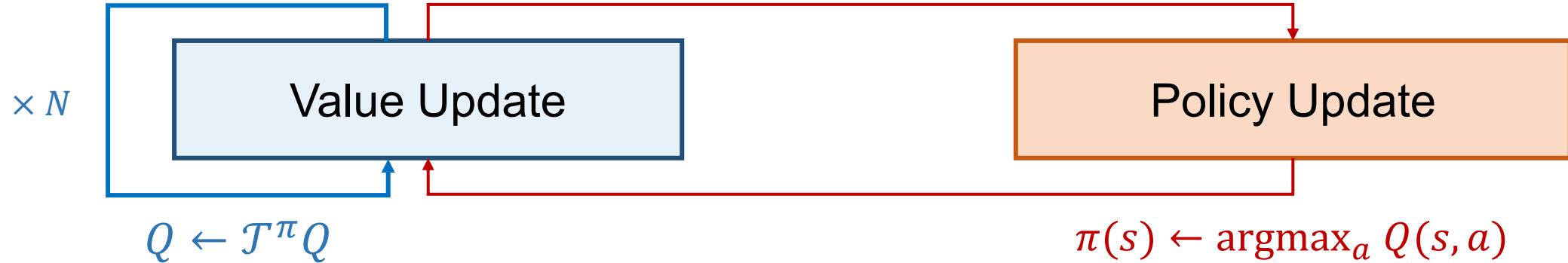


Model Learning and Model-Based RL

Chen-Yu Wei

Recap: Model-Free RL



$Q \leftarrow \mathcal{T}^\pi Q$ means $Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s', a'} P(s'|s, a) \pi(a'|s') Q(s', a')$ for all s, a

Recap: Generalized Policy Iteration with Samples

For $k = 1, 2, \dots$

For $i = 1, 2, \dots, N$:

Choose action a_i with the current policy

Receive reward $r_i \sim R(s_i, a_i)$ and $s'_i \sim P(\cdot | s_i, a_i)$

$s_{i+1} = s'_i$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

Push (s_i, a_i, r_i, s'_i) to \mathcal{B}

Data collection

Draw (s, a, r, s') from \mathcal{B} , and use them to update policy/value

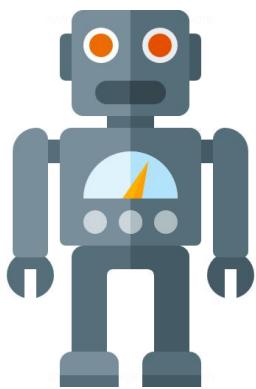
Empty \mathcal{B} if under on-policy training

Policy / value update

When Is Model-Based Method Helpful?

- Model (transition) is easy to learn
 - Deterministic transition could be easier to learn to stochastic one
 - System identification: known parameterized model with unknown parameters
- Model is known
 - The space/action space is too large for full policy/value iteration (Go, Chess)

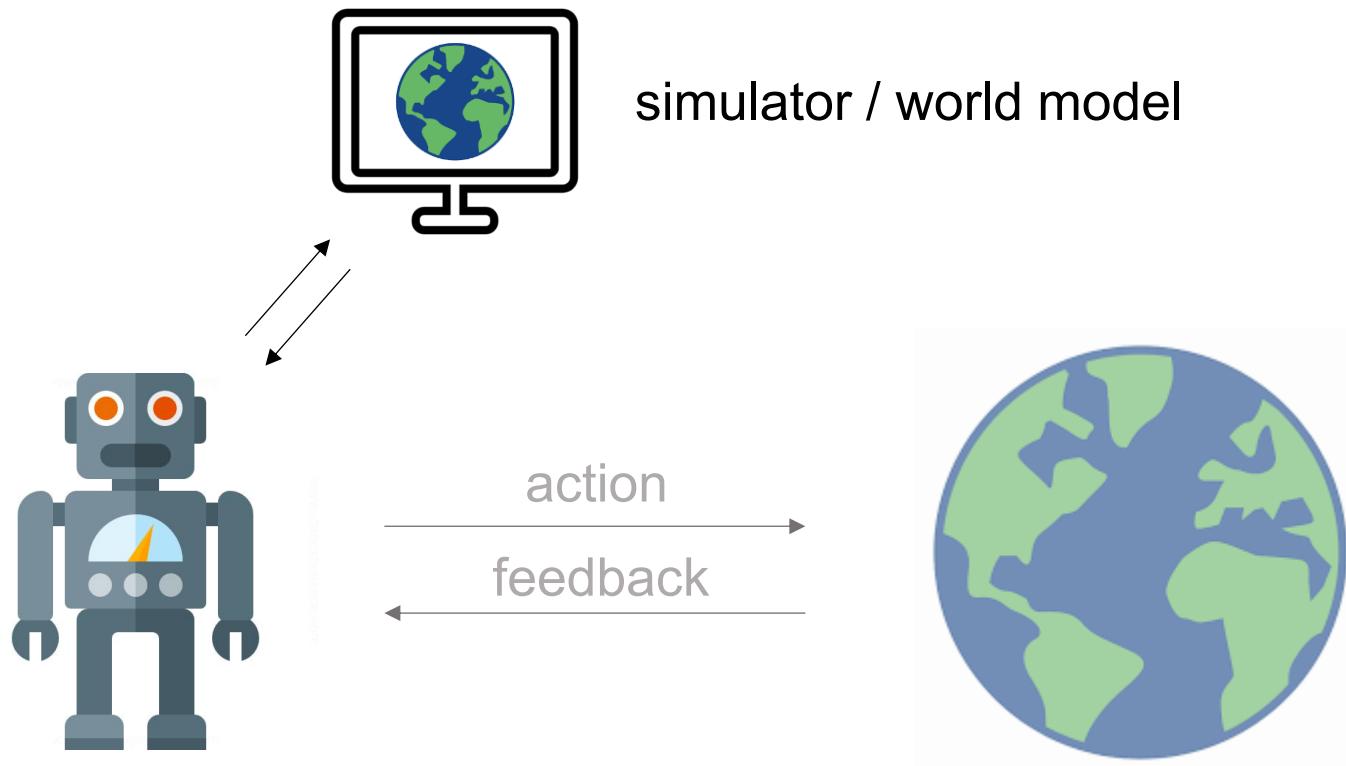
Model-Based RL



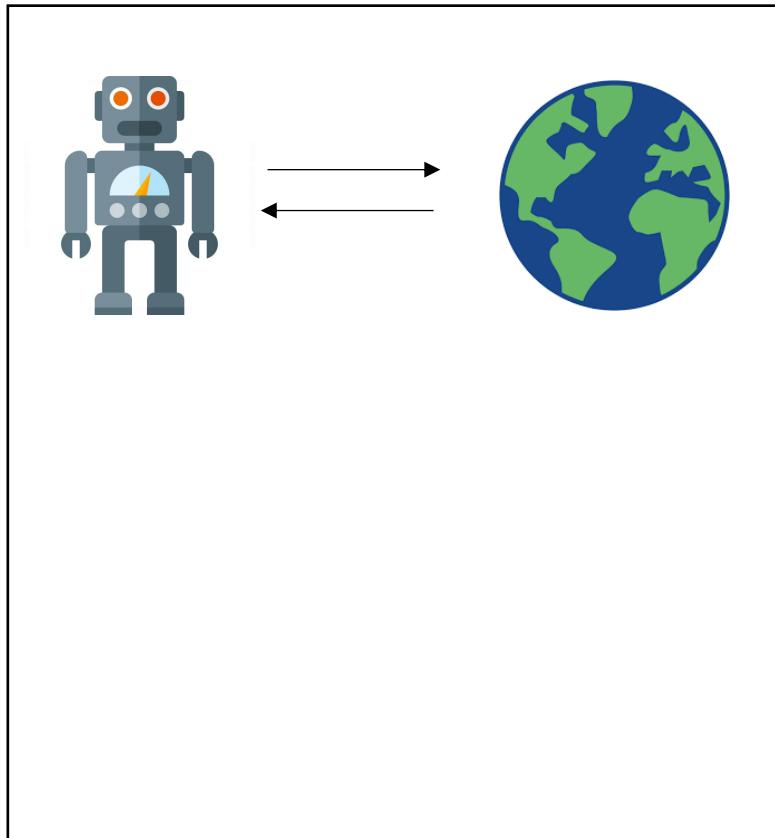
action
feedback

A horizontal double-headed arrow pointing from left to right, positioned between the robot and the Earth. The word "action" is above the arrow, and the word "feedback" is below it, both in a black sans-serif font.

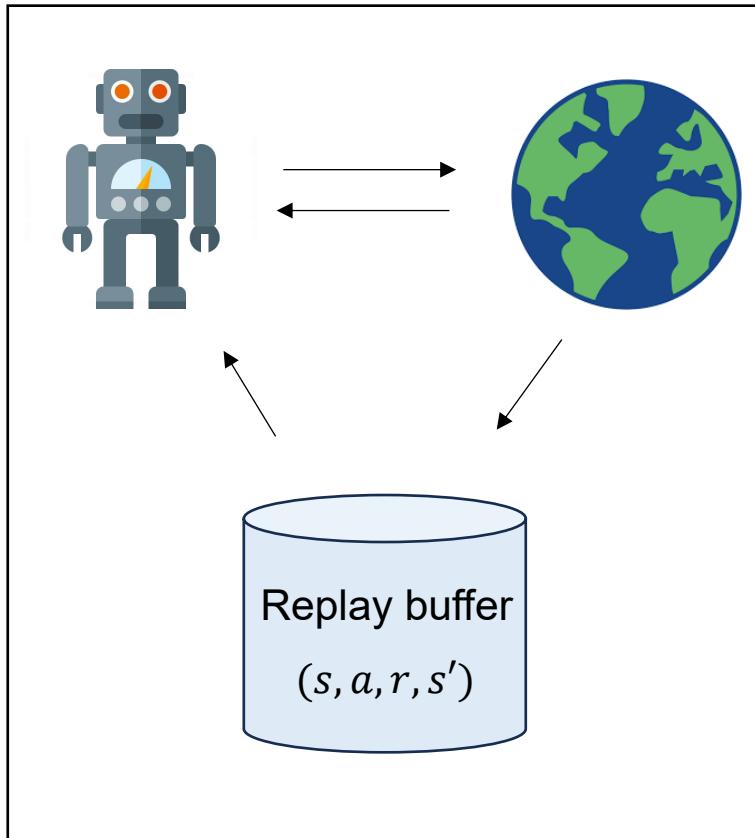
Model-Based RL



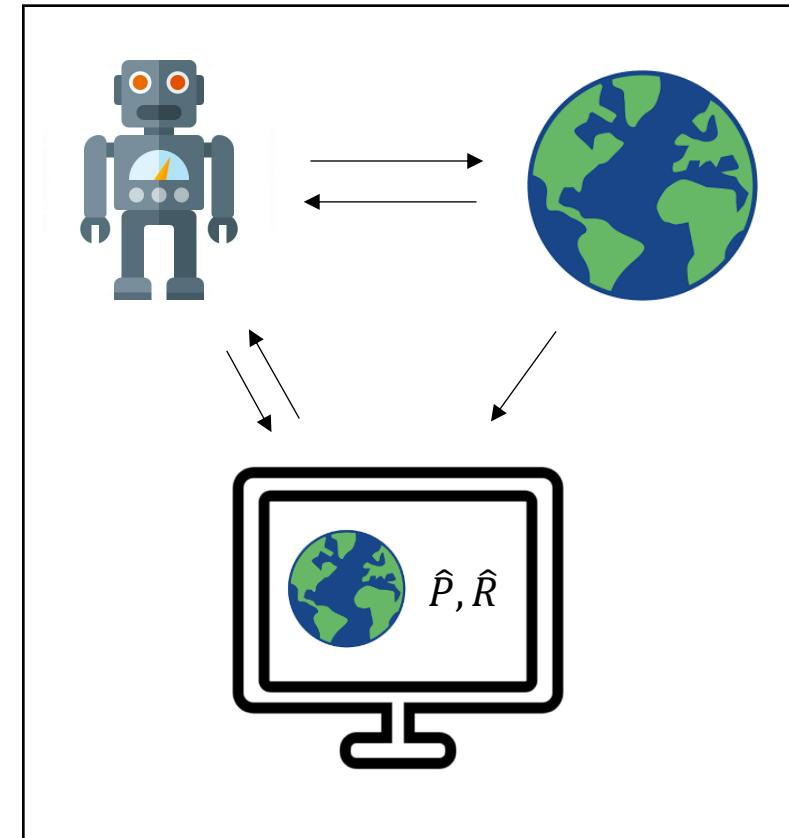
Comparison between training methods



Model-free On-policy



Model-free Off-policy

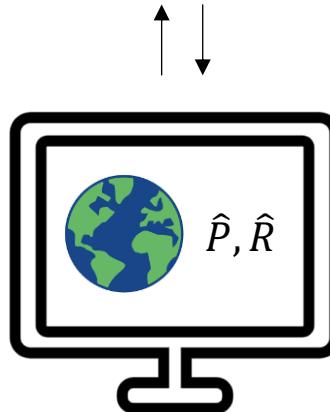
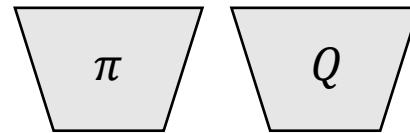


Model-based

Model-Based RL

Two ways to use the simulator / world model / model:

Any model-free algorithm:



Model-assisted model-free learning



Planning with a model

1. Model-Assisted Model-Free Learning (Dyna-style)

For $k = 1, 2, \dots$

For $i = 1, 2, \dots, N$:

Choose action a_i with the current policy π_k

Receive reward $r_i \sim R(s_i, a_i)$ and $s'_i \sim P(\cdot | s_i, a_i)$

$s_{i+1} = s'_i$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

Push (s_i, a_i, r_i, s'_i) to \mathcal{B}

Update model \hat{P}, \hat{R} with data in \mathcal{B}

Repeat several times:

Sample $(s, a) \sim \mathcal{B}$, or sample $s \sim \mathcal{B}$ and $a \sim \pi_k(\cdot | s)$ or uniform

Let $r \sim \hat{R}(s, a)$ and $s' \sim \hat{P}(\cdot | s, a)$

Update policy / value with sample (s, a, r, s')

Data collection

Model update

Policy / value update

1. Model-Assisted Model-Free Learning (Dyna-style)

Why still sample s from the buffer? Why not generate s randomly or generate it from the trained model?

1. Model-Assisted Model-Free Learning (Dyna-style)

Some Dyna-style algorithms:

Gu et al., [Continuous deep Q-Learning with model-based acceleration](#), 2016. (MBA)

Feinberg et al., [Model-based value expansion](#), 2018. (MVE)

Janner et al., [When to trust your model: model-based policy optimization](#). 2019. (MBPO)

The performance of MB-RL is heavily influenced by how to represent and train \hat{P} efficiently, while making it predictive and scalable:

Hafner et al., [Mastering Diverse Domains through World Models](#). 2023. (DreamerV3)

2. Planning with A Model

If we have a model / simulator, how to decide the next action without having a trained policy / value network?

Exact / closed-form solution: finite-state-finite-action, linear system



2. Planning with A Model

If we have a model / simulator, how to decide the next action without having a trained policy / value network?

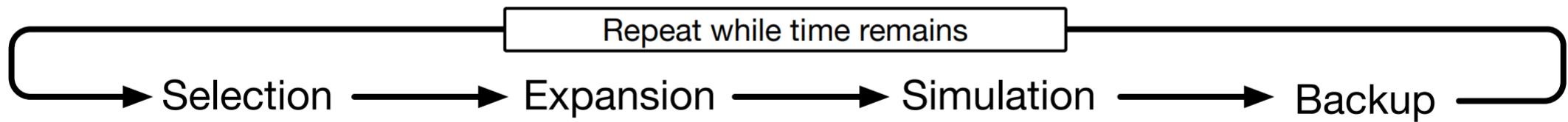
Search (for large state space without structure):

“Create the policy on the fly”: decide $\pi(\cdot | s)$ only when reaching s

This is often used when there is a **default** policy, but we aim to enhance it on the fly.



Monte-Carlo Tree Search (MCTS)

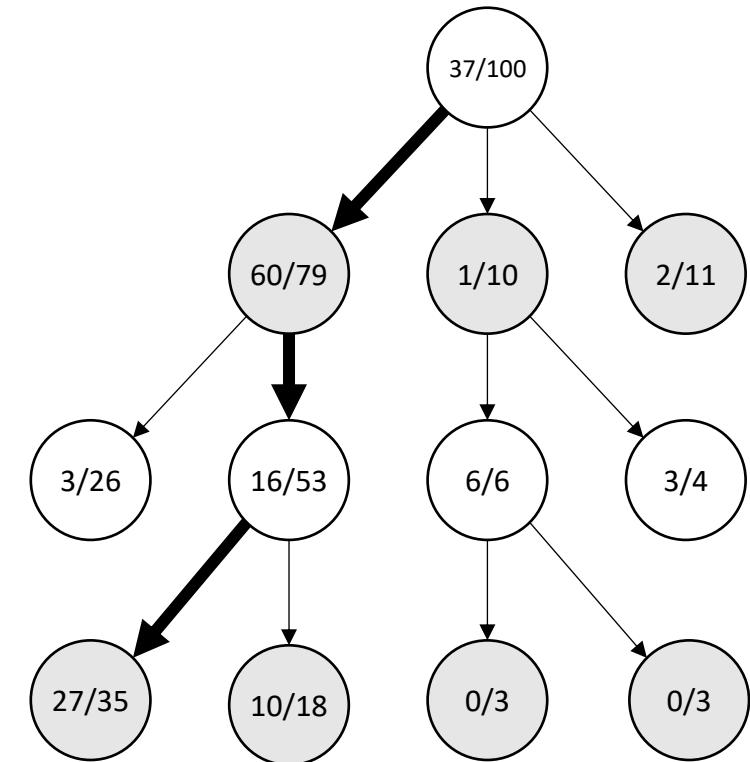


Monte-Carlo Tree Search (MCTS)

Selection

- Starting from the root node, execute **tree policy** until reaching a leaf node
- One effective tree policy is given by UCB1, which chooses an action based on

$$\frac{W(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{parent}(n))}{N(n)}}$$



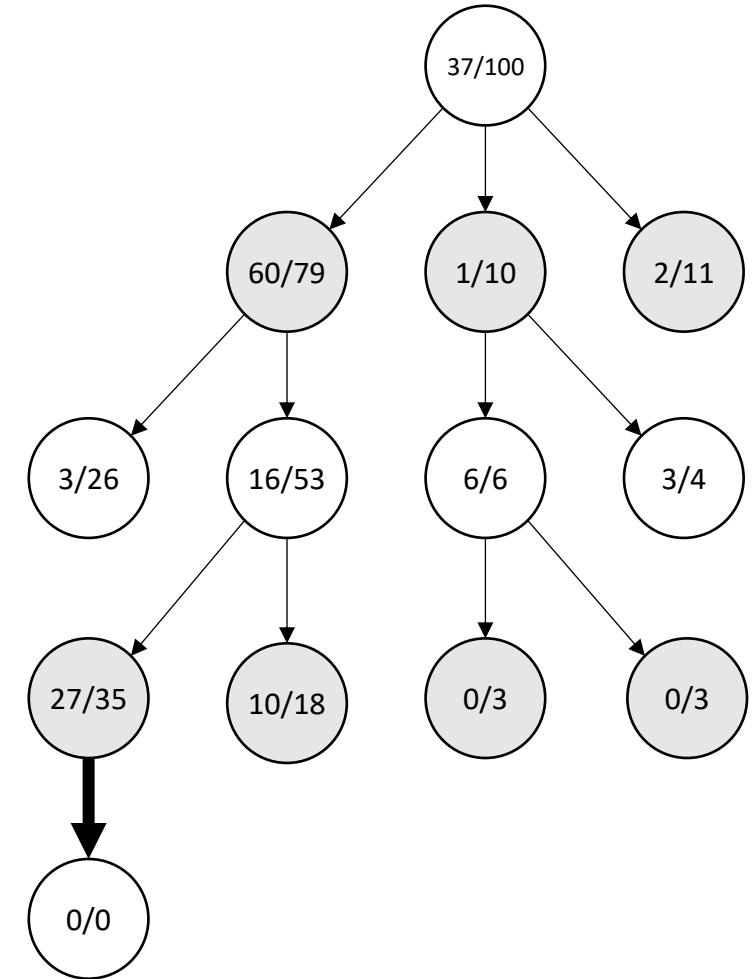
$W(n)$: total #wins of all playouts that went through node n

$N(n)$: total #playouts that went through node n

Monte-Carlo Tree Search (MCTS)

Expand

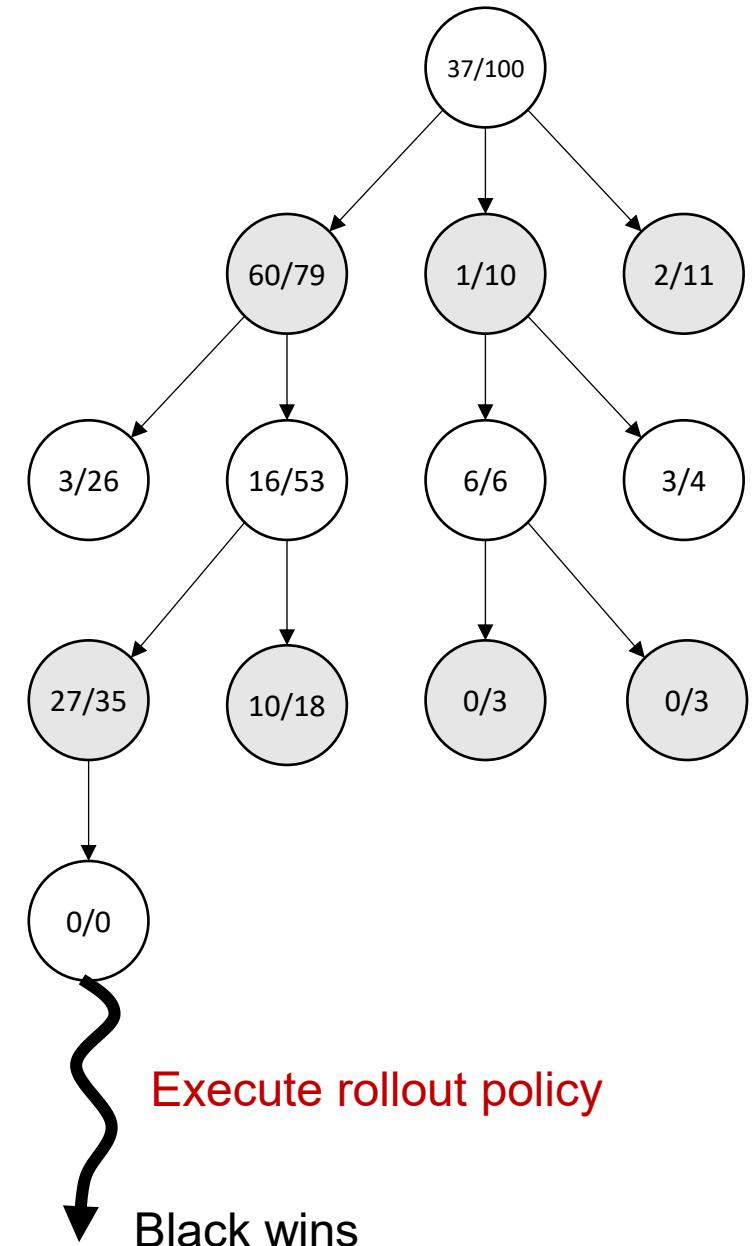
- On some iterations, grow the search tree from selected leaf nodes by adding one or more child nodes



Monte-Carlo Tree Search (MCTS)

Simulation

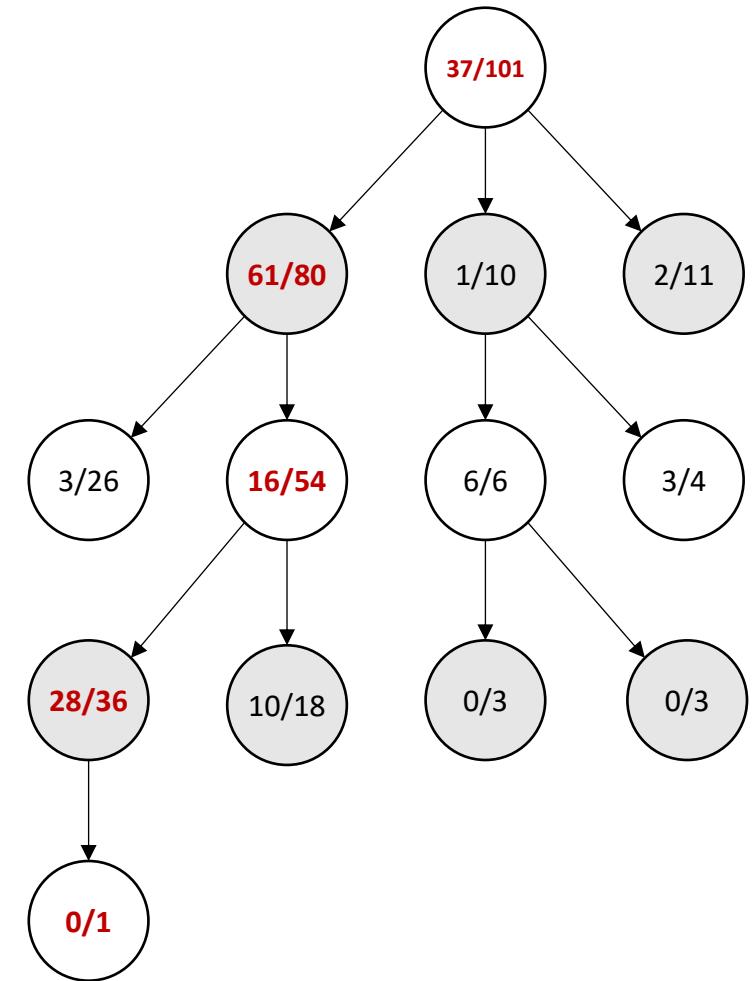
- From the selected or expanded node (if any), execute the **rollout policy** to the end of the game
- Rollout policy
 - Could be heuristics, such as “consider capture moves” in chess
 - Could be learned through neural networks by self-play



Monte-Carlo Tree Search (MCTS)

Backup

- Update the #wins and #playouts on nodes along the tree policy



Monte-Carlo Tree Search (MCTS)

Finally,

- Choose the action from the root node that has the largest visit count.
- After the opponent's move, start the same procedure from the new state
(can keep the statistics from the previous state)

2. Planning with A Model

If we have a model / simulator, how to decide the next action without having a trained policy / value network?

Search (for large state space without structure):

“Create the policy on the fly”: decide $\pi(\cdot | s)$ only when reaching s

This is often used when there is a **default** policy, but we aim to enhance it on the fly.



