



CS 540 Introduction to Artificial Intelligence

Reinforcement Learning II / Summary

Fred Sala
University of Wisconsin-Madison

April 22, 2021

Announcements

- **Homeworks:**
 - HW8 graded; released after class.
- **OH:** Monday, 4-7 pm
 - Chat about whatever you like.

Tuesday, April 20	Reinforcement Learning I
Thursday, April 22	RL II + Search Summary
Tuesday, April 27	AI in the Real World
Thursday, April 29	AI Ethics

- **Class roadmap:**

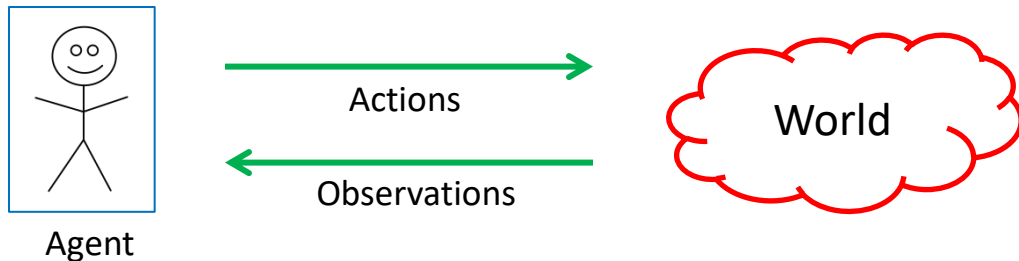
Outline

- Review of reinforcement learning
 - MDPs, value functions, value iteration
- Q-learning
 - Q function, SARSA, deep Q-learning
- Search + RL Review
 - Uninformed/informed search, optimization, RL

Building The Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A
- Information: at time t , observe state $s_t \in S$. Get reward r_t
- Agent makes choice $a_t \in A$. State changes to s_{t+1} , continue



Goal: find a map from **states to actions** maximize rewards.

↑
A “policy”

Markov Decision Process (MDP)

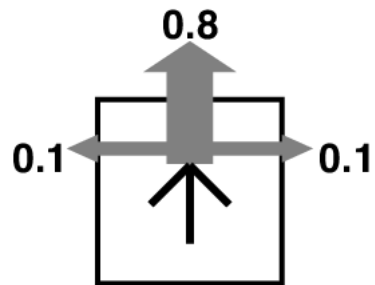
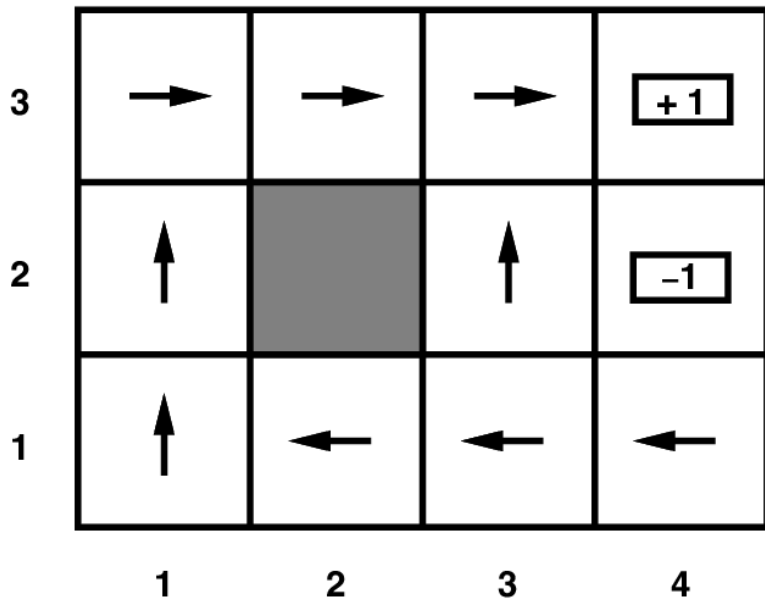
The formal mathematical model:

- **State set** S . Initial state s_0 . **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
 - Markov assumption: transition probability only depends on s_t and a_t , and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy:** $\pi(s) : S \rightarrow A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Grid World Optimal Policy

Note: (i) Robot is unreliable (ii) Reach target fast



$r(s) = -0.04$ for every non-terminal state

Defining the Optimal Policy

For policy π , **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^\pi(s_0) = \sum_{\text{sequences starting from } s_0} P(\text{sequence})U(\text{sequence})$$

Called the **value function** (for π, s_0)



Discounting Rewards

One issue: these are infinite series. **Convergence?**

- Solution

$$U(s_0, s_1 \dots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \dots = \sum_{t \geq 0} \gamma^t r(s_t)$$


- Discount factor γ between 0 and 1
 - Set according to how important **present** is VS **future**
 - Note: has to be less than 1 for convergence

Values and Policies

Now that $V^\pi(s_0)$ is defined what a should we take?

- First, set $V^*(s)$ to be expected utility for **optimal** policy from s
- What's the expected utility of an action?
 - Specifically, action a in state s ?

$$\sum_{s'} P(s'|s, a) V^*(s')$$



All the states we
could go to

Transition probability

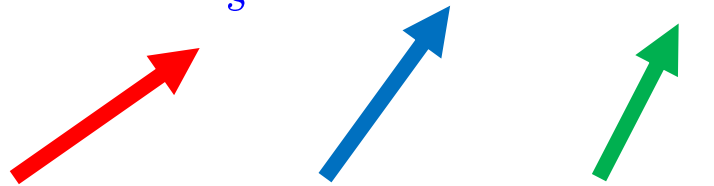
Expected rewards

Obtaining the Optimal Policy

We know the expected utility of an action.

- So, to get the optimal policy, compute

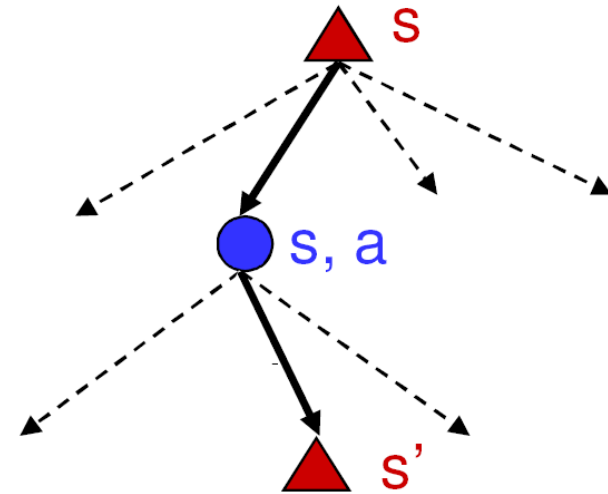
$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) V^*(s')$$



All the states we
could go to

Transition
probability

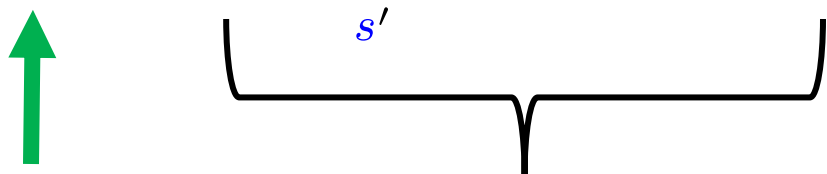
Expected
rewards



Credit L. Lazbenik

Bellman Equation

Let's walk over one step for the value function:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V^*(s')$$


Current state
reward

Discounted expected
future **rewards**

- Bellman: inventor of dynamic programming



Value Iteration

Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s' | s, a)$
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0$. Then, update

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s' | s, a) V_i(s')$$

Q-Learning

What if we don't know transition probability $P(s' | s, a)$?

- Need a way to learn to act without it.
- **Q-learning**: get an action-utility function $Q(s, a)$ that tells us the value of doing a in state s
- Note: $V^*(s) = \max_a Q(s, a)$
- Now, we can just do $\pi^*(s) = \arg \max_a Q(s, a)$
 - But need to estimate Q !



Q-Learning Iteration

How do we get $Q(s, a)$?

- Similar iterative procedure

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Learning rate



Idea: combine old value and new estimate of future value.

Note: We are using a policy to take actions; based on Q!

Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
 - **Pros:**
 - Get a more accurate model of the environment
 - Discover higher-reward states than the ones found so far
 - **Cons:**
 - When exploring, not maximizing your utility
 - Something bad might happen
- **Exploitation:** go with the best strategy found so far
 - **Pros:**
 - Maximize reward as reflected in the current utility estimates
 - Avoid bad stuff
 - **Cons:**
 - Might also prevent you from discovering the true optimal strategy

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

- With some $0 < \epsilon < 1$ probability, take a random action at each state, or else the action with highest $Q(s, a)$ value.

$$a = \begin{cases} \operatorname{argmax}_{a \in A} Q(s, a) & \text{uniform}(0, 1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

Q-Learning: SARSA

An alternative:

- Just use the next action, no max over actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate

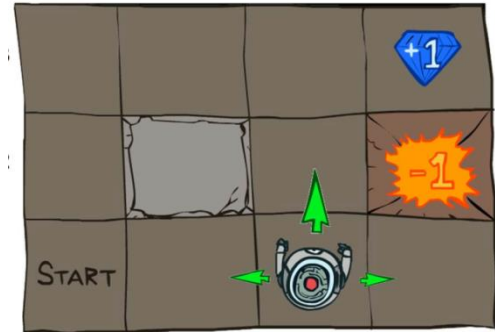


- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Q-Learning Details

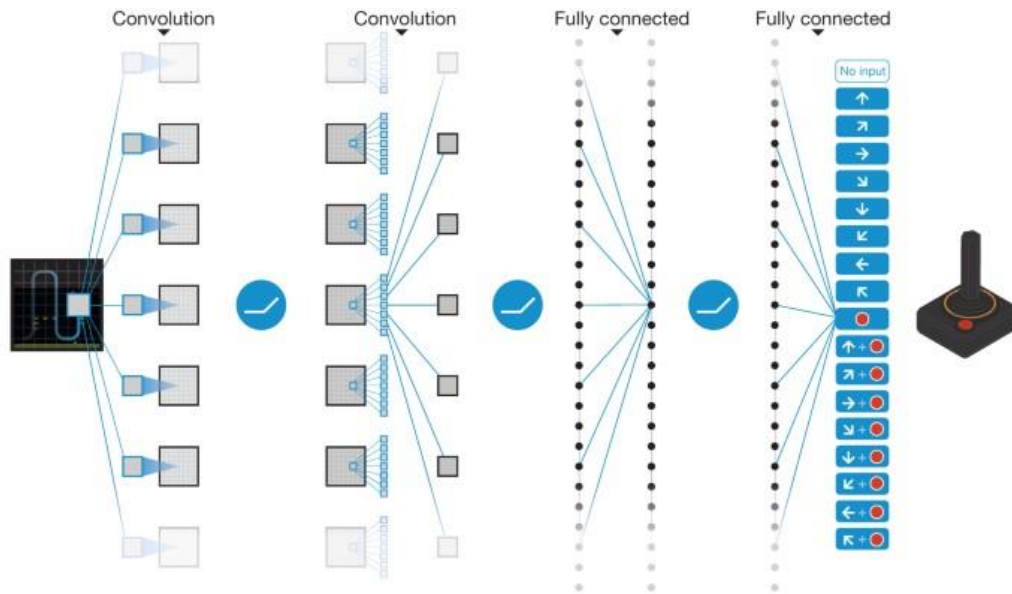
Note: if we have a **terminal** state, the process ends

- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states (see homework!)



Deep Q-Learning

How do we get $Q(s, a)$?



Mnih et al, "Human-level control through deep reinforcement learning"

Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning

Search and RL Review

- Search
 - Uninformed vs Informed
 - Optimization
- Games
 - Minimax search
- Reinforcement Learning
 - MDPs, value iteration, Q-learning

Uninformed vs Informed Search

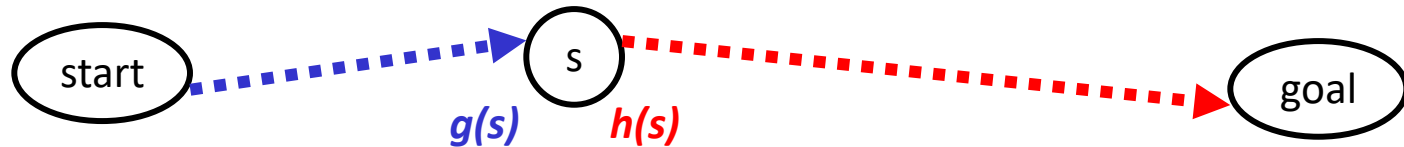
Uninformed search (all of what we saw). Know:

- Path cost $g(s)$ from start to node s
- Successors.



Informed search. Know:

- All uninformed search properties, plus
- Heuristic $h(s)$ from s to goal (recall game heuristic)

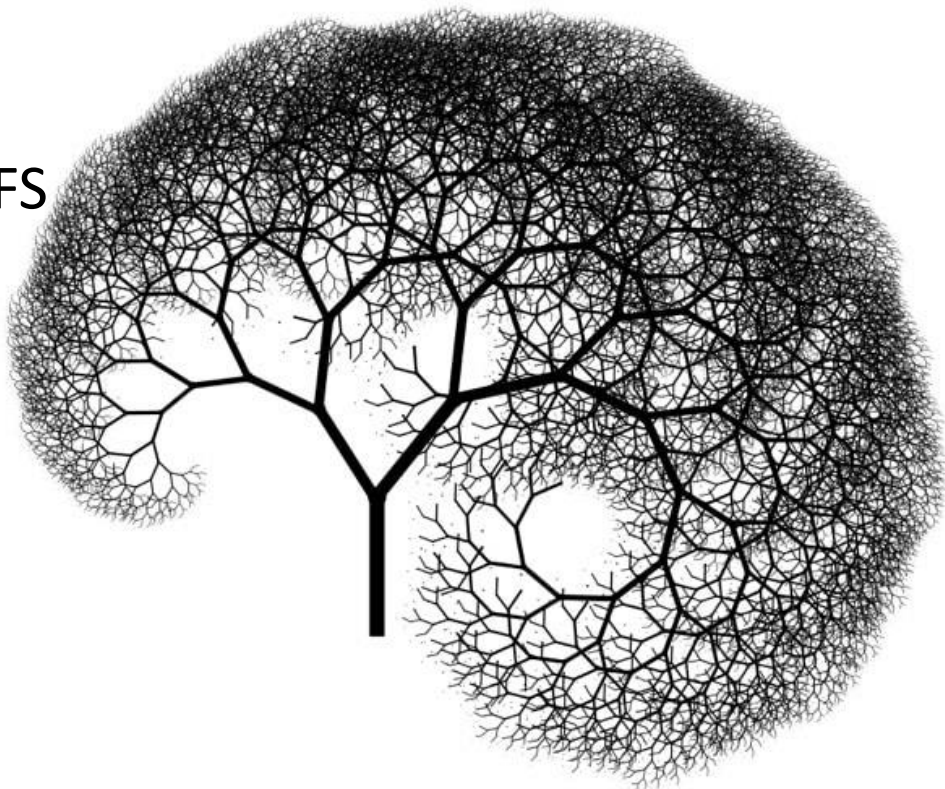


Uninformed Search: Iterative Deepening DFS

Repeated limited DFS

- Search like BFS, fringe like DFS
- **Properties:**
 - Complete
 - Optimal (if edge cost 1)
 - Time $O(b^d)$
 - Space $O(bd)$

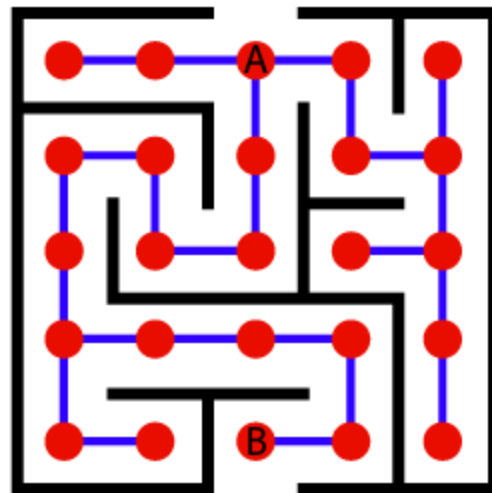
A good option!



Informed Search: A* Search

A*: Expand best $g(s) + h(s)$, with one requirement

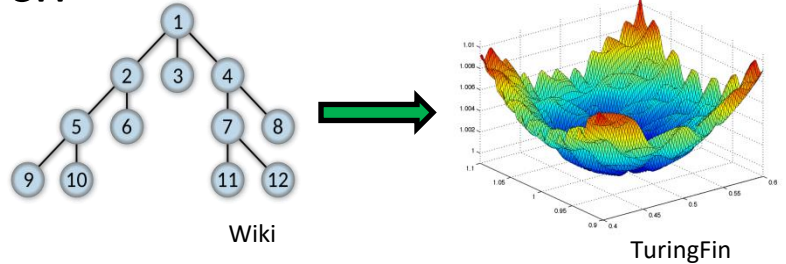
- Demand that $h(s) \leq h^*(s)$
- If heuristic has this property, “admissible”
 - Optimistic! Never over-estimates
- Still need $h(s) \geq 0$
 - Negative heuristics can lead to strange behavior



Search vs. Optimization

Before: wanted a path from start state to goal state

- Uninformed search, informed search



New setting: optimization

- States s have values $f(s)$
- Want: s with optimal value $f(s)$ (i.e, **optimize** over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a continuous function for SGD.

Hill Climbing Algorithm

Pseudocode:

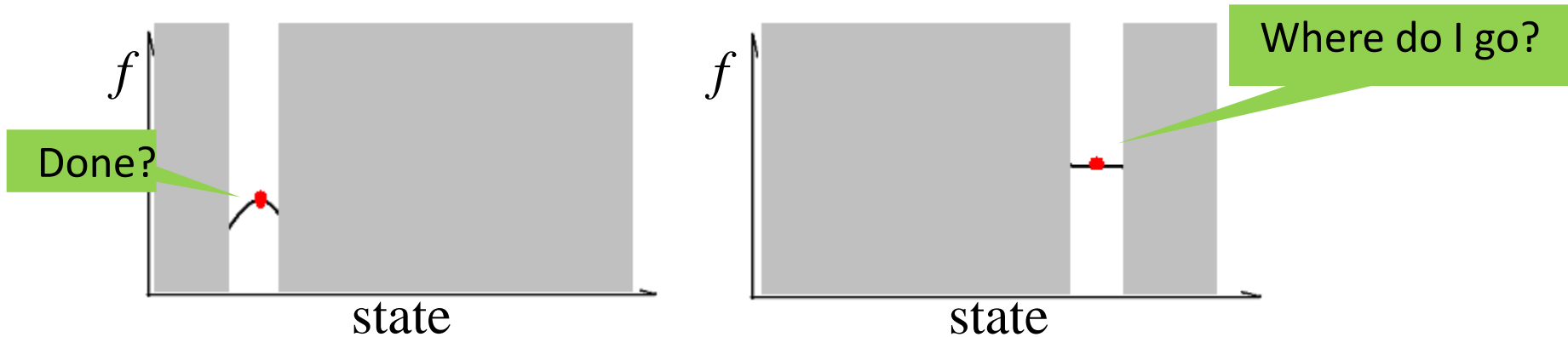
1. Pick initial state s
2. Pick t in **neighbors**(s) with the largest $f(t)$
3. if $f(t) \leq f(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**



Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?



Simulated Annealing

A more sophisticated optimization approach.

- **Idea:** move quickly at first, then slow down
- Pseudocode:

Pick initial state s

For $k = 0$ through k_{\max} :

$T \leftarrow \text{temperature}((k+1)/k_{\max})$

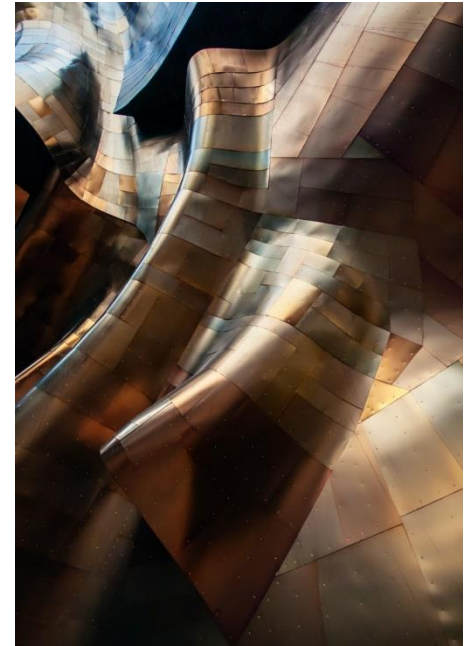
Pick a random neighbour, $t \leftarrow \text{neighbor}(s)$

If $f(s) \leq f(t)$, then $s \leftarrow t$

Else, with prob. $P(f(s), f(t), T)$ then $s \leftarrow t$

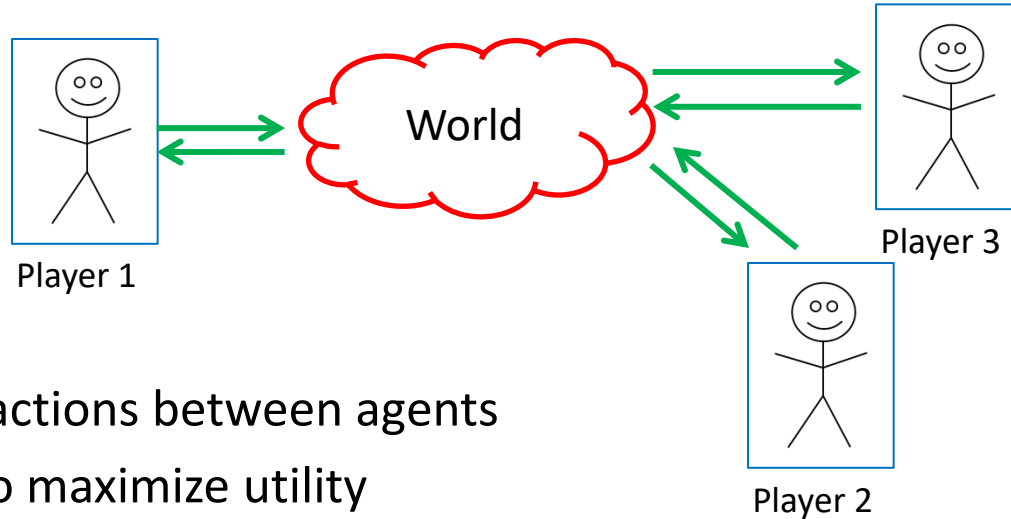
Output: the final state s

The interesting bit



Games Setup

Games setup: **multiple** agents



- Now: interactions between agents
- Still want to maximize utility
- **Strategic** decision making.

Minimax Search

Note that long games are yield huge computation

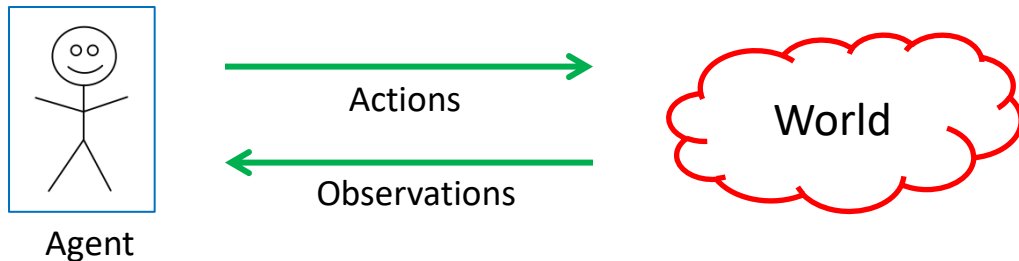
- To deal with this: limit d for the search depth
- **Q:** What to do at depth d , but no termination yet?
 - **A:** Use a heuristic evaluation function $e(x)$

```
function MINIMAX( $x, d$ ) returns an estimate of  $x$ 's utility value
  inputs:  $x$ , current state in game
            $d$ , an upper bound on the search depth
  if  $x$  is a terminal state then return Max's payoff at  $x$ 
  else if  $d = 0$  then return  $e(x)$ 
  else if it is Max's move at  $x$  then
    return  $\max\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$ 
  else return  $\min\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$ 
```

Building The Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A
- Information: at time t , observe state $s_t \in S$. Get reward r_t
- Agent makes choice $a_t \in A$. State changes to s_{t+1} , continue



Goal: find a map from **states to actions** maximize rewards.

↑
A “policy”



Acknowledgements: Based on slides from Yin Li, Jerry Zhu, Svetlana Lazebnik, Yingyu Liang, David Page, Mark Craven, Pieter Abbeel, Dan Klein