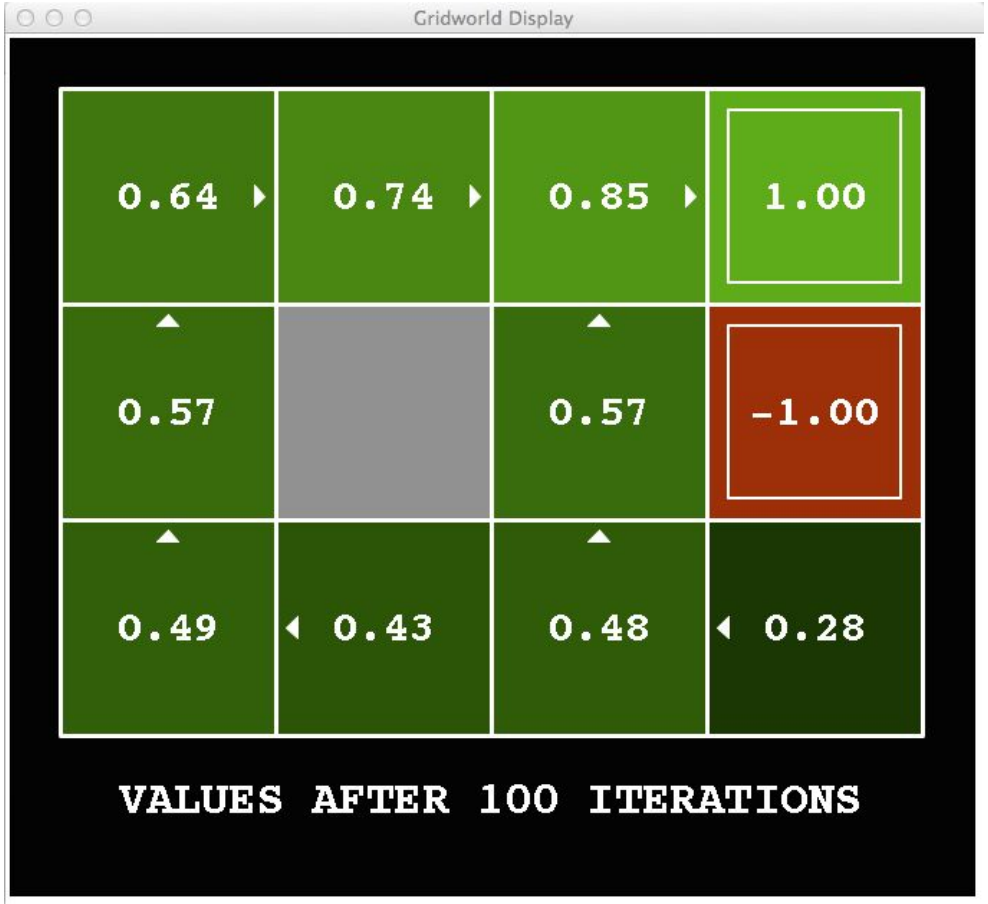
Gridworld Q Values





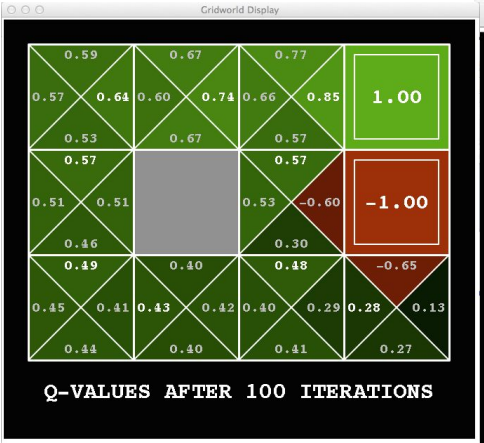
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

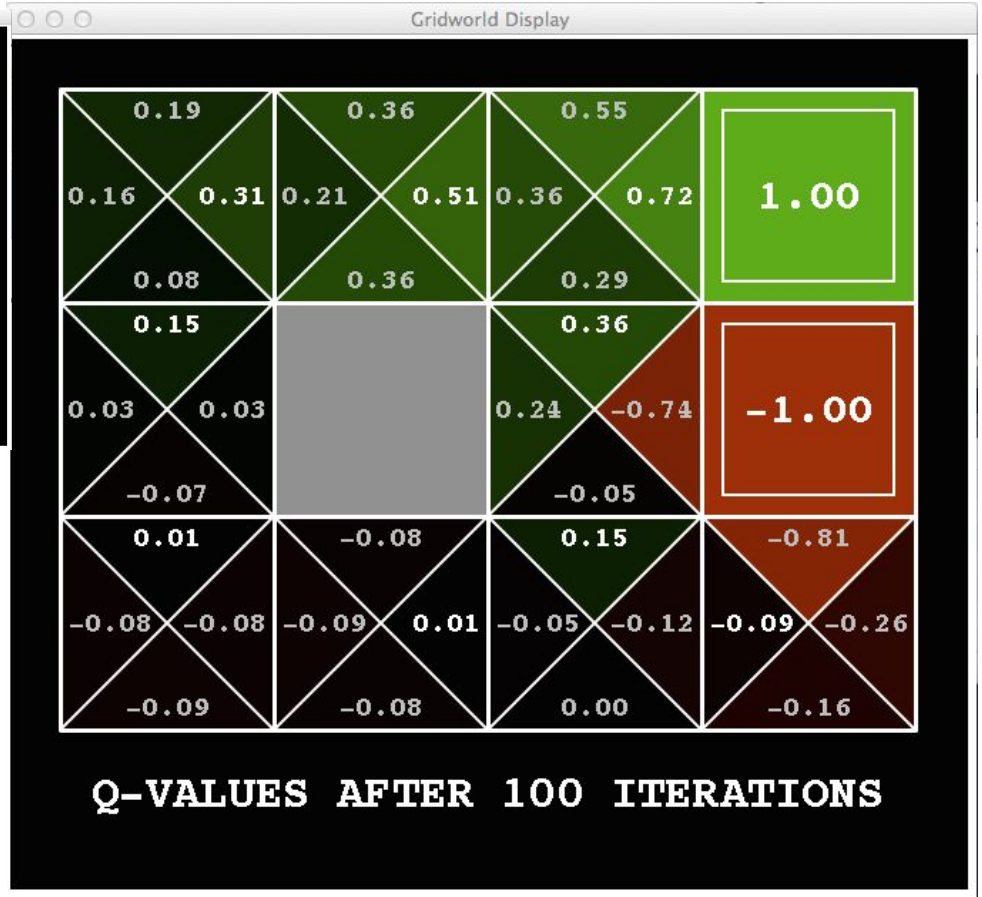
Gridworld V Values



Noise = 0.2
Discount = 0.9
Staying alive reward = 0

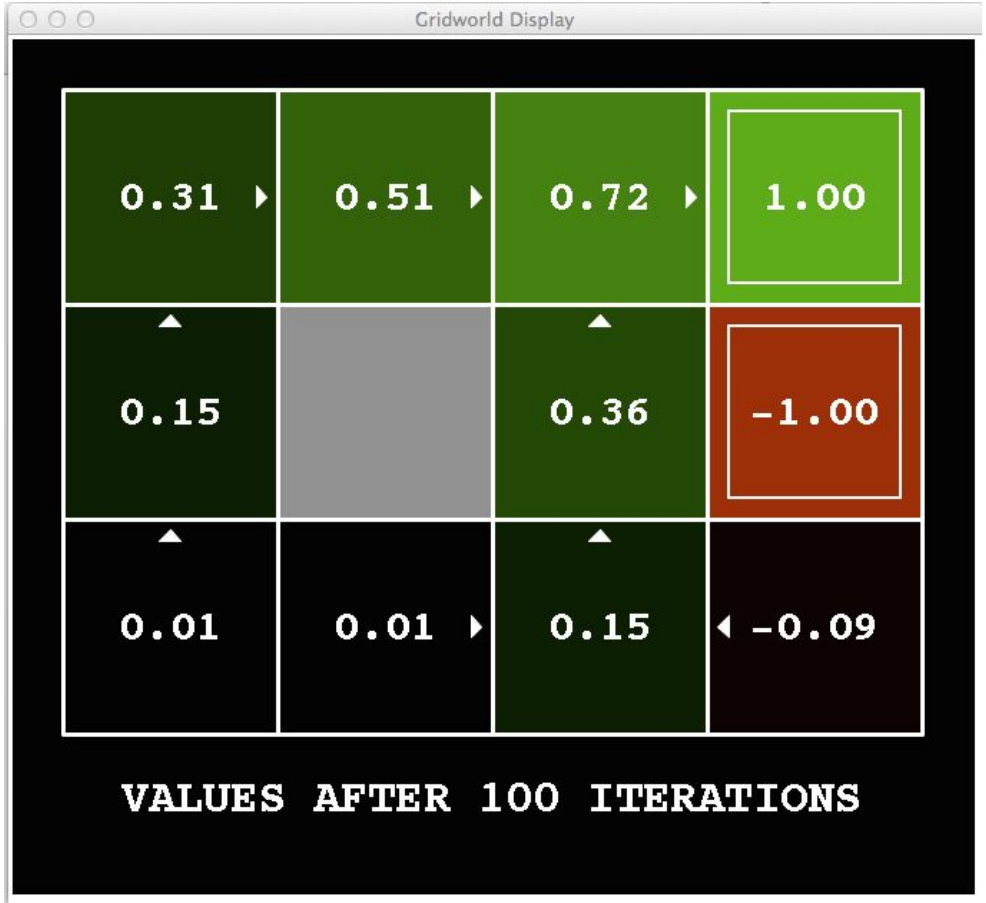
Gridworld Q Values





Noise = 0.2
Discount = 0.9
Staying alive reward = -0.1

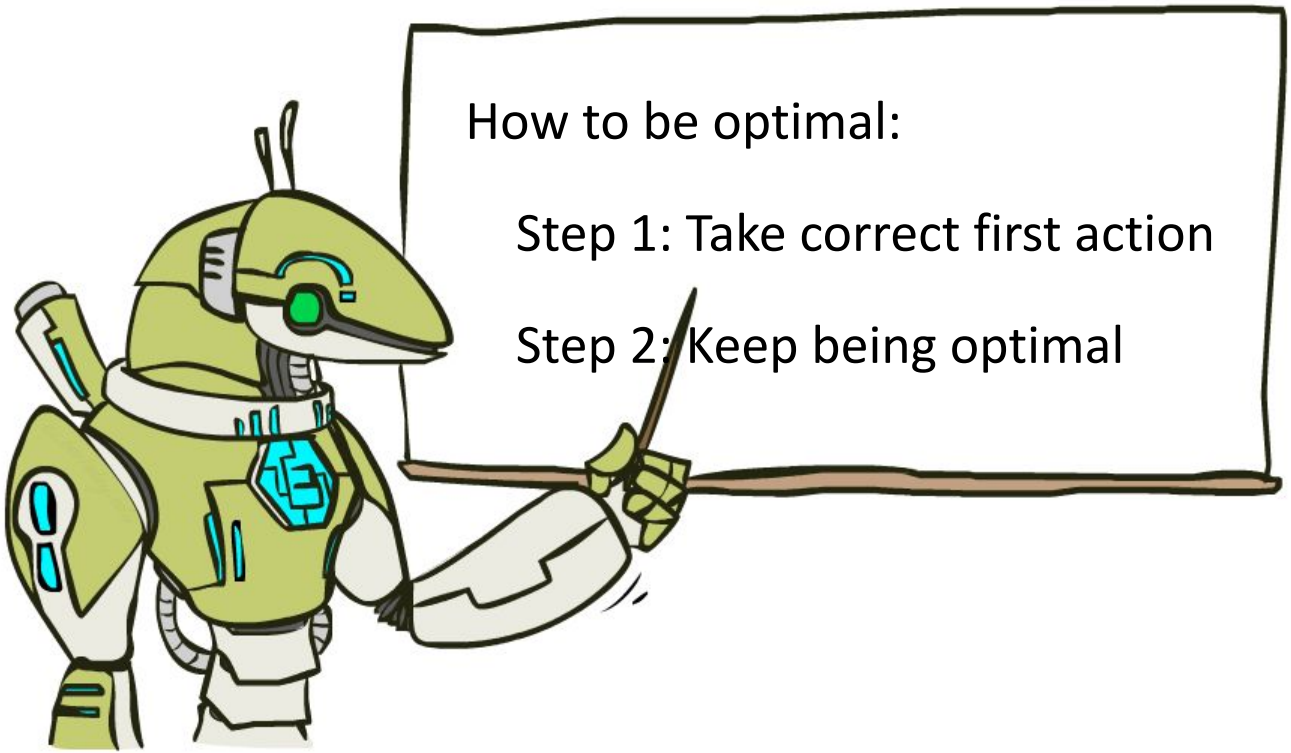
Gridworld V Values



Noise = 0.2
Discount = 0.9
Staying alive reward = -0.1

An agent will want to take the optimal action based on Q-values

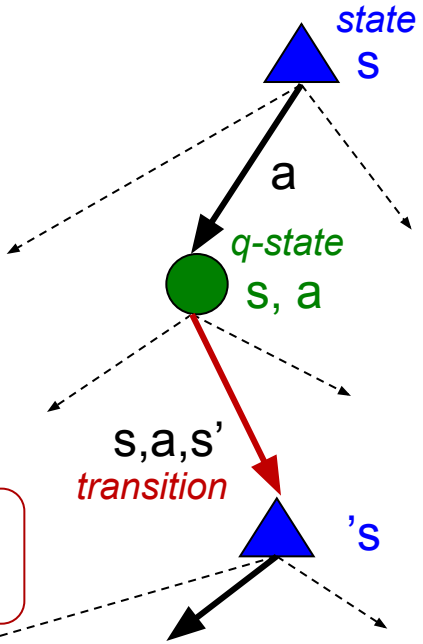
The Bellman Equations



How to compute value

How to compute the **value** of a state, V*(s):

- **Expected utility** under **optimal** action (denoted by *)
- Average **sum** of (discounted) **rewards**



Recursive definition of **value**:

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

Expected utility (reward) from state s

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

R(s, a, s') + lambda *

Expected utility (reward) of action a from state s

Discount factor

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

Probability of the action

Reward in taking the action

Value from new state

The Bellman Equations

- Definition of “optimal utility” is a recurrence

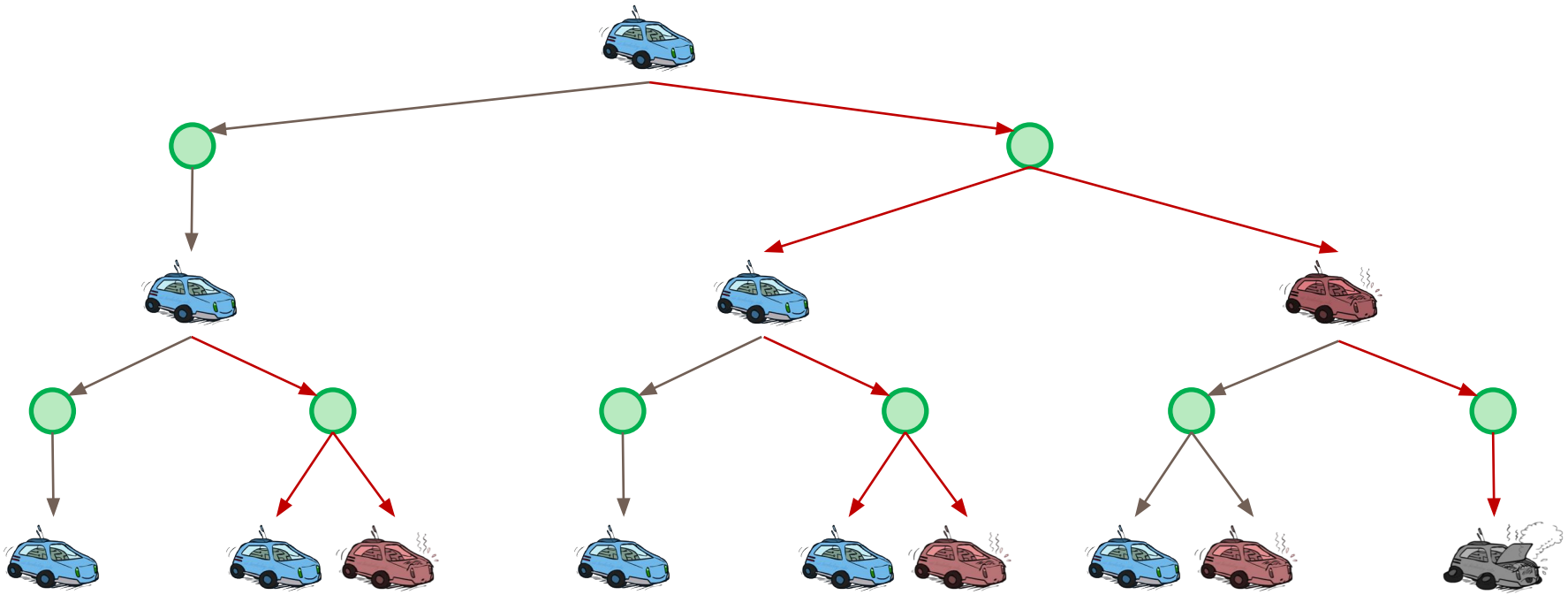
$$V^*(s) = \max_a Q^*(s,a)$$

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

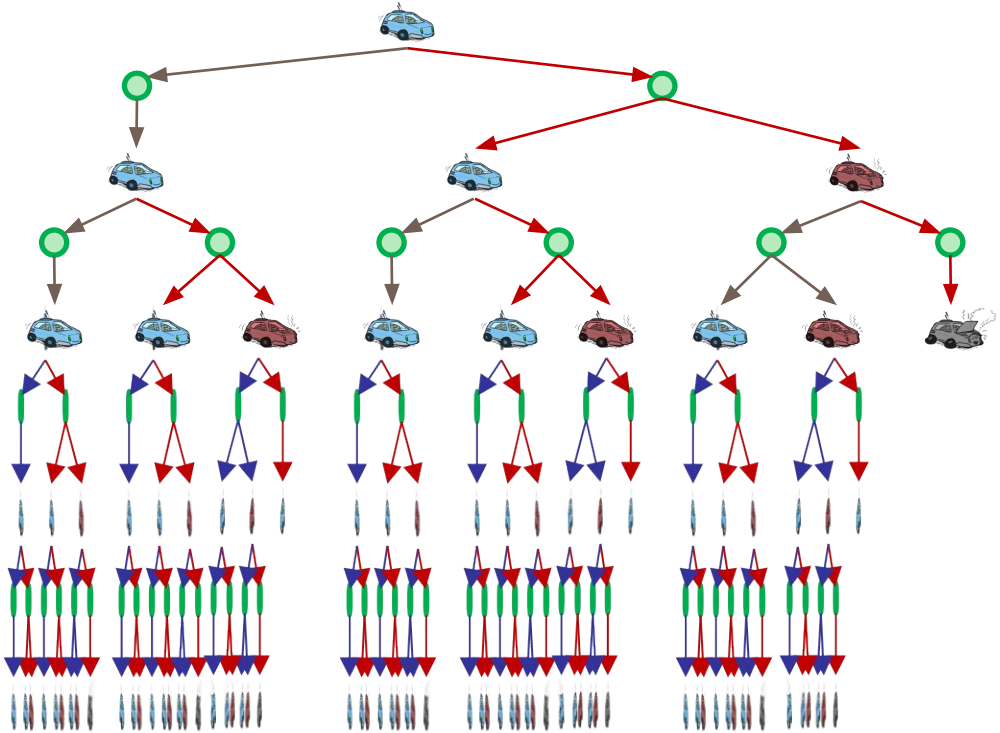
- These are the Bellman equations, and they characterize optimal values in a way we’ll use over and over

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

Racing Search Tree



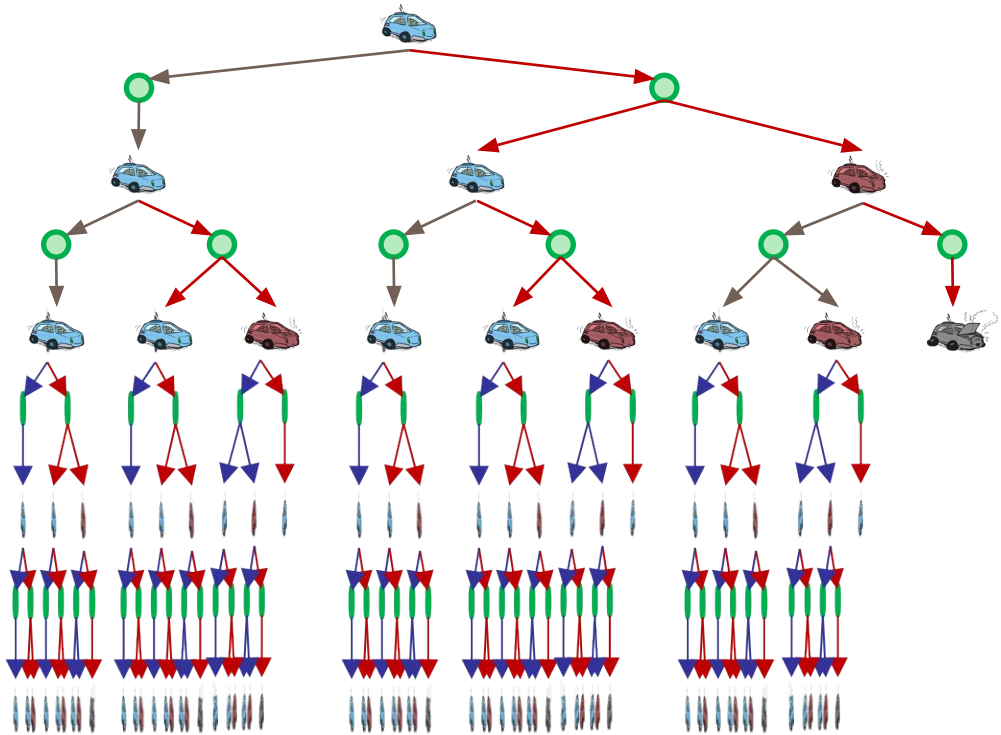
Racing Search Tree



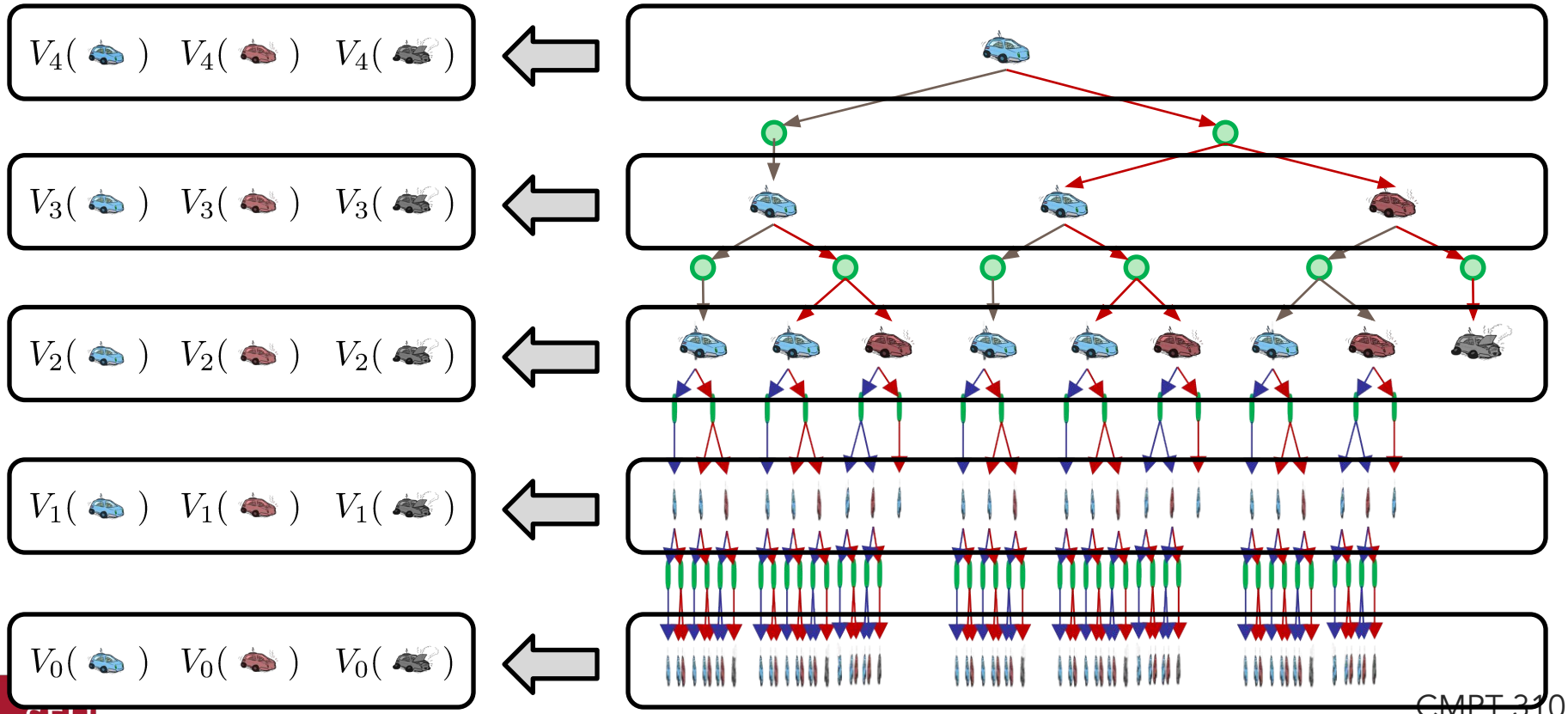
Racing Search Tree

- Notice: 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 $\gamma < 1$

What happens if lambda is small, but the rewards are so big that you still explore deep parts of the tree?

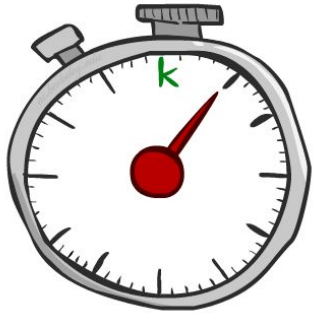


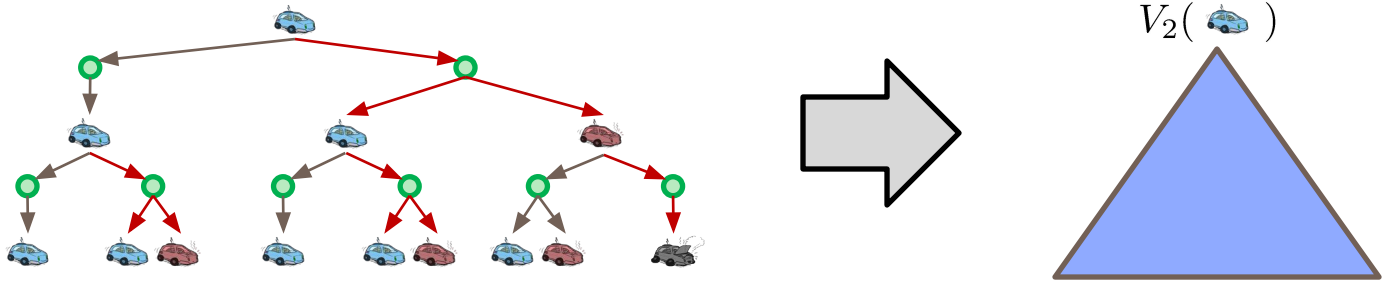
Computing Time-Limited Values



Time-Limited Values

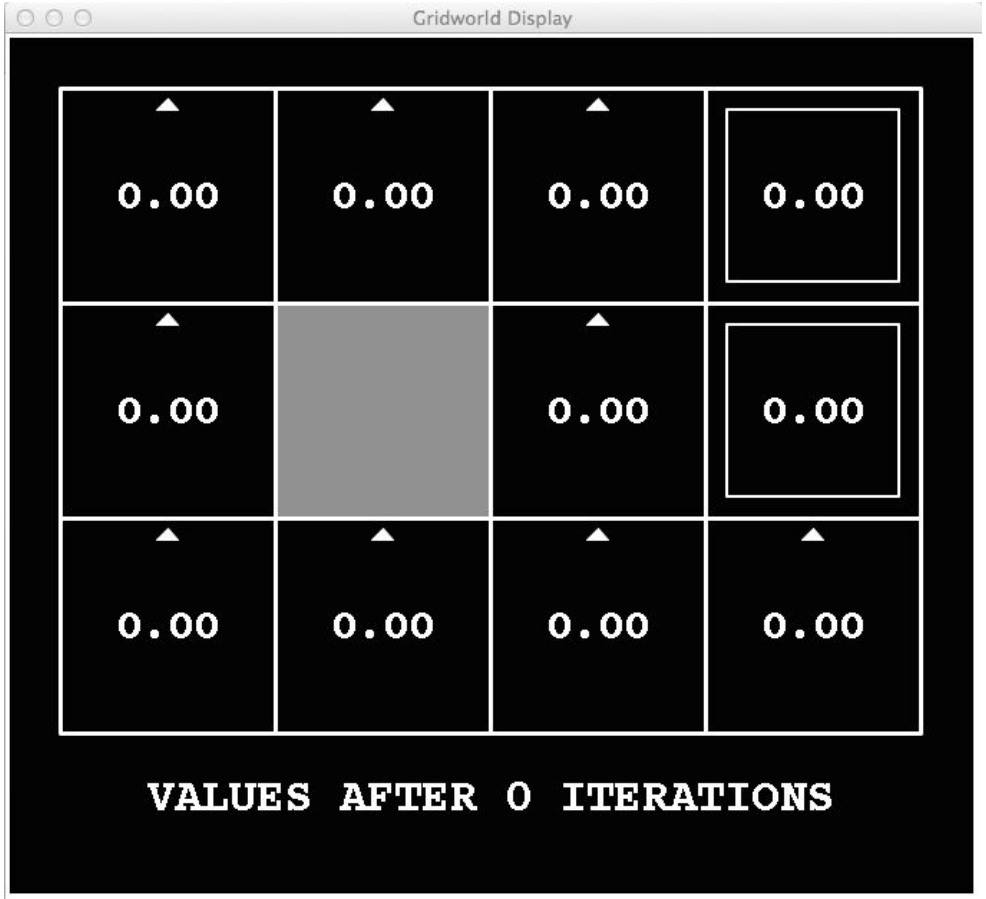
- **Key idea**: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps





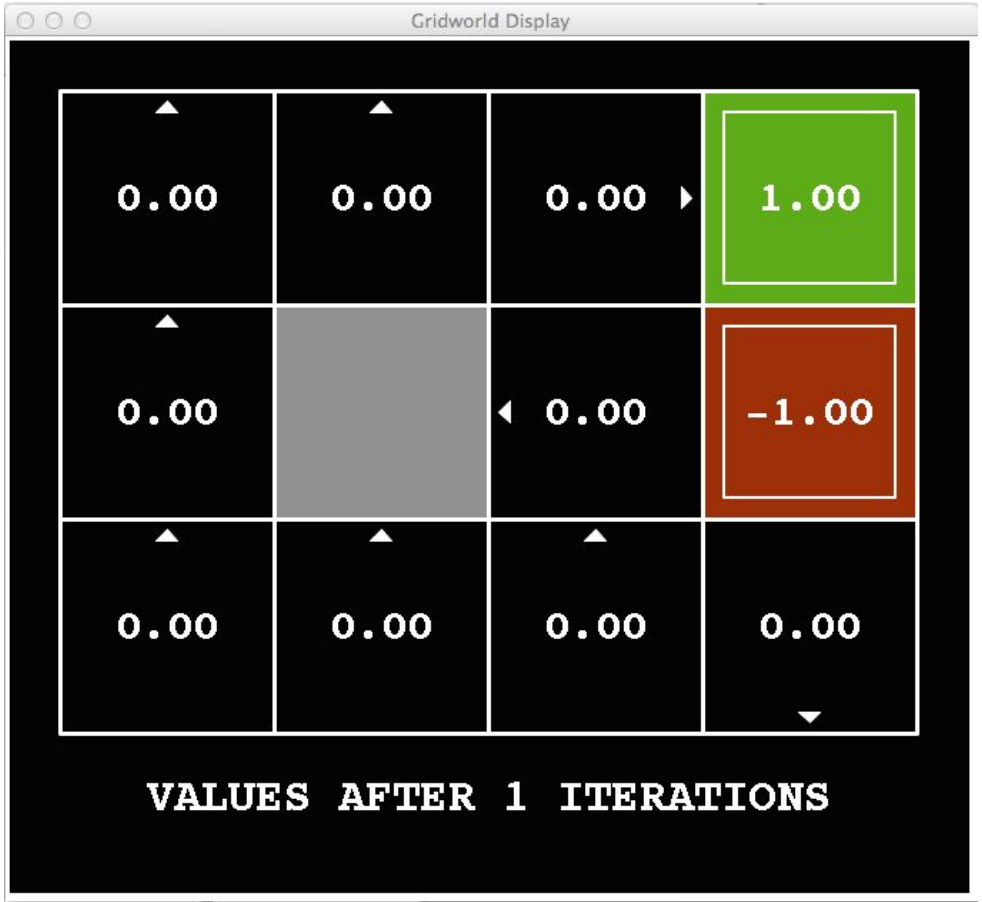
[Demo – time-limited values (L8D6)]

k=0



Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=1



Noise = 0.2
Discount = 0.9
Staying alive reward = 0

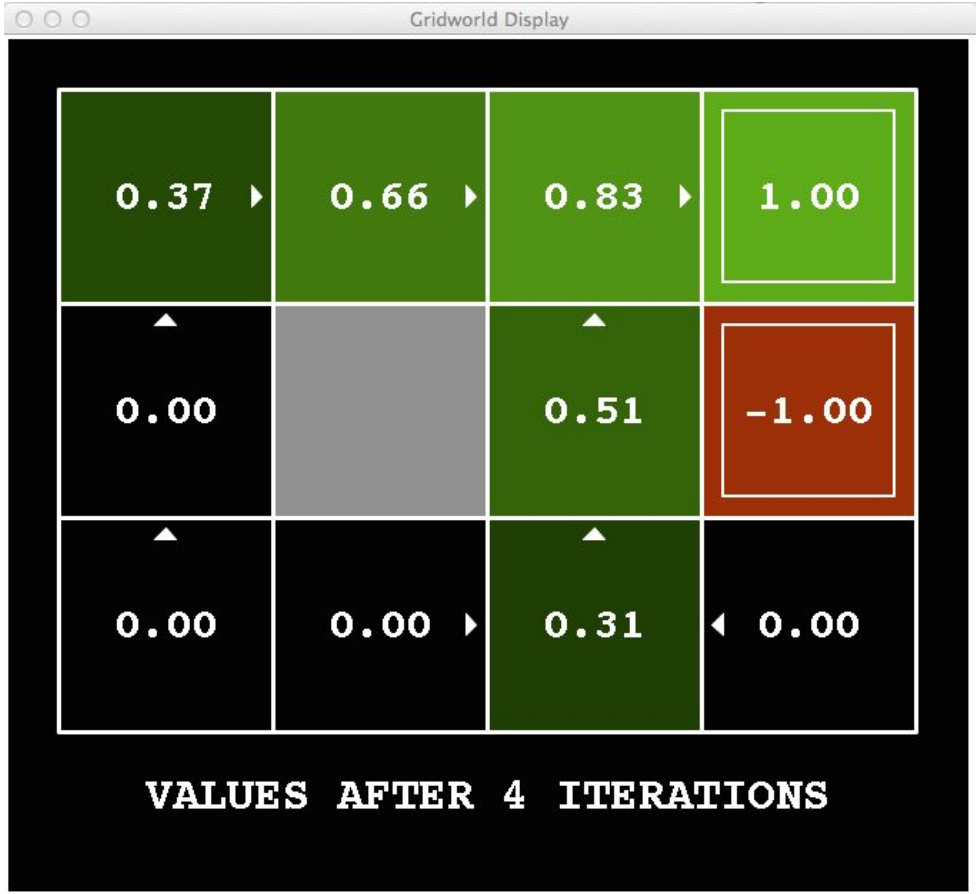
k=2

Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=3

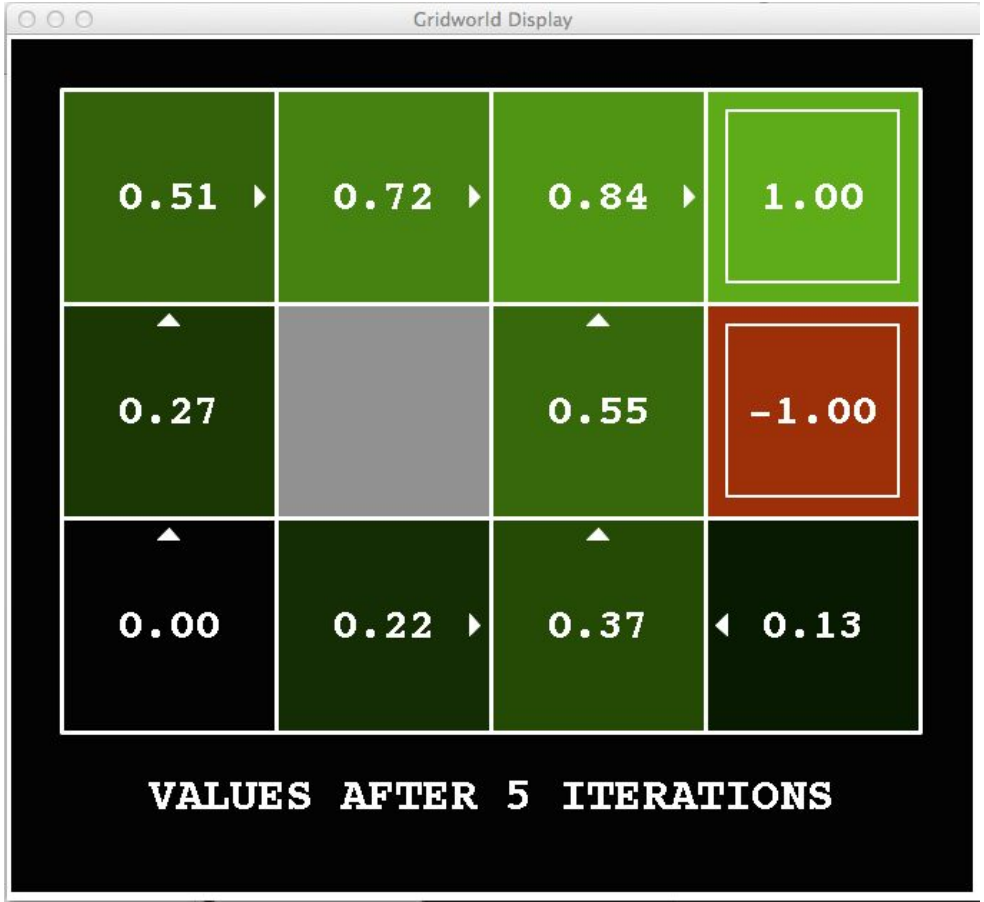
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=4



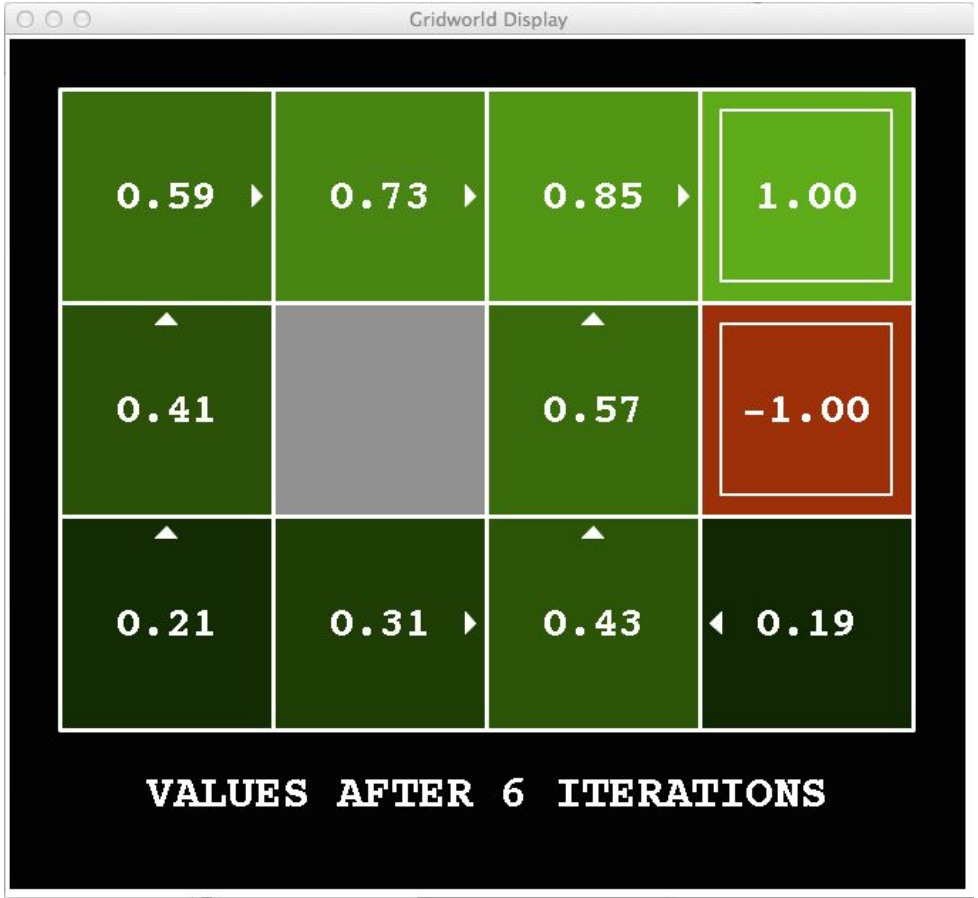
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=5



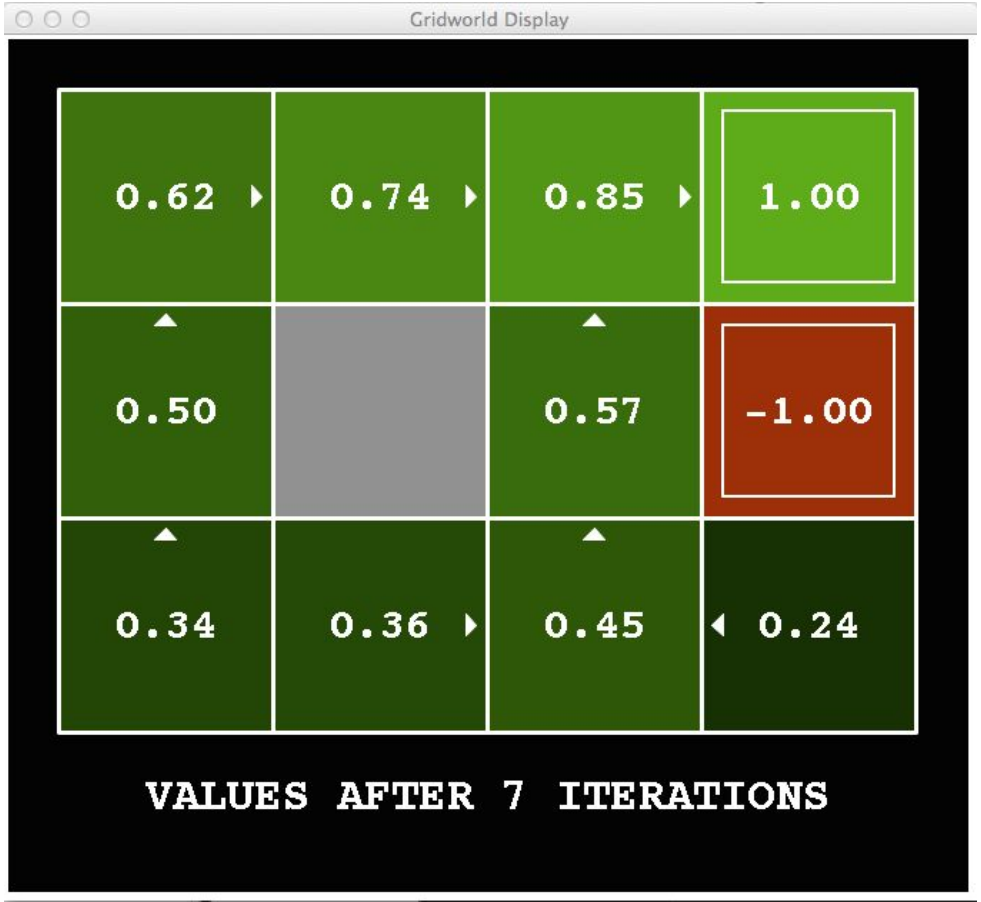
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=6



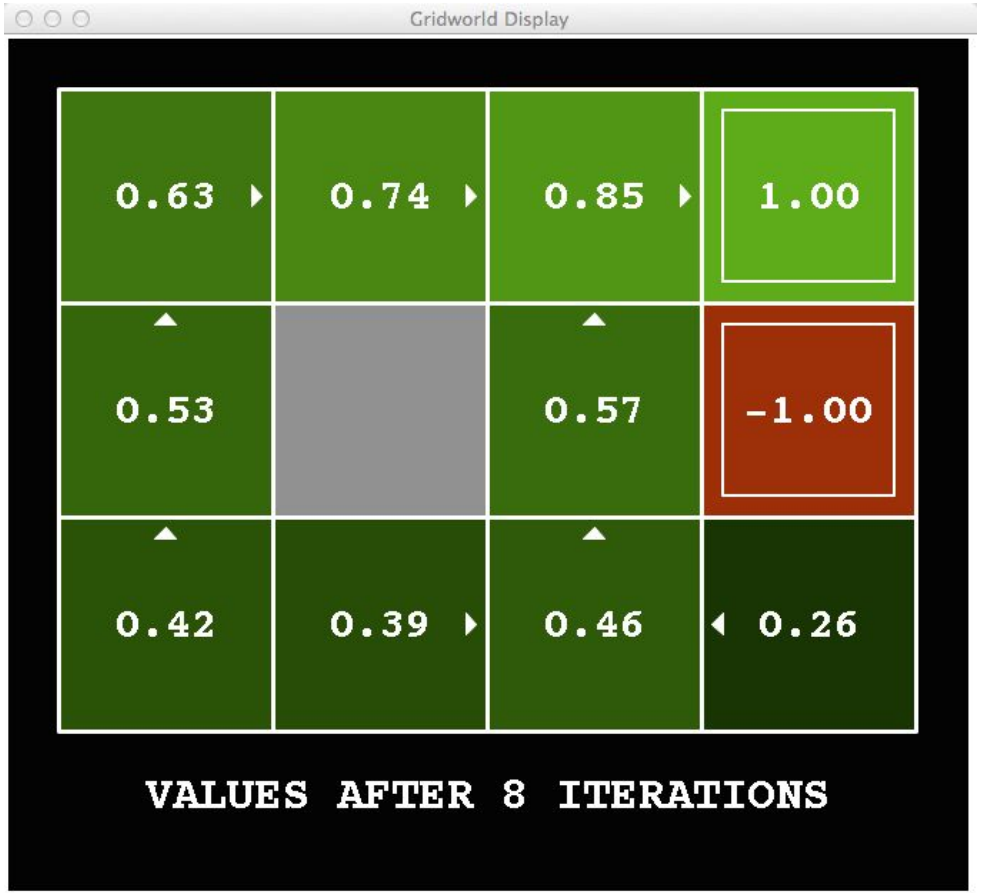
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=7



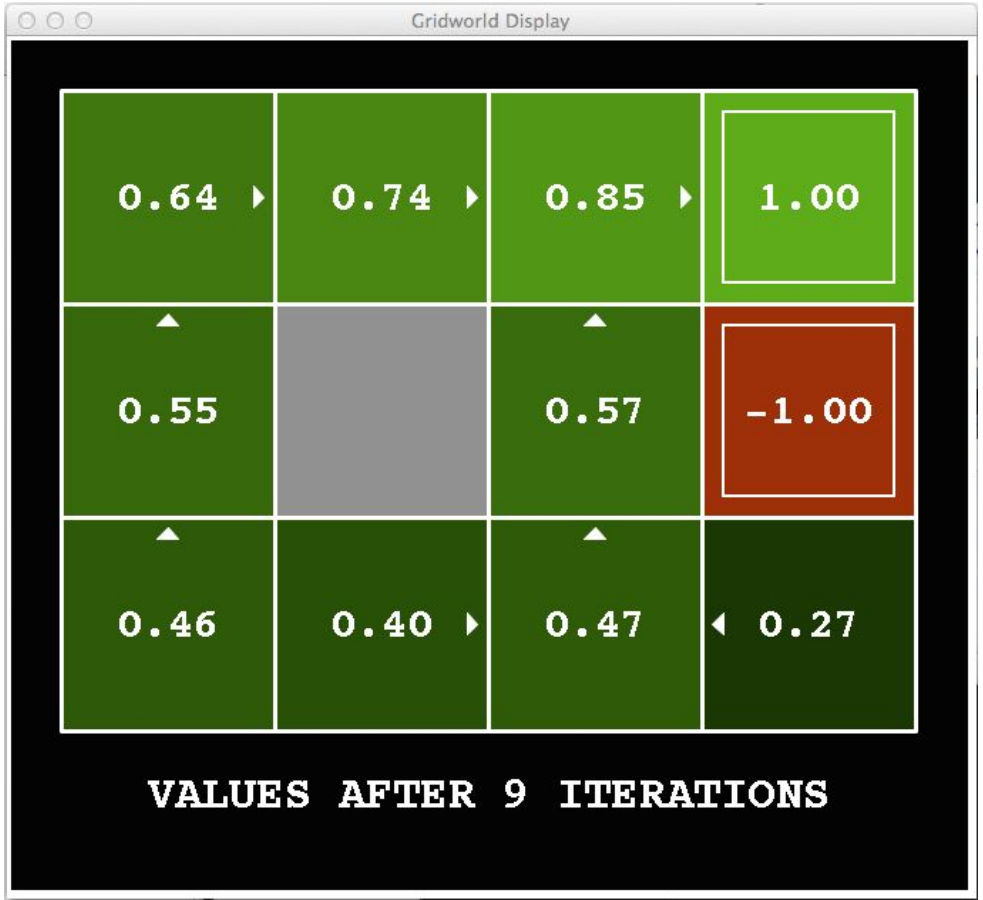
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=8



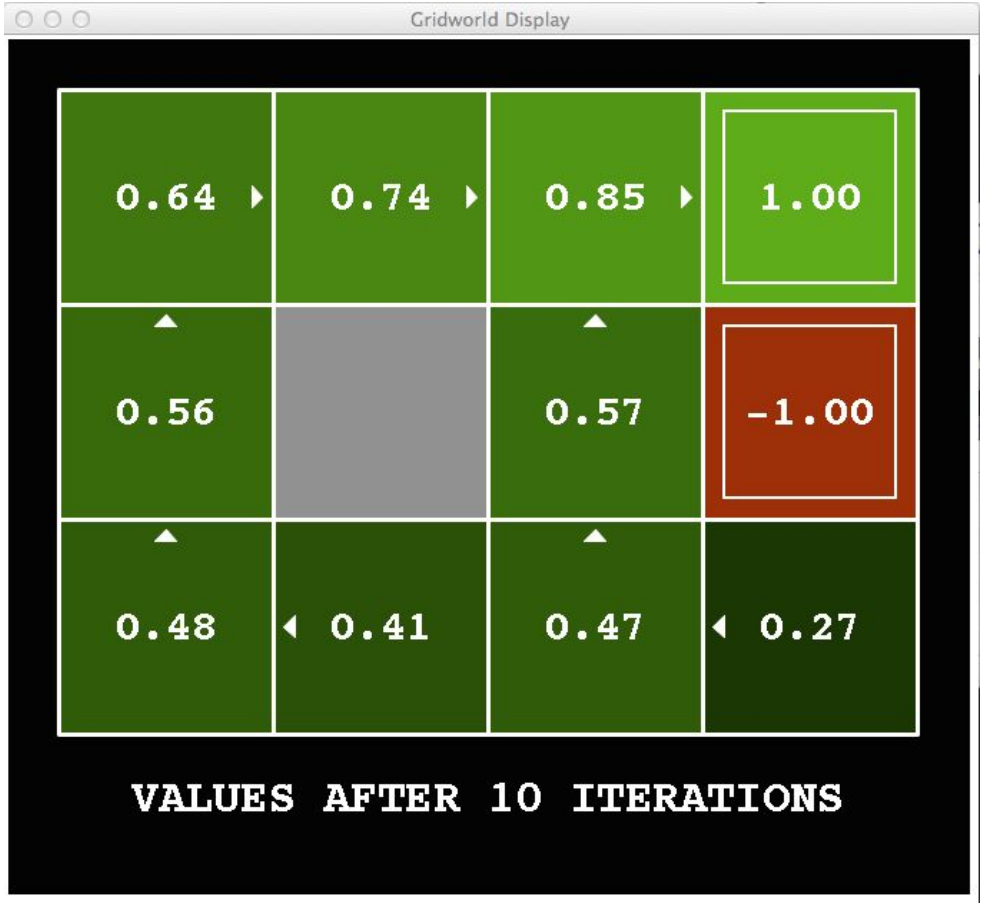
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=9



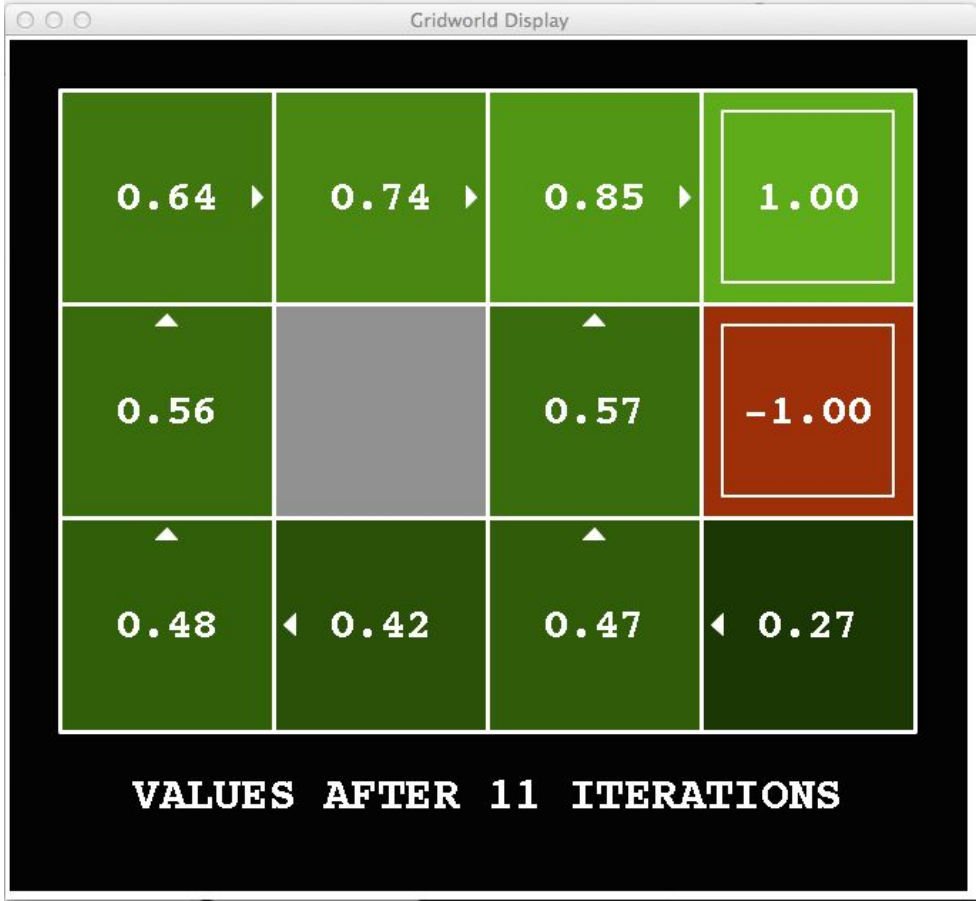
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=10



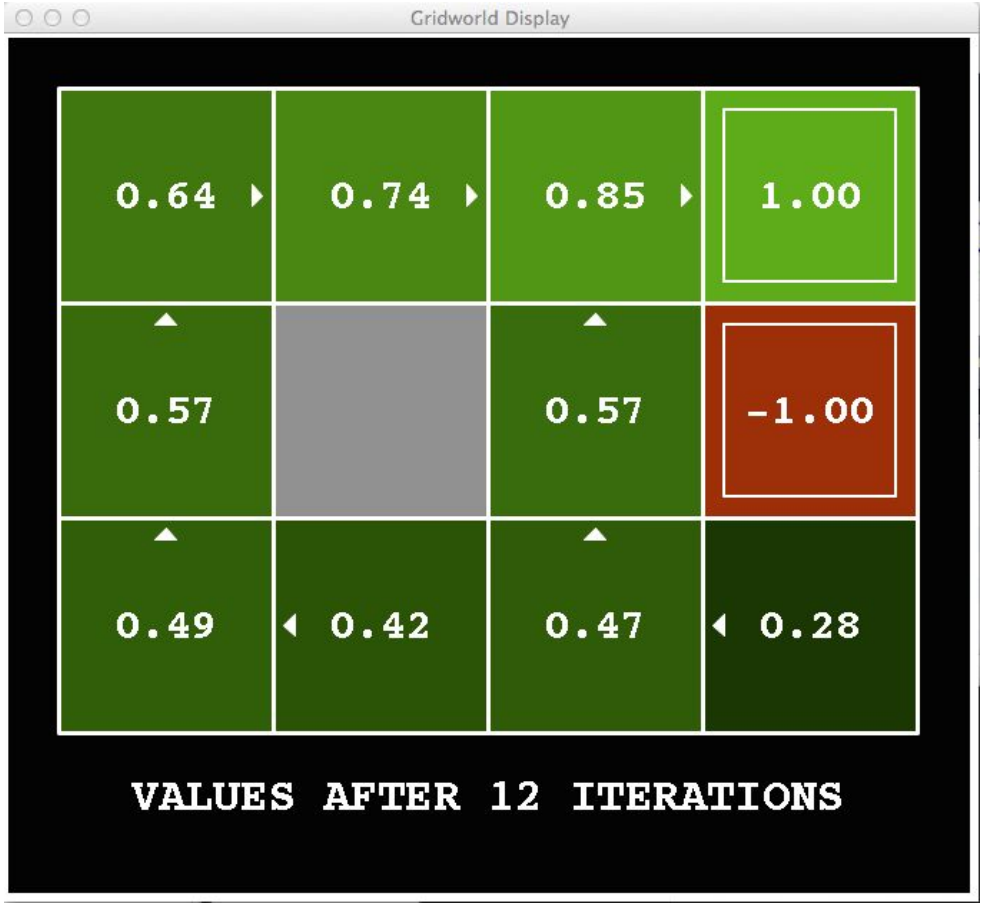
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=11



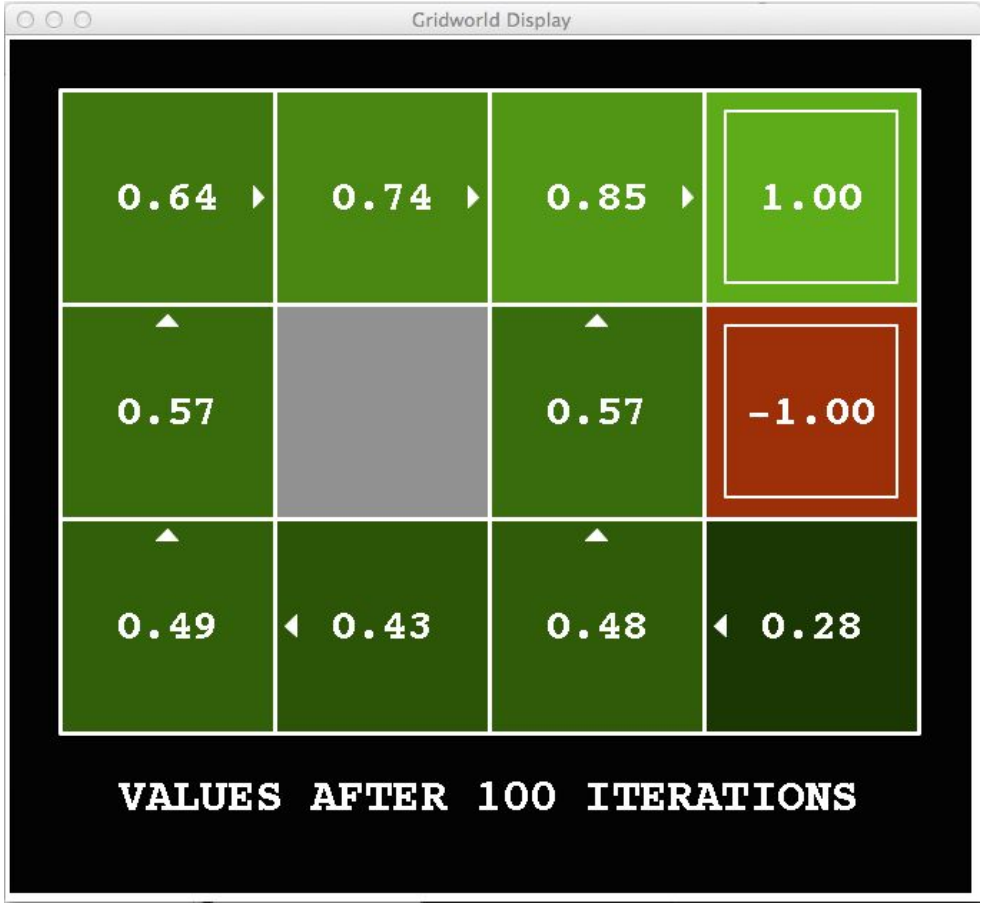
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=12



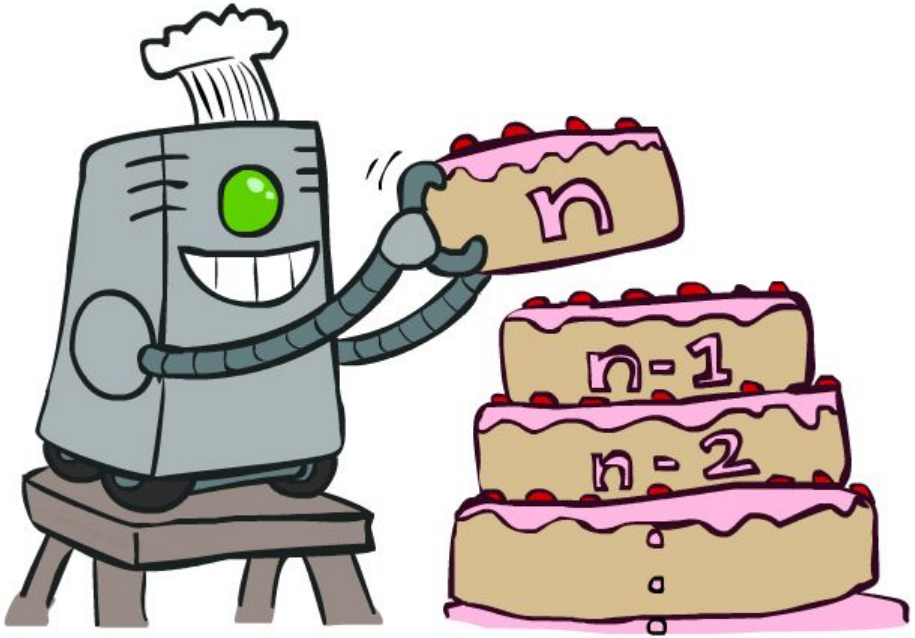
Noise = 0.2
Discount = 0.9
Staying alive reward = 0

k=100



Noise = 0.2
Discount = 0.9
Staying alive reward = 0

Value Iteration



a

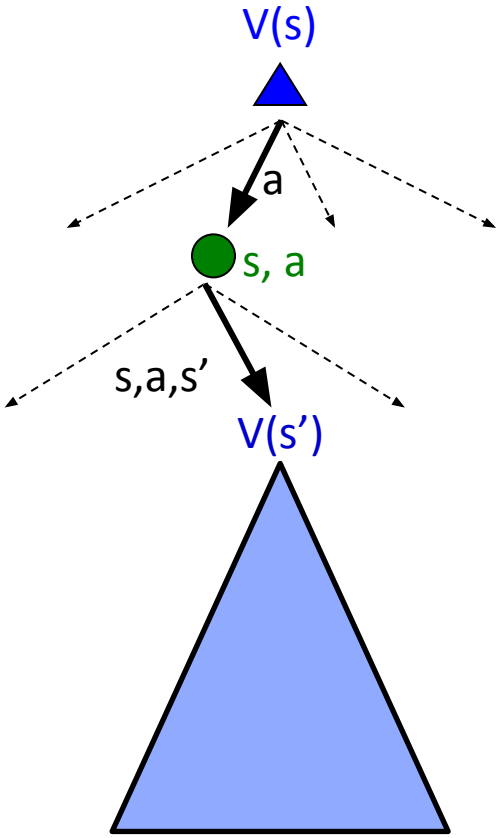
Value Iteration

- Bellman equations characterize the optimal values:

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

- Value iteration computes them:

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



Example: Value Iteration

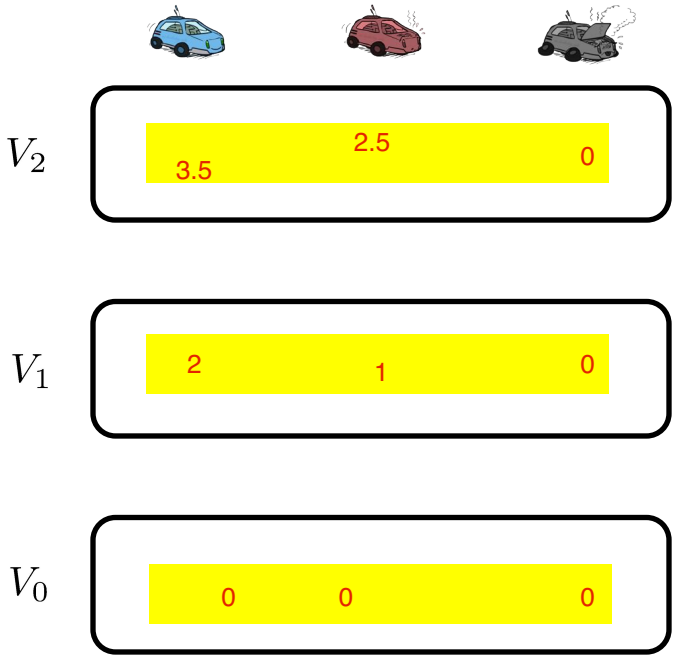
s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	–10
	(end)		1.0	0

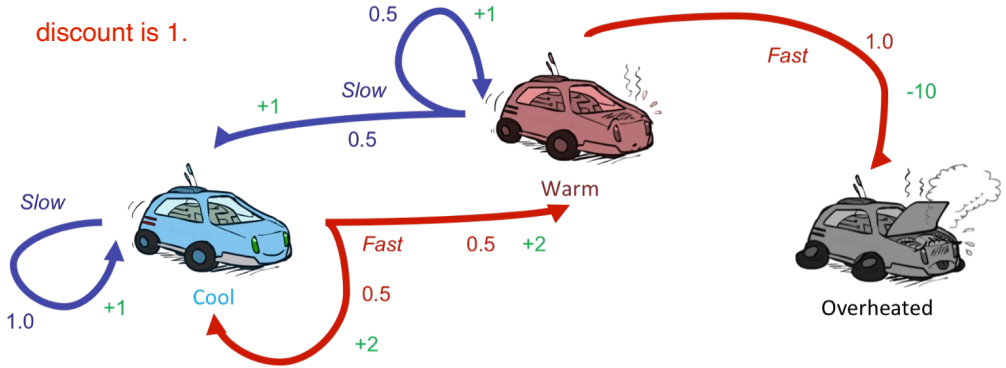
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Assume no discount!

Example: Value Iteration

Ask Angelica about these values.



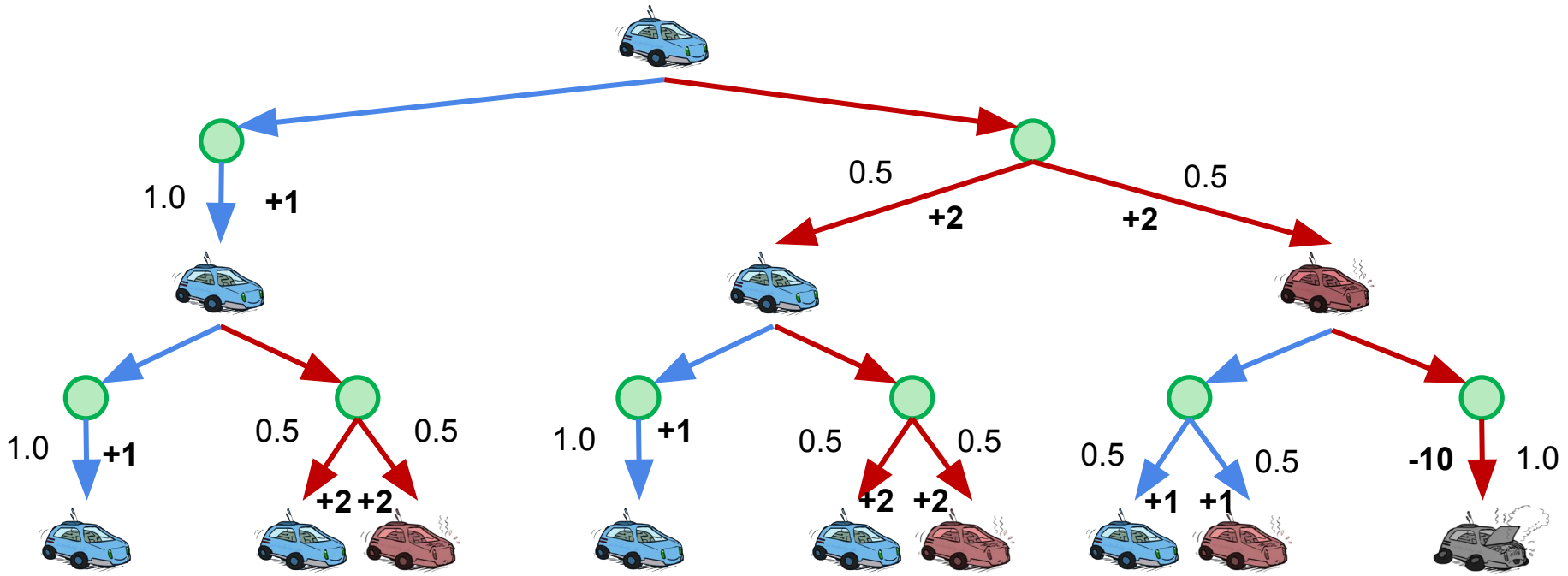


Assume no discount!

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

$$\max_a Q\ (s,a)$$

Racing Search Tree: Transition Probabilities and **Rewards**



Recap: MDPs

- **Markov decision processes:**
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0

- **Quantities**:
  - **Policy** = map of states to actions
  - **Utility** = sum of discounted rewards
  - **Values** = expected future utility from a state (max node)
  - **Q-Values** = expected future utility from a q-state (chance node)

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:

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

- 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

O(SA)

If the number of potential states, action, new state triples to check is O(SA) and it has to be done for each triple, and this occurs recursively, would it not be O((SA)^k)?

Next Time: Policy-Based Methods