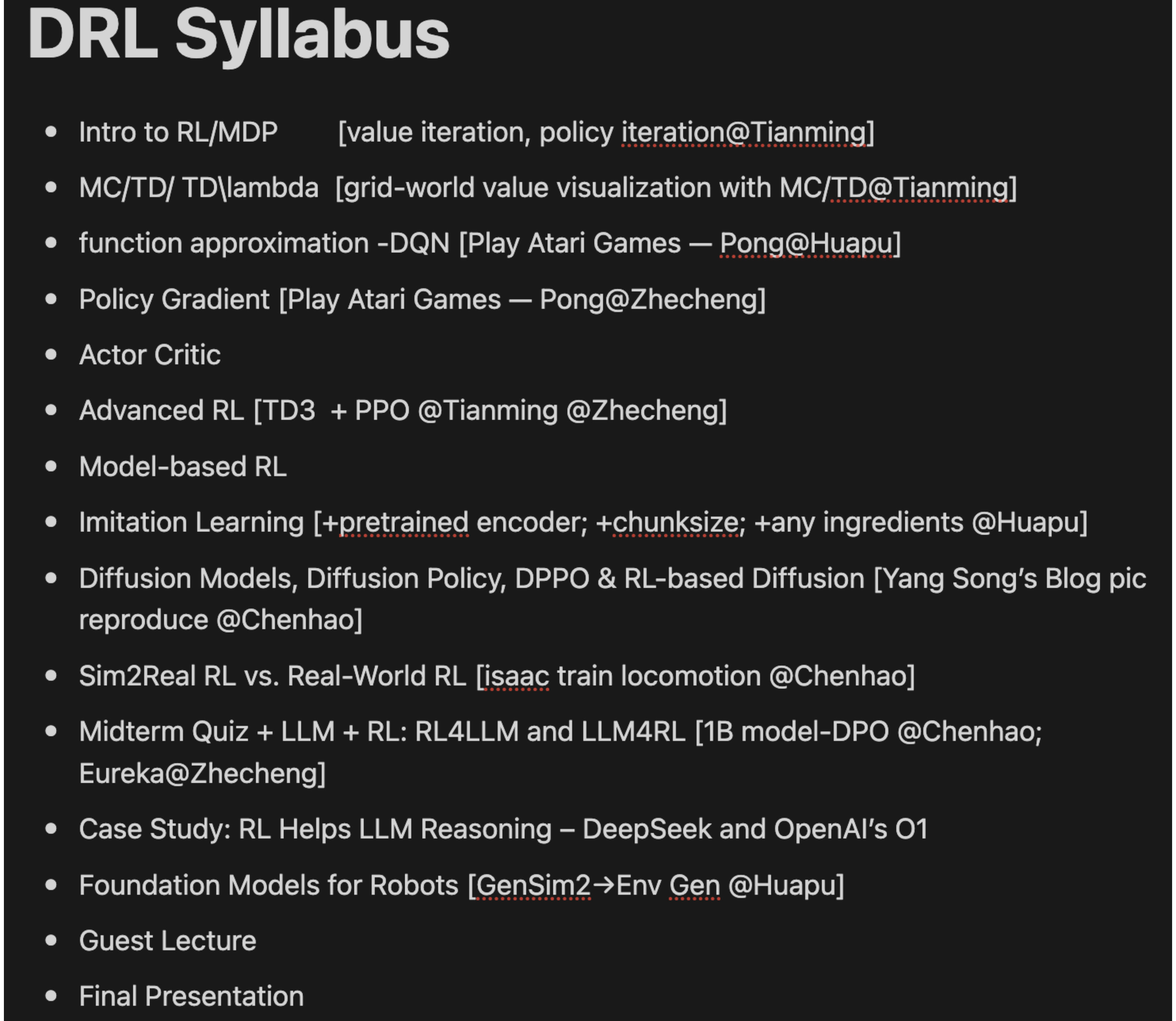
Deep Reinforcement Learning **(Ultra!)**

Lecture 1: From MDP to RL

Huazhe Xu
Tsinghua University



DRL Syllabus

- Intro to RL/MDP [value iteration, policy iteration@Tianming]
- MC/TD/ TD\lambda [grid-world value visualization with MC/TD@Tianming]
- function approximation -DQN [Play Atari Games — Pong@Huapu]
- Policy Gradient [Play Atari Games — Pong@Zhecheng]
- Actor Critic
- Advanced RL [TD3 + PPO @Tianming @Zhecheng]
- Model-based RL
- Imitation Learning [+pretrained encoder; +chunksize; +any ingredients @Huapu]
- Diffusion Models, Diffusion Policy, DPPO & RL-based Diffusion [Yang Song's Blog pic reproduce @Chenhao]
- Sim2Real RL vs. Real-World RL [isaac train locomotion @Chenhao]
- Midterm Quiz + LLM + RL: RL4LLM and LLM4RL [1B model-DPO @Chenhao; Eureka@Zhecheng]
- Case Study: RL Helps LLM Reasoning – DeepSeek and OpenAI's O1
- Foundation Models for Robots [GenSim2→Env Gen @Huapu]
- Guest Lecture
- Final Presentation

Logistics

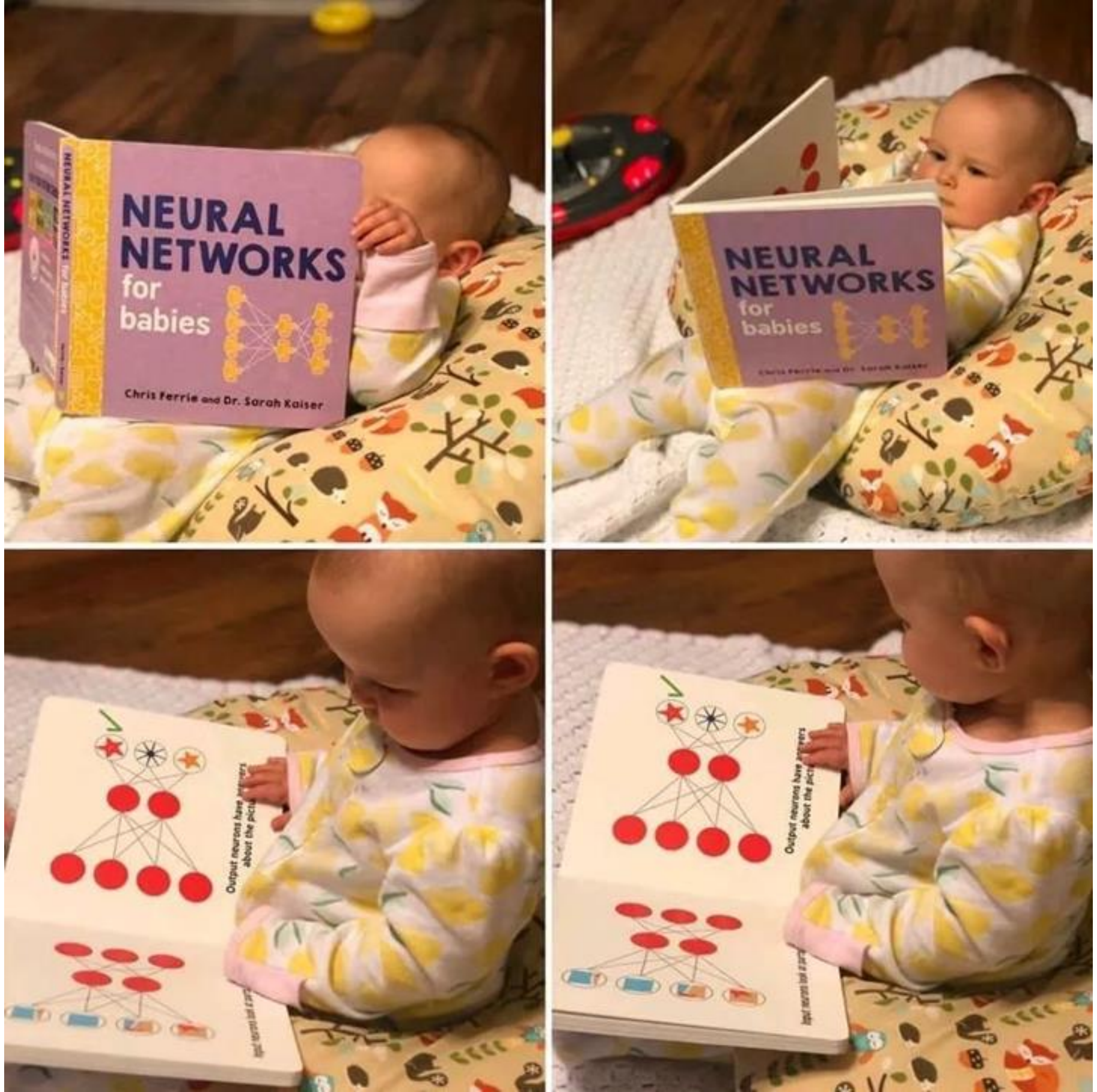
Recitation:

- Time: After the lecture. Recitation materials **will not** appear in exams.
- location: Here! Might change, stay tuned!
- Extended materials of lectures

- Office Hour:
  - Time: Upon Request
  - Location: 清华科技园启迪大厦c座19层/东升大厦A座601A
  - （需邮件/微信联系我开门）

Logistics

- *An Introduction to Reinforcement learning, Barto and Sutton 1998. (Optional) + Past Notes (no correctness guarantee!)*
- You’d better know deep learning. If not, you can search for CS231N from Stanford.
- If you don’t know neural networks yet. Drop it! Or work as hard as the dude in the next slide.
- This course has a bit workload. If you want to spend 1 week per semester on a course. Drop it!
- If you are too busy with your research projects. Drop it!


NEURAL NETWORKS for babies
Chris Ferrie and Dr. Sarah Kaiser
Output neurons have answers about the picture

Logistics

- 4 coding homeworks
  - each of 10%
  - Will always be due by 11:59:00 PM. One second later than that means you are late for a day.
- Late midterm quiz (~ week 10)
  - 10 %, mostly theoretical

Logistics

- Project
  - 40 % --- proposal 5%, oral presentation/poster 17 %, report 18 %
  - A group of 2-5 students. No solo project allowed. Contribution should be listed honestly. **More people = higher bar**

  Phase 1:

  1-page proposal

  Phase 2:

  Milestone materials for oral selection: 2-page report/demo video/live demo/website/slides

  Phase 3 [**oral presentation will have a bonus credit for final rating**]:

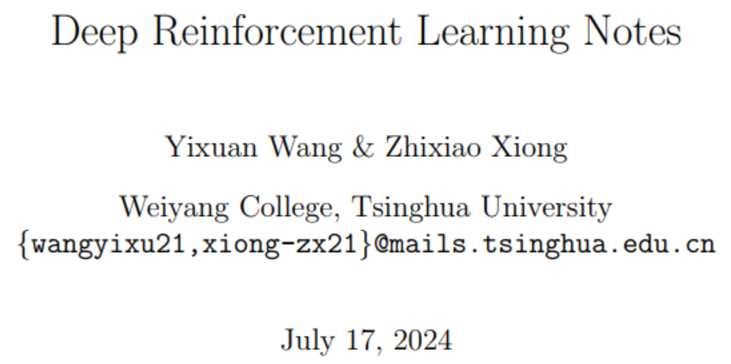
  presentation (oral/poster)

  Phase 4:

  6-page final report for rating

  - More details will be released around midterm.
  - Paper-level or workshop paper-level.

Logistics


Deep Reinforcement Learning Notes

Yixuan Wang & Zhixiao Xiong

Weiyang College, Tsinghua University
{wangyixu21,xiong-zx21}@mails.tsinghua.edu.cn

July 17, 2024

- Course Note (10%)
  - Starting from next lecture. You may refer to the lecture and past notes.
  - Our TAs will assign groups randomly.
  - Each student needs to take notes for at least once.
  - PDF + **LaTex** are required.
  - Scores:
    - Submit with enough info (6 pts)
    - Format, minimal typos/grammatical mistakes, clarity, logical narration (3 pts)
    - Nice or funny figures, your own insights, questions, or anything that makes it better (1 pts)
  - I hope we can release the notes online. If the quality is high enough, we can publish a DRL handbook with all the students as the authors.

Logistics

- Late homework:
  - Your homework will *NOT* be accepted if you are late.
  - But everyone has 3 free late days in total for this whole semester. TAs will track this.

Logistics

- Honor Code
  - You agree not to perform any action that may cause plagiarism (including using chatGPT or deepseek)
  - You will have zero point for that homework/project/quiz
  - Zero tolerance. I promise both of us will be very very unhappy.

- But at least top 50% of this course will have A- or above.

Some of my bad habits

- inspiration > knowledge
- intuition > rigorous math
- Cold call lover/There is no stupid questions/There is no bad answer even if it is wrong.
- English course + Chinese for better explanation

Why this course is a must for your study?

- AGI is coming!
- If you don’t know AI, guess who will be revolutionized in the next decade? Who writes quicksort faster, you or DeepSeek?
- Many of the biggest things happen in AI. RLHF in DeepSeek and RL controller in humanoid robot on the Chinese New Year’s Gala.
- Even if you don’t plan to work on AI, AI is your powerful tool!

AI This Week:

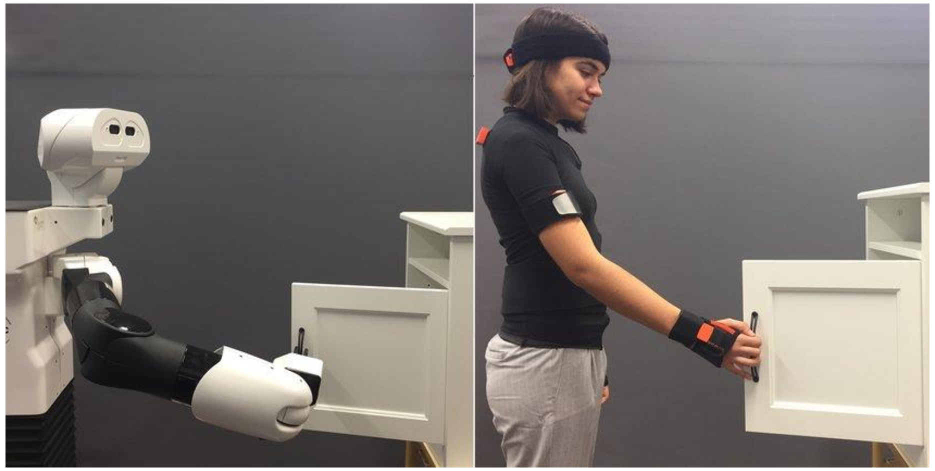
Let’s start & buckle up!

Any example of reinforcement learning (RL)?

What can you learn from this video?

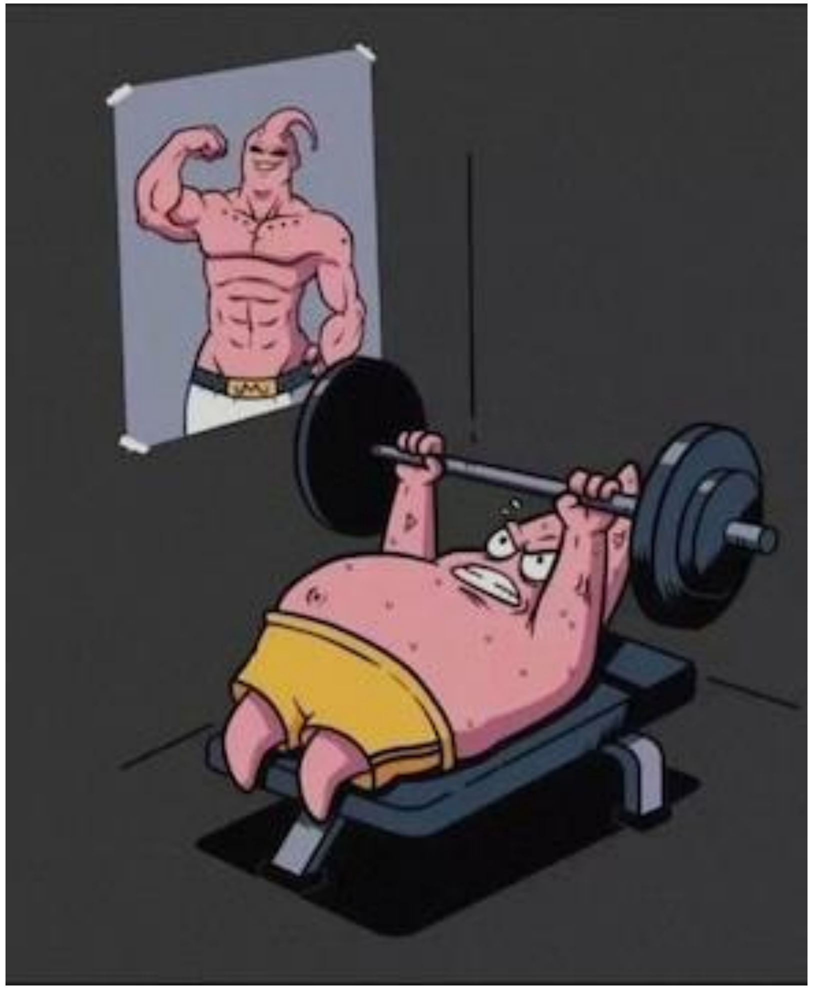
RL vs. supervised learning

- No supervision, only a reward signal



RL vs. supervised learning

- Delayed

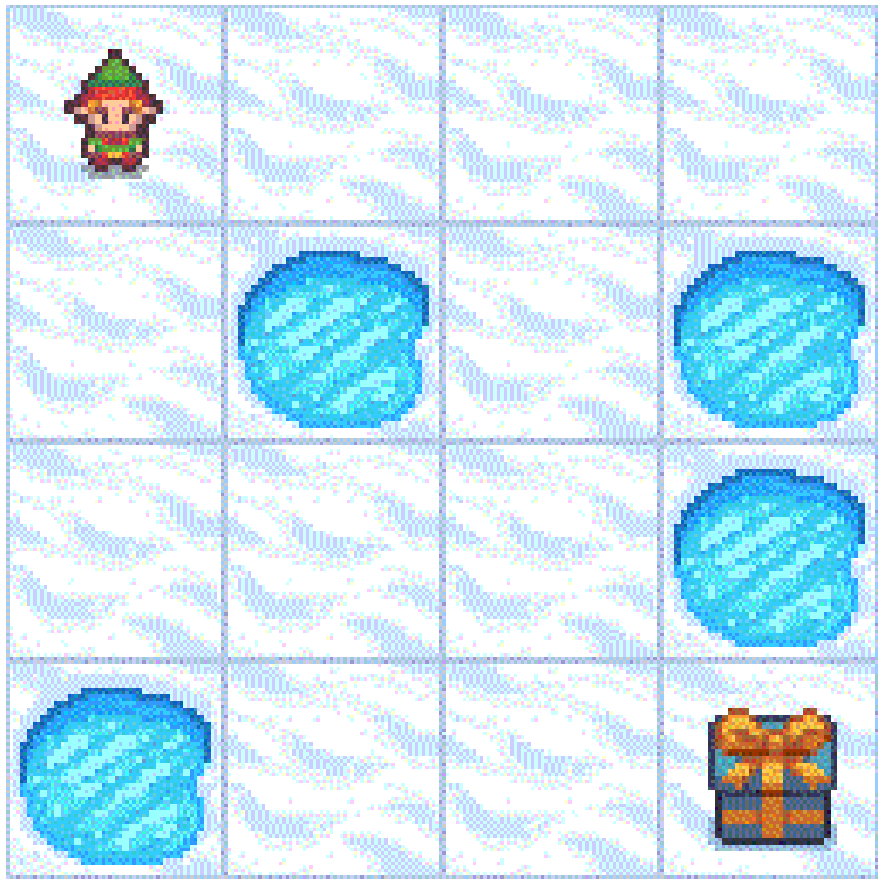


RL vs. supervised learning

- Not i.i.d data, it’s sequential!
- Your action matters.
    - You will never see the yolk if you don’t break the egg.

HOW TO MAKE PANCAKES
1 PREPARE OIL, FLOUR, SUGAR, EGGS, SALT, MILK
2 SIFT FLOUR, ADD SUGAR AND SALT
3 WHISK MILK AND EGGS
4 COMBINE DRY AND WET INGREDIENTS, ADD OIL AND WHISK
5 HEAT THE PAN, ADD OIL
6 PUT DOUGH INTO PAN
7 FRY AND FLIP
8 STACK AND SERVE WITH TOPPING

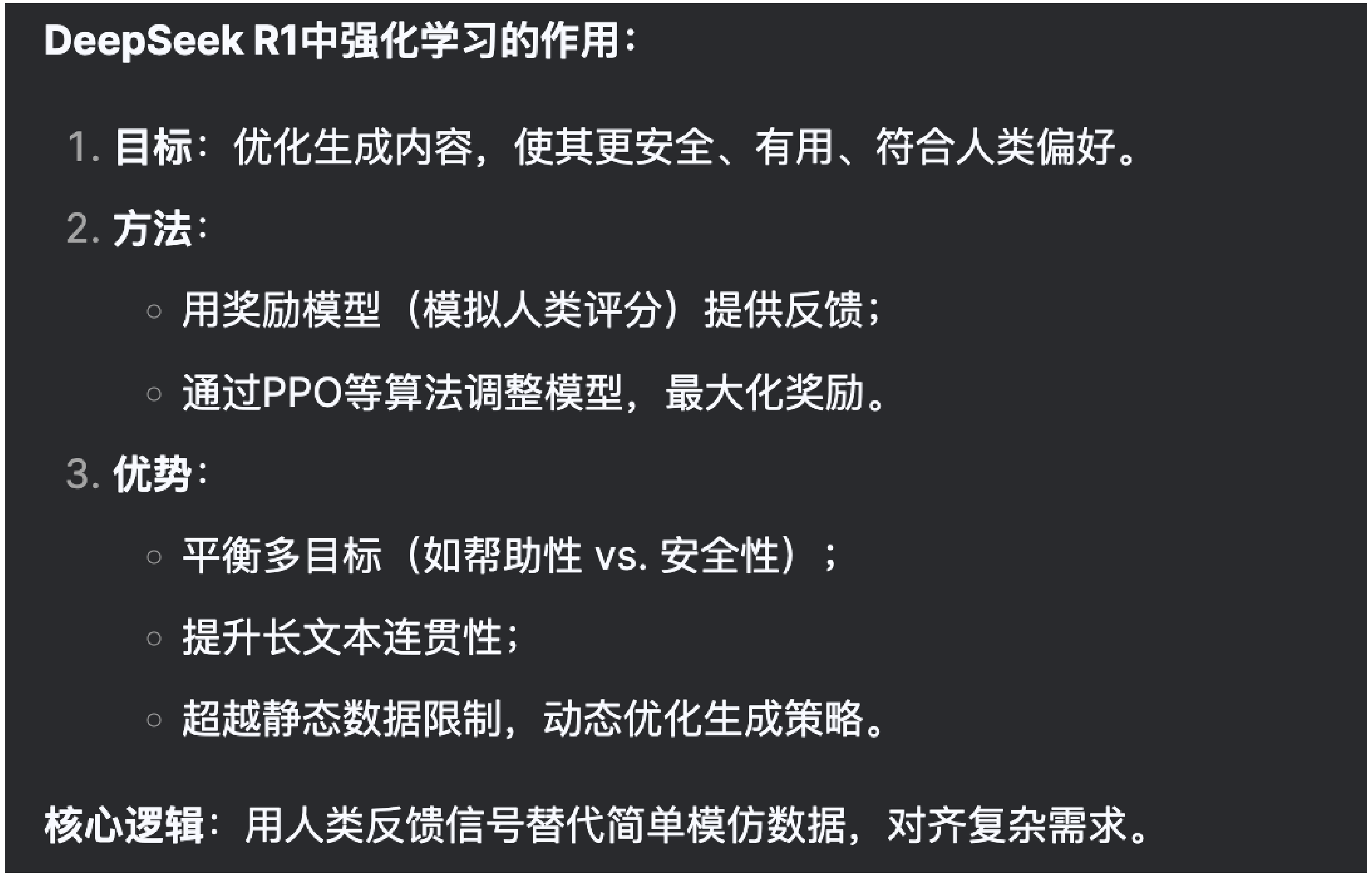
What can RL do?



What can RL do if you make it a bit deeper?

He et al, 2023

What can RL do if you make it a bit deeper?



DeepSeek R1中强化学习的作用：

1. **目标**：优化生成内容，使其更安全、有用、符合人类偏好。
2. **方法**：
   - 用奖励模型（模拟人类评分）提供反馈；
   - 通过PPO等算法调整模型，最大化奖励。
3. **优势**：
   - 平衡多目标（如帮助性 vs. 安全性）；
   - 提升长文本连贯性；
   - 超越静态数据限制，动态优化生成策略。

核心逻辑：用人类反馈信号替代简单模仿数据，对齐复杂需求。

Source:DeepSeek

What can RL do if you make it a bit deeper?

Source: youtube, OPENAI vs OG Dota2 game in The

What can RL do if you make it a bit deeper?

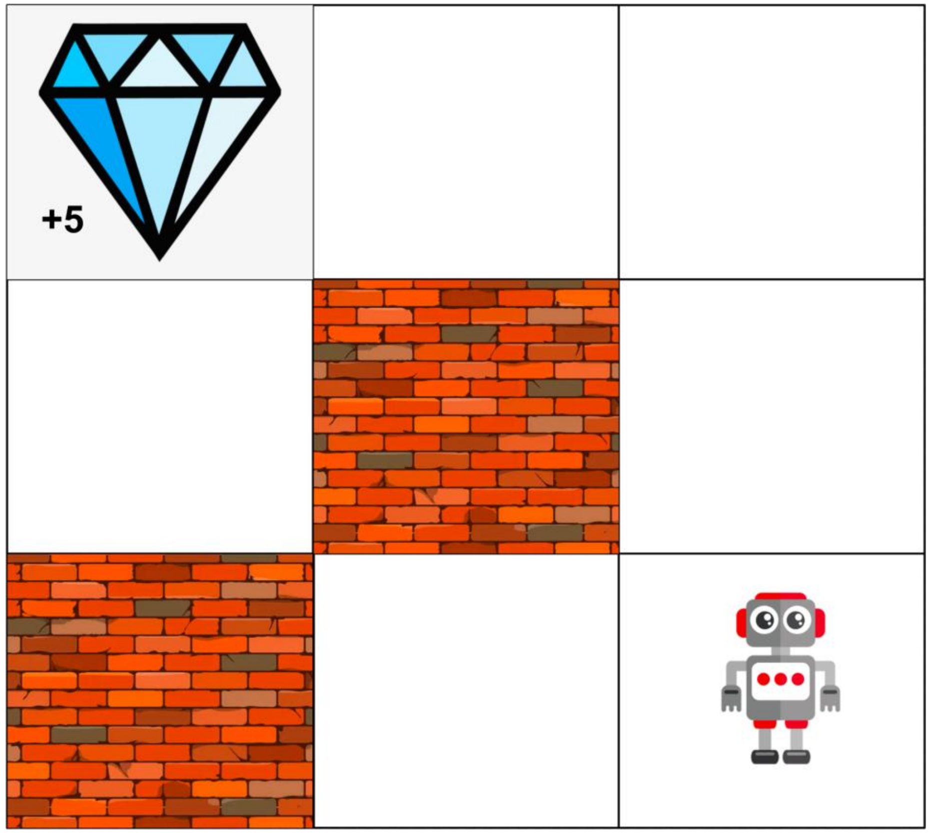
- Make a lot of money in trading
- Alleviate traffic scheduling problems
- What else?

Outline

Outline

Grid World

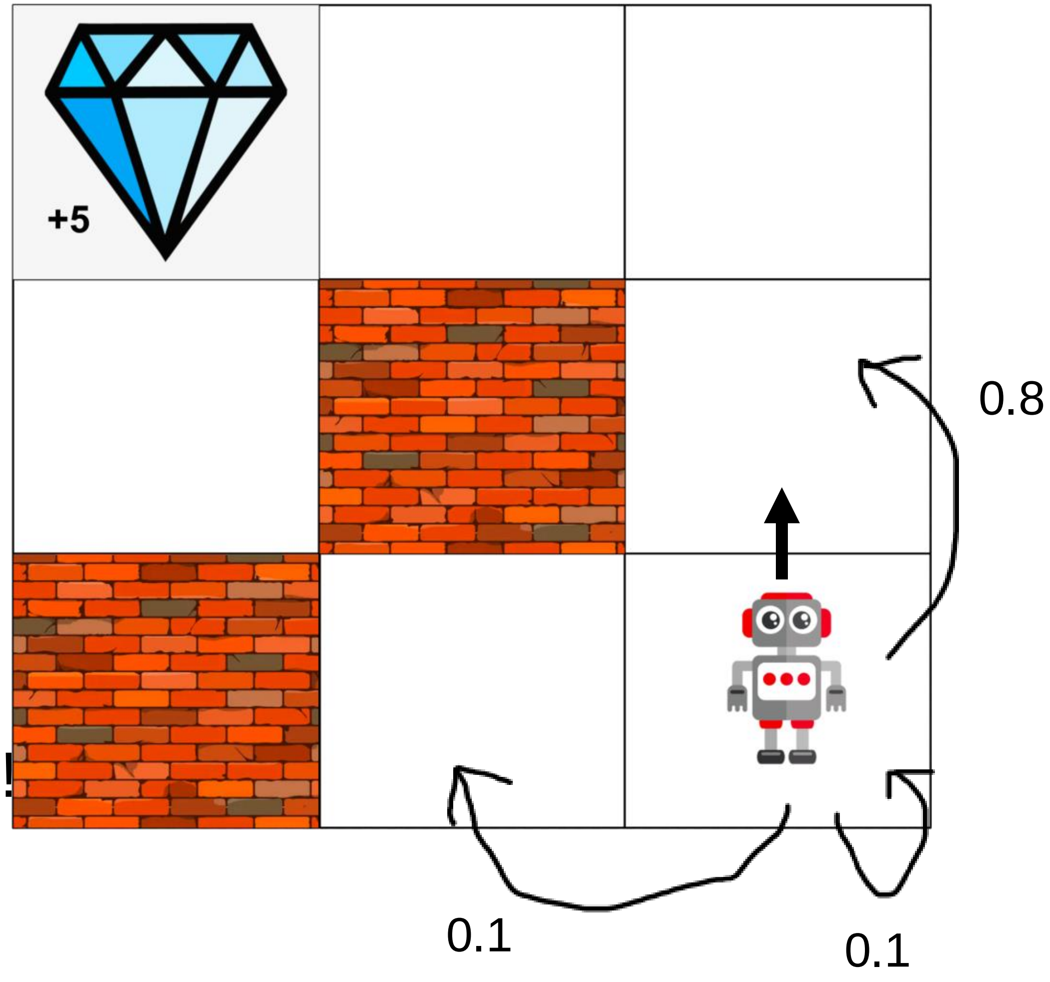
- A maze-like problem
  - The agent lives in this world
  - Walls and map boundaries block it
- Random movement
  - 80% chance the agent controls itself
  - 10% to the left, no matter what action
  - 10% to the right, no matter what action
- Reward from each time step
  - Big success reward
- Goal: maximize sum of reward



What is a Markov Decision Process (MDP)?

- The definition:
  - A set of **states s** $\in$ **S**
  - Sometimes a set **observations o** $\in$ **O**
  - A set of **actions a** $\in$ **A**
  - A **transition function T(s, a, s')**
    - Probability that action a at s leads to s', i.e., P(s'| s, a)
    - Also called the model or dynamics
  - A reward function **R(s, a, s')**
    - Sometimes simplified as R(s,a)
- An initial state distribution
- A terminal state

- MDPs are non-deterministic search problems!
  - You can do search!
  - What else can you think of?



"… actions can be any decision we want to learn how to make, states can be anything we can know that might be useful in making them."

—— Richard Sutton

Motors and linkage, muscles and skeletons, are these in an agent or the environment?

Why is it "Markov"?

- Given the present state, the future and the past are independent.

- MDP means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, \underbrace{S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0}_{\text{past states and actions}})$$

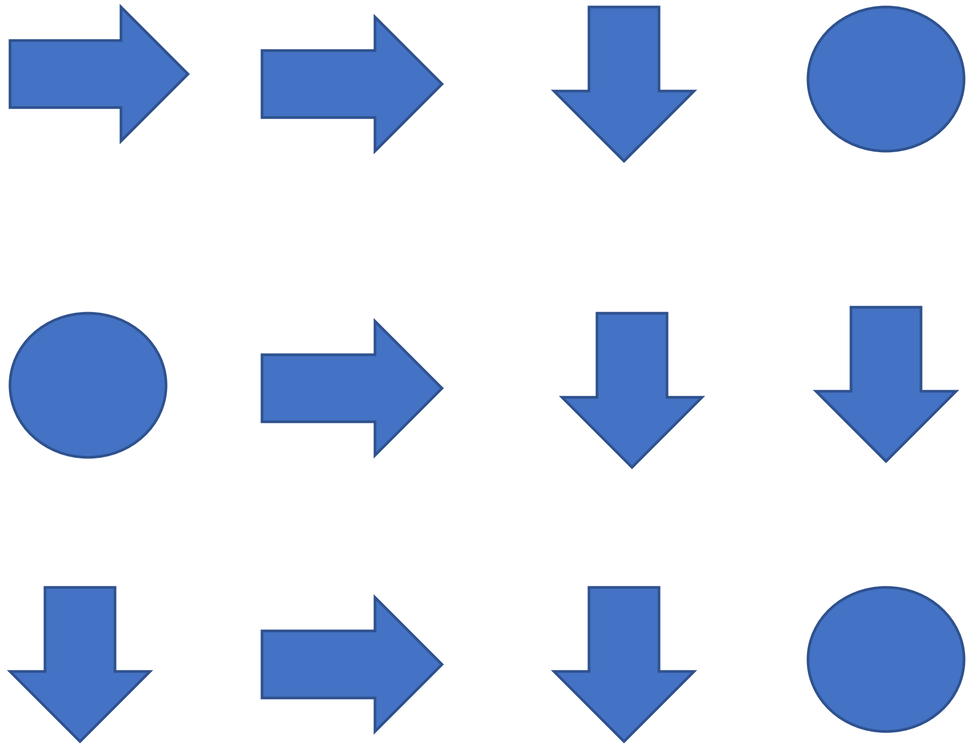
$$=$$

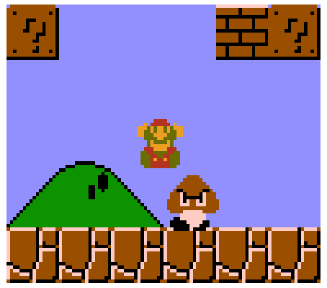
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Fully Observable or Partially Observable

- Fully observable: state is the same as observation
  - Go
- Partially Observable: only part of the states is observed
  - Robot camera
  - Poker
- If the agent is partially observable, the world is transformed from MDP into a Partially Observable MDP (POMDP).

A thought experiment



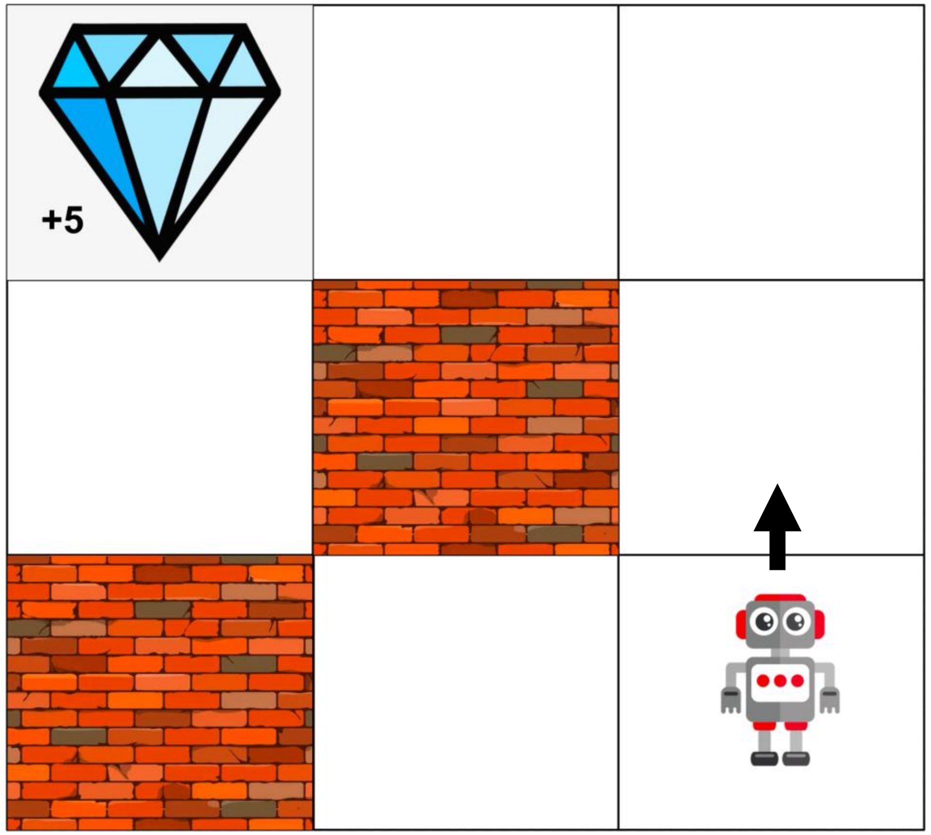


?

These are states!

If the agent makes a plan, it is a policy!

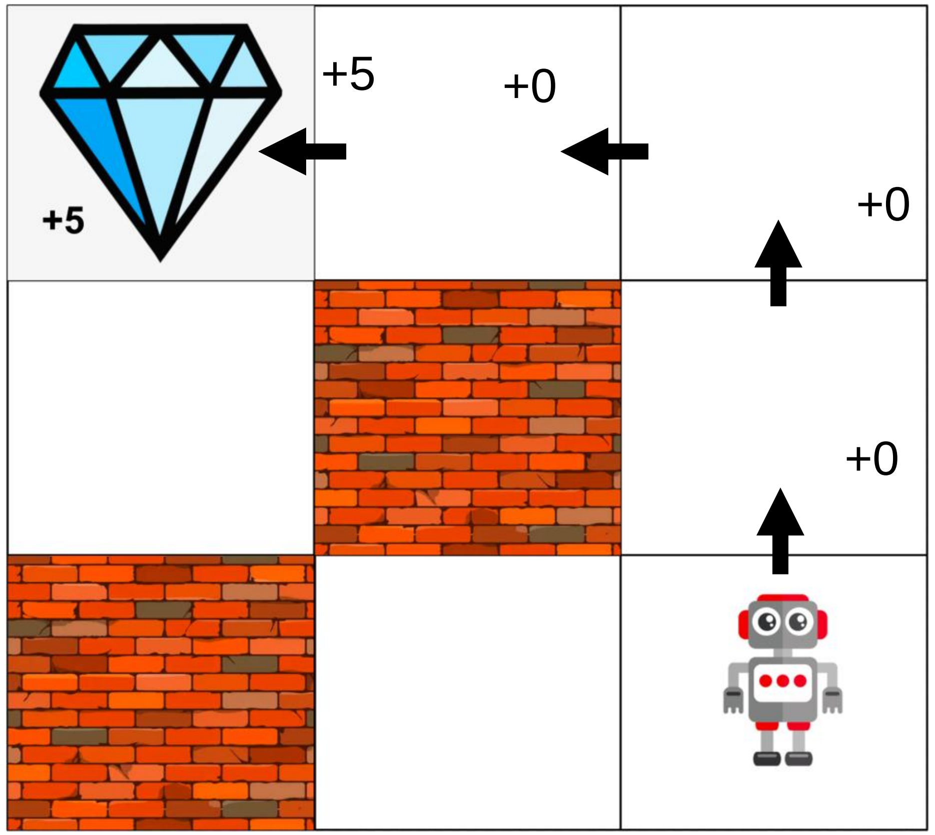
- For MDPs, we want an optimal policy π*: S → A
  - A policy π gives an action for each state
  - An optimal policy is the one that maximizes expected utility (sum of rewards)
  - An explicit policy defines a reflex agent
  - Sometimes we use $\pi(a|s)$ to express the probability of taking a certain action under a state.



Optimal policies to get max utility?

- Utilities are functions from outcomes (states) to real numbers that describe an agent's preferences.
- Utilities: [0, 0, 0, 5]

- Reward Hypothesis:
- All the goals are described by rewards.



Utilities of a Sequence

- More vs. less? [1, 2, 3] or [2, 5, 7]
- Now vs. later? [0, 0, 1] or [1, 0, 0]
- What about this one? [0, 0, 2] or [1, 0, 0]

Discount Factor on Rewards/Utilities

- You can maximize the sum of rewards
  - Additive utilities: U($[r_0, r_1, r_2, \ldots]$) = $r_0 + r_1 + r_2 + \cdots$
- You can also prefer rewards now to rewards later:
  - Discounted utilities: U($[r_0, r_1, r_2, \ldots]$) = $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$

1

Worth Now

γ

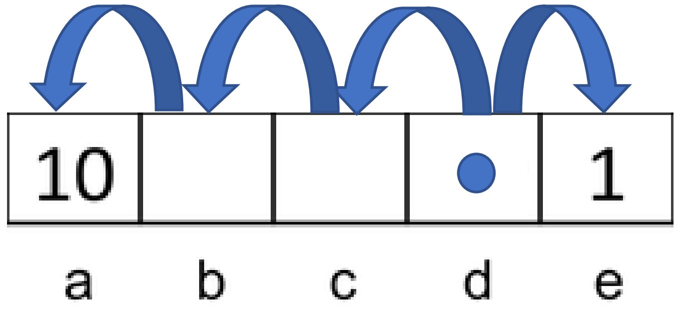
Worth Next Step

γ^2

Worth In Two Steps

Quiz time!

- Starting from d, which γ makes a and e equally good?



$$10\gamma^3 = 1\gamma$$

Most of the MDPs are discounted. Why?

- Avoid cyclic reward
- Infinite horizon
- Uncertainty about the future
- Time value
- Human like

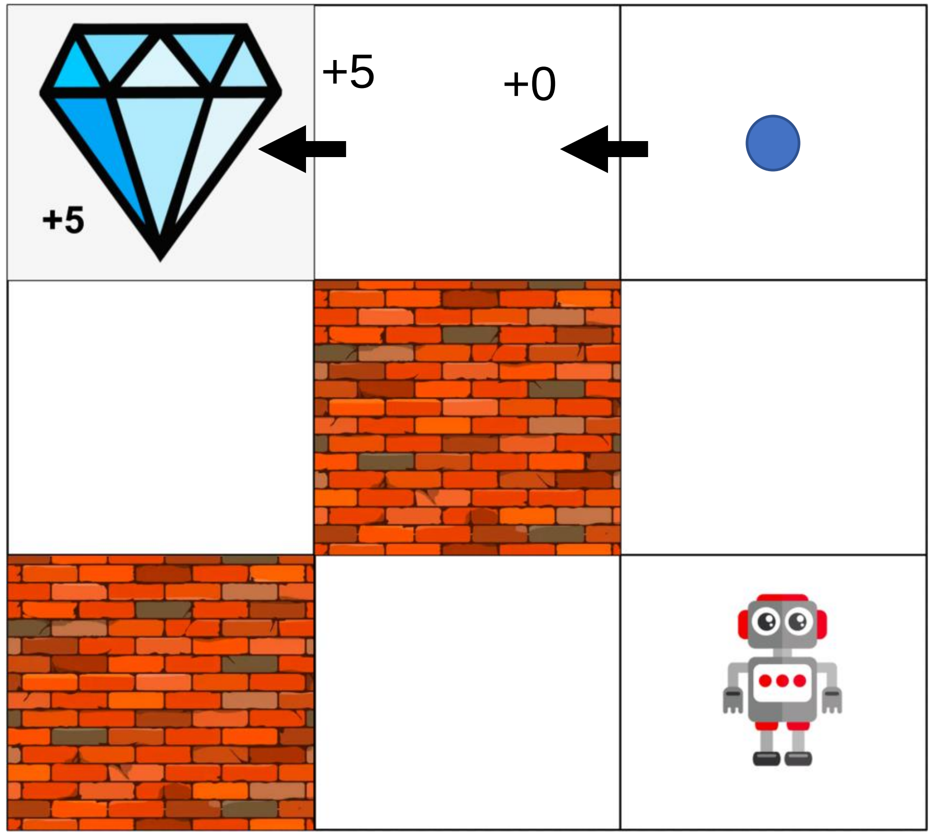
Recap of MDP

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

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

Optimal Quantities

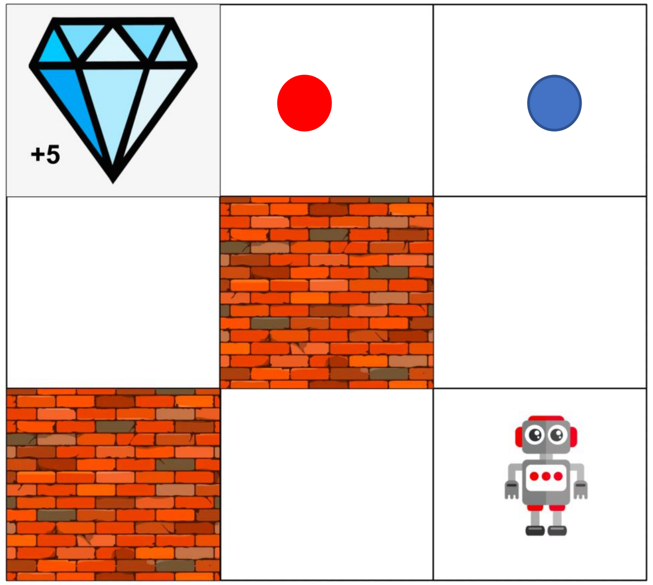
- The optimal policy
  - $\pi^*(s)$ = optimal action from state s
- The value (utility) of a state s:
  - V*(s) = expected utility starting in s and act optimally
- The Q value:
  - Q*(s, a) = expected utility taking action a from state s and acting optimally



Recursive Definition of Value: Bellman Equation!

- $V^*(s) = \max_a Q^*(s,a)$
  - V*(blue dot) = $\max\_a$ {Q*(bd, N), Q*(bd, S), Q*(bd, W), Q*(bd, E)}
- $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
  - Q*(bd, W) = (0.8+0.1)(reward_if_go_W + γ V*(rd)) +

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

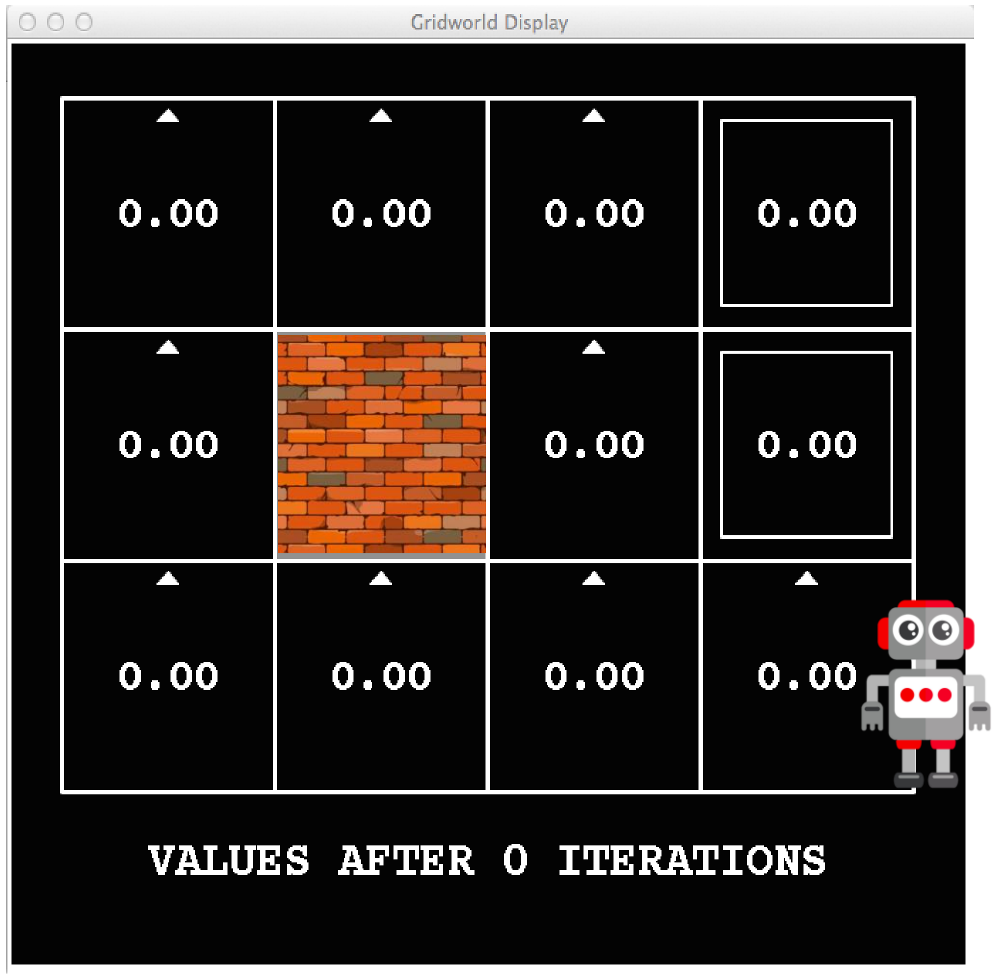


Outline

How can we get these quantities? **Value Iteration!**

- Define $V_{\mathrm{k}}(s)$ to be the optimal value of s if the game ends in k more time steps. （It works backward!）
- Start with $V_0(s) = 0$: Zero utility if the game ends now!
- Given $V_{\mathrm{k}}(s)$, we can compute $V_{\mathrm{k}+1}(s)$:
  - $V_{\mathrm{k}+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{\mathrm{k}}(s')]$
- Repeat until convergence
- Does the policy converge earlier or later than value?
  - Earlier!
- What's the complexity of each iteration in terms of state space cardinality S and action space cardinality A?
  - O(SA) or O(S^2A)?
  - O(S^2A), because you need to operate on every state!

k=0



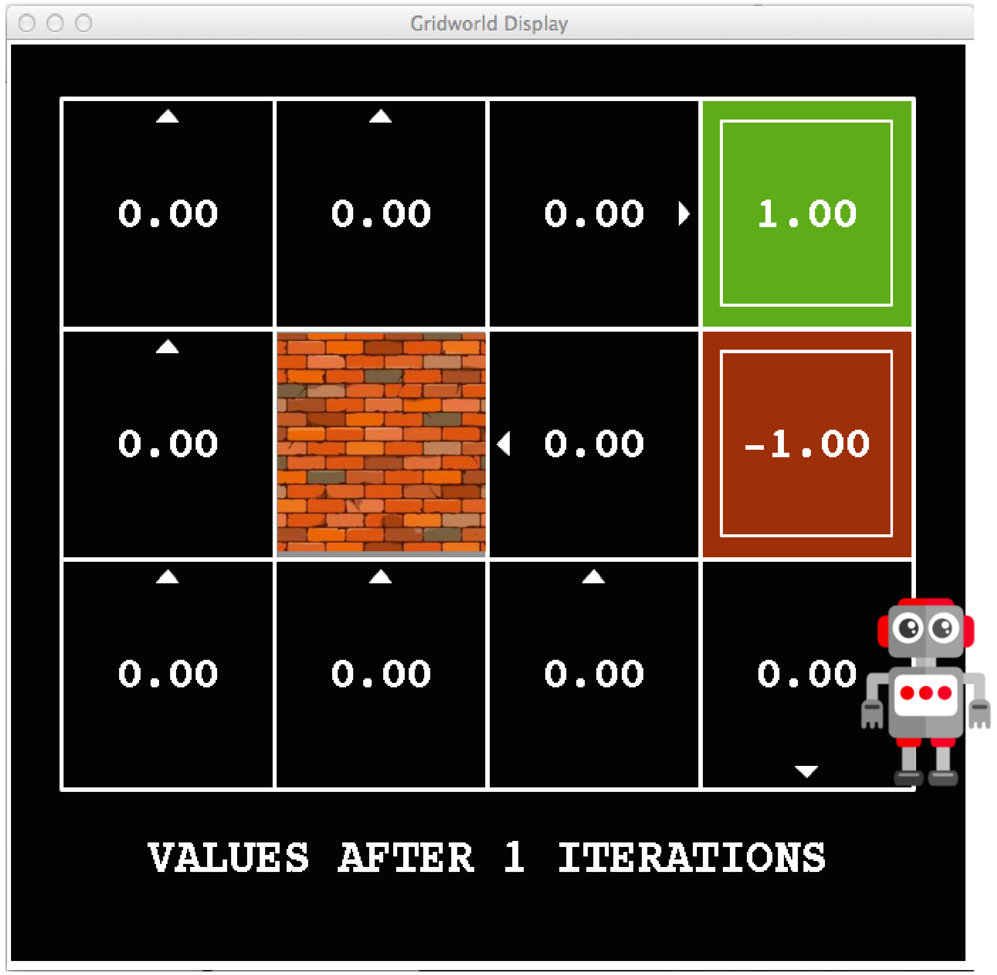
Gridworld Display

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=1

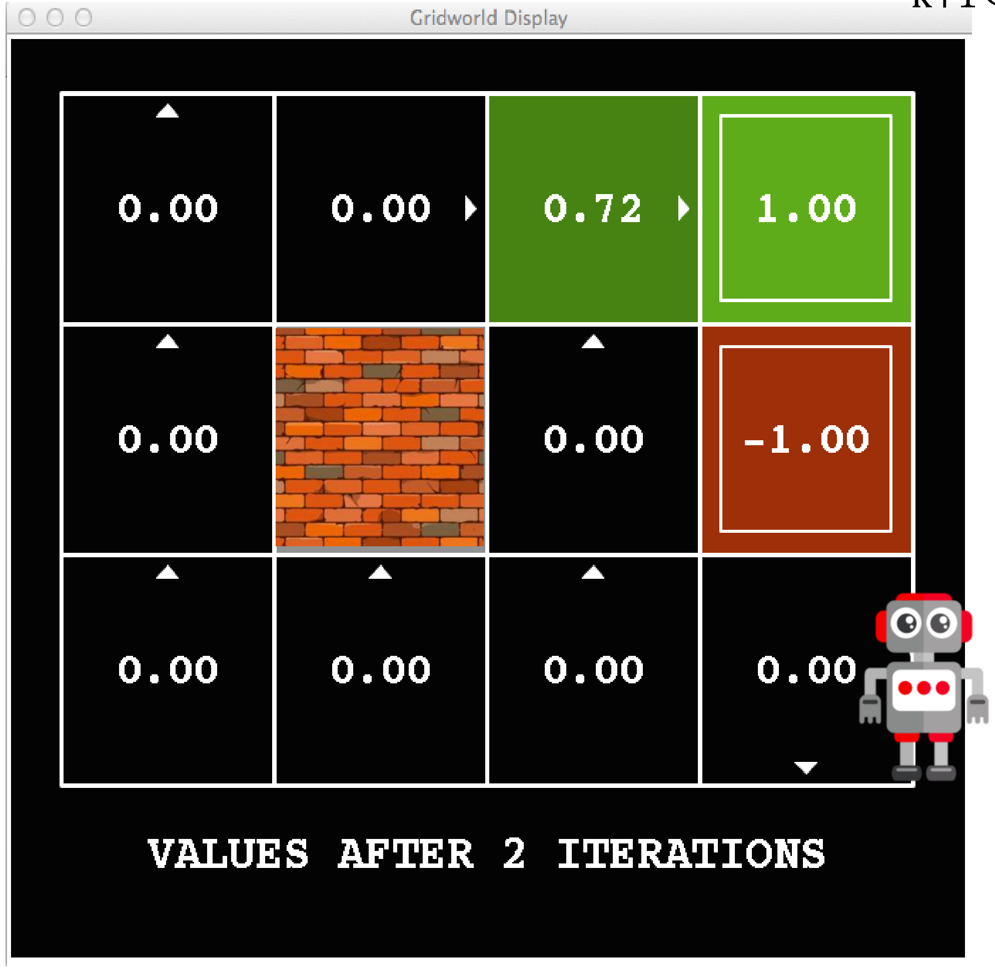


Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

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

0.72 = [0.8(0+0.9 x 1) + 0.2(0 + 0.9 x 0)]

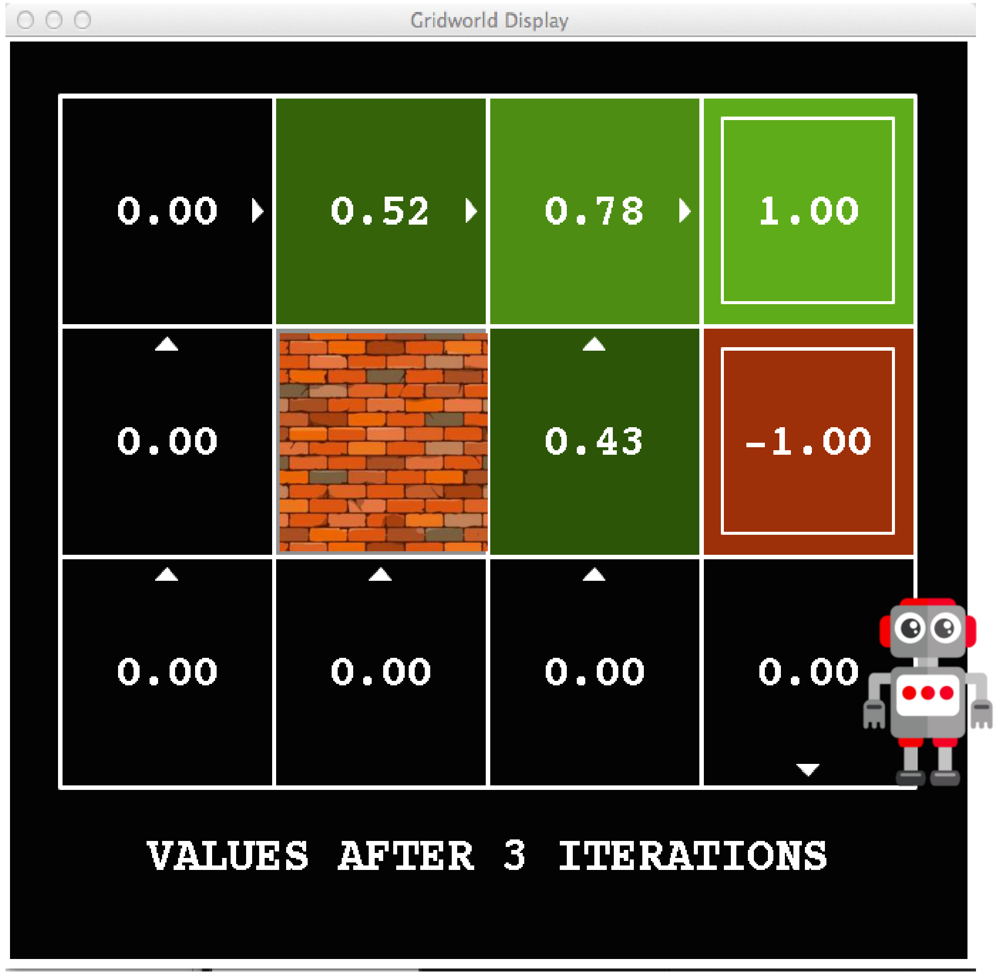


0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

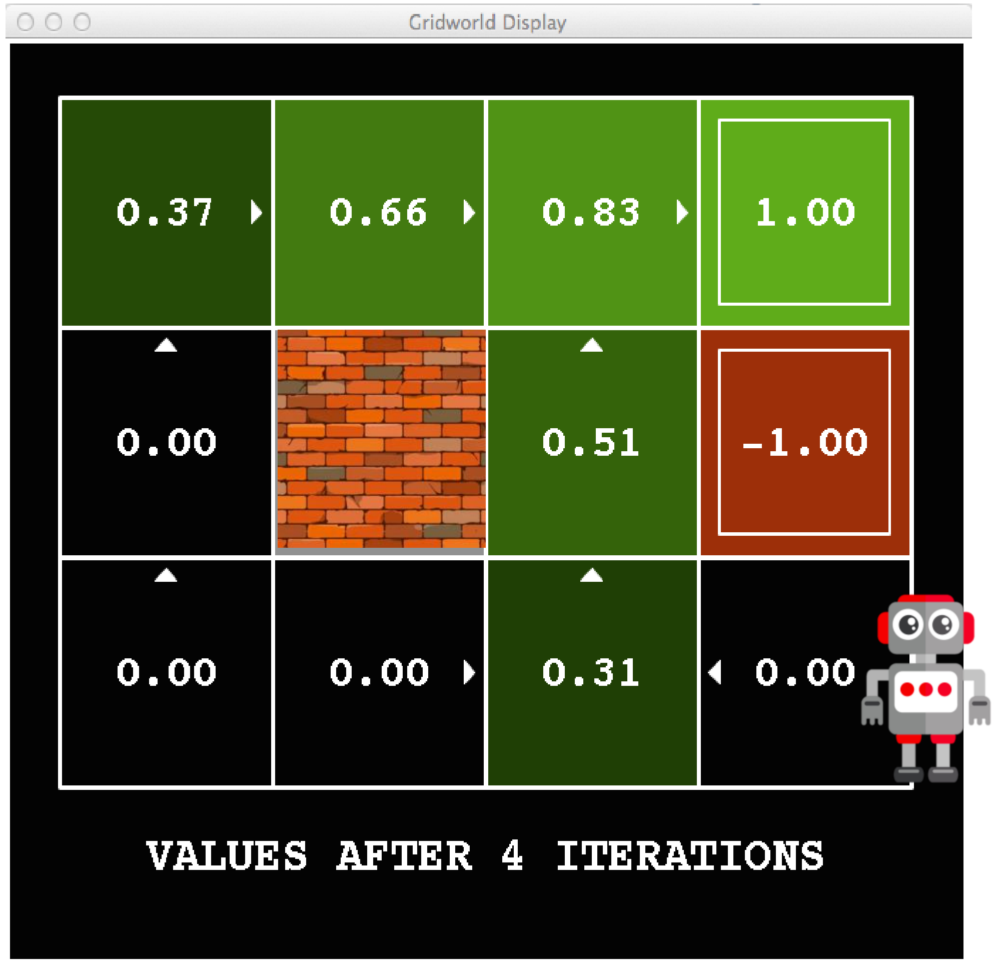

Gridworld Display

0.00 ▶	0.52 ▶	0.78 ▶	1.00
▲ 0.00		▲ 0.43	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=4

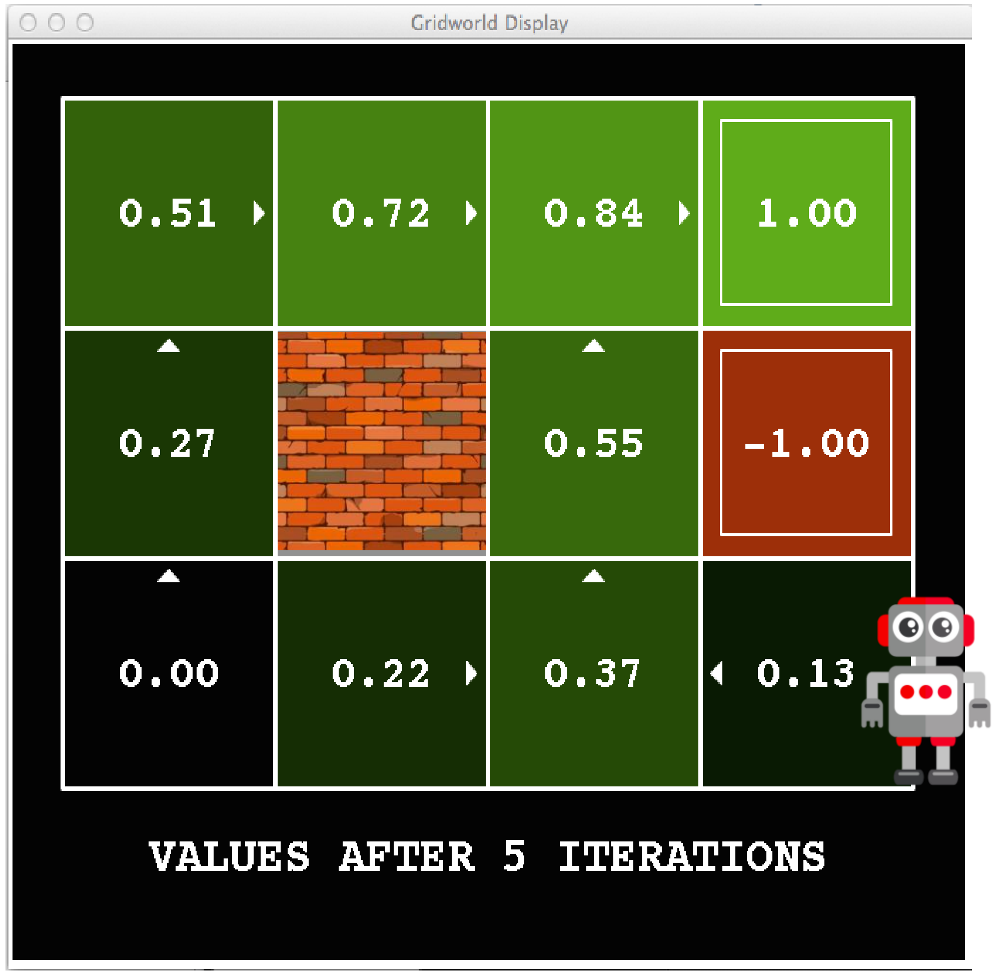


0.37 ▶	0.66 ▶	0.83 ▶	1.00
▲ 0.00		▲ 0.51	-1.00
▲ 0.00	0.00 ▶	▲ 0.31	◀ 0.00

VALUES AFTER 4 ITERATIONS

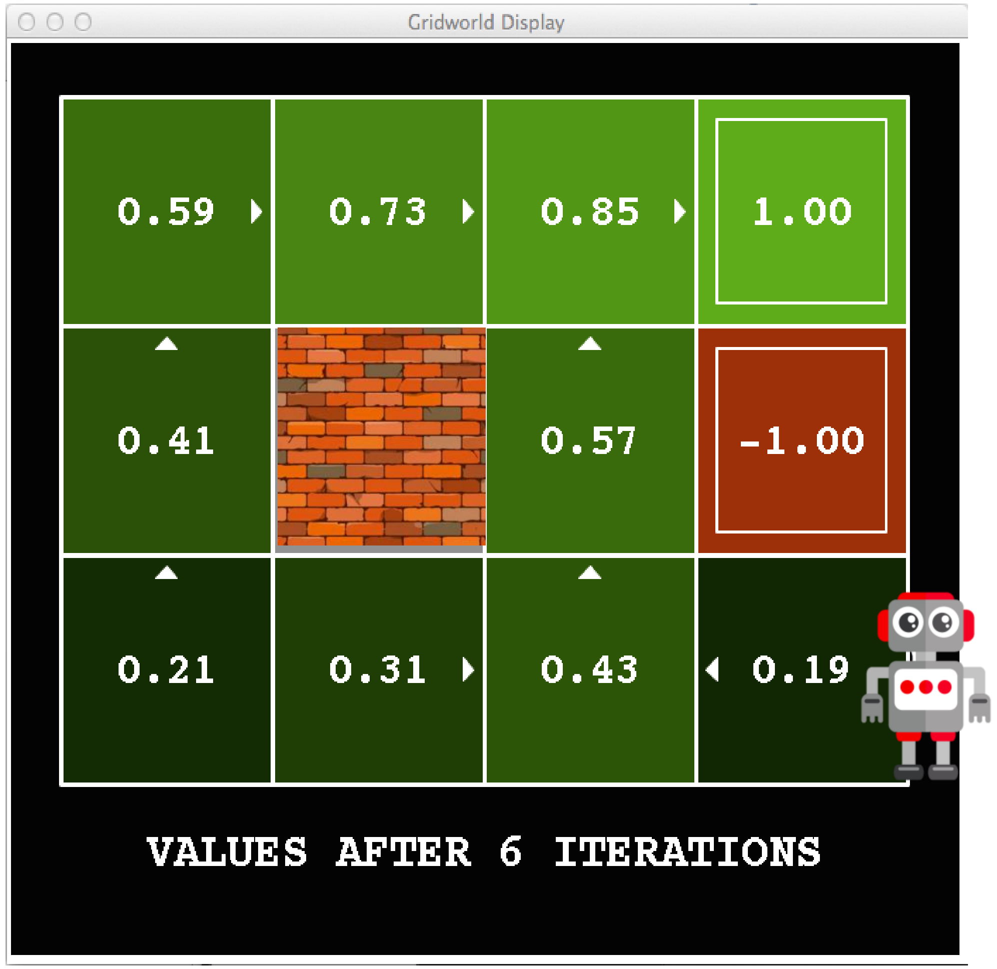
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

k=6

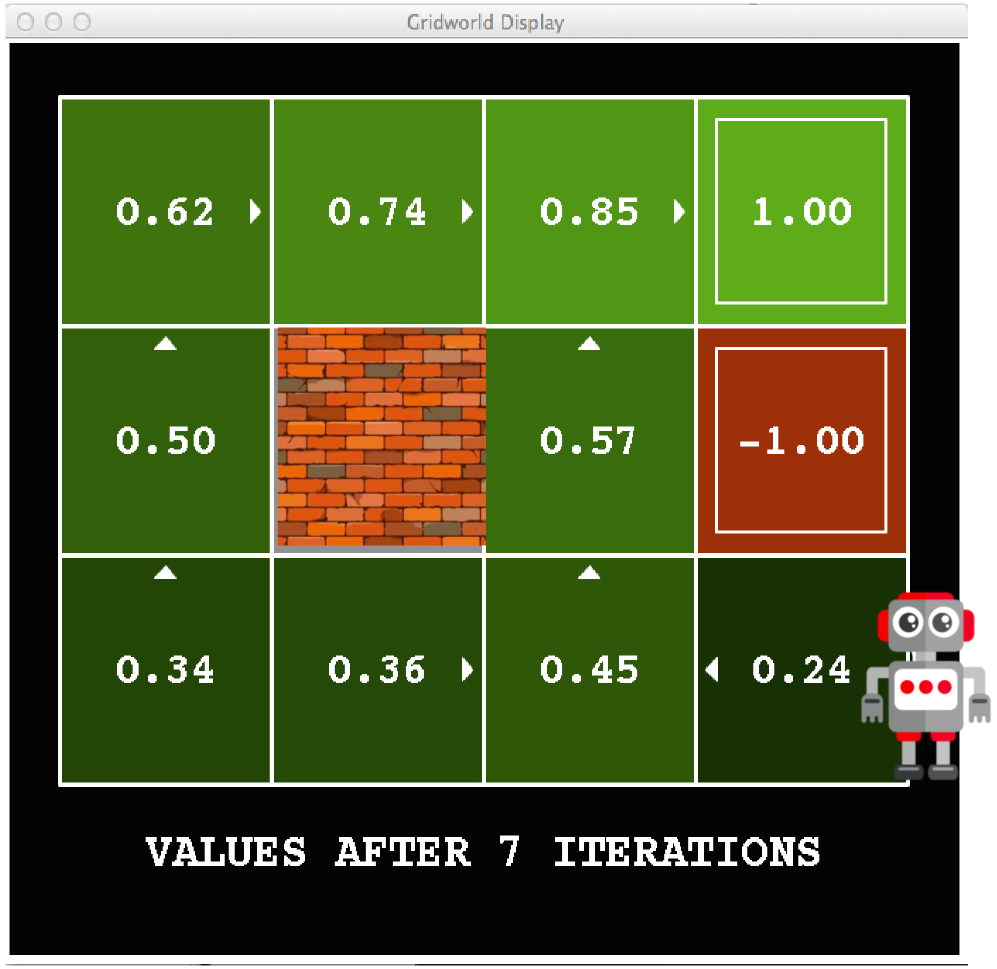


0.59 ▸	0.73 ▸	0.85 ▸	1.00
▴ 0.41		▴ 0.57	-1.00
▴ 0.21	0.31 ▸	▴ 0.43	◂ 0.19

VALUES AFTER 6 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

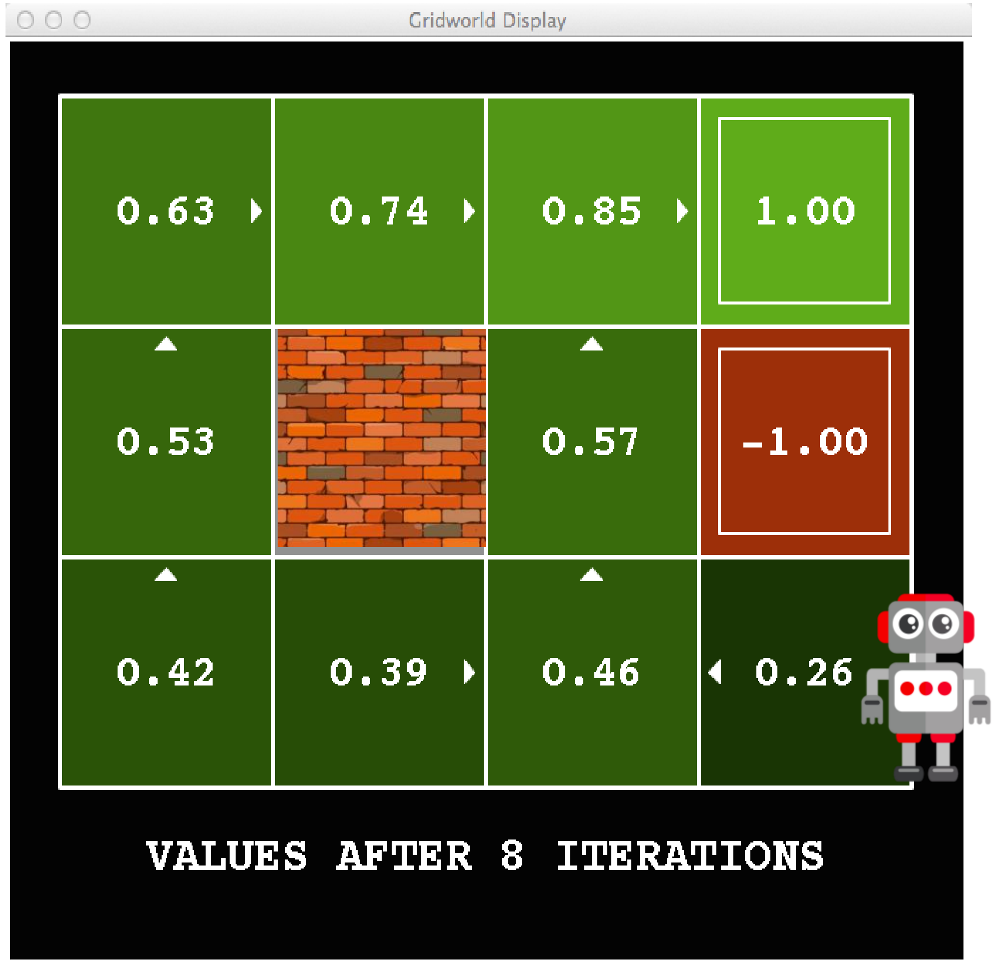


0.62 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.50		▴ 0.57	-1.00
▴ 0.34	0.36 ▸	▴ 0.45	◂ 0.24

VALUES AFTER 7 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

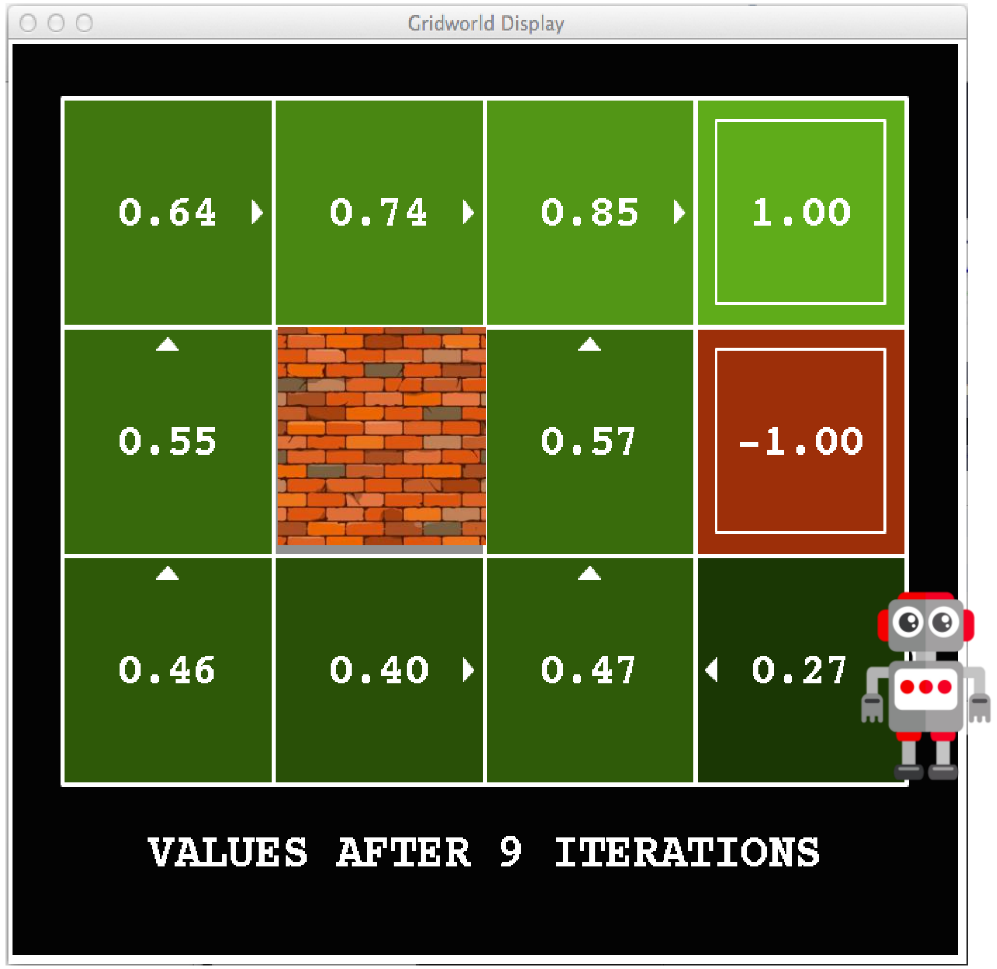


0.63 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.53		▴ 0.57	-1.00
▴ 0.42	0.39 ▸	▴ 0.46	◂ 0.26

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

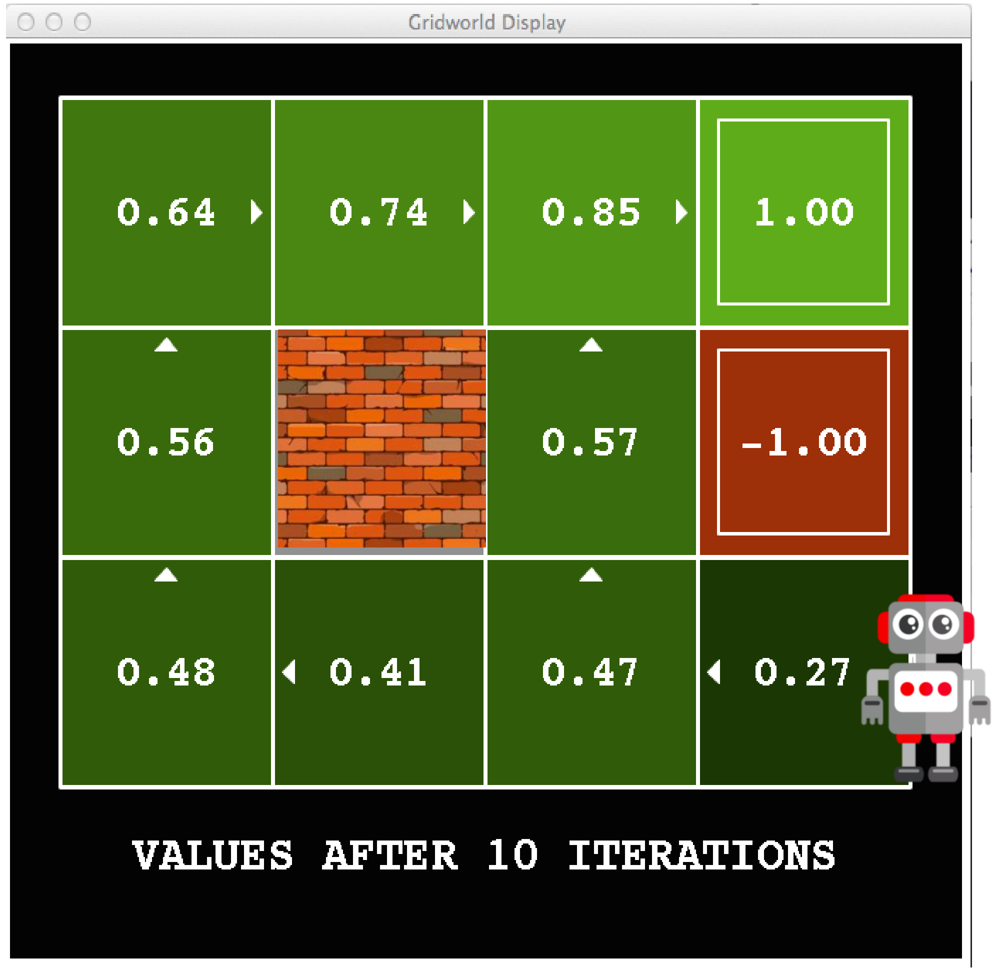


Noise = 0.2

Discount = 0.9

Living reward = 0

k=10

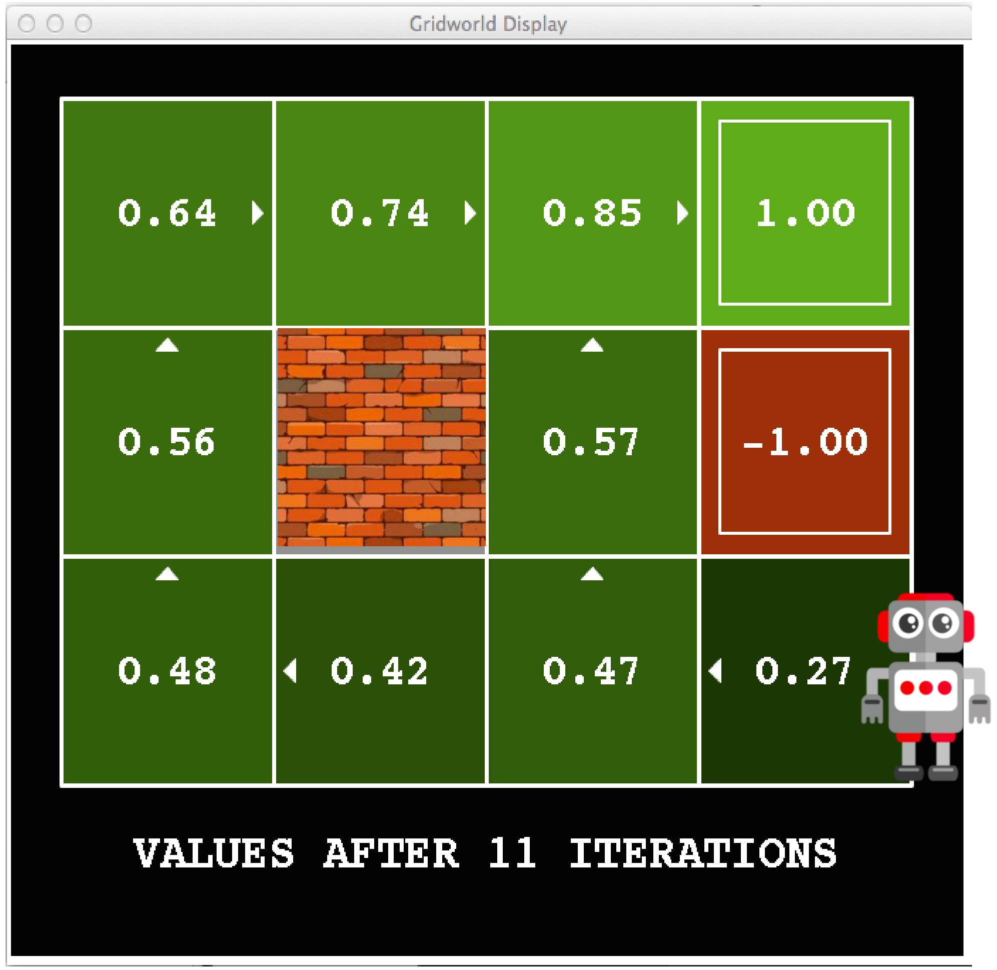


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.56		▲ 0.57	-1.00
▲ 0.48	◀ 0.41	▲ 0.47	◀ 0.27

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

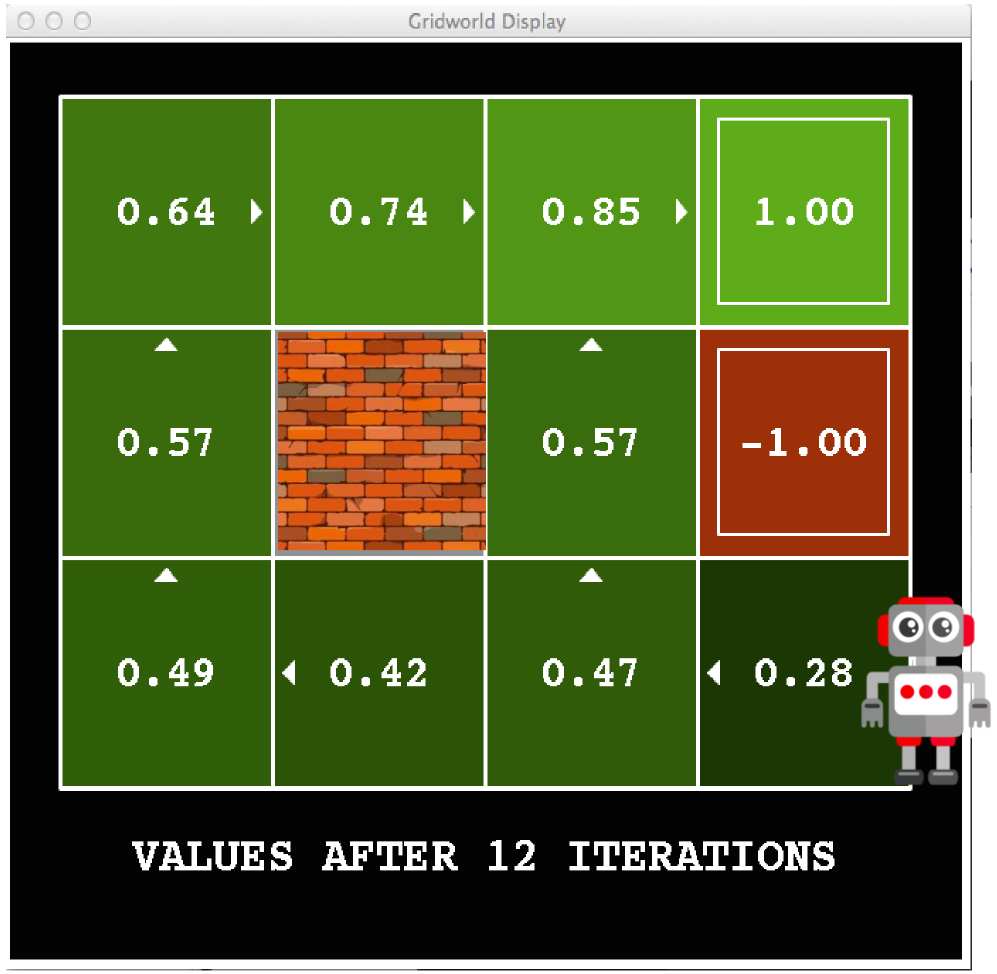


0.64 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.56		▴ 0.57	-1.00
▴ 0.48	◂ 0.42	▴ 0.47	◂ 0.27

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

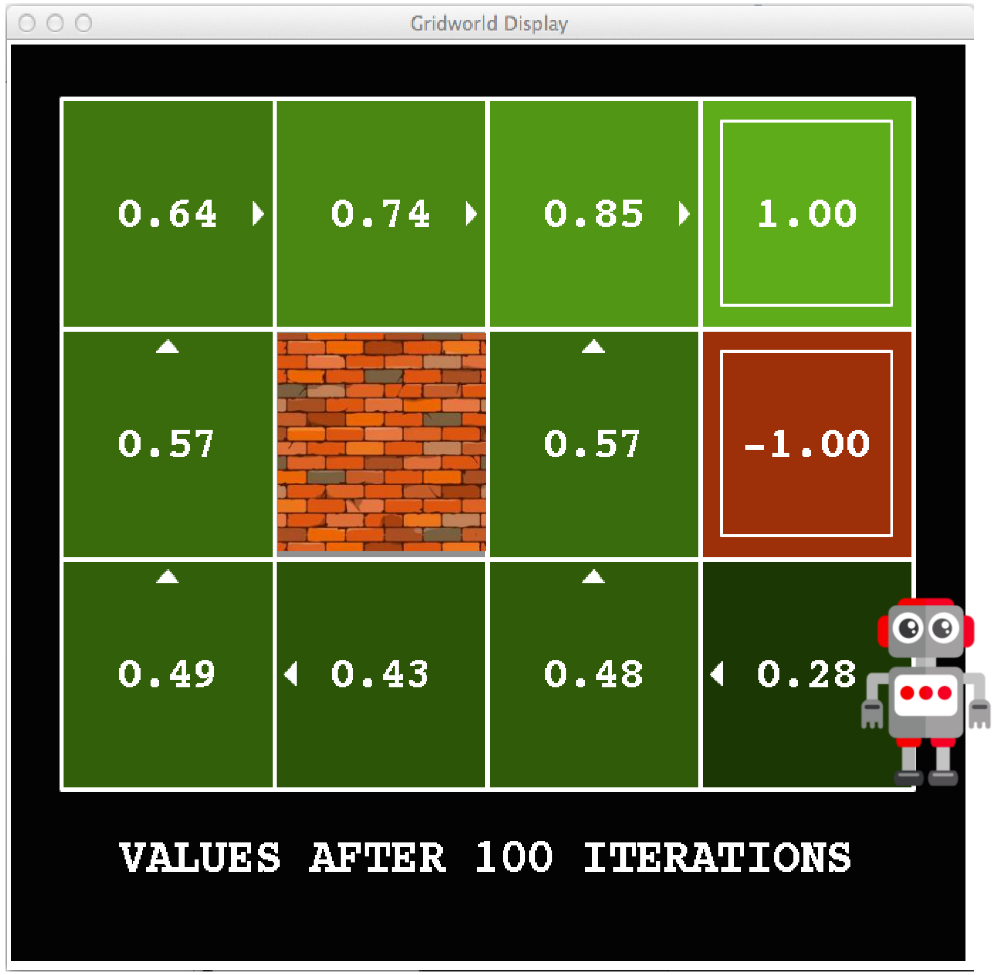


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.42	▲ 0.47	◀ 0.28

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

Until k=100

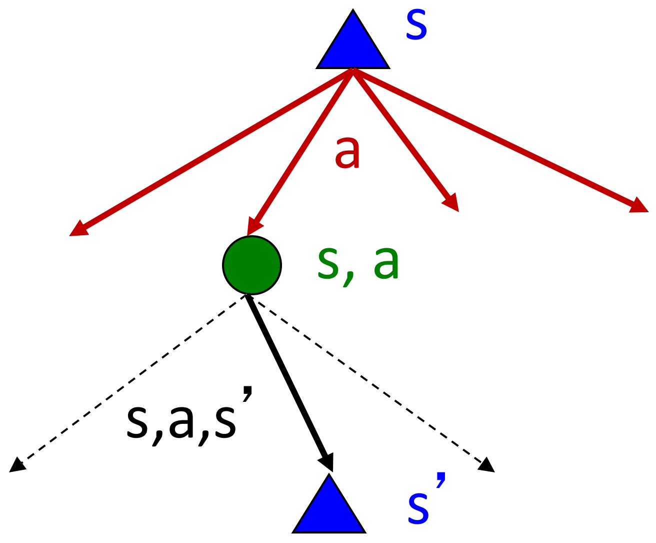


Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

- Value iteration repeats the Bellman update:

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

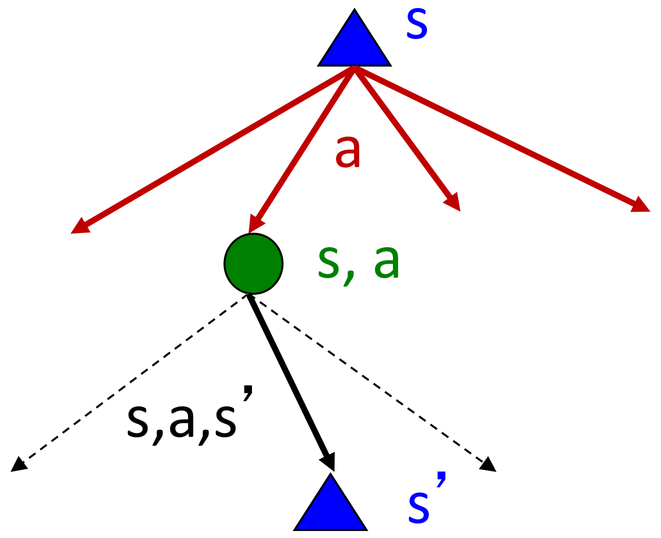


- Problem 1: It's slow – O(S^2A) per iteration

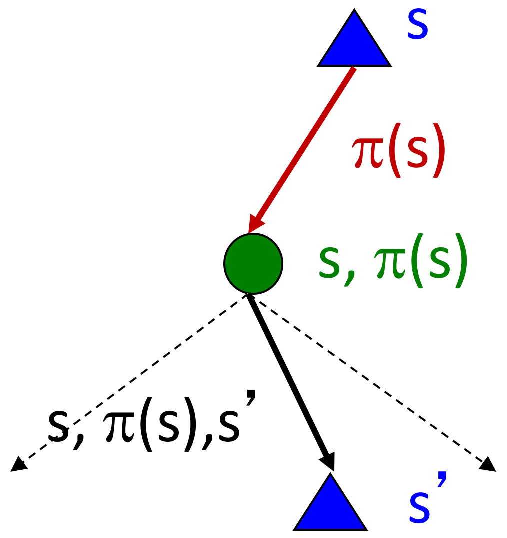
- Problem 2: The policy often converges long before the values

Fixed Policies

Do the optimal action



Do what π says to do



- Value iteration max over all actions to compute the optimal values
- If we fixed some policy π(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

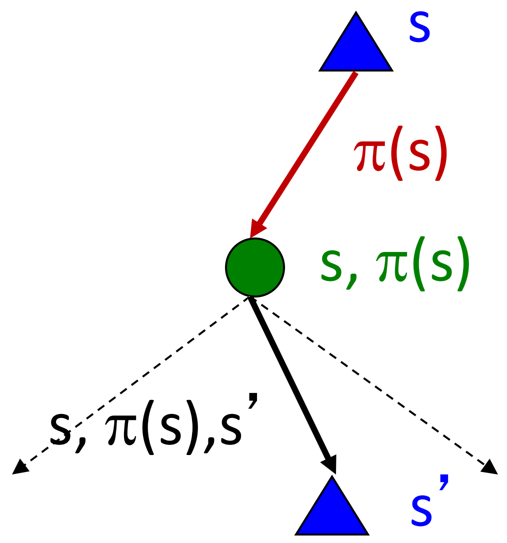
- Define the utility of a state s, under a fixed policy π:

  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

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

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

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



Outline

Policy Evaluation

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

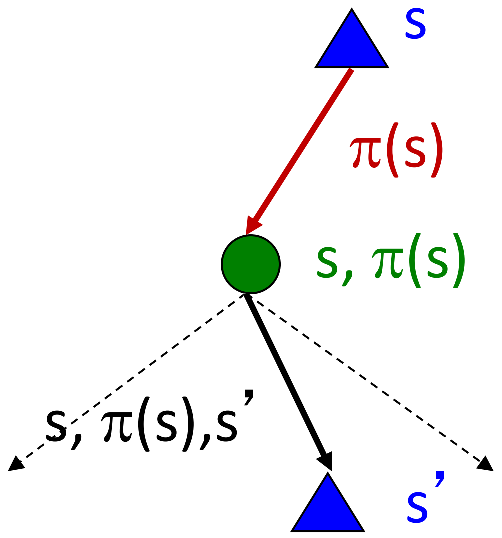
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

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

- 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)



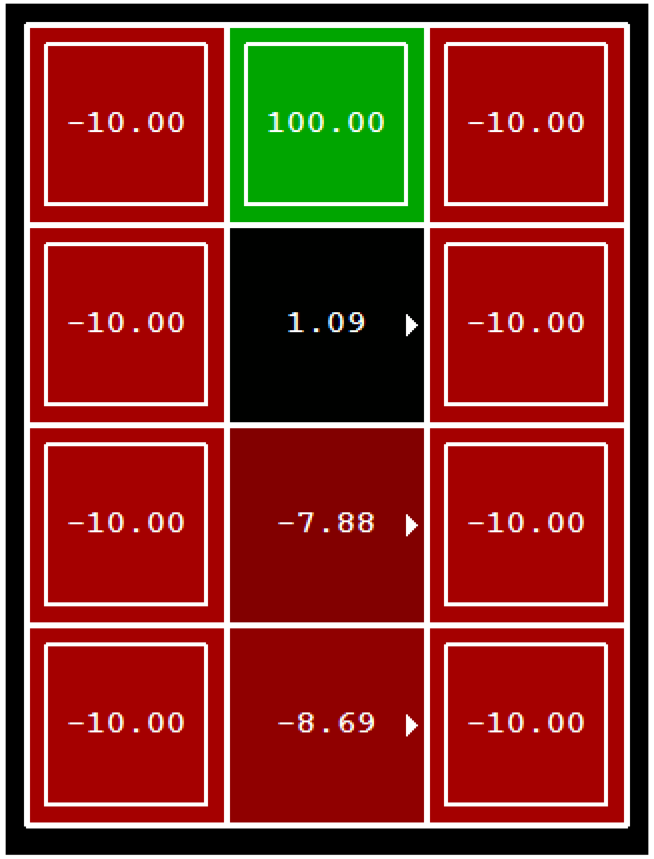
A detour to idea 2 first

- Your bellman equation is this (if you abuse the notation a bit..):
  - $V \ = P(r + \gamma V)$
- Let's assume $R = Pr$
- Then we have $(I - \gamma P)V = R$
- $V = (I - \gamma P)^{-1}R$

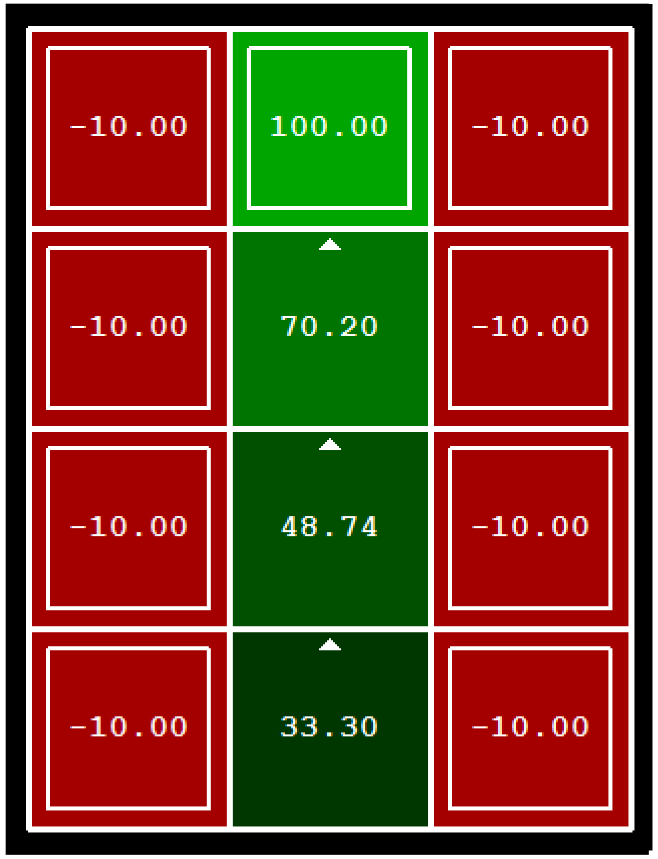
If the MDP is very small, then you can use this method.

Example: Policy Evaluation

Always Go Right

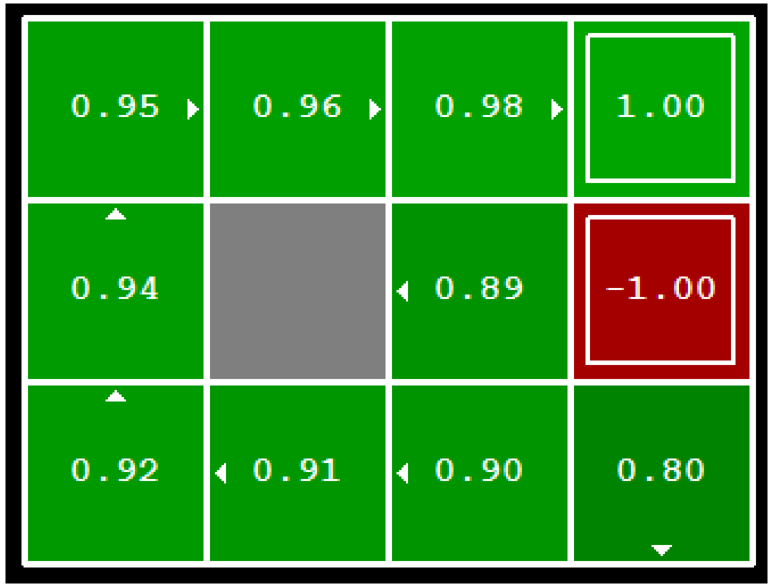


Always Go Forward



Computing Actions from Values --- Policy Improvement

- Let's imagine we have the values V(s)
- How should we act?
  - It's not obvious!
- We need to do a one step look-up:



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

- This is called policy improvement, since it gets the policy implied by 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:
  - Iterate until values converge:

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

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

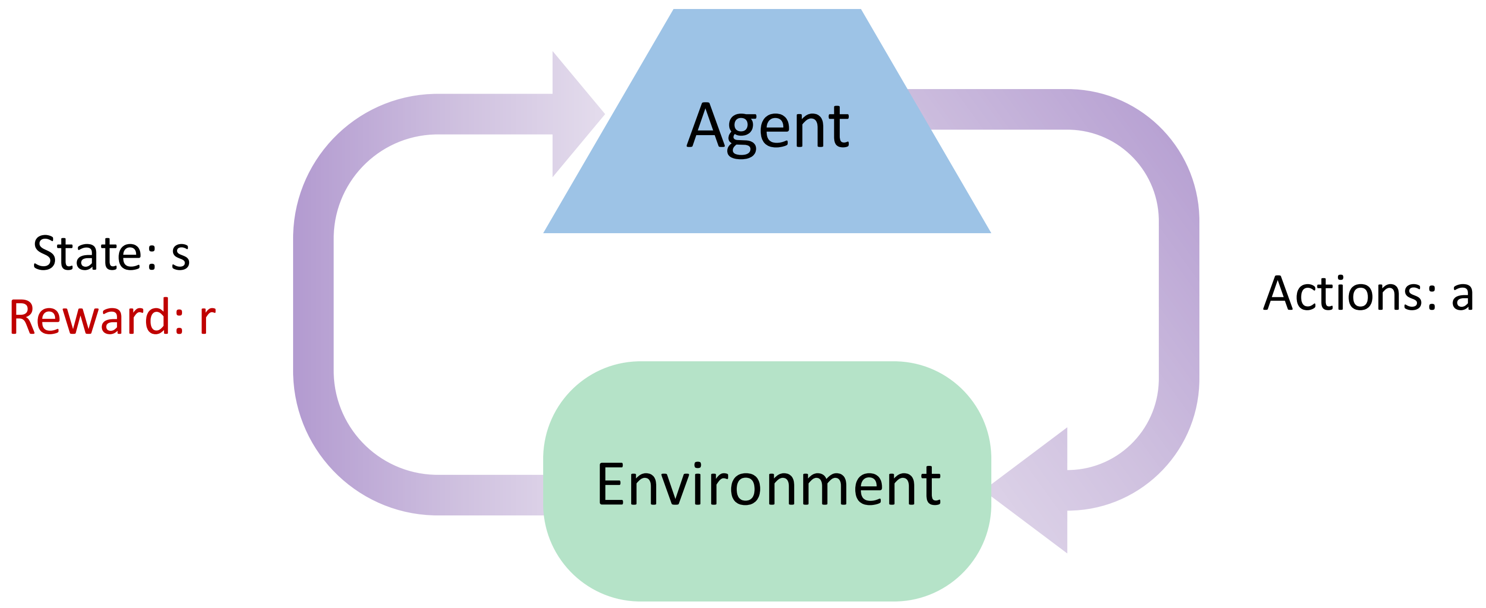
$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Outline

What if we don’t know the transition?

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

Next big idea: Reinforcement Learning



Basic idea:

- Receive feedback in the form of rewards
- Agent's utility is defined by the reward function
- Must act to maximize expected rewards
- All learning is based on observed **samples** of outcomes!

Summary

1. MDP, optimal quantities, bellman equation
2. Value Iteration: a recursive way to use MDP
3. Policy Iteration: two-stage algorithm that converges faster
4. Reinforcement Learning: learn without transition