Reinforcement learning

Textbook

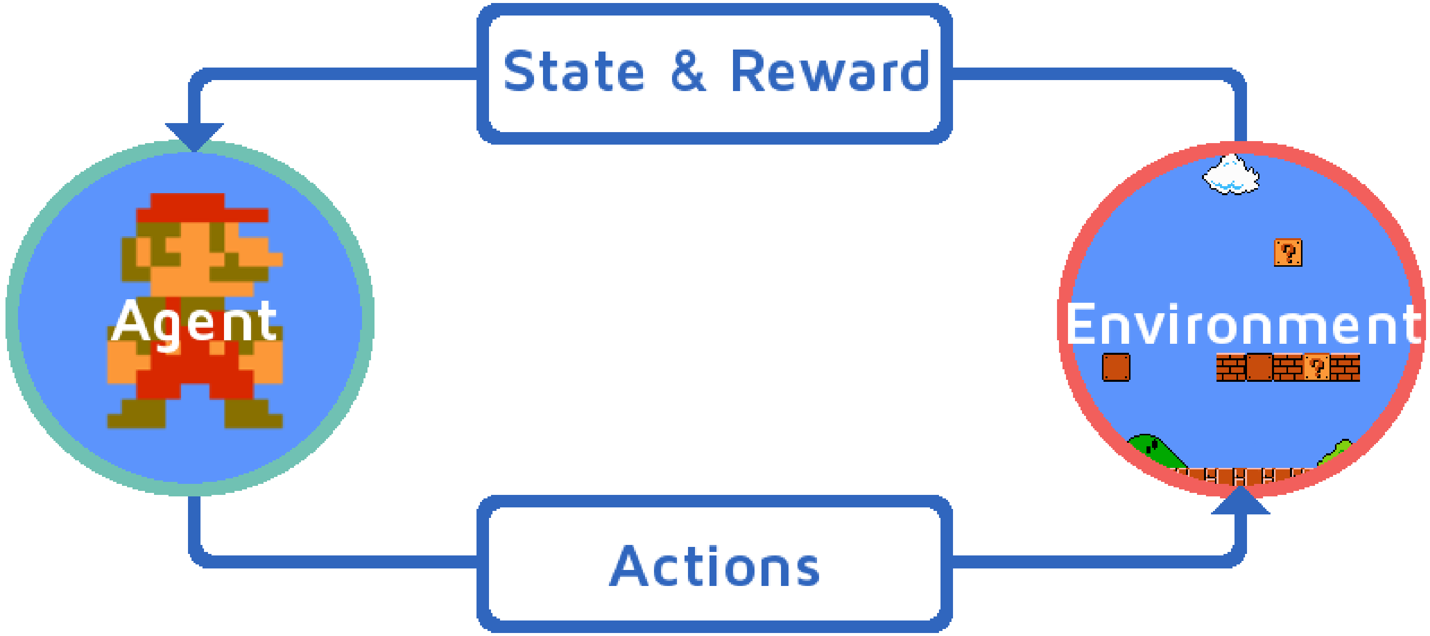
Read the first few chapters (up to and including "Dynamic programming") of

- *Reinforcement Learning: An Introduction* by Sutton and Barto (2018)

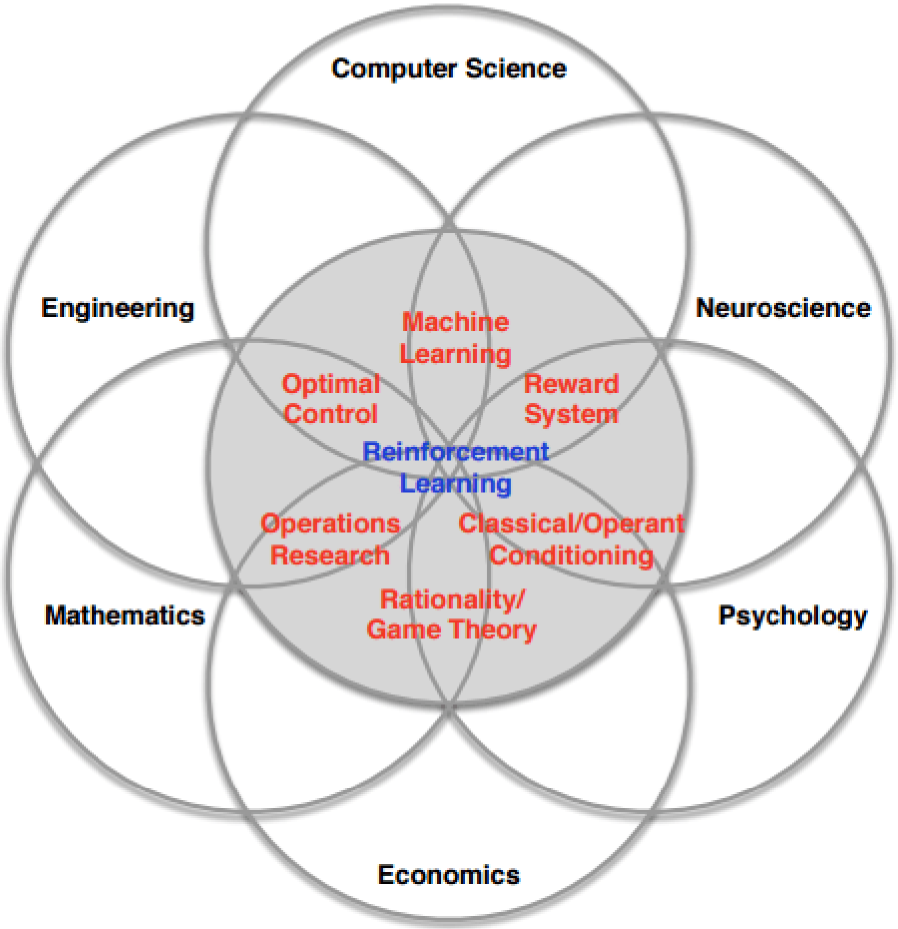
What is reinforcement learning?

- Learning how to map situations to actions, so as to maximize a numerical reward.
- Features:
  - Trial and error search
  - Delayed rewards
  - Dilemma of exploration vs exploitation
- Applications:
  - Games
  - Robotics

What is reinforcement learning?



Reinforcement learning from different views



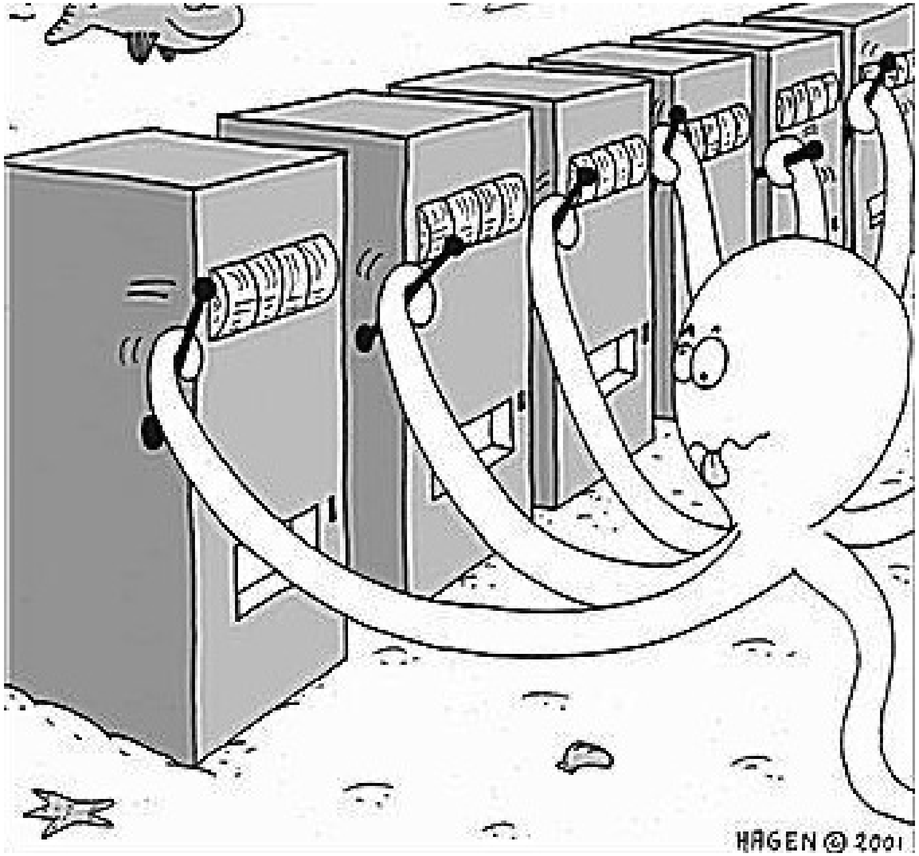
Elements of reinforcement learning

- Policy:
  - defines the agent's behaviour at a given time
  - it is a mapping from states to actions
  - can be stochastic
- Reward signal:
  - at each time step, the environment sends a number to the agent called the reward
  - ultimate goal of the agent is to maximize total reward

Elements of reinforcement learning cont.

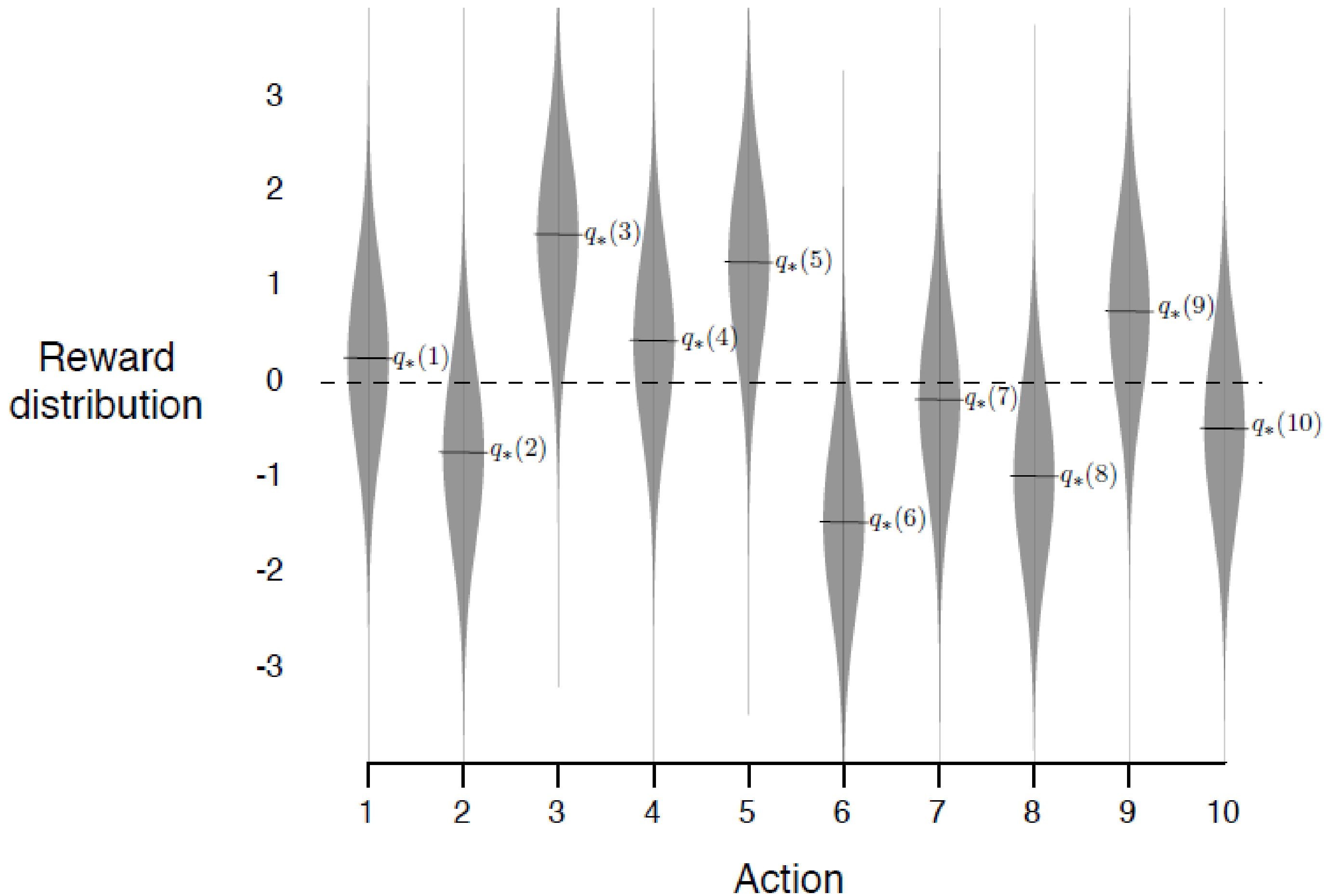
- Value function:
  - the value of a state is the total amount of reward the agent can expect to accumulate over the future, starting from that state
  - reward signal indicates what is good in the immediate sense; the value function indicates what is good in the long run
- Model (optional):
  - approximation of the environment that can be used to predict the next state and reward given the current state and action
  - used for planning
  - model-free methods vs model-based methods

Multi-armed bandit problem



- Imagine there are k jackpot machines in front of you. When you pull the lever of machine i, with some unknown probability p_i you will win \$1, and with probability $1 - p_i$ you will receive nothing.
- The average payouts of the machines may all be different, and the casino affords you 1000 lever pulls before asking you to leave. What should your strategy be?
- Here we are assuming the rewards are distributed as Bernoulli random variables, but they can also be modeled with other distributions (eg. Gaussian).

Gaussian rewards



Action-values

- Before choosing the strategy, let's keep track of our current estimates of how good pulling lever i is, or in more general terms, how good taking action i is. We calculate

$$Q_t(i) = \frac{\text{Sum of rewards from action i before t}}{\text{Number of times action i is taken prior to t}}$$

for each i and at all times t.

- This is an estimate of the true action-value $q_*(i)$, which is the expected reward one were to obtain if one were to keep selecting action i.
- By the law of large numbers, we have

$$Q_t(i) \xrightarrow{t \to \infty} q_*(i)$$

for all i.

Incremental implementation

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n+1}\sum_{i=1}^{n+1} R_i \\
&= \frac{1}{n+1}\left(\sum_{i=1}^{n} R_i + R_{n+1}\right) \\
&= \frac{1}{n+1}\left(n\left(\frac{1}{n}\right)\sum_{i=1}^{n} R_i + R_{n+1}\right) \\
&= \frac{1}{n+1}\left(nQ_n + Q_n - Q_n + R_{n+1}\right) \\
&= Q_n + \frac{1}{n+1}\left(R_{n+1} - Q_n\right)
\end{aligned}
$$

- The formula for incrementally updating the mean estimate is of the form

$$\mathit{NewEstimate} \leftarrow \mathit{OldEstimate} + \mathit{StepSize}[\mathit{Target} - \mathit{OldEstimate}].$$

- The expression $[\mathit{Target} - \mathit{OldEstimate}]$ is called the *error* of the estimate. It is reduced by taking a step in the direction of Target.

Greedy policy

- The simplest strategy is the following, at time t, take the action with the best return so far:

$$A_t = \arg\max_i Q_t(i).$$

- This strategy fully exploits, but does not explore.

Exploitation-Exploration trade-off

- Exploitation:
    - We exploit our current knowledge of the action-values to select the best action.
    - This is the right thing to do to maximize expected reward in one step, or when there is no more uncertainty.

- Exploration:
    - When we select a non-greedy action, we are exploring.
    - We do so despite receiving a smaller reward in the short term, in the hopes of finding better actions, which we can then exploit at later times to maximize reward in the long run.

5 min break

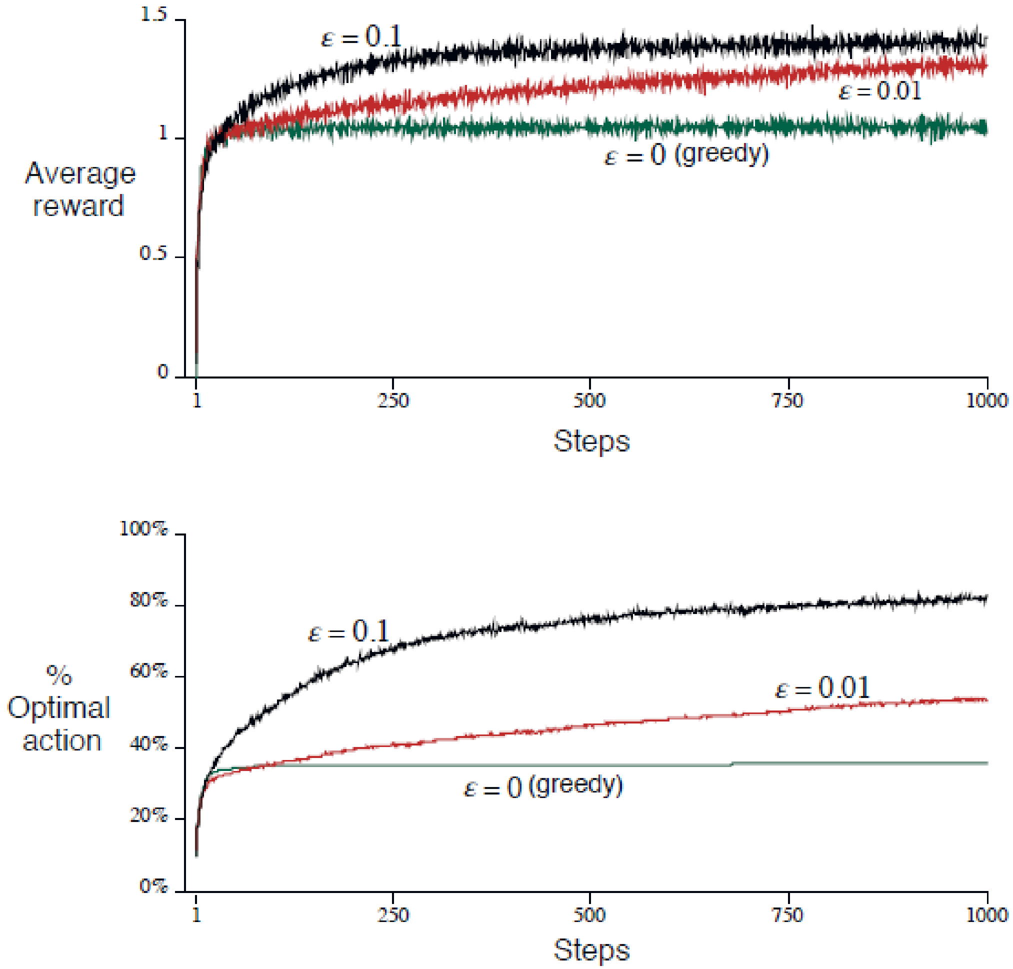
ϵ-greedy policy

- Behave greedily most of the time, but explore once in a while:

$$A_t = \begin{cases} i^* = \arg\max_i Q_t(i) & \text{with probability 1 - } \epsilon \\ j,\ j \neq i^* & \text{each with probability } \frac{\epsilon}{k-1} \end{cases}$$

- Balances exploitation vs exploration, but does not select intelligently between the $k-1$ non-greedy actions.

Performance (averaged over 2000 runs/episodes)



Upper confidence bound (UCB or UCB1)

- "Optimism in the face of uncertainty."
- Select action at time t according to

$$A_t = \arg\max_i \left(Q_t(i) + c\sqrt{\frac{\log t}{N_t(i)}} \right).$$

- $N_t(i)$ denotes the number of times action i has been selected prior to time t.
- c is the exploration constant; increasing it favours exploration and decreasing it favours exploitation.

Hoeffding's inequality

Theorem

Let $X_1, \ldots, X_n$ *be i.i.d. random variables with mean* μ *such that* $X_i \in [0, 1]$ *for all* i*. Then*

$$P\left(\mu \geq \bar{X}_n + U\right) \leq e^{-2nU^2},$$

where $\bar{X}_n = \frac{1}{n}(X_1 + \ldots + X_n)$*.*

Explaining the exploration term

- We want our upper bound U to be set such that the probability that the true mean exceeds our sample estimate $\bar{X}_n$ plus U is very low.
- We also expect our estimate to get better with time, so, although somewhat arbitrary, we can demand that

$$P\left(\mu \geq \bar{X}_n + U\right) \leq \frac{1}{t^k},$$

  for some k. Here t denotes the total number of actions taken so thus far.

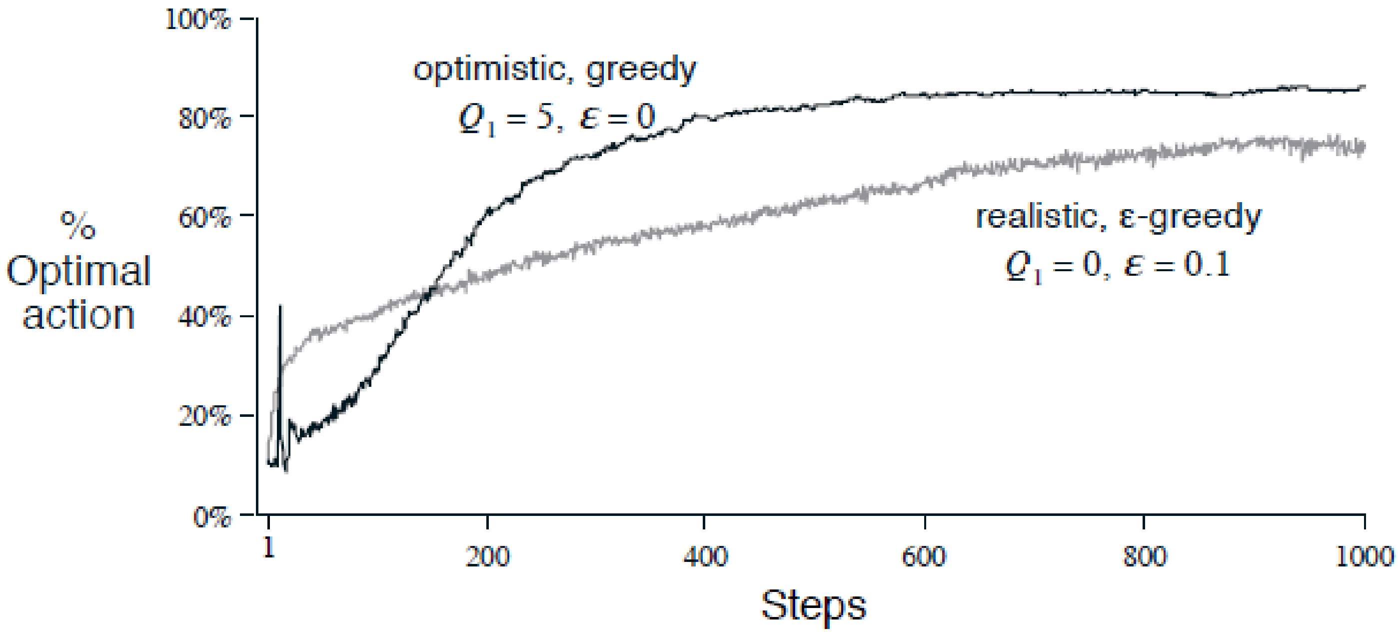
Explaining the exploration term

- Thus using Hoeffding's inequality, we require

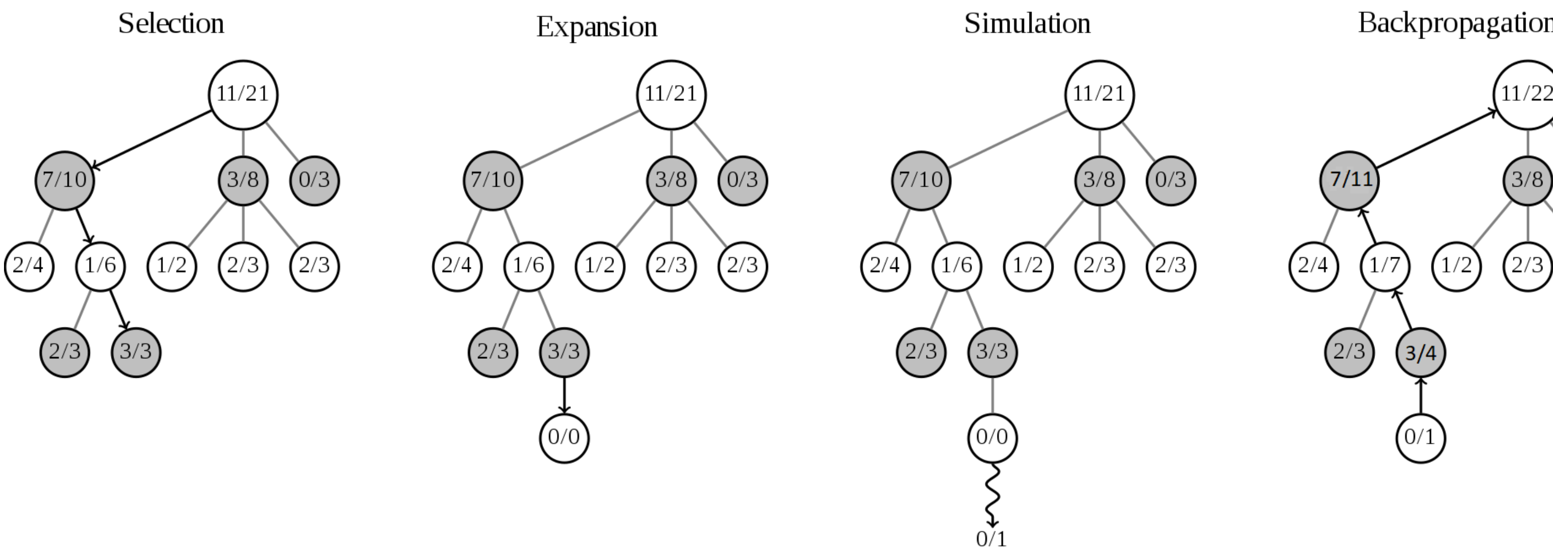
$$\begin{aligned} e^{-2nU^2} \leq \frac{1}{t^k} &\iff -2nU^2 \leq -k \log t \\ &\iff U \geq \sqrt{\frac{k}{2}}\sqrt{\frac{\log t}{n}} \end{aligned}$$

Optimistic initial values

- Set $Q_0(i)$ to be something high, rather the expected value of the true means.
- This encourages exploration in the initial stages.



Monte-Carlo tree search (MCTS)



- White: Max player, Black: Min player
- Values are recorded from the point of view for the max player white

Steps of MCTS

(i) Selection: Starting from the root node, the search process descends down the tree by successively selecting child nodes according to the *tree policy*, the most common of which is UCB1.

(ii) Expansion: When the simulation phase reaches a leaf node, children of the leaf node are added to the tree, and one of them is selected by UCB1.

(iii) Simulation: One random playout, or multiple if parallel processing is employed, is performed until a terminal node is reached.

(iv) Back-propagation: The result of the playout is computed and used to update the nodes visited in the selection phase.

- Pros:
  - Does not require an evaluation function at leaf nodes.
  - Compared to alpha-beta search, which primarily uses depth-first search, MCTS is a best-first search algorithm, and search can be terminated at any time with relatively good results.
  - More robust than minimax or alpha-beta search.
- Cons:
  - At each node, UCB1 assumes a stationary distribution for the children (actions) of the node, but more often than not the actual distribution is non-stationary.
  - Simple averaging of the rewards to determine the means of the child nodes causes convergence to the optimal action to be slow, particularly in minimax trees.

Non-stationary problems

- Give more weight to recent rewards than older rewards:

$$Q_{n+1} = Q_n + \alpha[R_{n+1} - Q_n], \qquad \alpha \in (0, 1].$$

- Expanding the expression, we have

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_{n+1} - Q_n] \\ &= \alpha R_{n+1} + (1-\alpha)Q_n \\ &= \alpha R_{n+1} + (1-\alpha)[\alpha R_n + (1-\alpha)Q_{n-1}] \\ &= \alpha R_{n+1} + (1-\alpha)\alpha R_n + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_{n+1} + (1-\alpha)\alpha R_n + (1-\alpha)^2 \alpha R_{n-1} \\ &\qquad + \cdots + (1-\alpha)^n \alpha R_1 + (1-\alpha)^{n+1} Q_0. \end{aligned}$$

- This is also called exponential recency-weighted average.