CS 540 Introduction to Artificial Intelligence
Reinforcement Learning I

University of Wisconsin-Madison
Spring 2026 Sections 1 & 2

Announcements

- **Homework**:
  - HW9 online, due on Wednesday April 22nd at 11:59PM

- Class roadmap:

Introduction to Reinforcement Learning
Reinforcement Learning II

Outline

- Review/Complete Sequential-move games
- Introduction to reinforcement learning
  - Basic concepts, mathematical formulation, MDPs, policies.
- Learning policies
  - Q-learning, action-values, exploration vs exploitation.

Our Approach So Far

We find the minimax value/strategy bottom up

- Minimax value: score of terminal node when both players play optimally
  - **Max's** turn, take max of children
  - **Min's** turn, take min of children

- Can implement this as depth-first search: **minimax algorithm**

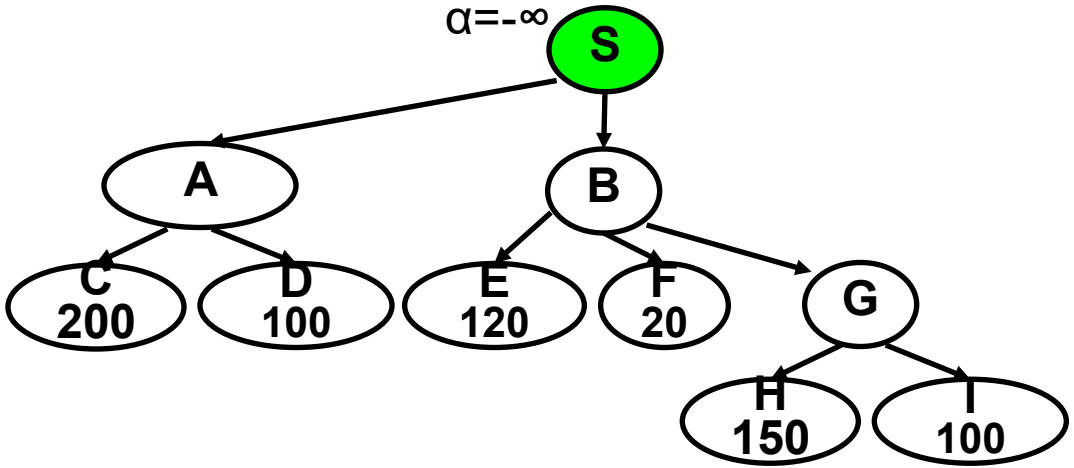
Minimax algorithm in execution

max

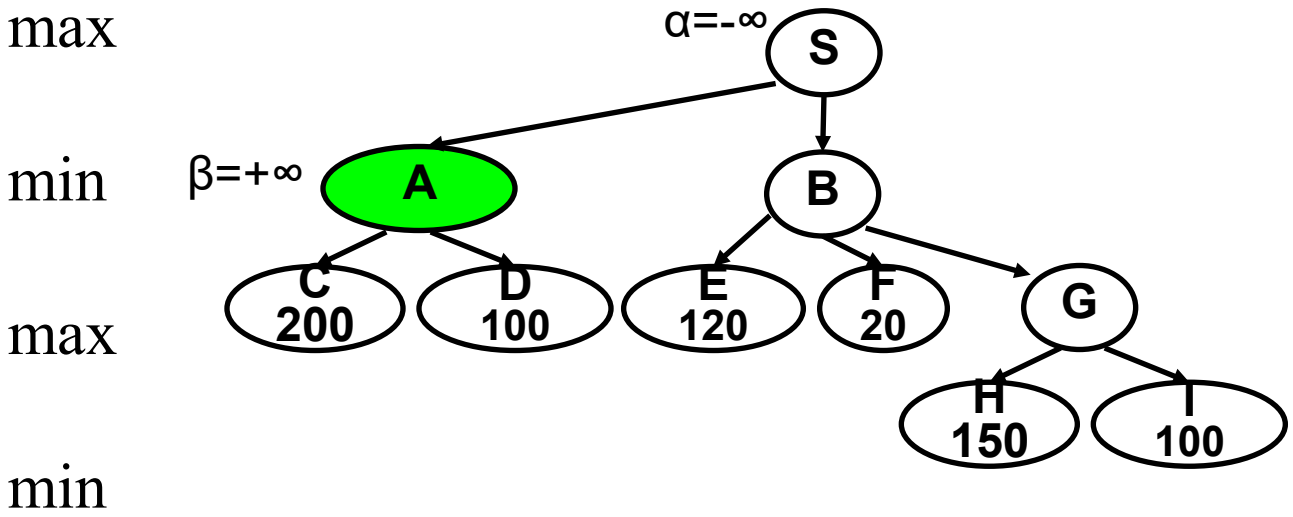
min

max

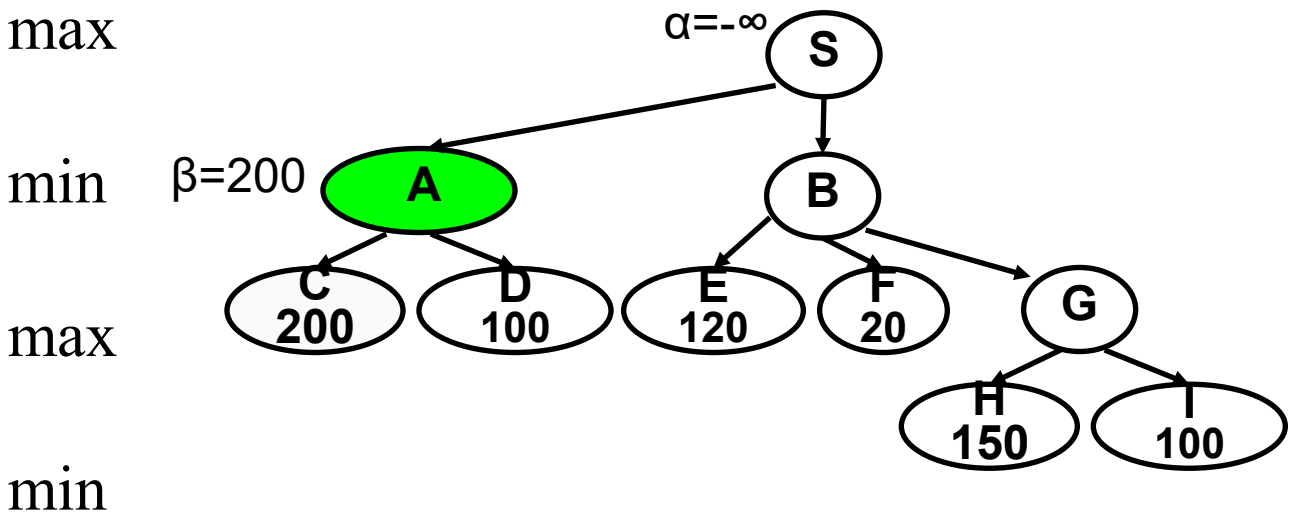
min



Minimax algorithm in execution

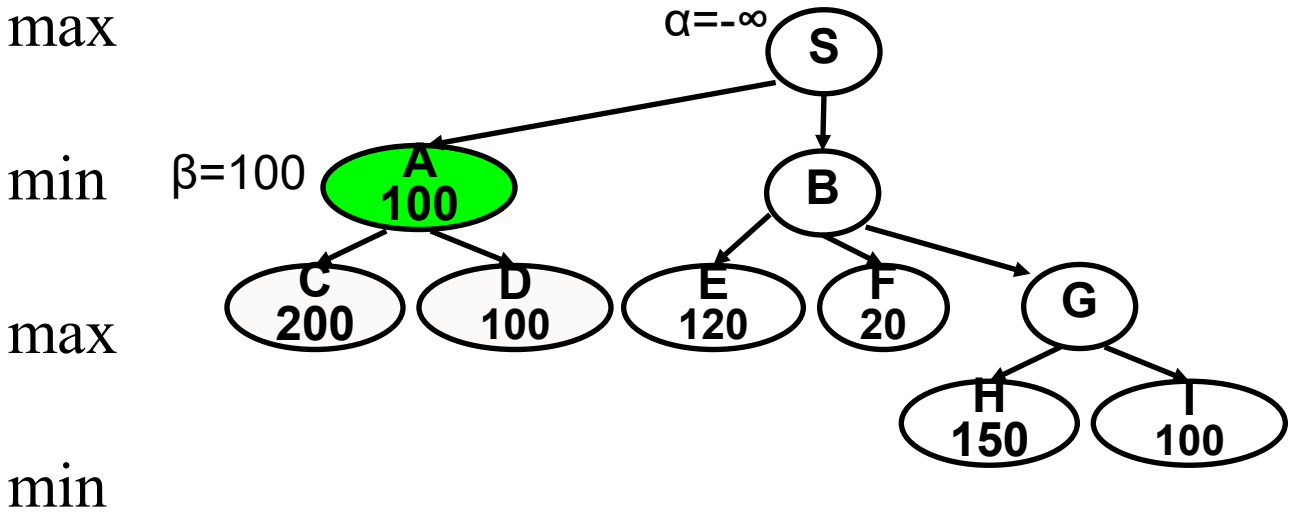


Minimax algorithm in execution

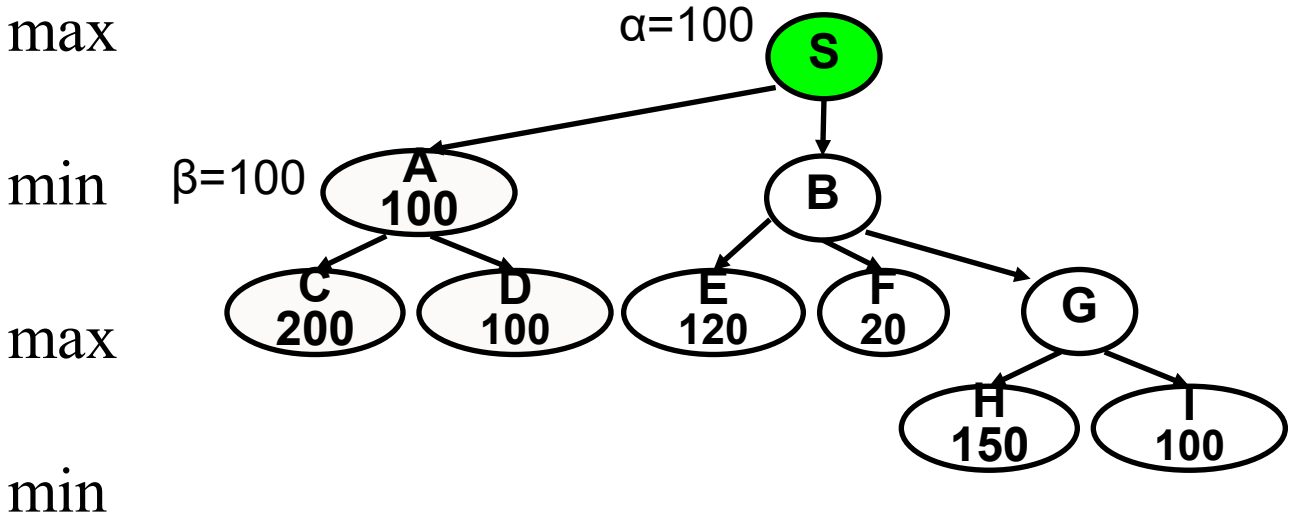


The execution on the terminal nodes is omitted.

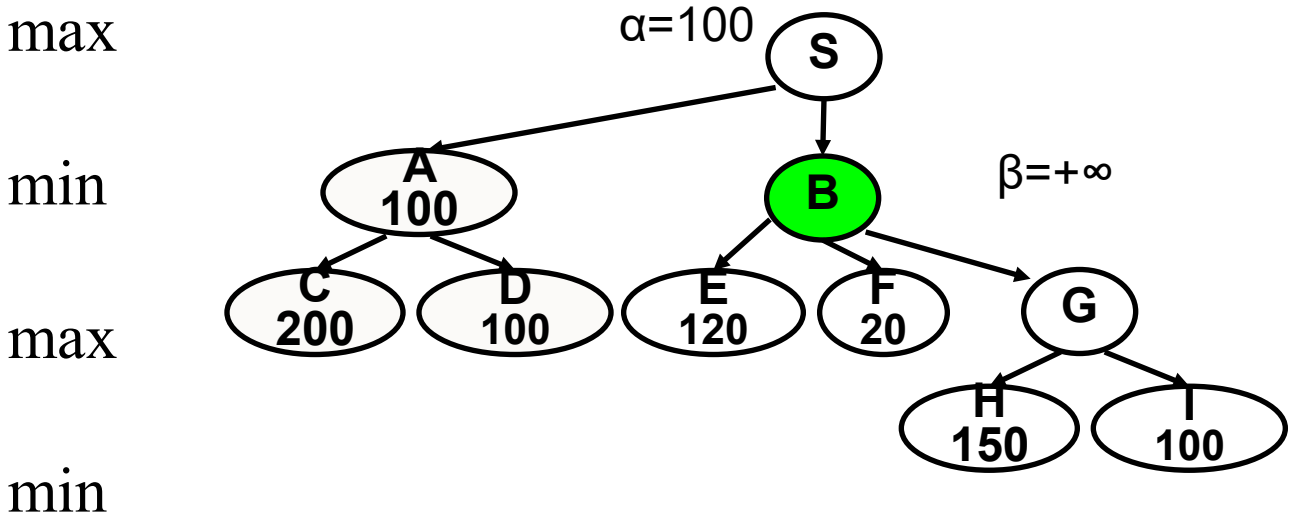
Minimax algorithm in execution



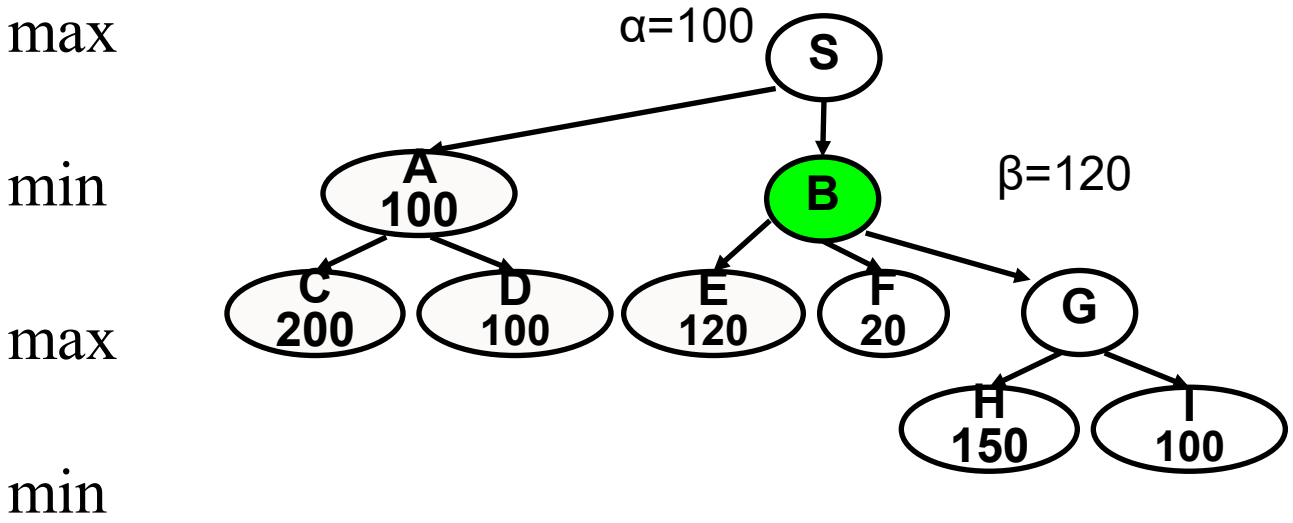
Minimax algorithm in execution



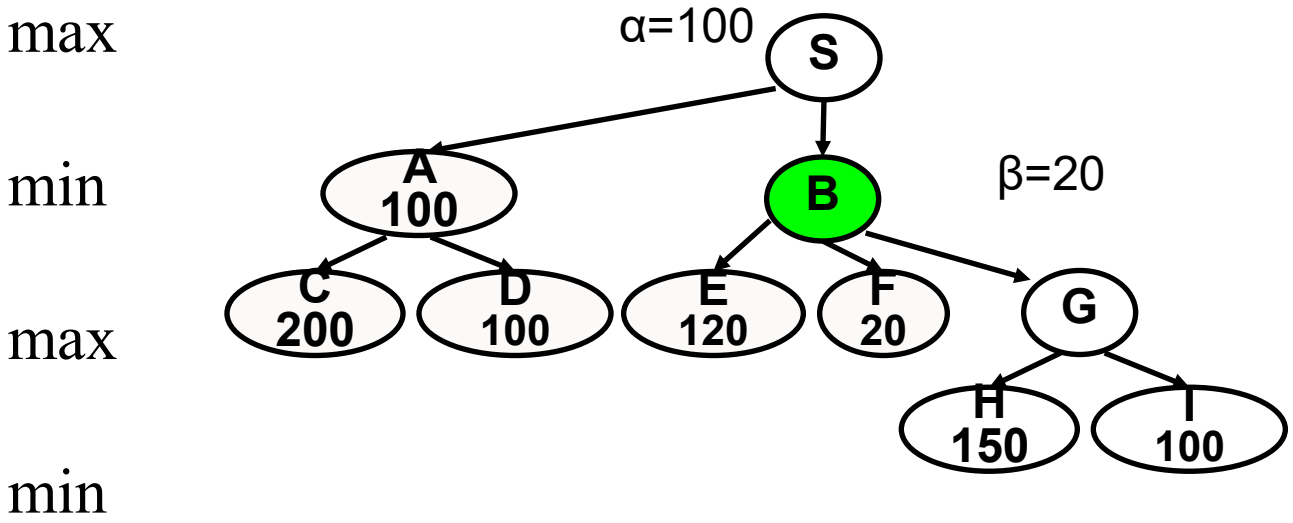
Minimax algorithm in execution



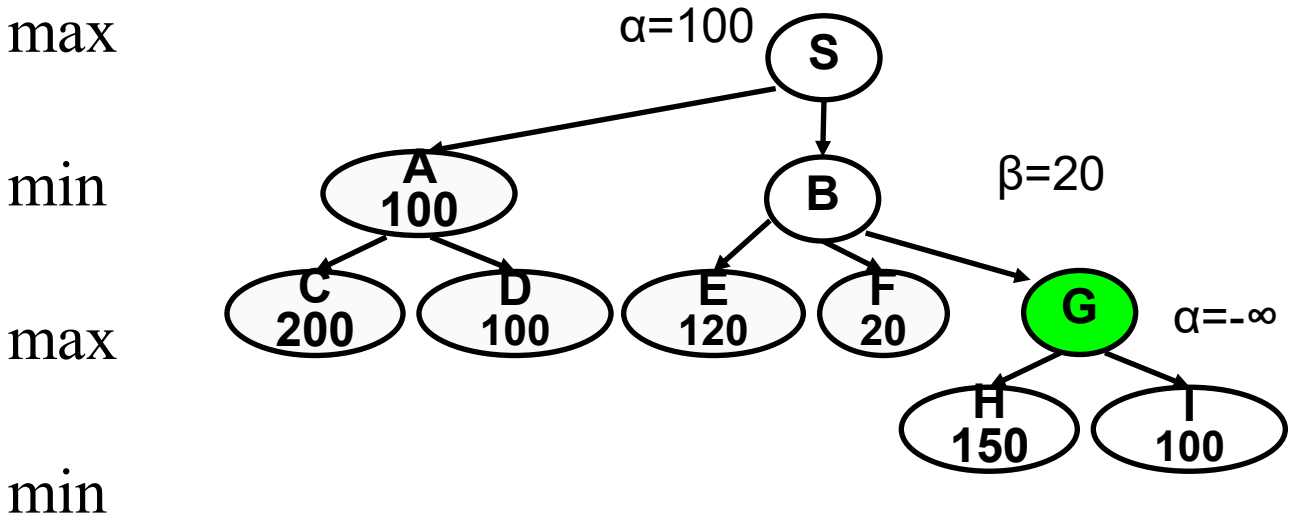
Minimax algorithm in execution



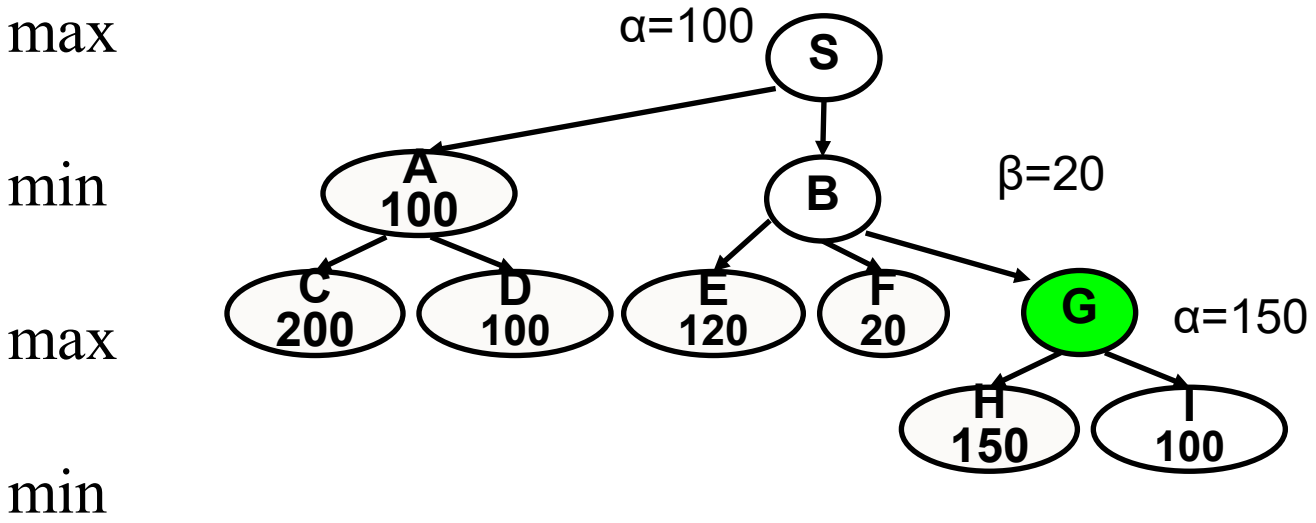
Minimax algorithm in execution



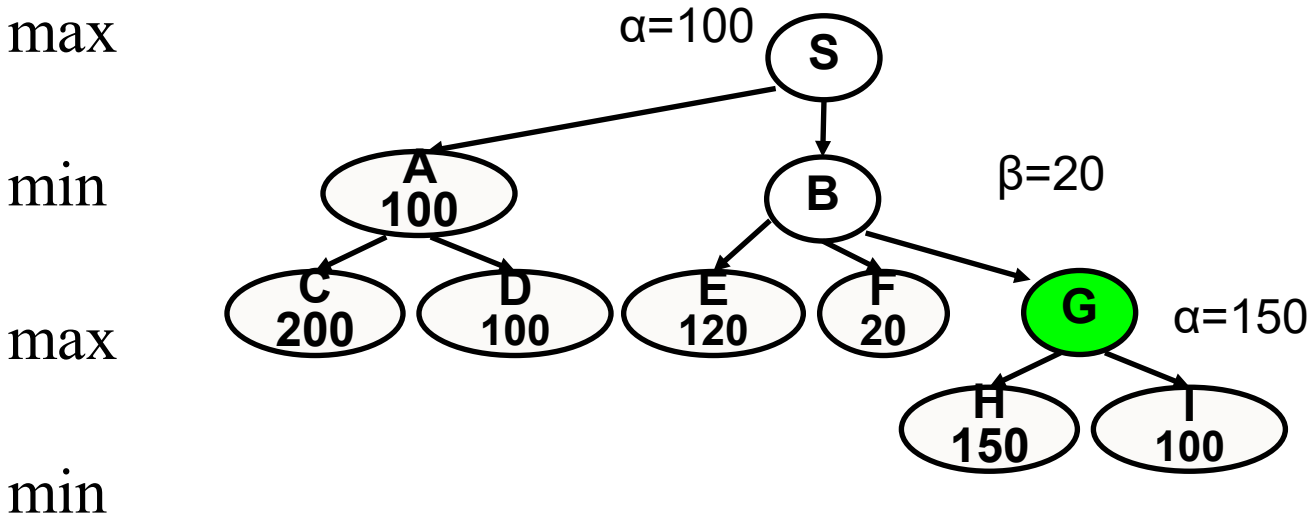
Minimax algorithm in execution



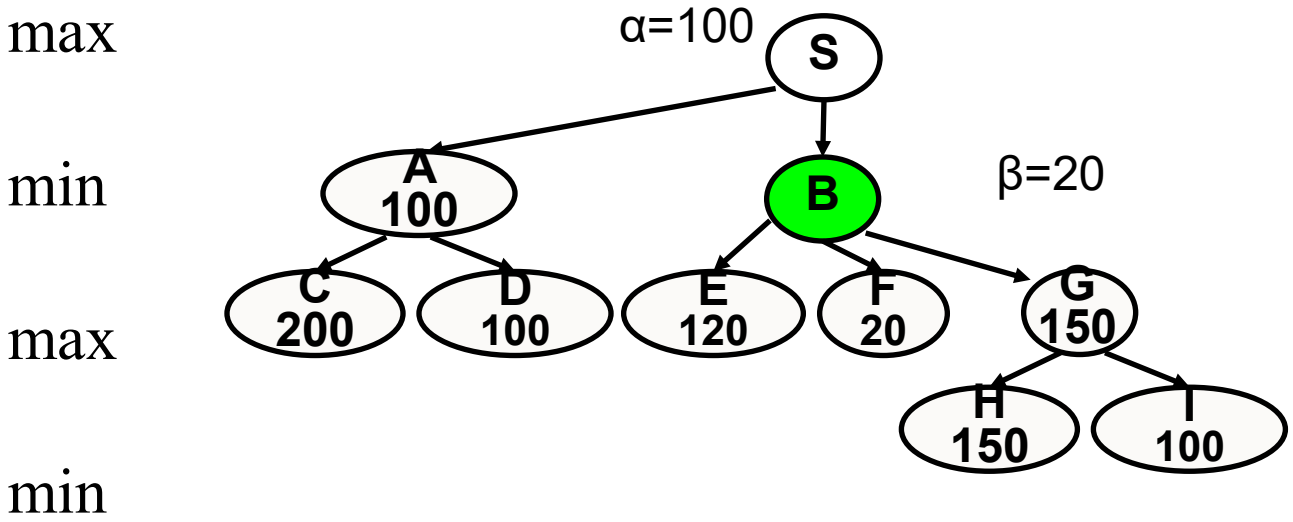
Minimax algorithm in execution



Minimax algorithm in execution



Minimax algorithm in execution



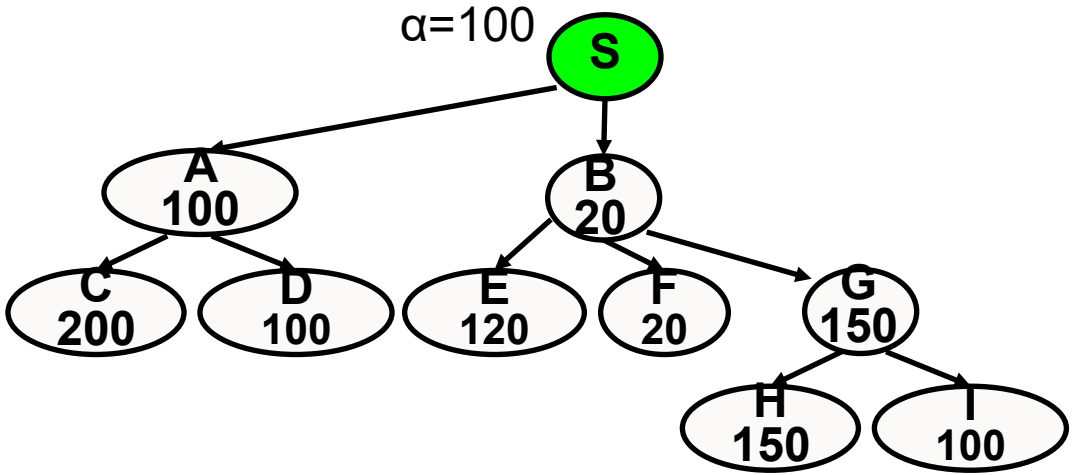
Minimax algorithm in execution

max

min

max

min



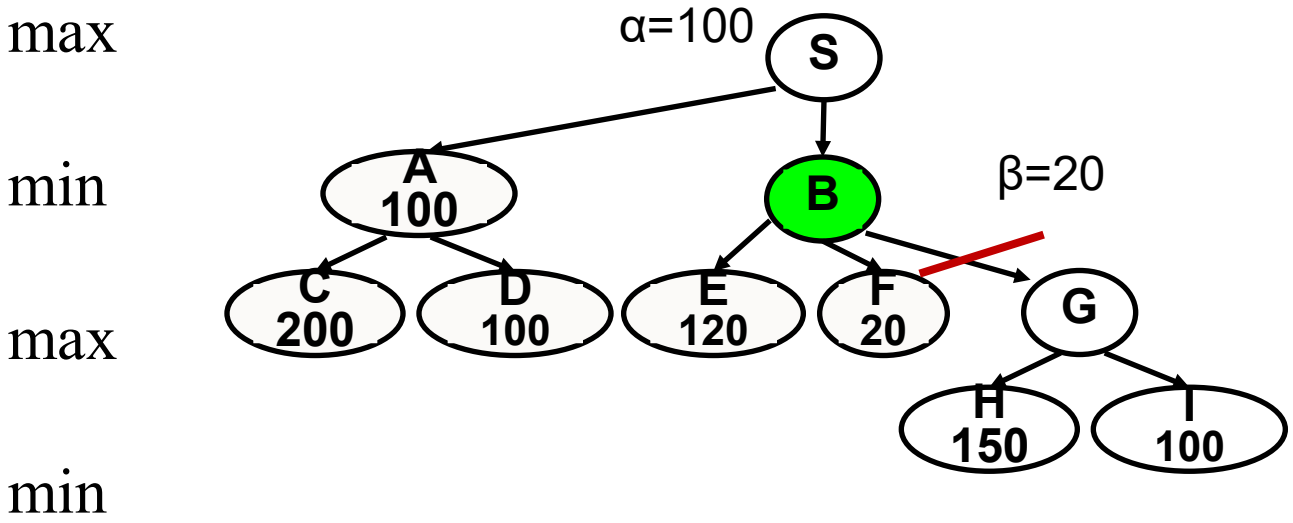
Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster
- We can get rid of bad branches
- Same principle as quiz question

Minimax algorithm in execution



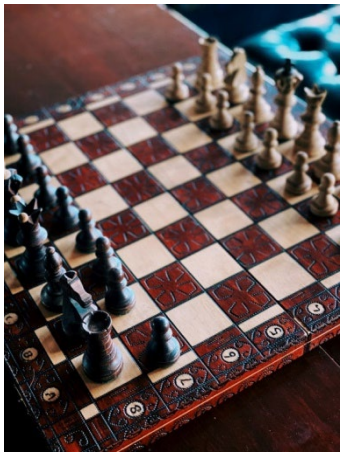
Alpha-beta pruning

```
function Max-Value (s,α,β)
inputs:
      s: current state in game, Max about to play
      α: best score (highest) for Max along path to s
      β: best score (lowest) for Min along path to s
output: min(β , best-score (for Max) available from s)

      if ( s is a terminal state )
      then return ( terminal value of s )
      else for each s' in Succ(s)
         α := max( α , Min-value(s',α,β))
         if ( α ≥ β ) then return β   /* alpha pruning */
      return α
```

```
function Min-Value(s,α,β)
output: max(α , best-score (for Min) available from s )

      if ( s is a terminal state )
      then return ( terminal value of s)
      else for each s' in Succs(s)
         β := min( β , Max-value(s',α,β))
         if (α ≥ β ) then return α   /* beta pruning */
      return β
```

Starting from the root:
Max-Value(root, -∞, +∞)

Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!
  - Best case, the #of nodes to search is O($b^{m/2}$)
  - Happens when each player's best move is the leftmost child.
  - The worst case is no pruning at all.

- In DeepBlue, the average branching factor was about 6 with alpha-beta instead of 35-40 without.

Minimax With Heuristics

Note that long games may require 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{MINIMAX(y, d−1) : y is a child of x}
   else return min{MINIMAX(y, d−1) : y is a child of x}
```

Credit: Dana Nau

Heuristic Evaluation Functions

- e(x) can be any computable function of x; e.g. a weighted sum of features (like our linear models)

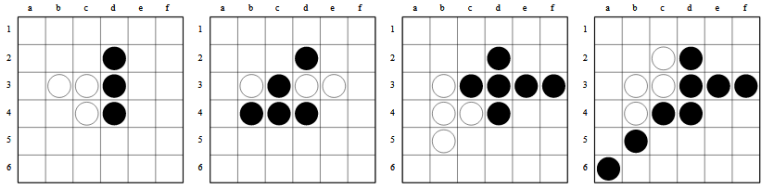
$$e(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$$

- Chess example: $f_i(x)$ = **difference** between number of white and black, with *i* ranging over piece types.
  - Set weights according to piece importance
  - E.g., 1(# white pawns - # black pawns) + 3(#white knights - # black knights)

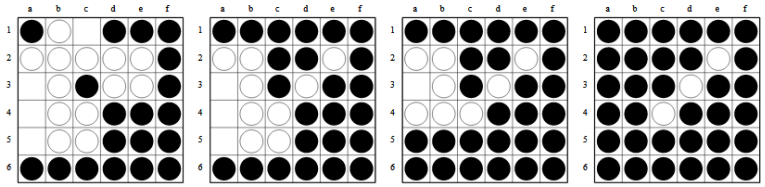
Going Further

- Monte Carlo tree search (MCTS)
  - Uses random sampling of the search space
  - Choose some children (heuristics to figure out #)
  - Record results, use for future play
  - Self-play

- AlphaGo and other big results!



The agent (Black) learns to capture walls and corners in the early game

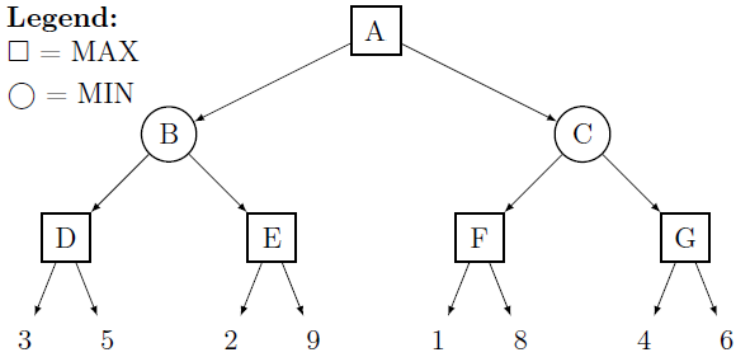


The agent (Black) learns to force passes in the late game

Credit: Surag Nair

Break & Quiz

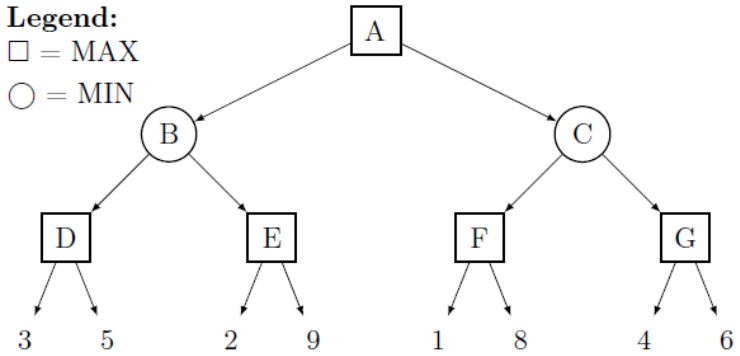
Q 1.1 Consider the game tree shown in the figure below. The root node (A) represents the current state of the game where it is the MAX player's turn.
Assuming both players play optimally according to the Minimax algorithm, what is the value of the root node A?



A. 4
B. 5
C. 6
D. 8

Break & Quiz

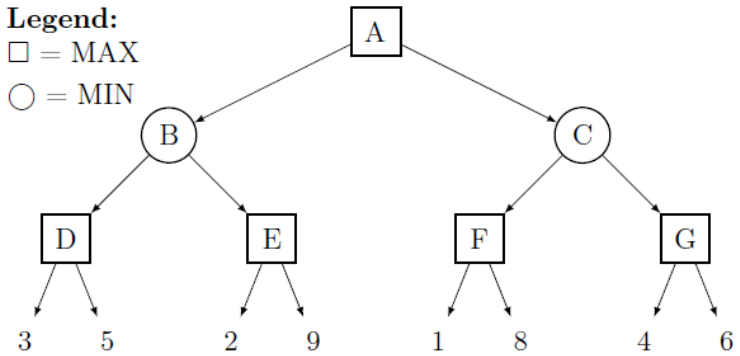
Q 1.1 Consider the game tree shown in the figure below. The root node (A) represents the current state of the game where it is the MAX player's turn.
Assuming both players play optimally according to the Minimax algorithm, what is the value of the root node A?



A. 4
B. 5
C. 6
D. 8

Break & Quiz

Q 1.1 Consider the game tree shown in the figure below. The root node (A) represents the current state of the game where it is the MAX player's turn.
Assuming both players play optimally according to the Minimax algorithm, what is the value of the root node A?



A. 4
B. 5
C. 6
D. 8

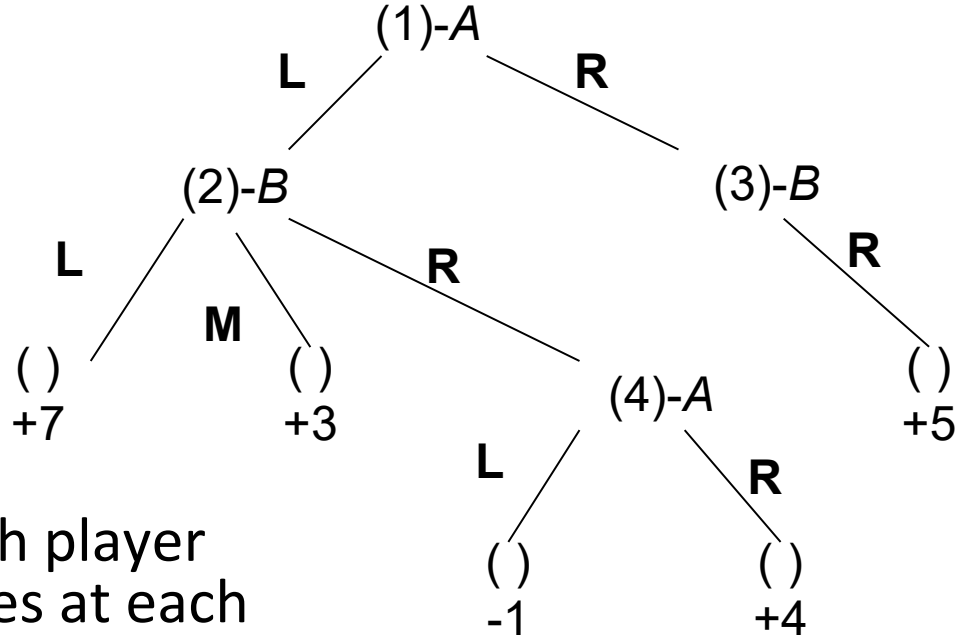
The Minimax algorithm propagates values from the leaf nodes up to the root.
1. Level 2 (MAX Nodes): The MAX player chooses the highest value among the children.
- Node D: max(3, 5) = 5
- Node E: max(2, 9) = 9
- Node F: max(1, 8) = 8
- Node G: max(4, 6) = 6

2. Level 1 (MIN Nodes): The MIN player chooses the lowest value among the children passed up from Level 2.
- Node B: min(D,E) = min(5, 9) = 5
- Node C: min(F,G) = min(8, 6) = 6

3. Level 0 (Root Node): The MAX player chooses the highest value among the children passed up from Level 1.
- Node A: max(B,C) = max(5, 6) = 6

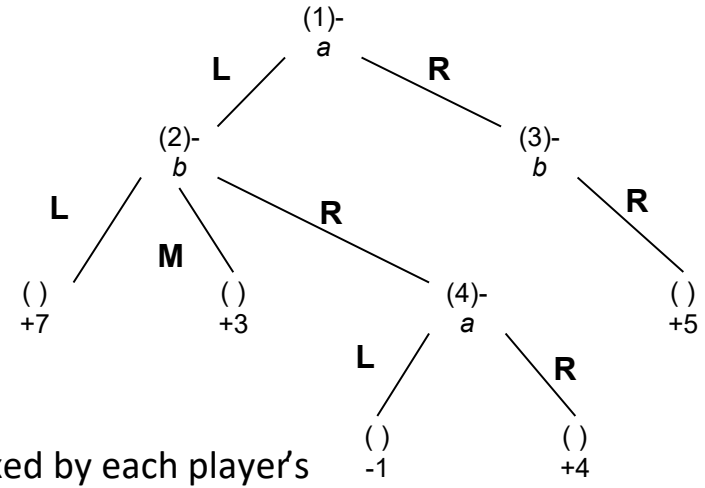
From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:
  - A's strategy I: (1→L, 4→L)
  - A's strategy II: (1→L, 4→R)
  - A's strategy III: (1→R, 4→L)
  - A's strategy IV: (1→R, 4→R)
- Player B has 3 pure strategies:
  - B's strategy I: (2→L, 3→R)
  - B's strategy II: (2→M, 3→R)
  - B's strategy III: (2→R, 3→R)
- How many pure strategies if each player can see *N* states, and has *b* moves at each state?



Matrix Normal Form of games

A's strategy I: (1→L, 4→L)
A's strategy II: (1→L, 4→R)
A's strategy III: (1→R, 4→L)
A's strategy IV: (1→R, 4→R)
B's strategy I: (2→L, 3→R)
B's strategy II: (2→M, 3→R)
B's strategy III: (2→R, 3→R)

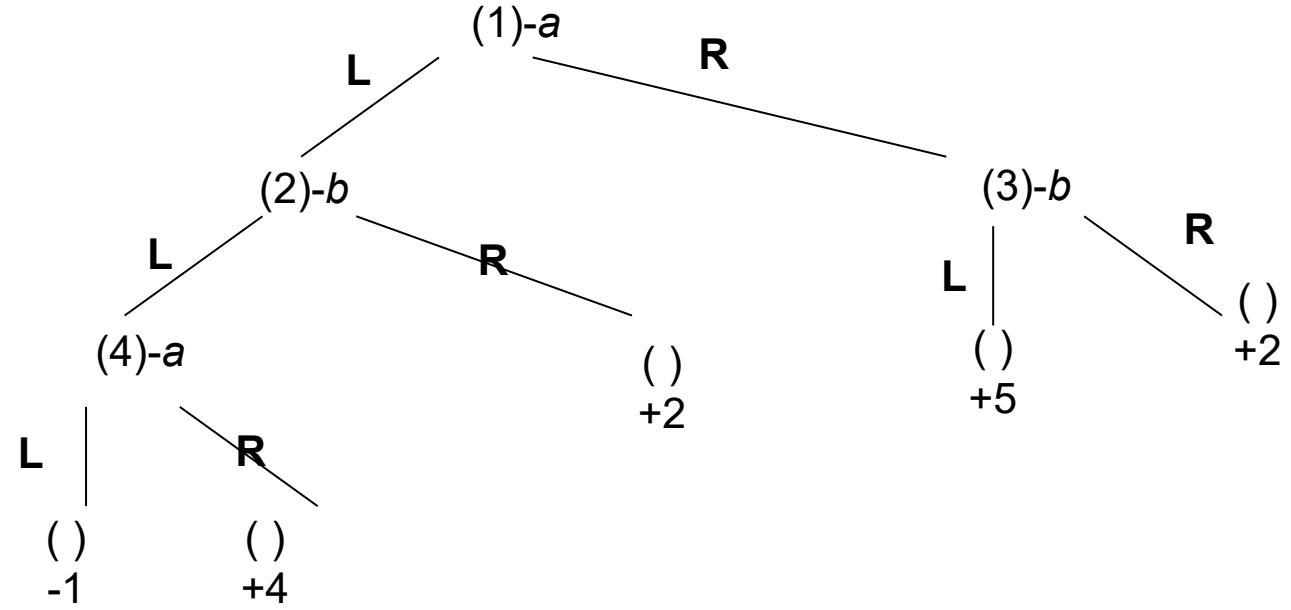


The matrix normal form is the game value matrix indexed by each player's strategies.

	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

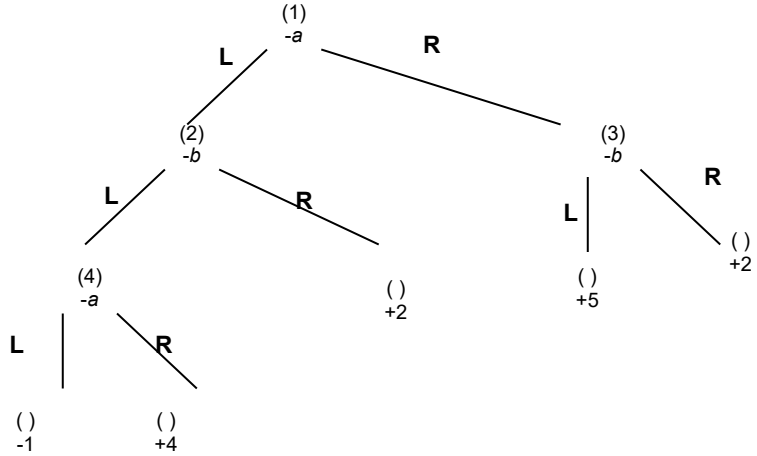
The matrix encodes every outcome of the game! The rules etc. are no longer needed.

Another example of normal form



- How many pure strategies does A have?
- How many does B have?
- What is the matrix form of this game?

Matrix normal form example



	B-I	B-II	B-III	B-IV
A-I	-1	-1	2	2
A-II	4	4	2	2
A-III	5	2	5	2
A-IV	5	2	5	2

- How many pure strategies does A have? 4

A-I (1→L, 4→L) A-II (1→L,4→R) A-III (1→R,4→L) A-IV (1→R, 4→R)

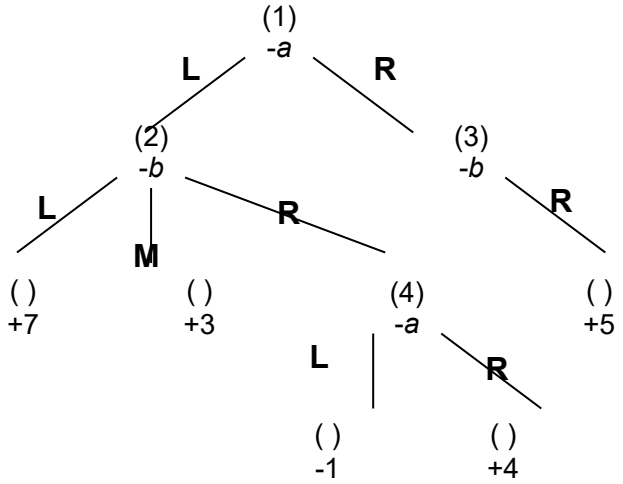
- How many does B have? 4

B-I (2→L, 3→L) B-II (2→L,3→R) B-III (2→R,3→L) B-IV (2→R, 3→R)

- What is the matrix form of this game?

Minimax in Matrix Normal Form

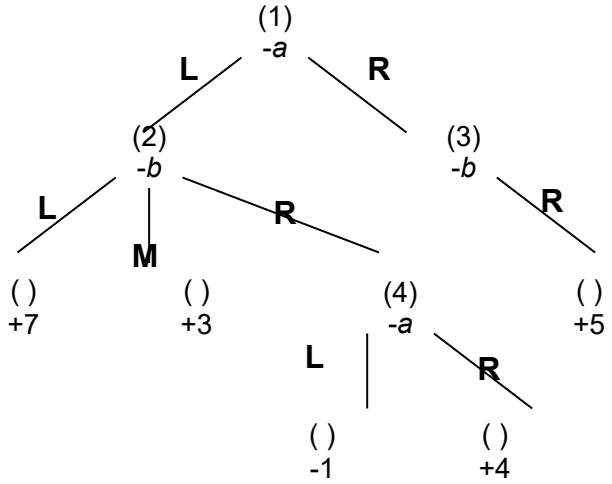
- Player A: for each strategy, consider all B's counter strategies (a row in the matrix), find the minimum value in that row. Pick the row with the maximum minimum value.
- Here maximin=5



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

- Player B: find the maximum value in each column. Pick the column with the minimum maximum value.
- Here minimax = 5



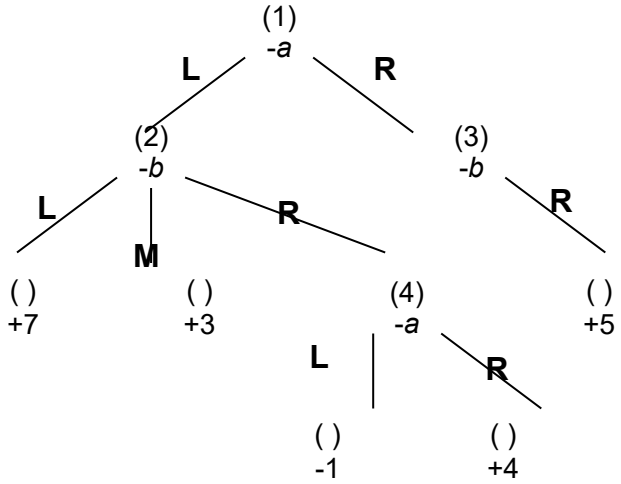
Fundamental game theory result (proved by von Neumann):

In a 2-player, zero-sum game of perfect information (sequential moves), Minimax==Maximin. And there always exists an optimal pure strategy for each player.

	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

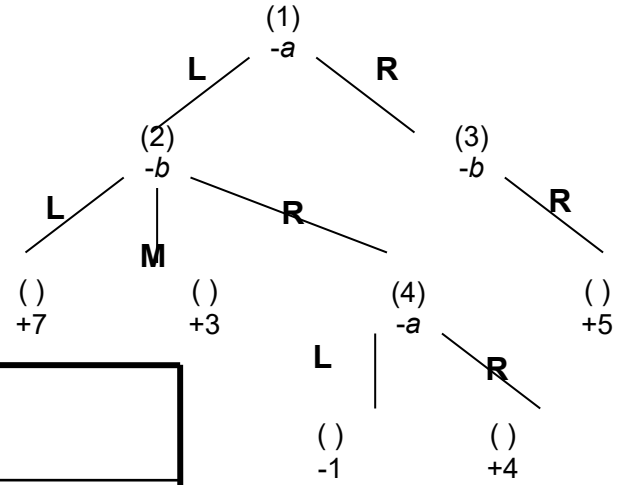
Interestingly, A can tell B in advance what strategy A will use (the maximin), and this information will not help B!
Similarly B can tell A what strategy B will use.
In fact A knows what B's strategy will be.
And B knows A's too.
And A knows that B knows
…
The game is at an equilibrium



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

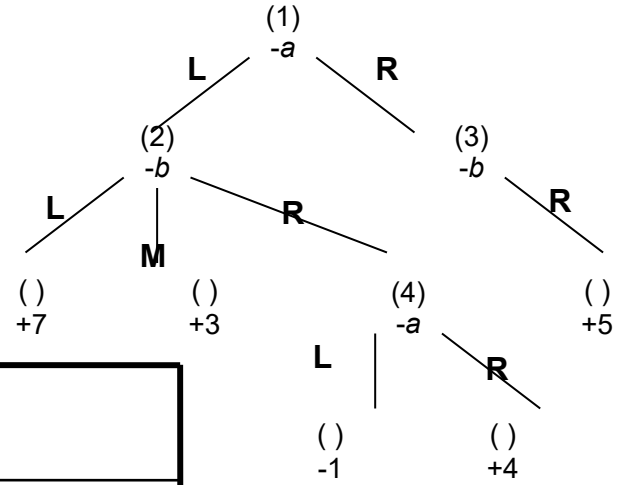
- We can also check for mutual best responses



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

- We can also check for mutual best responses



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Suggested Readings

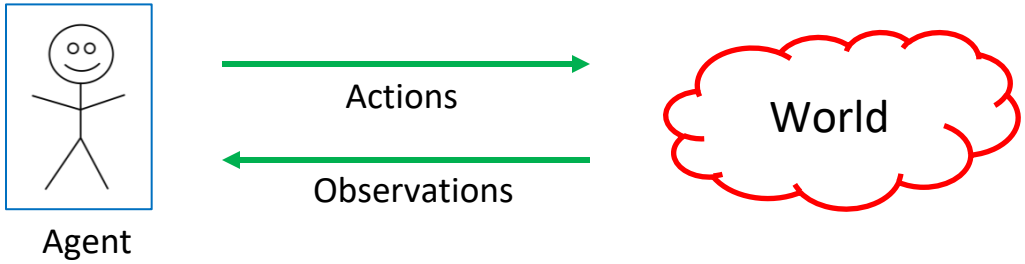
Textbook: Artificial Intelligence: A Modern Approach (4th edition). Stuart Russell and Peter Norvig. Pearson, 2020.

- Chapters 5, 18

Reinforcement Learning

Back to Our General Model

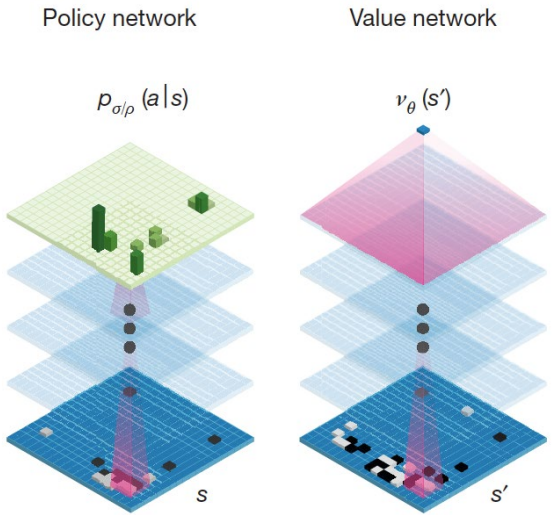
We have an **agent** **interacting** with the **world**



- Agent receives a reward based on state of the world
  - **Goal**: maximize reward / utility **($$$)**
  - Note: **data** consists of actions & observations
    - Compare to unsupervised learning and supervised learning

Examples: Gameplay Agents

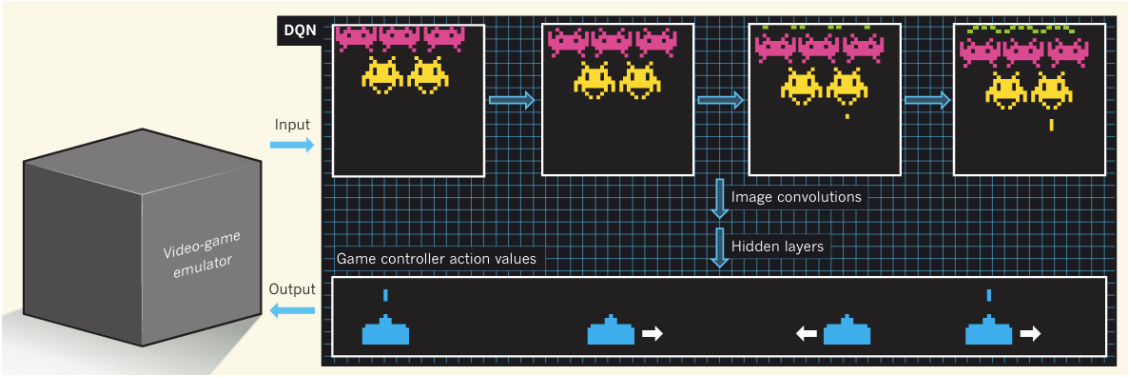
AlphaZero:



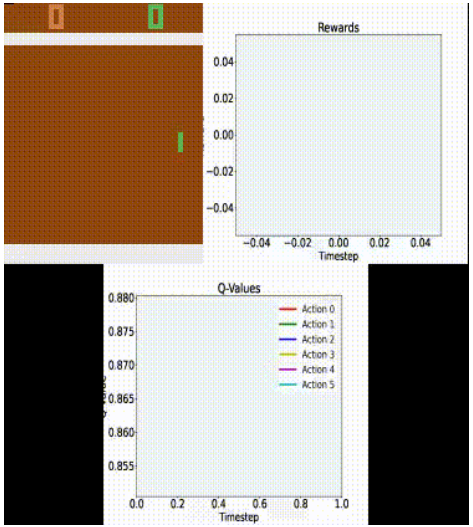
https://deepmind.com/research/alphago/

Examples: Video Game Agents

Pong, Atari



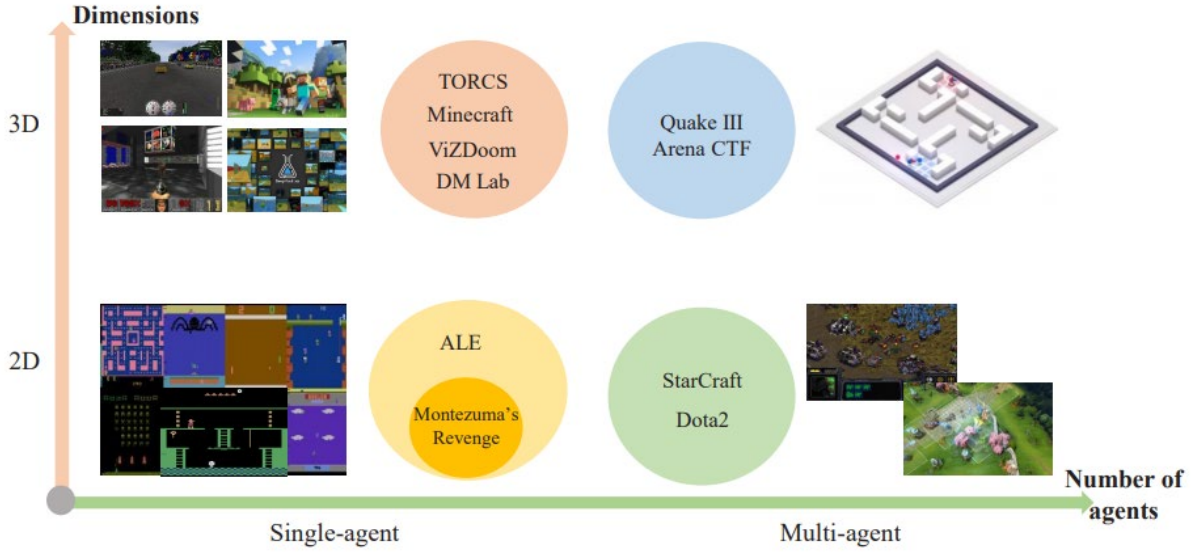
Mnih et al, “Human-level control through deep reinforcement learning”



A. Nielsen

Examples: Video Game Agents

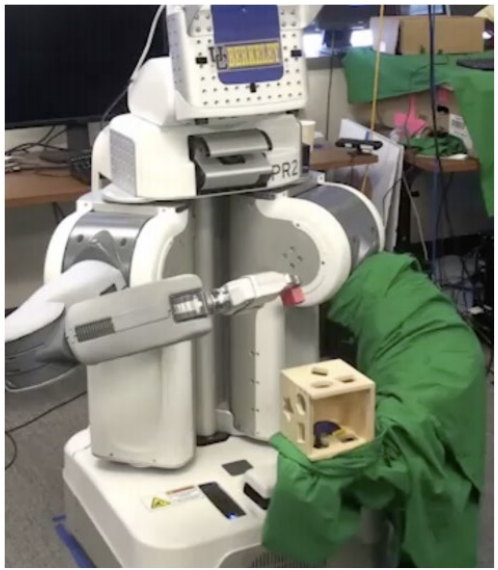
Minecraft, Quake, StarCraft, and more!



Shao et al, "A Survey of Deep Reinforcement Learning in Video Games"

Examples: Robotics

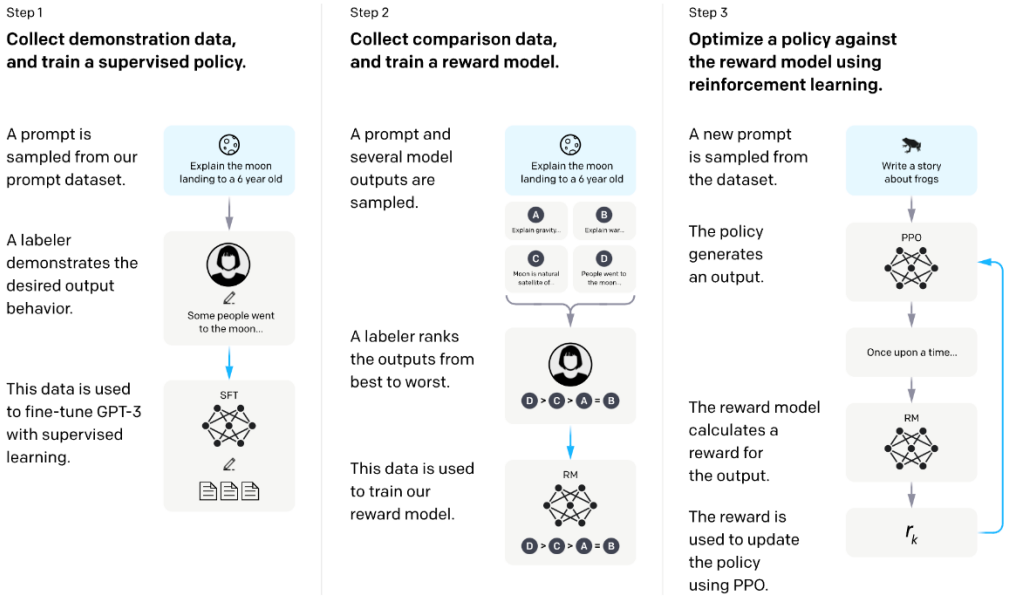
Training robots to perform tasks (e.g., grasp objects!)



Ibarz et al, " How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned "

Examples: Large Language Models

RL used to “align” model outputs to human preferences



C. Huyen, https://huyenchip.com/2023/05/02/rlhf.html

Unaligned Model

There are many ways to cheat in poker, including using marked cards, collusion, and using electronic devices to track other players' cards.

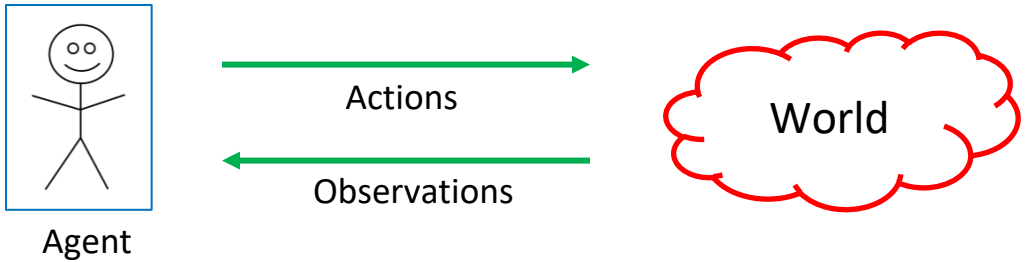
Aligned Model Post-RLHF

Cheating in poker is illegal and unethical, and can result in penalties such as fines or expulsion from the game. There here are many ways to win at poker legitimately, such as developing a strong strategy, practicing, and analyzing your opponents' behavior.

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** that maximize rewards.

A “policy”

Markov Decision Process (MDP)

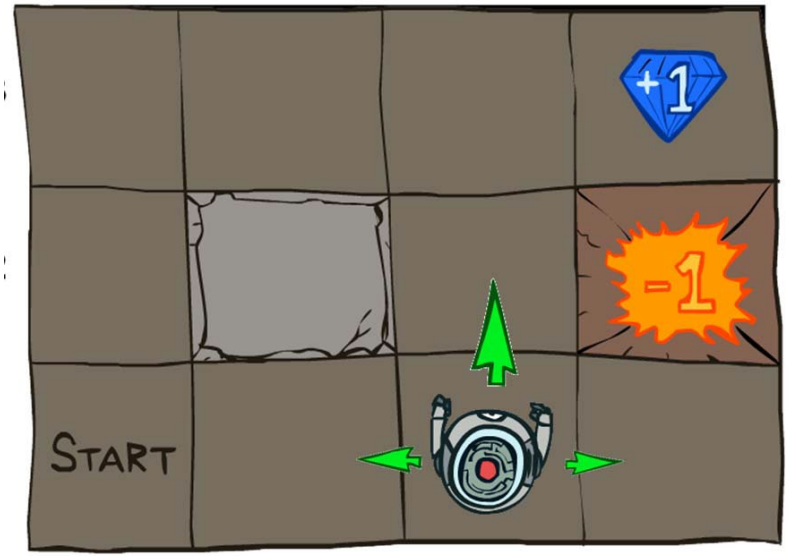
The formal mathematical model:

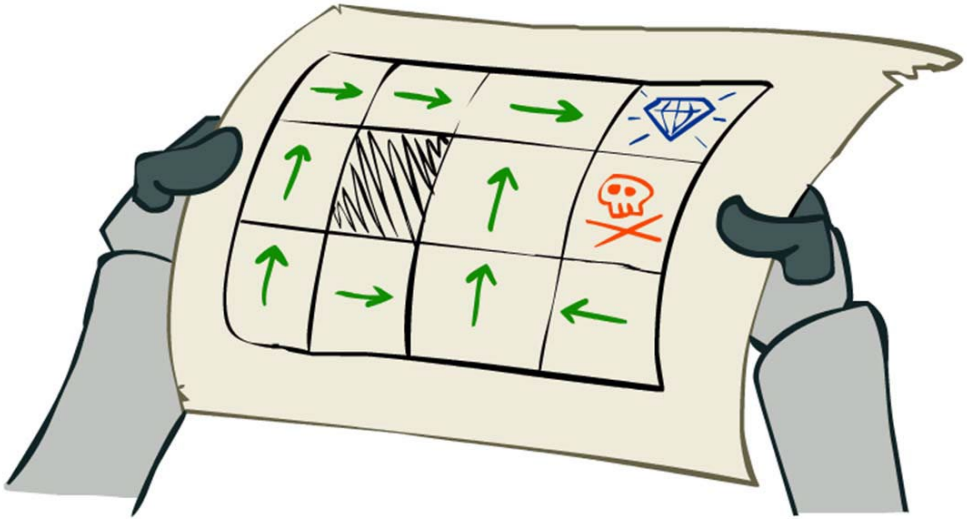
- **State set** S. Initial state s_0. **Action set** A
- **Reward function: *r*(s_t)**
- **State transition model**: $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not earlier history (previous actions or states)
- More generally: $r(s_t, a_t)$, potentially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

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

Example of MDP: Grid World

Robot on a grid; goal: find the best policy

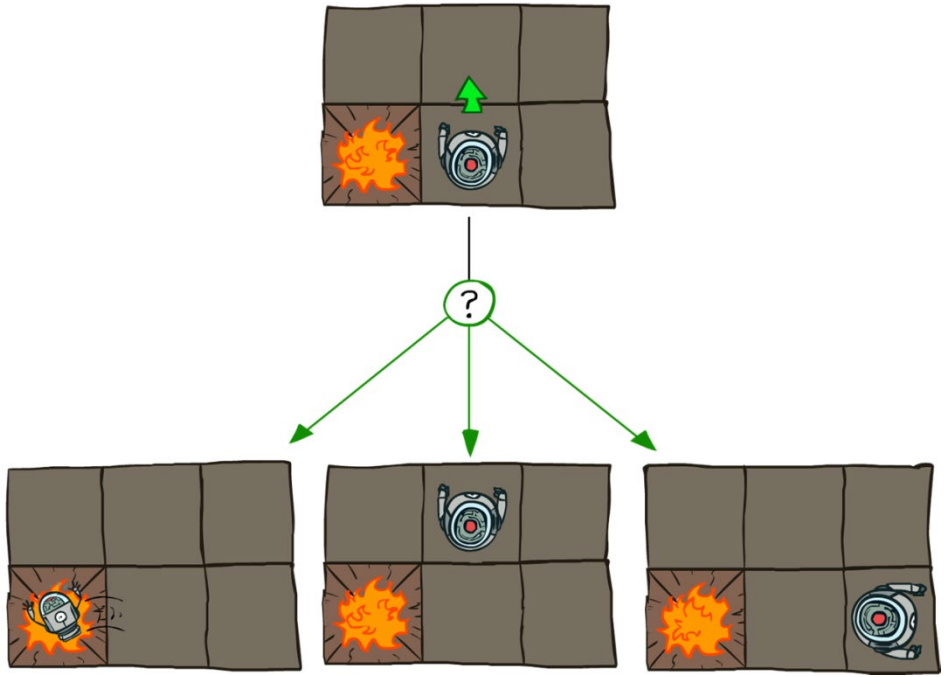


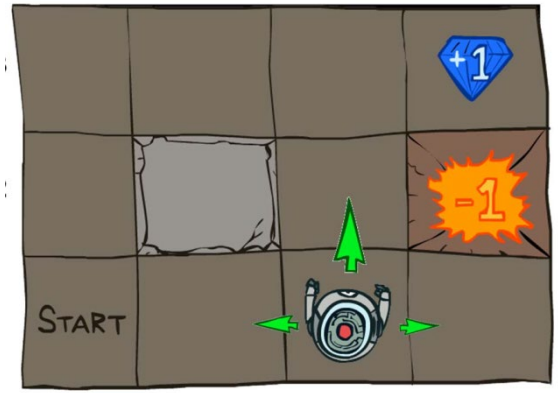


Source: P. Abbeel and D. Klein

Example of MDP: Grid World

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

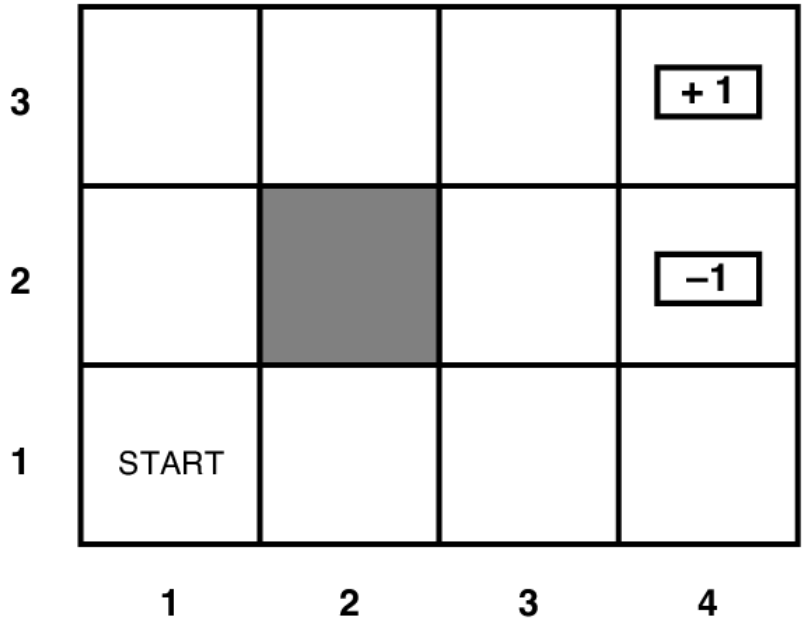


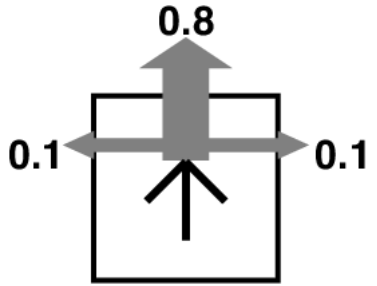


$r(s) = -0.04$ for every non-terminal state

Grid World Abstraction

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

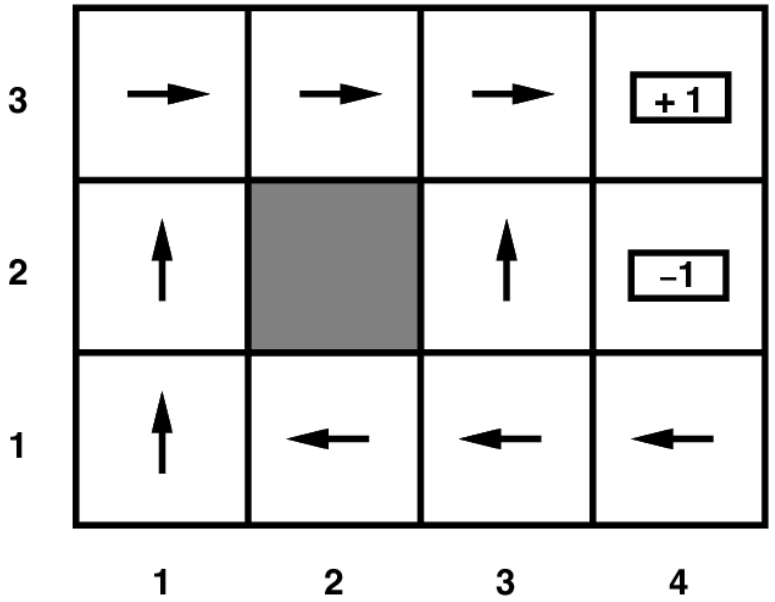


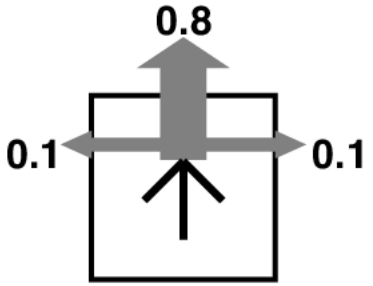


$r(s) = -0.04$ for every non-terminal state

Grid World Optimal Policy

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





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

Back to MDP Setup

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.

How do we find the best policy?

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

Reinforcement Learning Challenges

Credit-assignment:

- May take many actions before reward is received. Which ones were most important?
- Example: You study 15 minutes a day all semester. The morning of the final exam, you eat a bowl of yogurt. You receive an A on the final. Was it the studying or the yogurt that led to the A?

Exploration vs. Exploitation:

- Transition probabilities and reward may be unknown to the learner.
- Should you keep trying actions that led to reward in the past or try new actions that might lead to even more reward?

Break & Quiz

Q 1.1 Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value
- B. The policy maps states to actions
- C. The probability of next state can depend on current and previous states
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards

Break & Quiz

Q 1.1 Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value
- B. The policy maps states to actions
- **C. The probability of next state can depend on current and previous states**
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards

Break & Quiz

Q 1.1 Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value (**True: need to be able to compare**)
- B. The policy maps states to actions (**True: a policy tells you what action to take for each state**).
- **C. The probability of next state can depend on current and previous states** (**False: Markov assumption**).
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards (**True: want to maximize rewards overall**).

Defining the Optimal Policy

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

Utility of sequence

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

Probability of sequence when following π

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

Discounting Rewards

One issue: these are possibly infinite series.

Convergence?

- Solution: discount future rewards.

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

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

From Value to Policy

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

- First, let π^* be the **optimal** policy for $V^{\pi}(s_0)$, and $V^*(s_0)$ its expected utility.
- What's the expected utility following 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

Slight Problem…

Now we can get the optimal policy by doing

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

- So we need to know $V^*(s)$ (and P).
  - But it was defined in terms of the optimal policy!
  - So we need some other approach to get $V^*(s)$.
  - Instead, learn about the utility of actions directly.

Bellman Equation

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

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

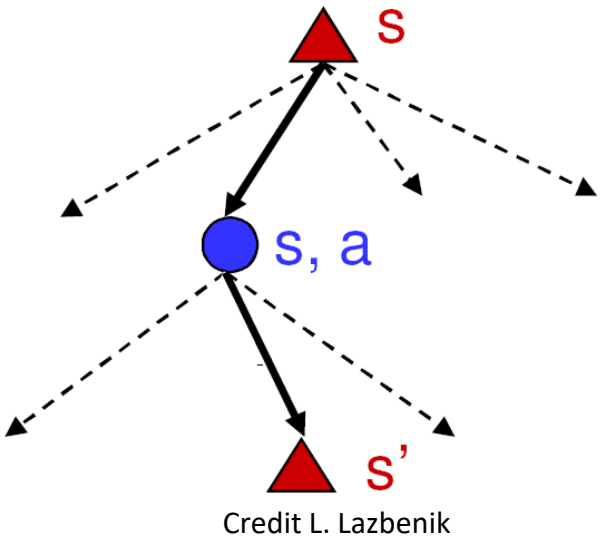
Current state reward

Discounted expected future **rewards**

- Richard Bellman: inventor of dynamic programming

Bellman Equation

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



Credit L. Lazbenik

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

Current state reward

Discounted expected future **rewards**

Value Iteration

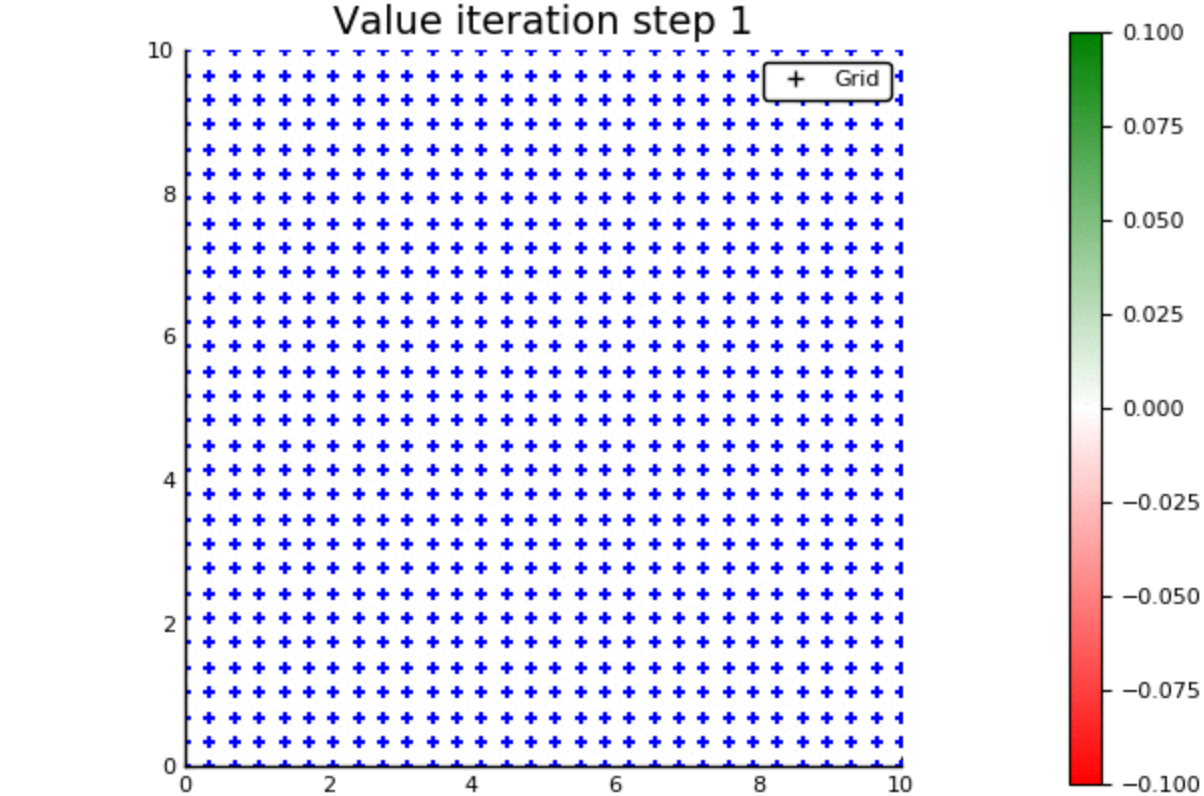
Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s' \mid s,a)$
  - Knowing r and P is the "planning" problem. In reality r and P must be estimated from interactions : "reinforcement learning"
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

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

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

Value Iteration: Demo



Source: POMDPBGallery Julia Package

Break & Quiz

Q 2.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Here, transitions are deterministic. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. Let π: π(*A*) = π(*B*) = **move** (i.e., an "always move" policy). What is the value function $V^{\pi}(A)$?

- A. 0
- B. 1 / (1 -γ)
- C. 1 / (1 -γ^2)
- D. 1

Q-Learning

- Our **next** reinforcement learning algorithm.
- Does not require knowing *r* or *P*. Learn from data of the form:$\{(s_t, a_t, r_t, s_{t+1})\}$.
- Learns an action-value function *Q**(*s*,*a*) that tells us the expected value of taking *a* in state *s*.
  - Note: $V^*(s) = \underset{a}{max} Q^*(s, a)$.
- Optimal policy is formed as $\pi^*(s) = \arg\underset{a}{max} Q^*(s, a)$

The Q*(s,a) function

- Starting from state s, perform (perhaps suboptimal) action a. THEN follow the optimal policy

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

- Equivalent to

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

Q-Learning Iteration

How do we get Q(s,a)?

- Iterative procedure

Current Q Value

Reward for taking that action at that state

Current Q Value

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t) \right]$$

New Q Value for that state and action

Learning rate

Maximum expected future reward

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_b Q(s_{t+1}, b) \right]$$

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

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

Q-Learning

Estimate Q*(*s*,*a*) from data $\{(s_t, a_t, r_t, s_{t+1})\}$:

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 prevent you from discovering the true optimal strategy

Q-Learning: ε-Greedy Behavior Policy

Getting data with both **exploration and exploitation**

- With probability ε, take a random action; else the action with the highest (current) Q(*s*,*a*) value.

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

Q-learning Algorithm

Input: step size α, exploration probability ϵ

1. set $Q(s,a) = 0$ for all s, a.
2. For each episode:
3. Get initial state s.
4. While (s not a terminal state):
5. Perform a = ϵ-greedy(Q, s), receive r, s'
6. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_b Q(s',b))$
7. $s \leftarrow s'$
8. End While
9. End For

Explore: take action to see what happens.

Update action-value based on result.

Break & Quiz

Q 3.1 For Q learning to converge to the true Q function, we must

- A. Visit every state and try every action
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

Summary

- Reinforcement learning setup
- Mathematical formulation: MDP
- Bellman Equation
- Value Iteration Algorithm
- The Q-learning Algorithm

Suggested Readings

Textbook: Artificial Intelligence: A Modern Approach (4th edition). Stuart Russell and Peter Norvig. Pearson, 2020.

- Chapter 22