

# COSC1125/1127 Artificial Intelligence

## Week 7: Markov Decision Process

[RN2] Section 17.1 – 17.3

[RN3] Section 17.1 – 17.3

Please also read Dan Klien and Pieter Abbeel's slides/youtube, as the slides here are mostly based on theirs:  
[https://www.youtube.com/watch?v=Oxqwwnm\\_x0s](https://www.youtube.com/watch?v=Oxqwwnm_x0s)

---

# House Keeping Information

---

- ❑ Class test will be run on **10 May during week 9 lecture**. It will be a supervised test, and will need to be completed by filling in online Googleform. The class test will account for 10% (covering topics from week 1 to 6).
- ❑ Assignment#1 results have been released. If any question, please talk to the your lab assistant during the lab time. We encourage you to do so, as this is how we learn from feedback!
- ❑ For Assignment#2, please make sure that you only submit work of your own. **We will run plagiarism detection software on all submissions!!**
- ❑ For Assignment#3, it will be important to learn Markov Decision Process, in order to complete it.

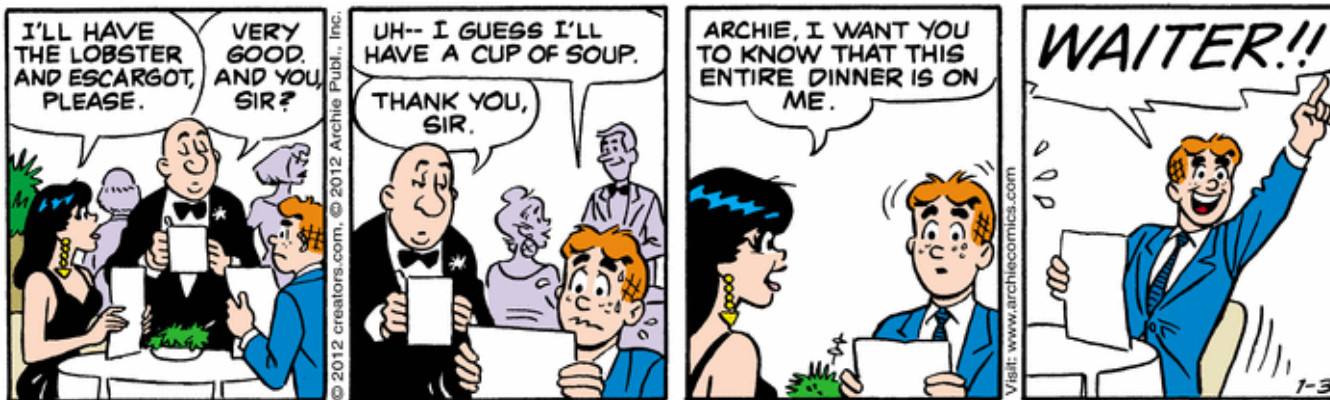
# Making Complex Decisions

Deterministic logical agent has a goal and executes (any) plan that achieves the goals. In an **uncertain** world, this is no longer the case:

- Flight A90: 95% making to airport on time.
- Flight A120: 98% making it on time.
- Flight A1440: 100% making it on time.

Relax goals: consider agent **preferences** for different outcomes.

**Utility theory** = associated degree of usefulness (utility) to each outcome, with outcomes of higher utility preferred.



---

# Simple vs Complex Sequential Decisions

---

**Simple decisions:** utility of plan based on **one action/decision**:

- Whether I take a taxi or train to airport?
- Where to go for a date?

**Complex Sequential decisions:** utility of outcomes are dependent on **a sequence of actions/decisions**:

- Chess.
- Financial planning.
- Path navigation.
- Deployment of mining trucks in an area.

---

# Overview of next 4 weeks

---

## Sequential Complex Decision Making: MDPs

- Environment known, but uncertainty in outcomes of actions/decisions
- Utility of plan dependent on sequence of actions

## Reinforcement Learning

- Environment unknown (transitions, rewards).
- But feedback available!

## Probability and Bayesian Inference

- How to represent beliefs and handle uncertainty?
- Given evidence, how to infer probabilities of outcomes?
- How to simplify probabilistic relationships to make it easier to reason and infer?

---

# Outline of MDP Content

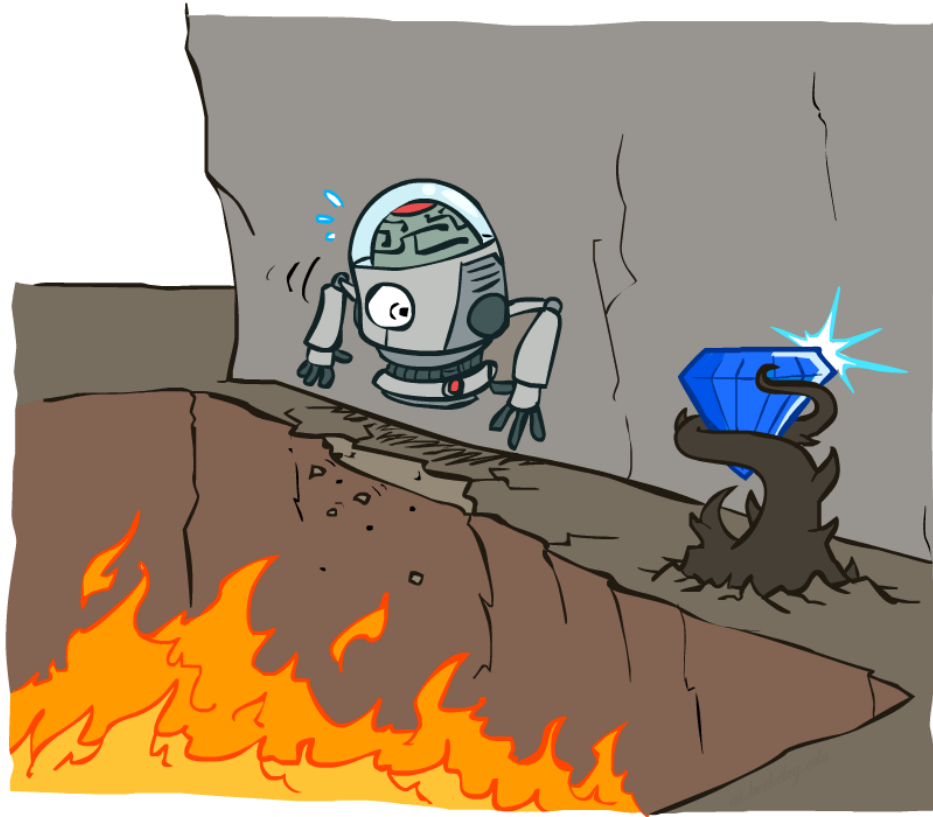
---

- Grid World Running Example.
- Definition of Markov Decision Processes.
- Policies.
- Utilities of Sequences.
- Value Iteration.
- Policy Evaluation & Policy Iteration.

---

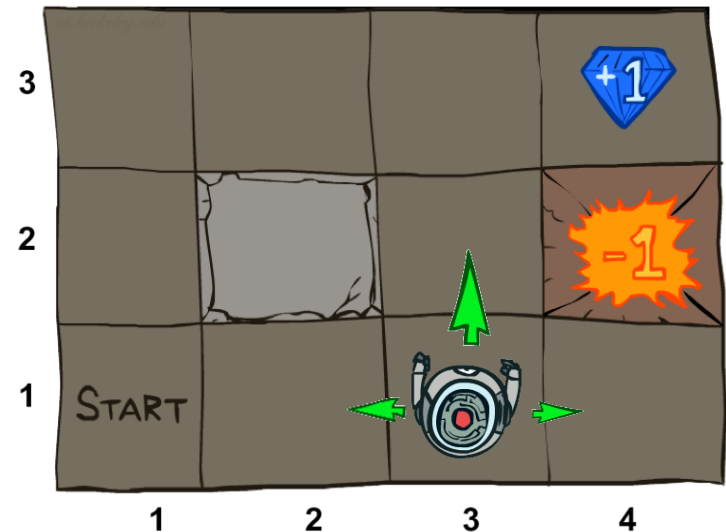
# Non-Deterministic Search

---



# Example: Grid World

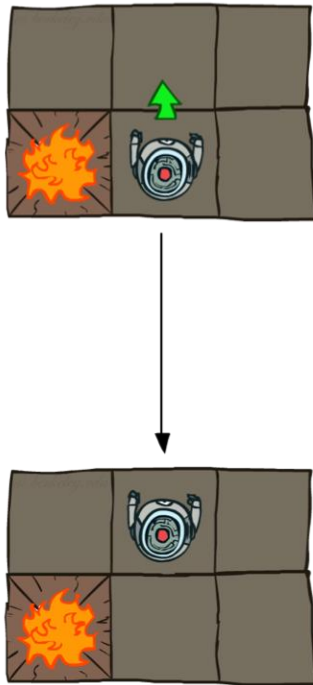
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is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



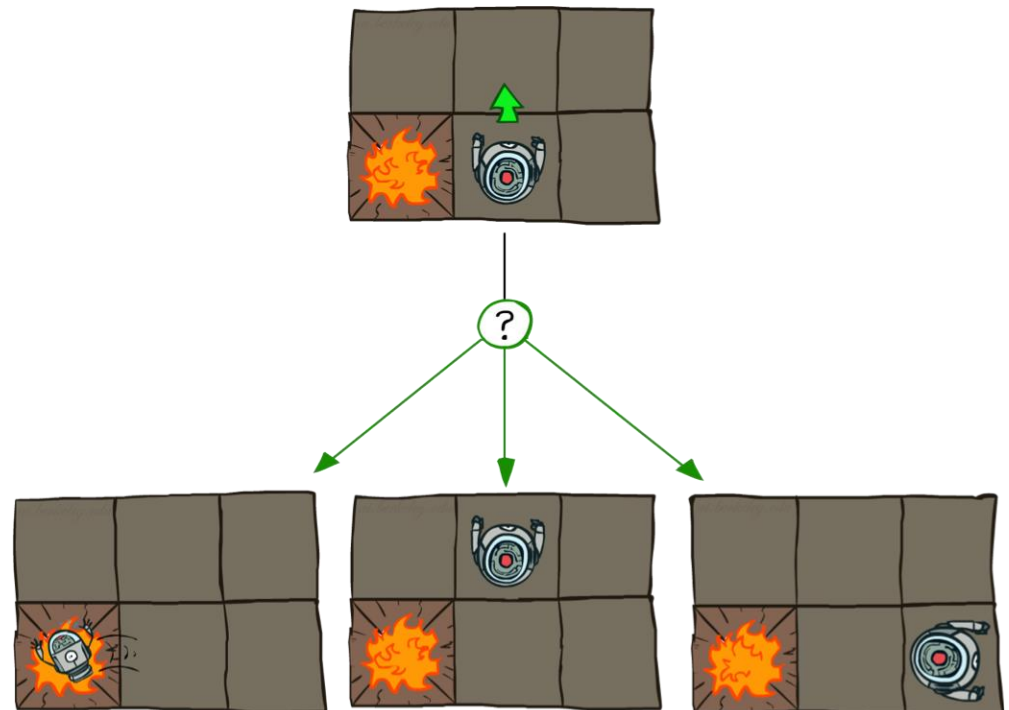


# Grid World Actions

Deterministic Grid World



Stochastic Grid World



---

# Characteristics of Grid World

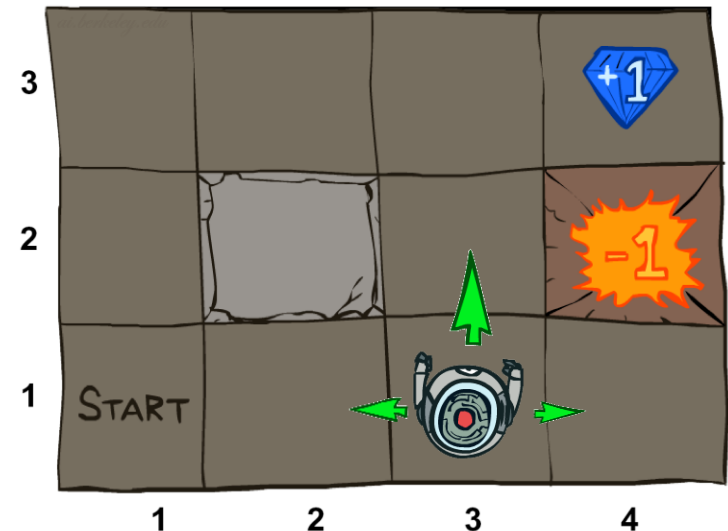
---

- Environment is fully observable:
  - Agent knows where it is at any time
- But actions are uncertain (movement)
- Each grid location is a state that the agent can be in
- Has terminating grid locations/states
- Sequential decision problem:
  - Each grid location/state has a reward
  - Final **utility** depends on a sequence of states agent traversed

**Question:** How to find the best sequence of actions an agent should take in the Grid World?

# Markov Decision Processes

- An MDP is defined by:
  - A **set of states**  $s \in S$
  - A **set of actions**  $a \in A$
  - 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**
- MDPs are non-deterministic search problems



---

# What is Markov about MDPs?

---

- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means action outcomes depend only on the current state

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

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

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



Andrey Markov  
(1856-1922)

---

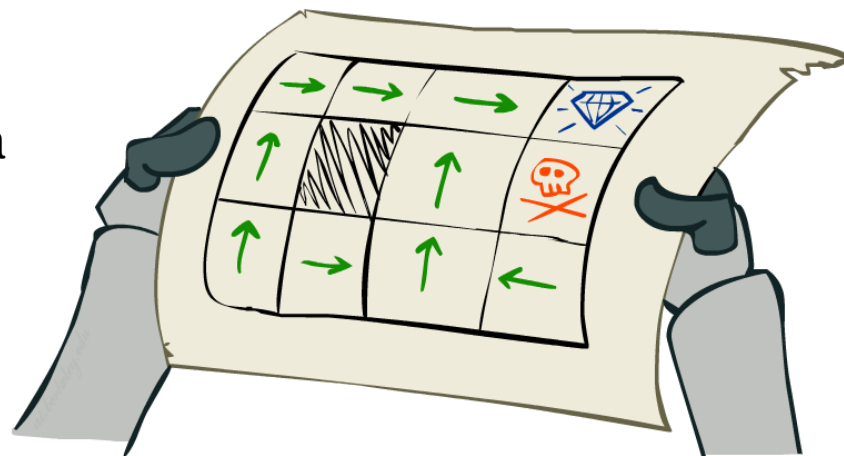
# Outline of MDP Content

---

- GridWorld Running Example
- Definition of Markov Decision Processes
- **Policies**
- Utilities of Sequences
- Value Iteration
- Policy Evaluation & Policy Iteration

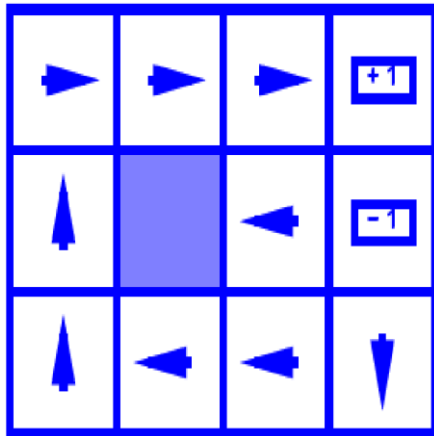
## Policies

- In deterministic single-agent search problems, we wanted an optimal **plan**, or sequence of actions, from start to a goal
- For MDPs, we want an optimal **policy**  
 $\pi^*: S \rightarrow A$ 
  - A policy  $\pi$  gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

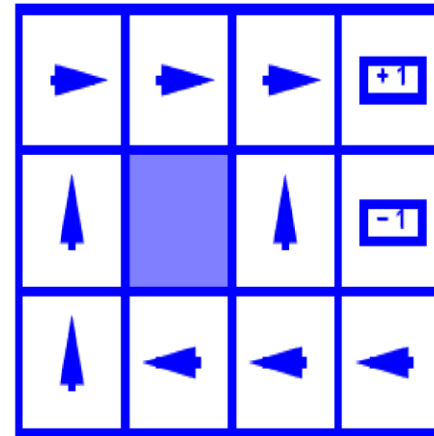


Optimal policy when  $R(s, a, s') = -0.03$   
for all non-terminals  $s$

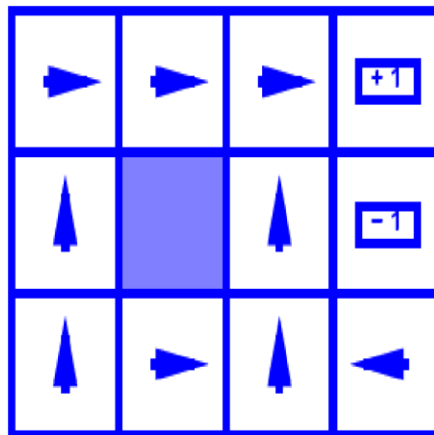
# Value/Utility of a State and a Policy



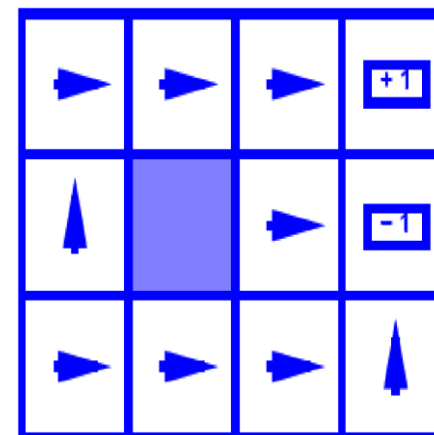
$$U(s) = -0.01$$



$$U(s) = -0.03$$



$$U(s) = -0.4$$

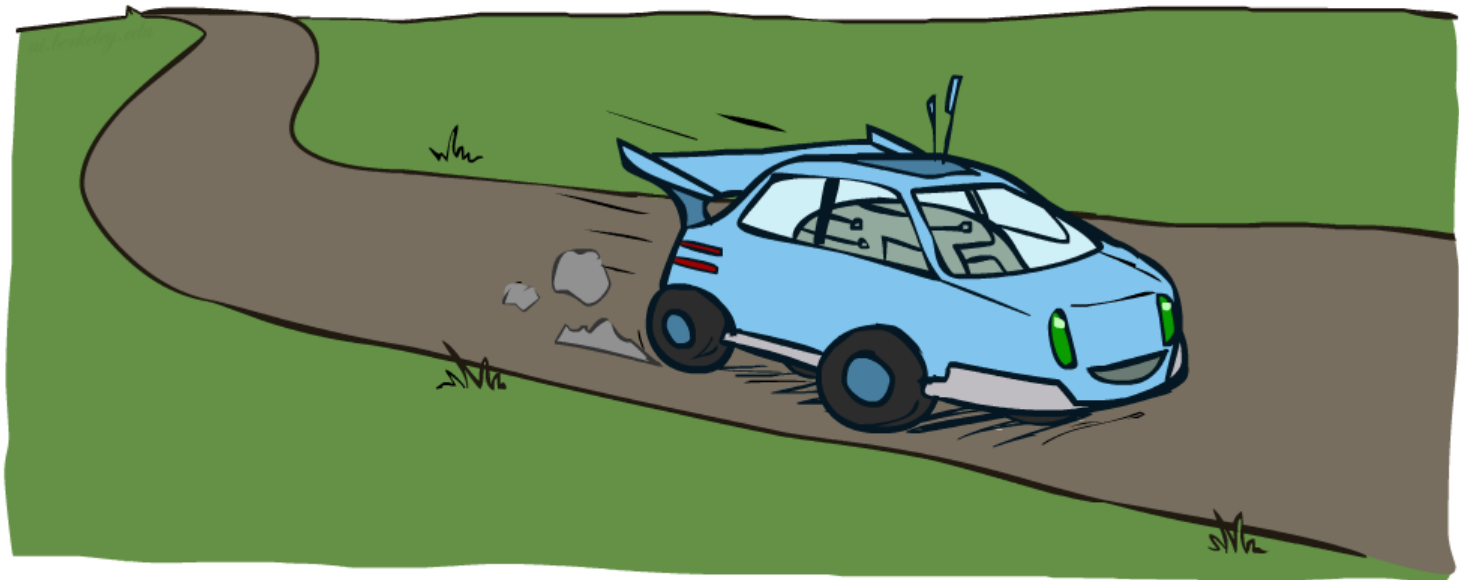


$$U(s) = -2.0$$

---

# Example: Racing

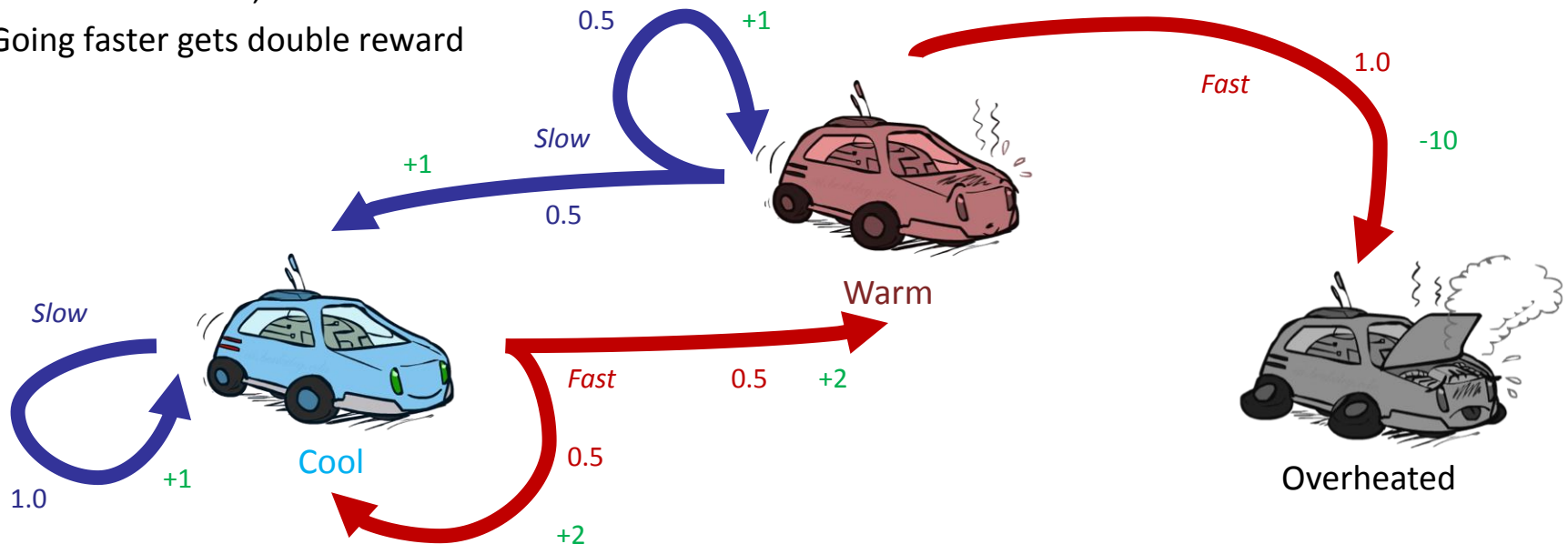
---



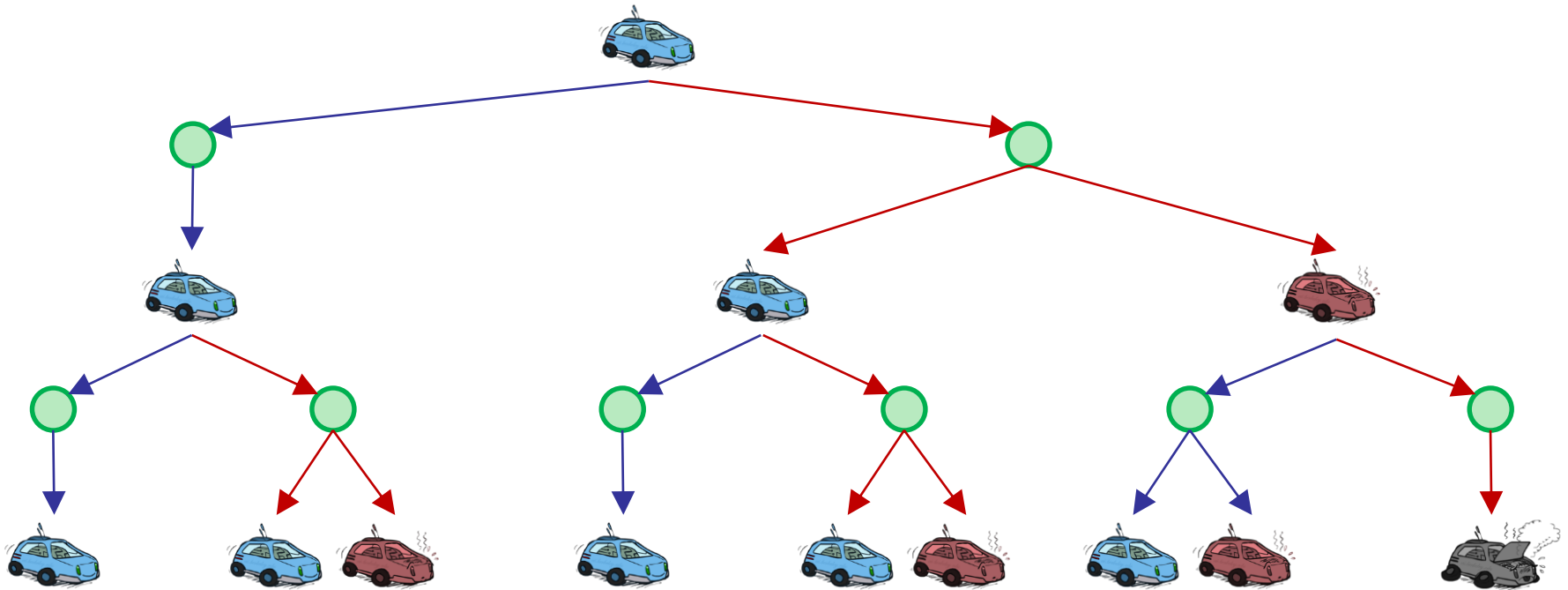


# Example: Racing

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

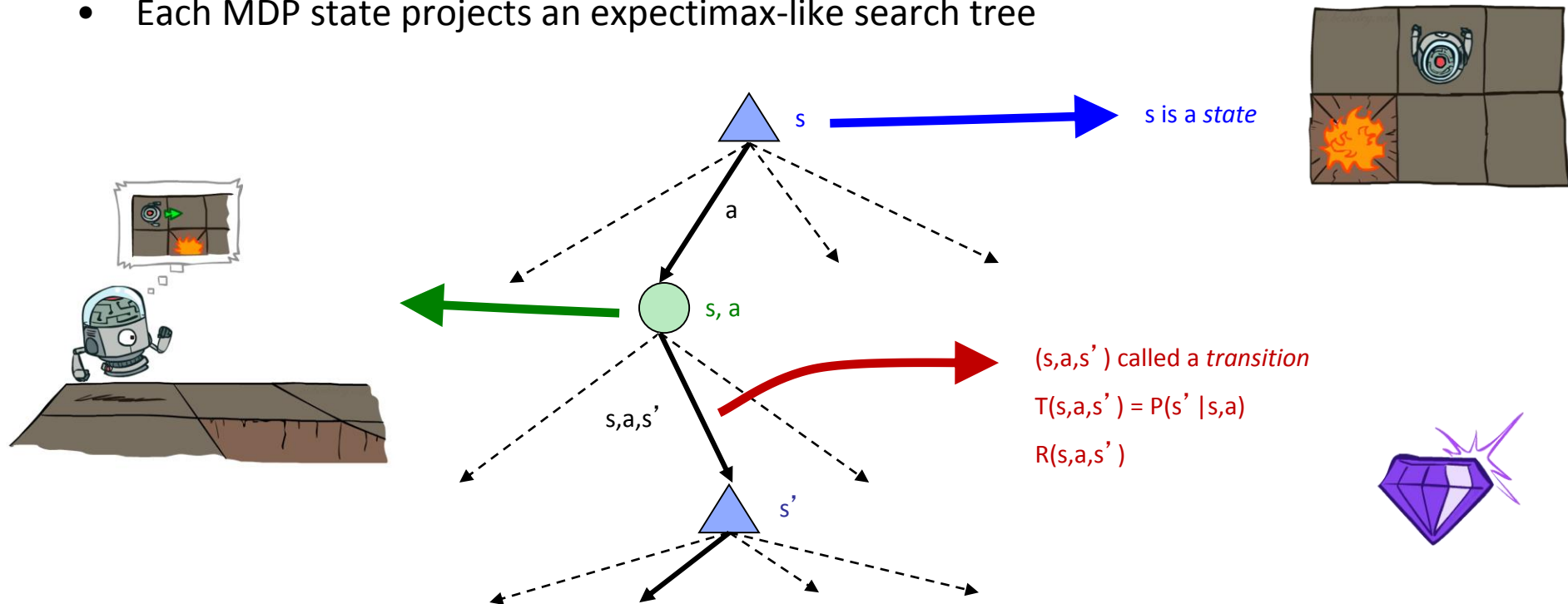


## Racing Search Tree



# MDP Search Trees

- Each MDP state projects an expectimax-like search tree



- We need to calculate **expected utilities**: weighted average (expectation) of values of children. See also examples from: [https://en.wikipedia.org/wiki/Expected\\_value](https://en.wikipedia.org/wiki/Expected_value)

---

# Outline of MDP Content

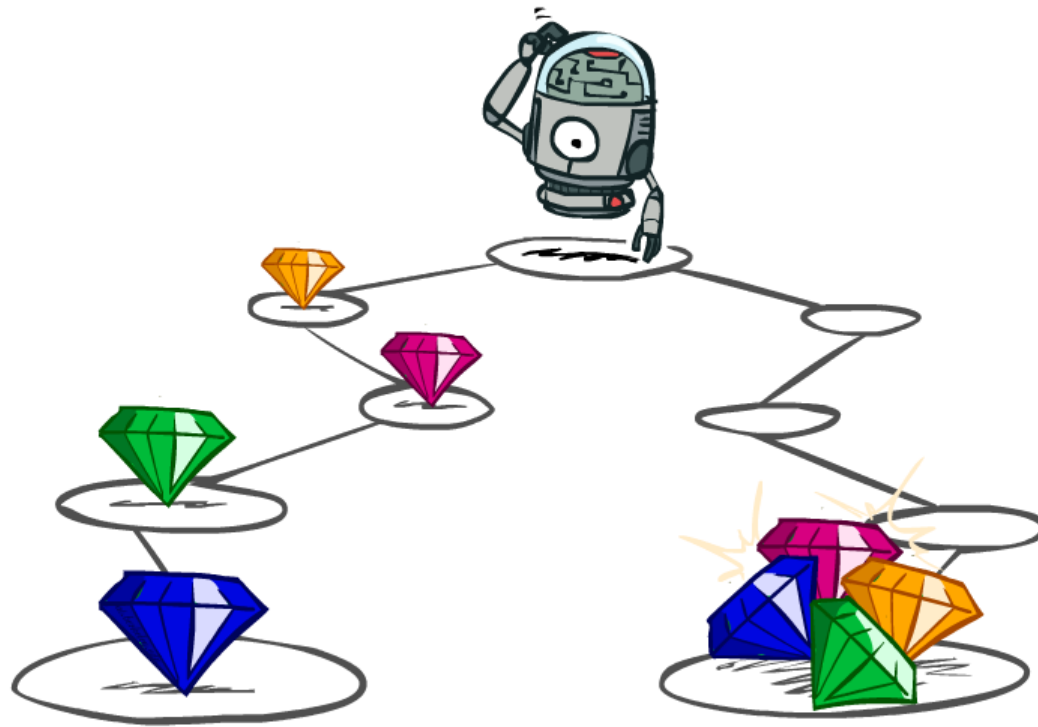
---

- GridWorld Running Example
- Definition of Markov Decision Processes
- Policies
- **Utilities of Sequences**
- Value Iteration
- Policy Evaluation & Policy Iteration

---

# Utilities of Sequences

---

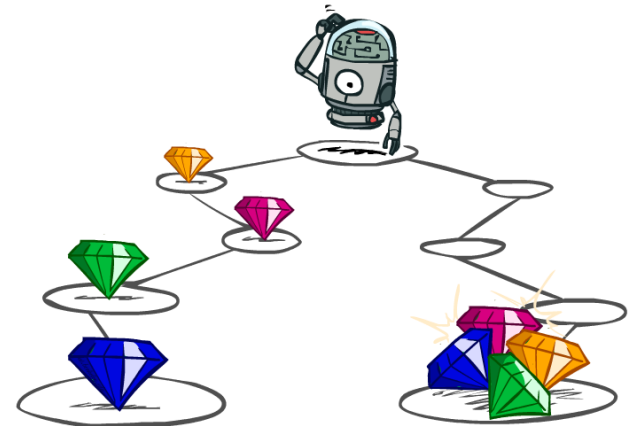


---

# Utilities of Sequences

---

- What preferences should an agent have over reward sequences?
- More or less?  
 $[1, 2, 2]$       or       $[2, 3, 4]$
- 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



$\gamma$

Worth Next Step

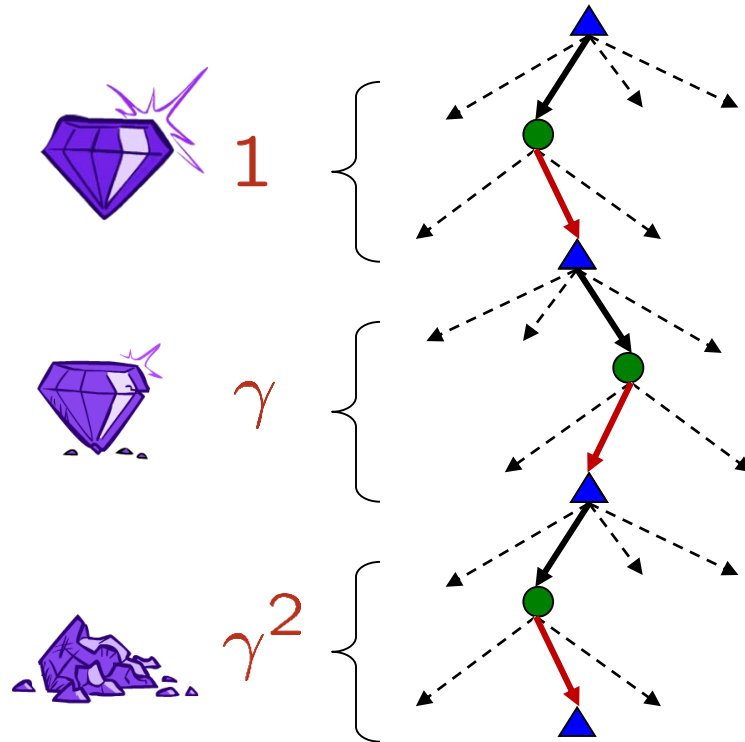


$\gamma^2$

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

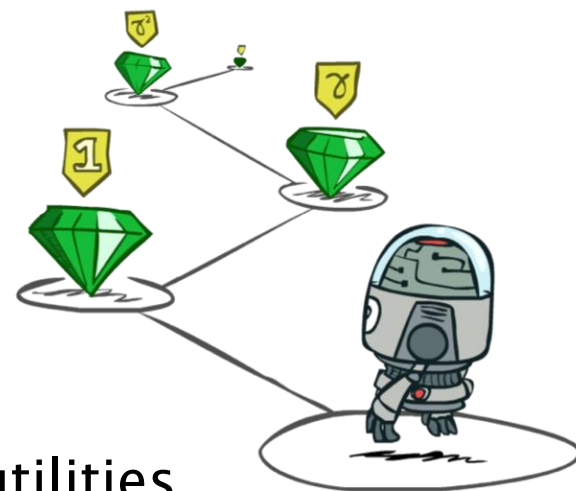




# Stationary Preferences

- Theorem: if we assume **stationary preferences**:

$$\begin{aligned} [a_1, a_2, \dots] &\succ [b_1, b_2, \dots] \\ \Leftrightarrow \\ [r, a_1, a_2, \dots] &\succ [r, b_1, b_2, \dots] \end{aligned}$$



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

---

# Infinite Utilities?!

---

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

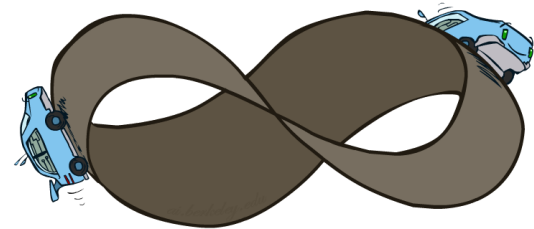
- Solutions:

- Finite horizon: (similar to depth-limited search)
  - Terminate episodes after a fixed T steps (e.g. life)
  - Gives nonstationary policies ( $\pi$  depends on time left)

- Discounting: use  $0 < \gamma < 1$

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

- Smaller  $\gamma$  means smaller “horizon” – shorter term focus
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)



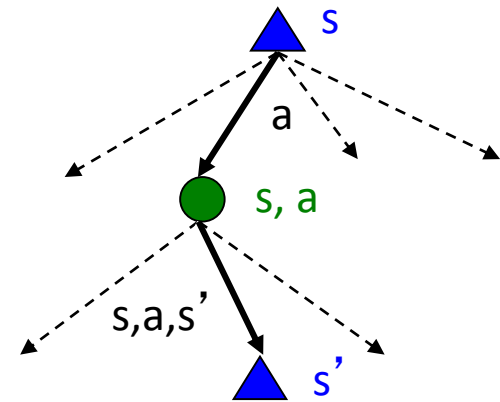
---

# 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  $\gamma$ )



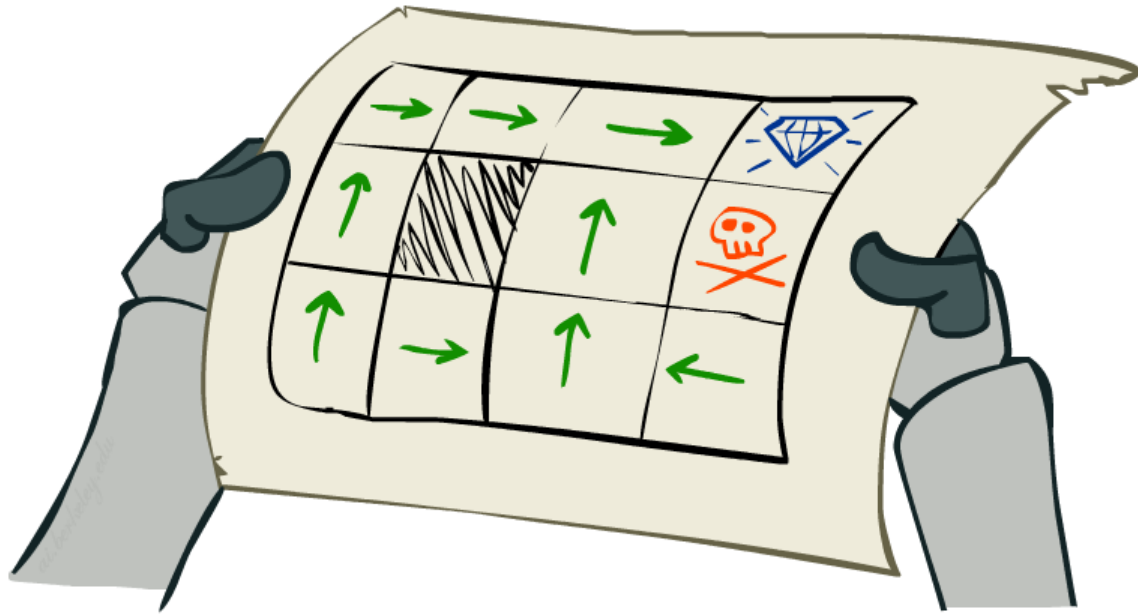
- MDP quantities so far:

- Policy = Choice of action for each state
- Utility = sum of (discounted) rewards

---

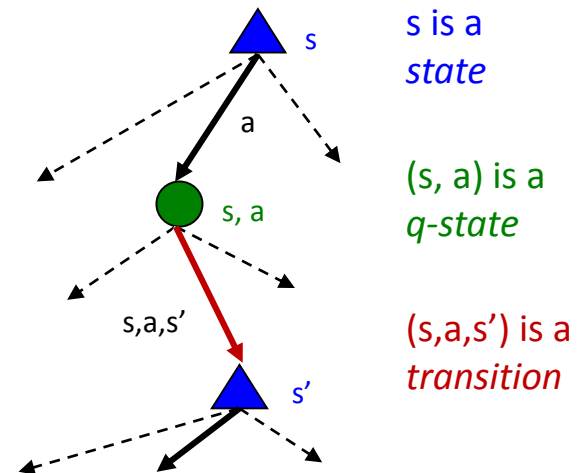
# Solving MDPs

---



# 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  $s$

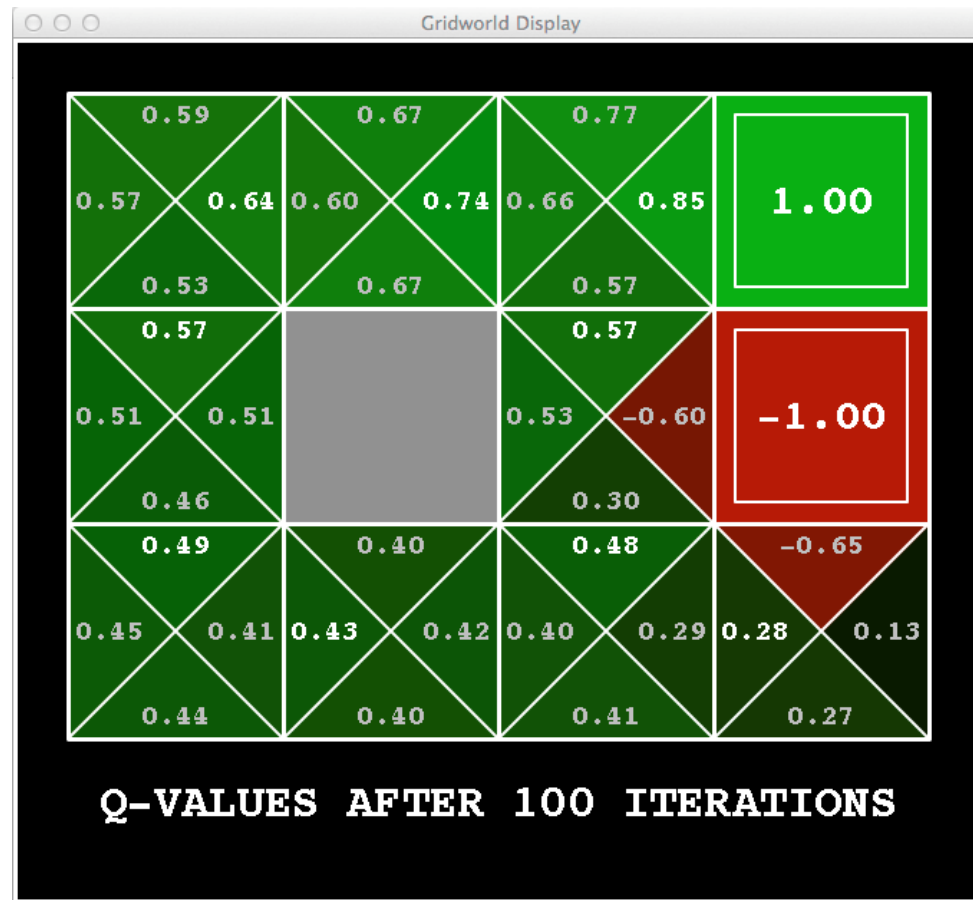


# Snapshot of Demo – Gridworld V Values



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Snapshot of Demo – Gridworld Q Values



Noise = 0.2  
Discount = 0.9  
Living reward = 0

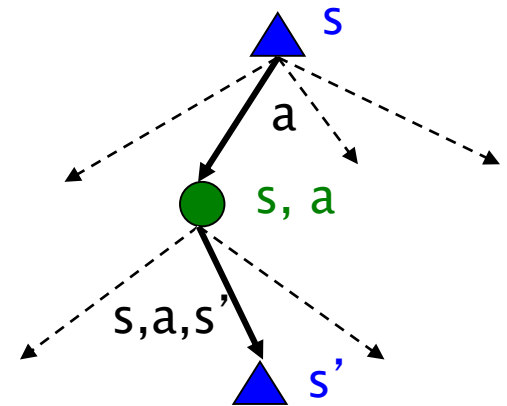
# Values of States

- 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:

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

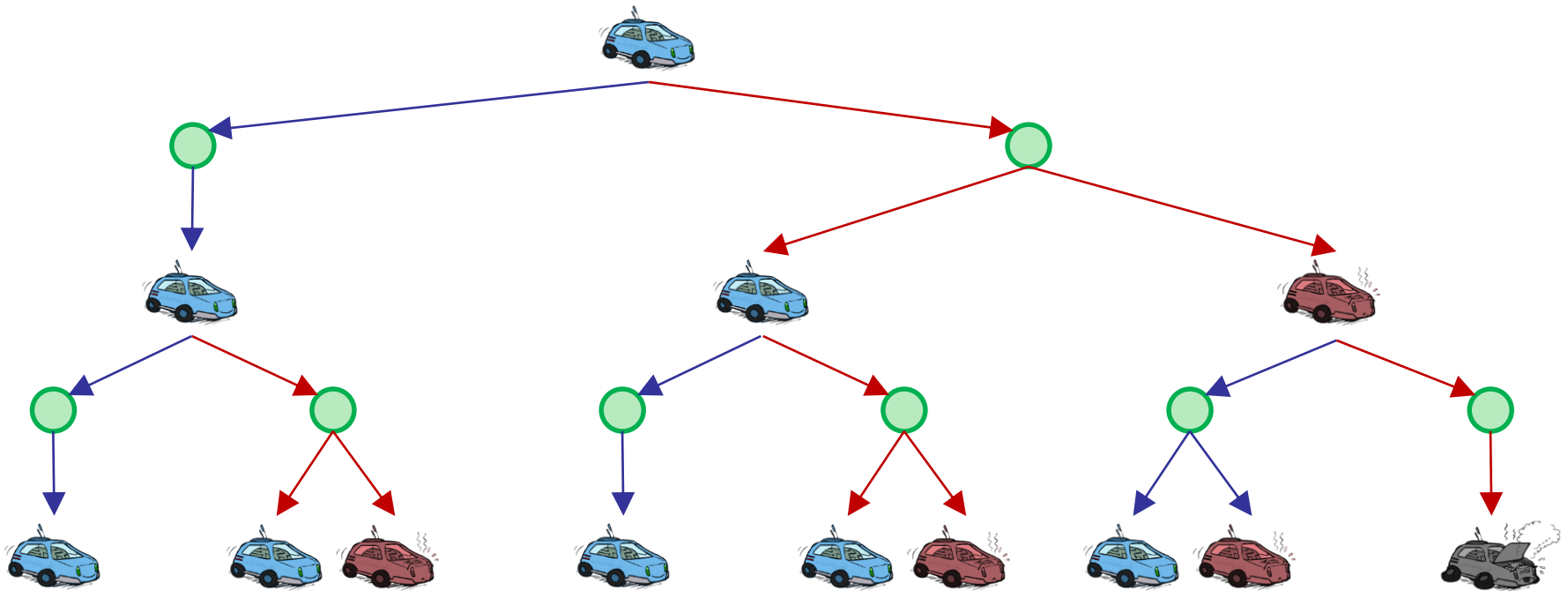
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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





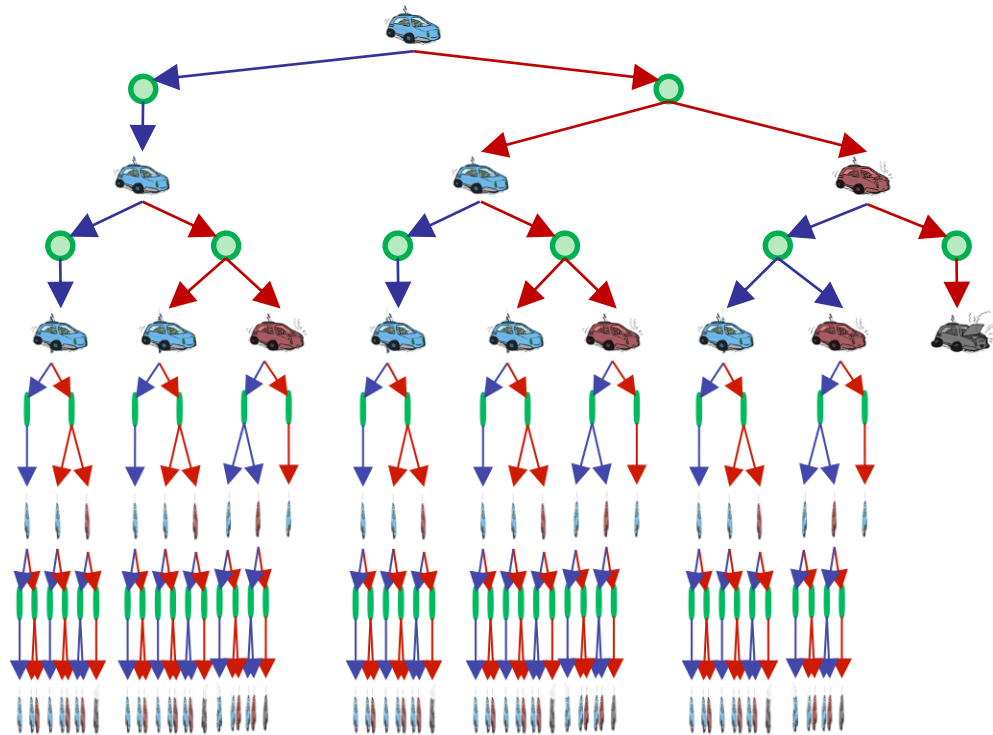
# Racing Search Tree



---

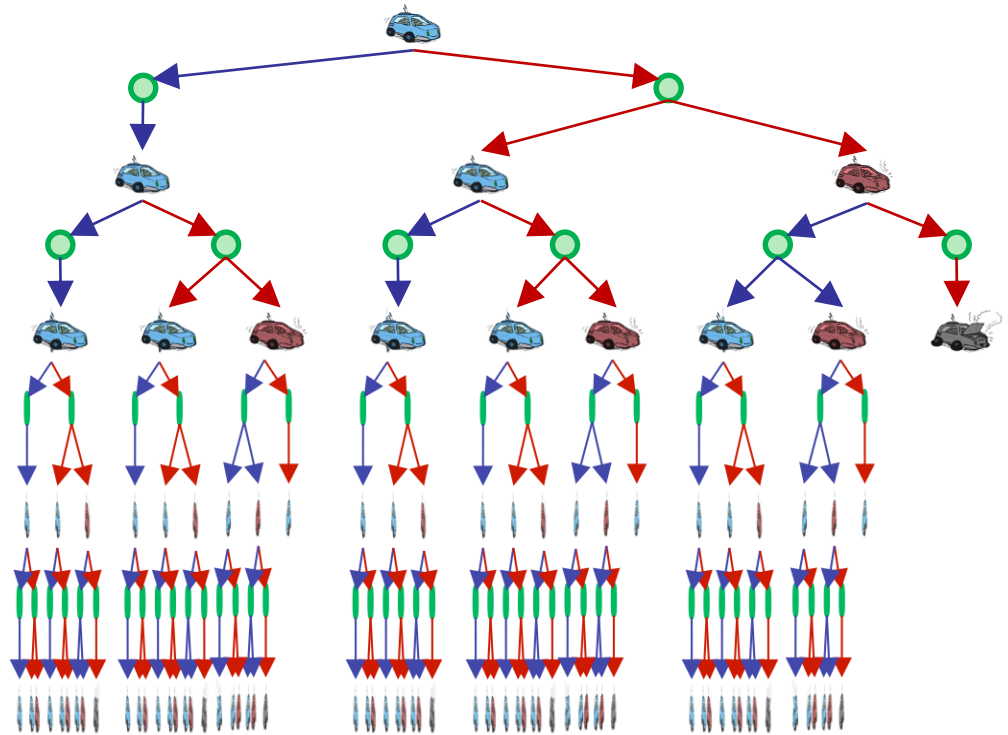
# Racing Search Tree

---



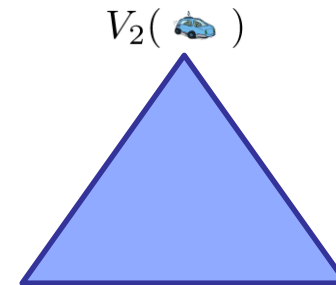
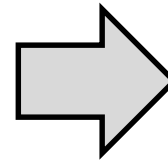
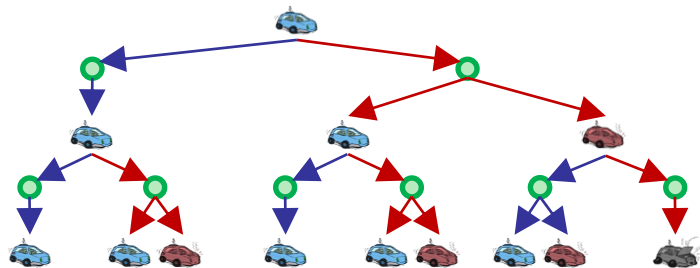
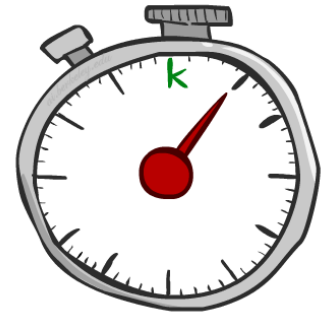
# Racing Search Tree

- 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

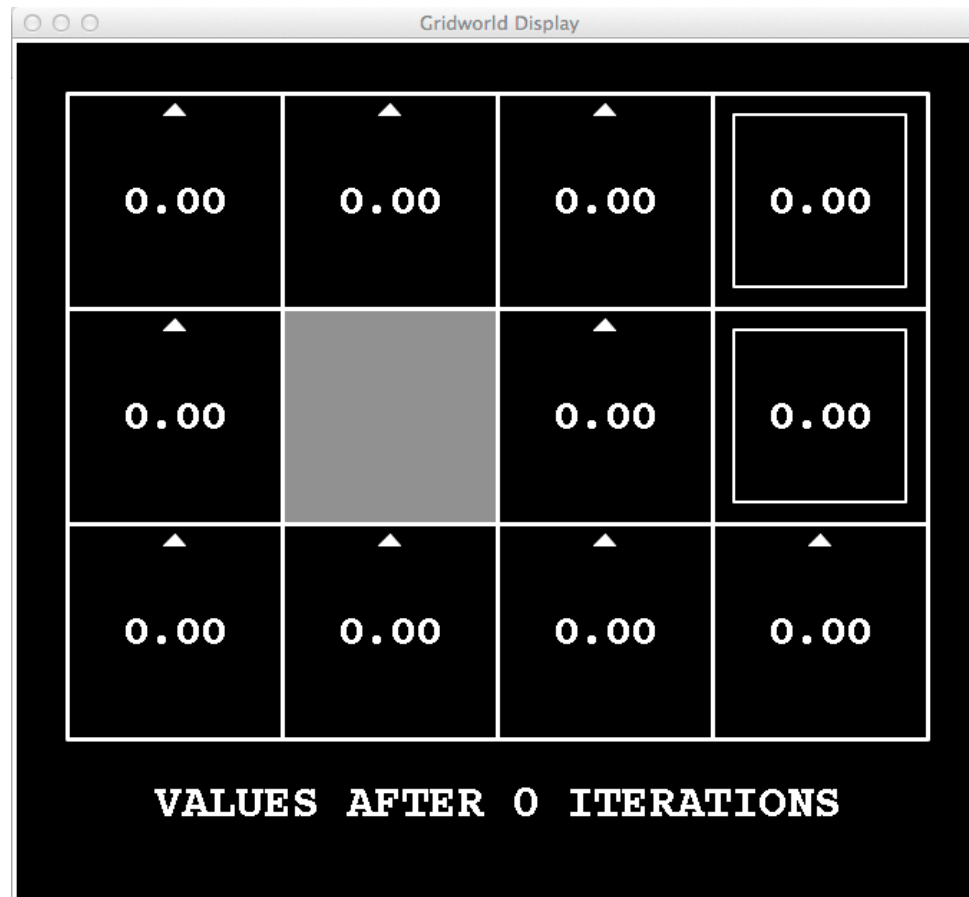
- Key idea: time-limited values
- Define  $V_k(s)$  to be the optimal value of  $s$  if the game ends in  $k$  more time steps
  - Equivalently, it's what a depth- $k$  expectimax would give from  $s$



---

# $k=0$

---

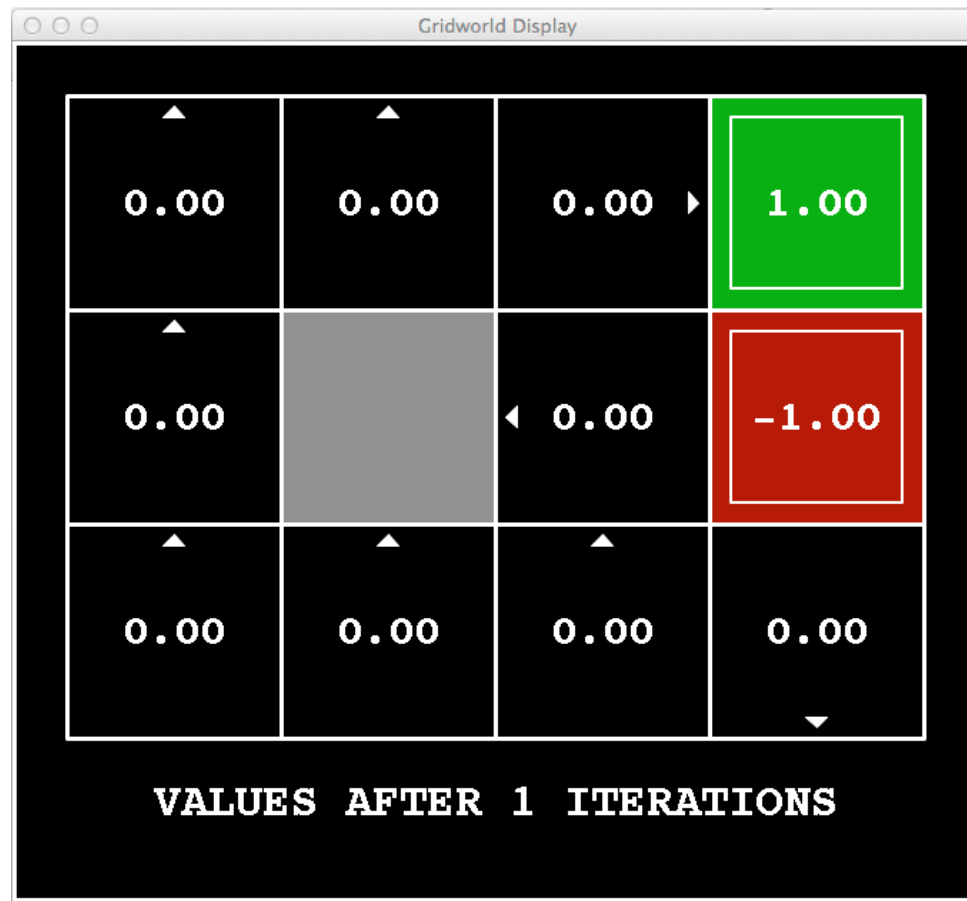


Noise = 0.2  
Discount = 0.9  
Living reward = 0

---

# $k=1$

---



Noise = 0.2  
Discount = 0.9  
Living reward = 0

---

# $k=2$

---

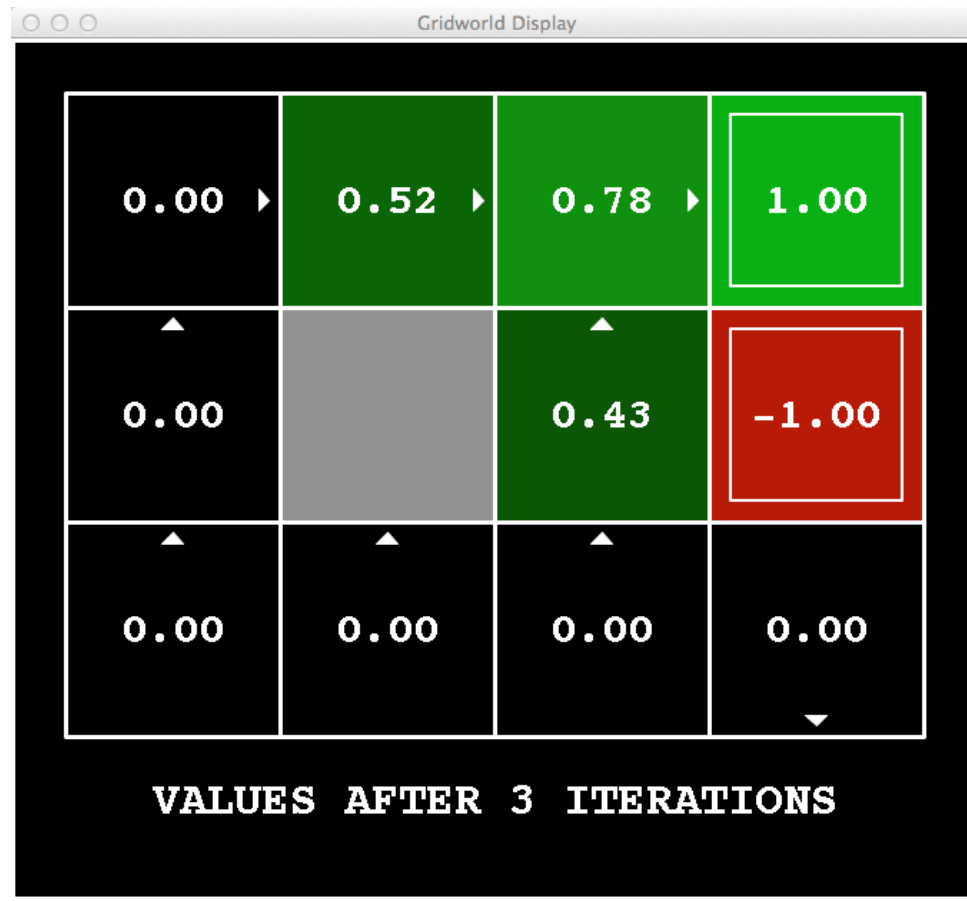


Noise = 0.2  
Discount = 0.9  
Living reward = 0

---

# k=3

---



Noise = 0.2  
Discount = 0.9  
Living reward = 0



---

# k=4

---



Noise = 0.2  
Discount = 0.9  
Living reward = 0

---

# k=5

---

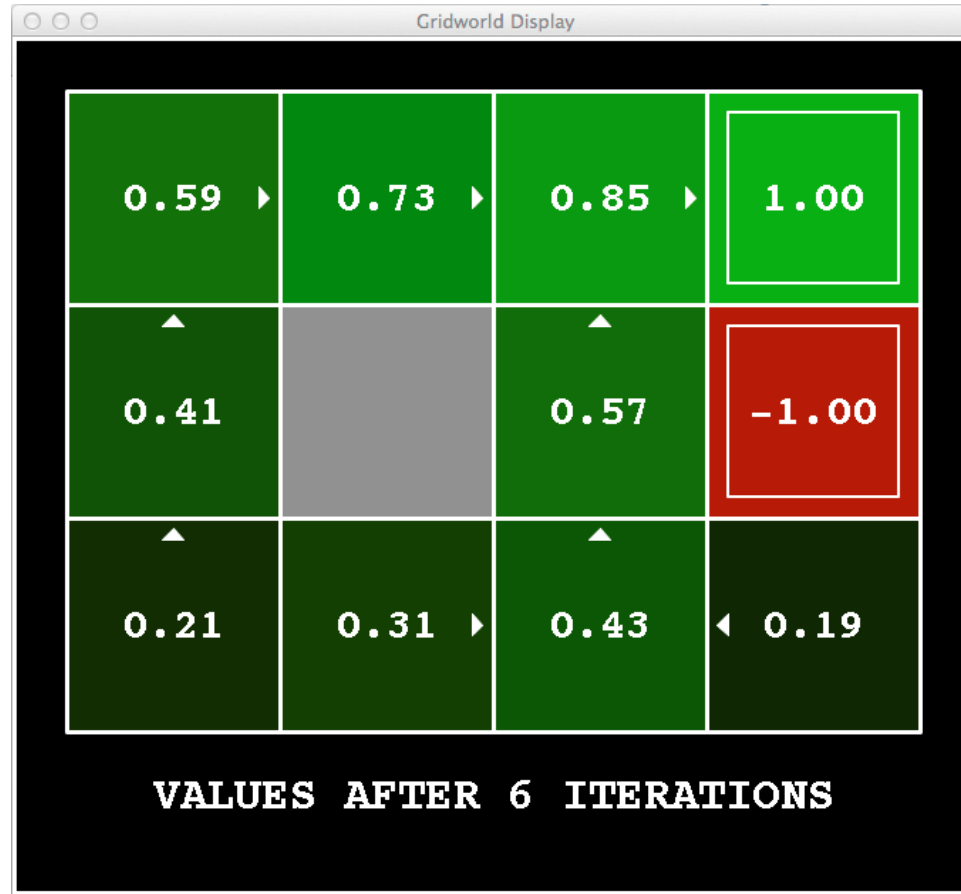


Noise = 0.2  
Discount = 0.9  
Living reward = 0

---

# 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

---

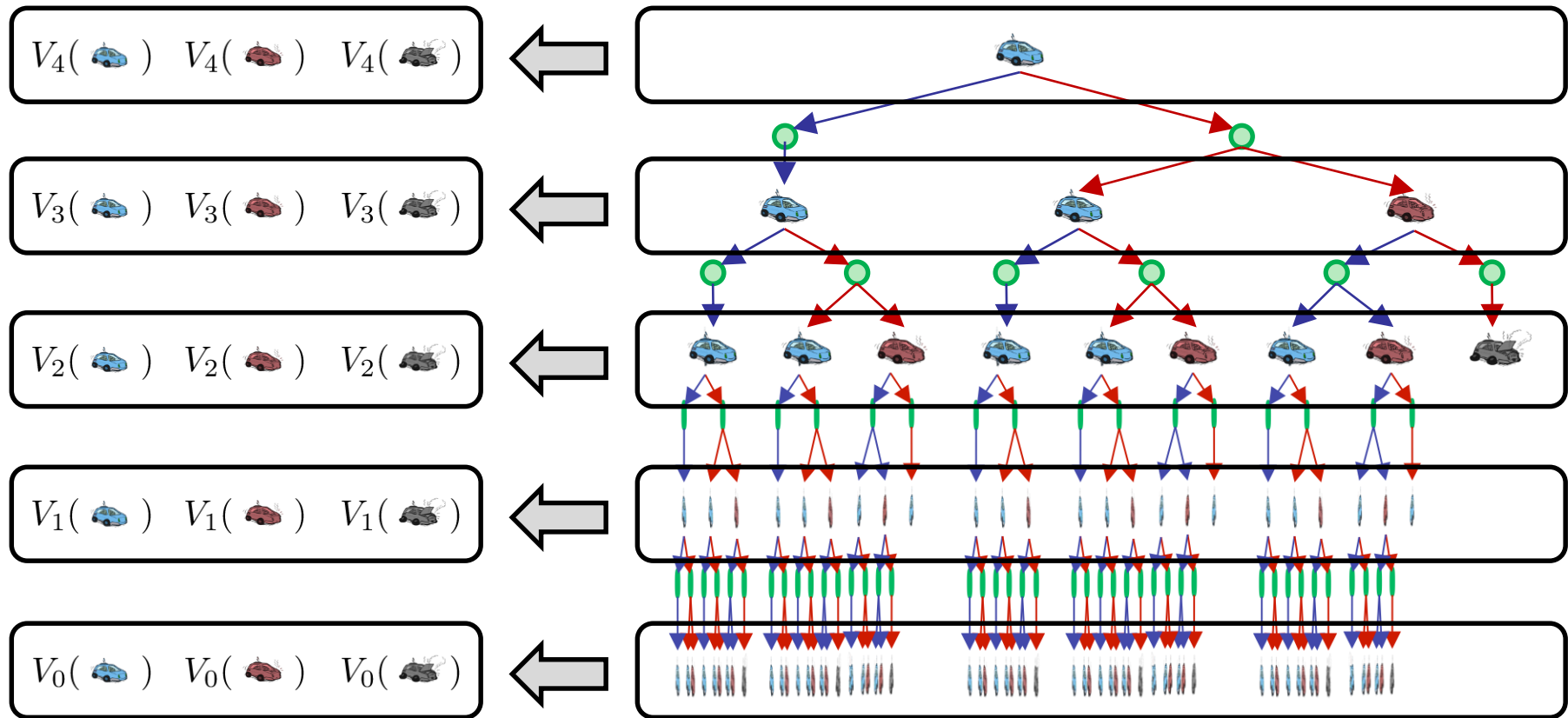
# $k=100$

---



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Computing Time-Limited Values



---

# Outline of MDP Content

---

- GridWorld Running Example
- Definition of Markov Decision Processes
- Policies
- Utilities of Sequences
- **Value Iteration**
- Policy Evaluation & Policy Iteration

---

# Value Iteration

---

Given a MDP: **how to find the optimal policy for it?**

- Optimal policy means for each state, what is the best action to perform?

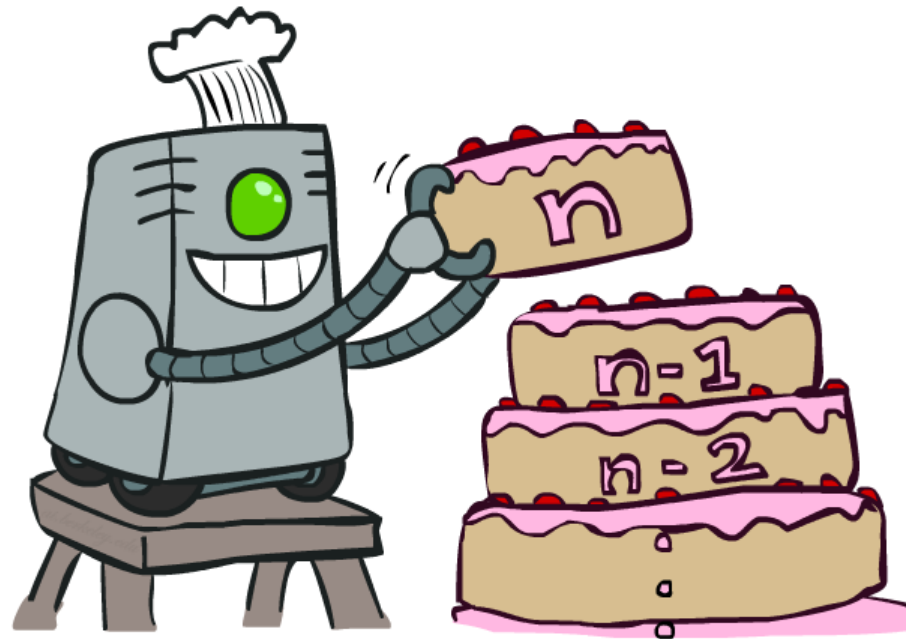
In order to determine what is the best action, need to calculate the utility of each state. Recursive? How to calculate this?

- Value Iteration

---

# Value Iteration

---



---

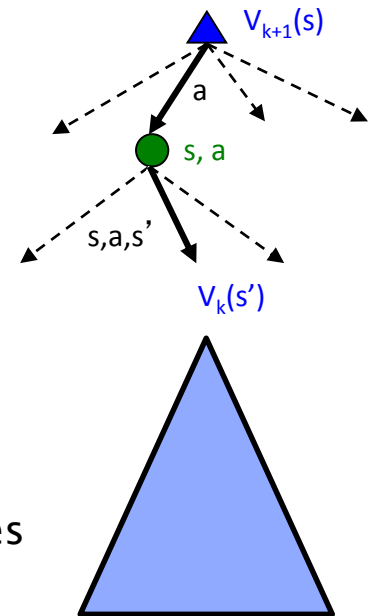
# Value Iteration

---




- 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:

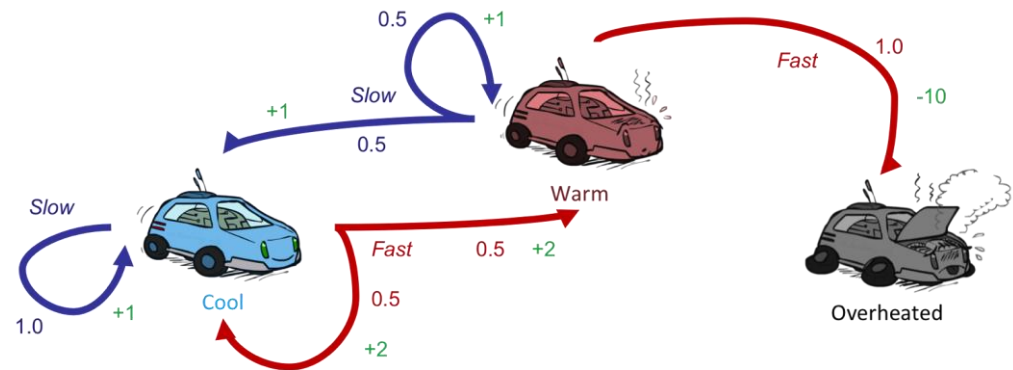
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- 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

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



Assume no discount!

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



---

## Value Iteration – Deterministic Actions

---

Assume **deterministic actions**, i.e, in any state, if we take an action, we always go to target state  $s'$ .

Accordingly, given a policy  $\pi$ , we can calculate utility  $U(s)$  as:

$$U(s) = R(s) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

---

# Value Iteration – Deterministic Actions

---

We can write equation as follows:

$$U(s) = R(s) + \gamma \underbrace{[R(s_1) + \gamma R(s_2) + \dots]}_{\text{Utility from state } s_1 \text{ onwards}}$$

Or

$$U(s) = R(s) + \gamma U(s_1)$$

What is state  $s_1$ ? It is determined by the policy, but this is what we are trying to find too! We want the optimal policy, hence we choose the optimal action for each subsequent state:

$$U(s) = R(s) + \gamma \max_a T(s, a, s_1) U(s_1)$$

---

# Utilities of States

---

Now let's go back to MDPs, where **actions are uncertain**, i.e., we perform an action in a state, the outcomes are not certain: there is a probability:  $P(s' | s, a)$

We follow similar logic to get to Value Iteration for MDPs.

For MDPs,  $U(s)$  = **expected** utility of the states that follow state  $s$ :

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s \right]$$

---

## Gridworld Utilities

---

3	0.812	0.868	0.918	<b>+ 1</b>
2	0.762		0.660	<b>-1</b>
1	0.705	0.655	0.611	0.388
	1	2	3	4

Utilities of state for  $\gamma=1$  and  $R(s) = 0.04$  for non-terminal states.

---

# Maximum Expected Utility Principle

---

The **Maximum Expected Utility (MEU) Principle** says to find the optimal action, we chose the action that **maximises** the **expected** utility of the **subsequent** states:

$$\pi^*(s) = \operatorname{argmax}_a \underbrace{\sum_{s'} T(s, a, s') U(s')}_{\text{expected utility over } s'}$$

---

# Bellman Equations

---

**Idea:** At any state, the utility is the reward + expected discounted utility of next possible states.

Isn't the next possible states dependent on my policy?

Recall we are trying to find the **optimal** policy. MEU says to pick the action that maximises expected discounted utility of next possible states.

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

This is called the **Bellman equation** (recursive!)

**Note** that the above  $R(s)$  is slightly different from  $R(s, a, s')$  in previous slides, but this does not change the problem in any fundamental way. See p.615 in R & N (2<sup>nd</sup> edition).

---

# Value Iteration

---

We use the Bellman equation to calculate the utility of states. For  $n$  states, we have  $n$  Bellman equations.

We cannot solve as simultaneous equations due to the max operator, but can solve iteratively.

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

Guaranteed to converge to an equilibrium and the utilities are unique also.

---

# Value Iteration

---

Function `valueIteration(MDP,  $\epsilon$ ,  $\gamma$ )`

$U_{new}$  (assign to all 0's initially)

Repeat

$U_{cur} = U_{new}; \delta = 0$

For each state  $s$  in  $S$  do:

$$U_{new}[s] = R[s] + \gamma \max_a \sum_{s'} T(s, a, s') U_{cur}[s']$$

If  $|U_{new} - U_{cur}| > \delta$  then  $\delta = |U_{new} - U_{cur}|$

Until termination (  $\delta < \epsilon(1 - \gamma)/\gamma$  )

Return  $U_{cur}$



---

## Extracting Optimal Policy

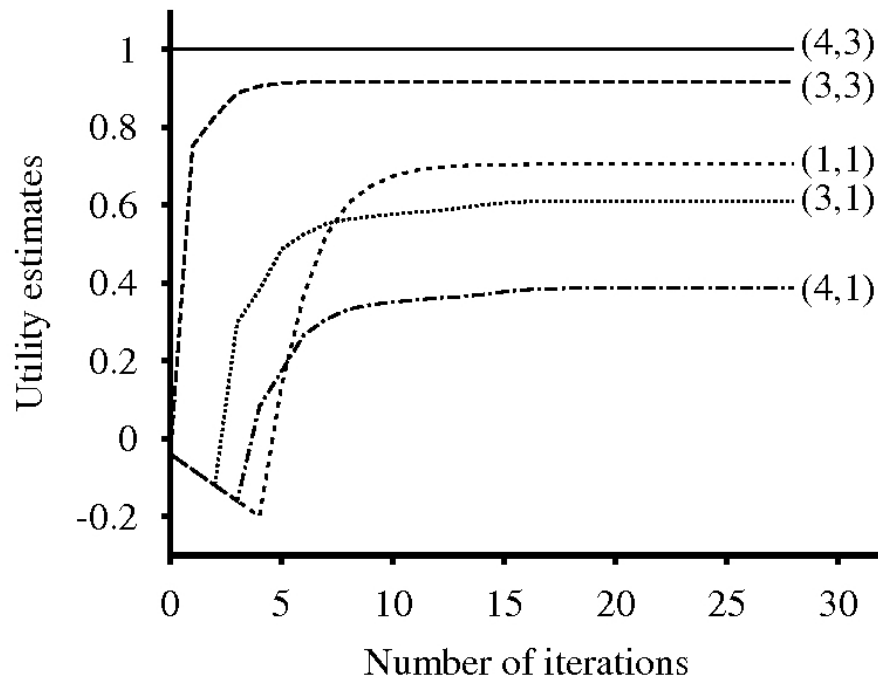
---

After value iteration, we know the utility of each state for an optimal policy.

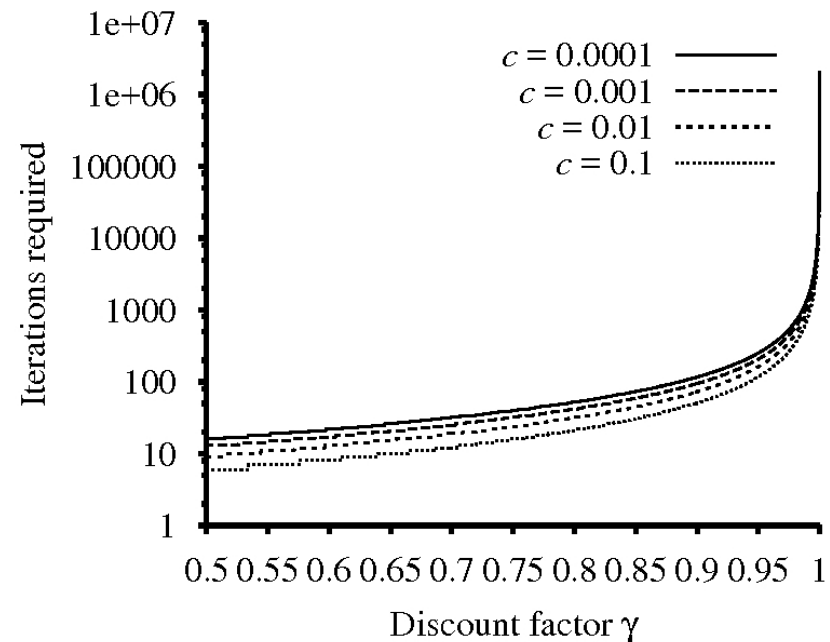
To extract the optimal action for each state (hence the policy), we use MEU:

$$\pi^*(s) = \operatorname{argmax}_a \underbrace{\sum_{s'} T(s, a, s') U(s')}_{\text{expected utility over } s'}$$

# Gridworld using Value Iteration



States and their evolution of utilities.



Number of iterations required to guarantee an error of  $c \cdot R_{\max}$ , as the discount factor is varied.

---

# Outline of MDP Content

---

- GridWorld Running Example
- Definition of Markov Decision Processes
- Policies
- Utilities of Sequences
- Value Iteration
- **Policy Evaluation & Policy Iteration**

---

# Policy Evaluation

---

In the case where the policy is **fixed** (e.g, always move right if possible), policy evaluation is to calculation the utility of each state for the fixed policy.

Given policy  $\pi$ :

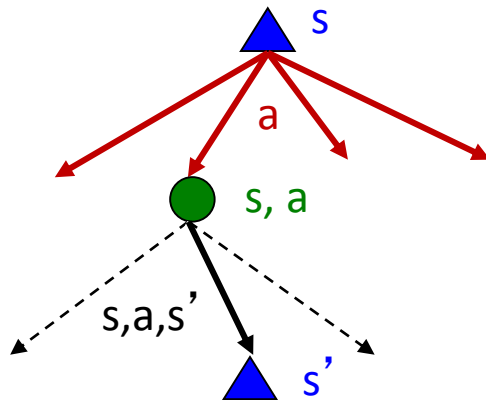
$$U_{k+1}^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') U_k^{\pi}(s')$$

---

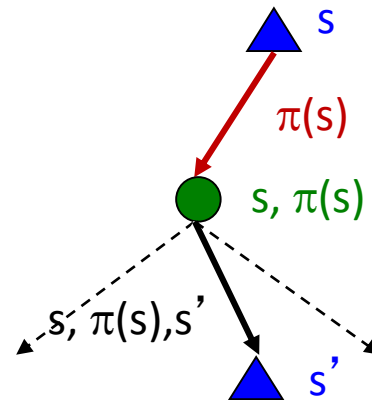
# Fixed Policies

---

Do the optimal action



Do what  $\pi$  says to do



- 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

---

# Policy Iteration

---

- Value Iteration can be slow to converge and finding best actions can be costly.
- 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  $\pi$ , find values with policy evaluation:
  - Iterate until values converge:

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

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

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

---

# Value Iteration vs Policy Iteration

---

- Both value iteration and policy iteration compute the optimal values
- **Value iteration:**
  - Every iteration updates utility values and policies implicitly
  - We don't track the policy but taking the max over all actions implicitly computes it
- **Policy iteration:**
  - Utilities are updated over a fixed policy (fast)
  - After utilities calculated, find optimal policy (similar to value iteration)

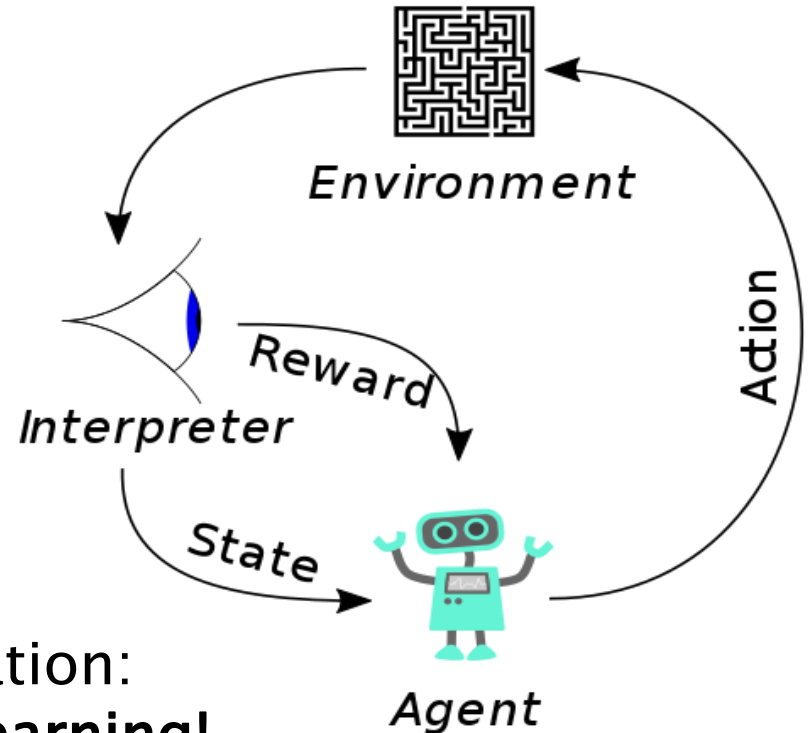


---

# Summary

---

- Sequential decision problems.
- Markov decision processes:
  - Definition
  - Policy
  - Utilities of Sequences
- Value Iteration:
  - Utilities of States
  - Bellman equations
  - Iterative algorithm
- Policy Evaluation & Policy Iteration:
- Next lecture: Reinforcement learning!
- Please also read Dan Klien and Pieter Abbeel's slides and youtube, as the slides here are mostly based on theirs:  
[https://www.youtube.com/watch?v=Oxqwwnm\\_x0s](https://www.youtube.com/watch?v=Oxqwwnm_x0s)



**Acknowledgement:** the slides were developed based those from Dan Klien and Pieter Abbeel (ai.berkeley.edu), and Sebastian Sardina and Jeffrey Chan (RMIT University).