Reinforcement Learning

Group Project

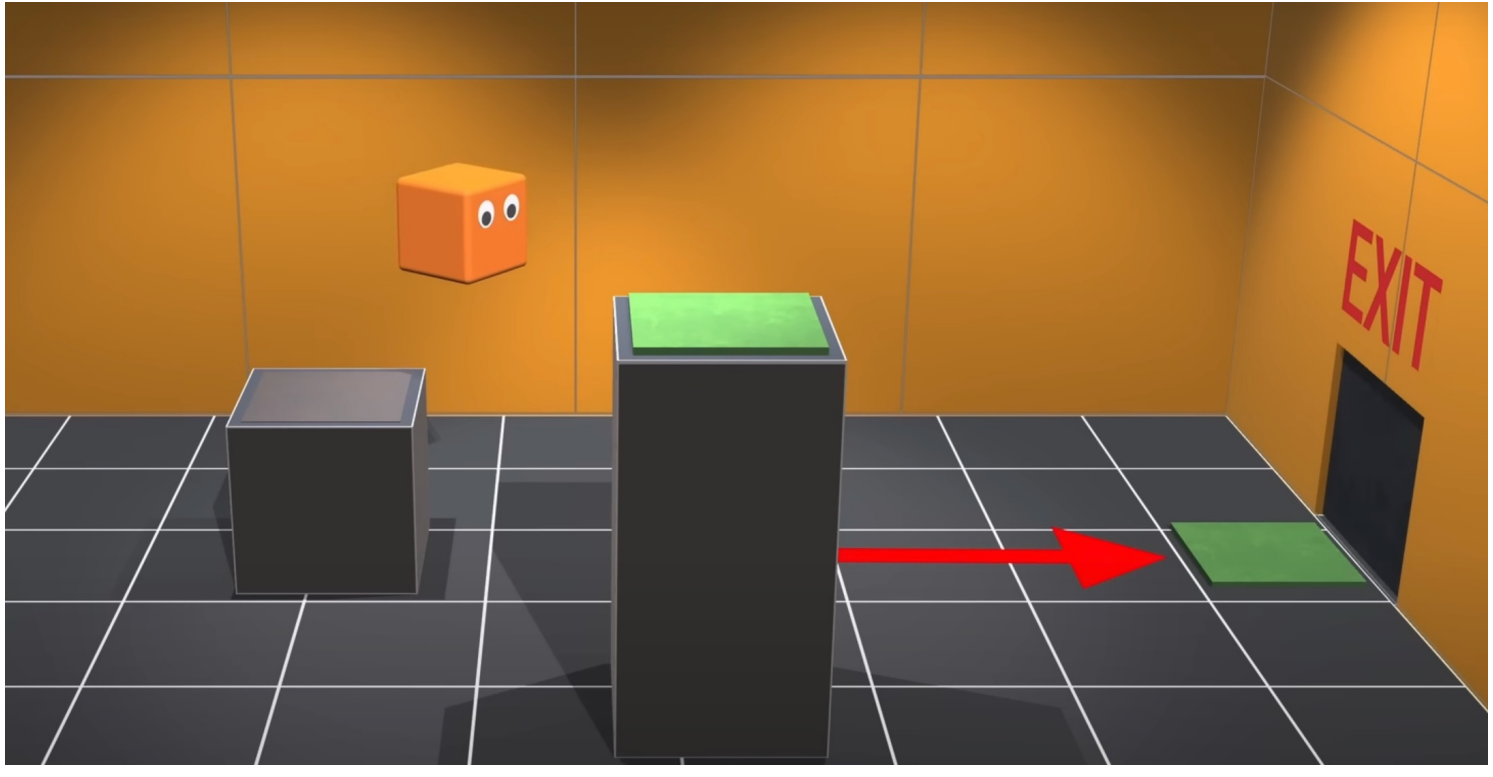
Reinforcement Learning

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

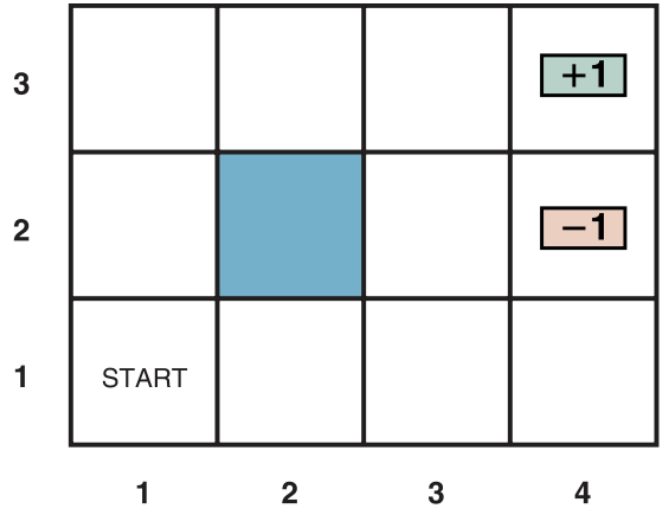
The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

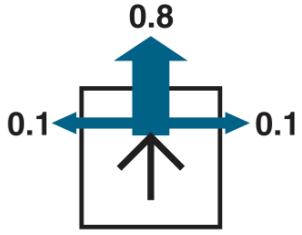
Imagine playing a new game whose rules you don't know; after a hundred or so moves, the referee tells you "You lose." You repeat this until you can find an optimal strategy for playing the game.

Albert



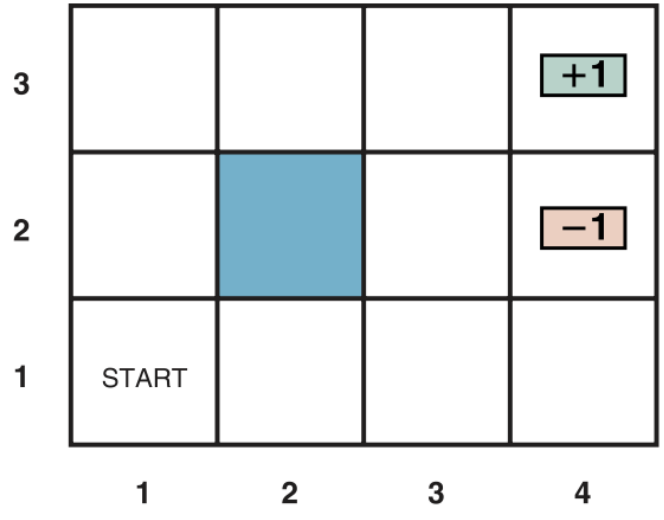
https://youtu.be/v3UBlEJDXR0?si=Ga13NYEoFXkHUARW

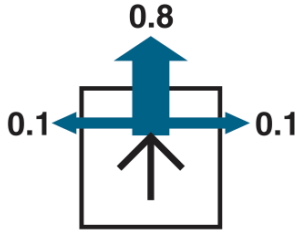




Imagine that robot is trying to learn the following task: Exit the the maze by finding the +1(green) cell. Successfully exiting the cell results in a win and gain a +1 **reward**.

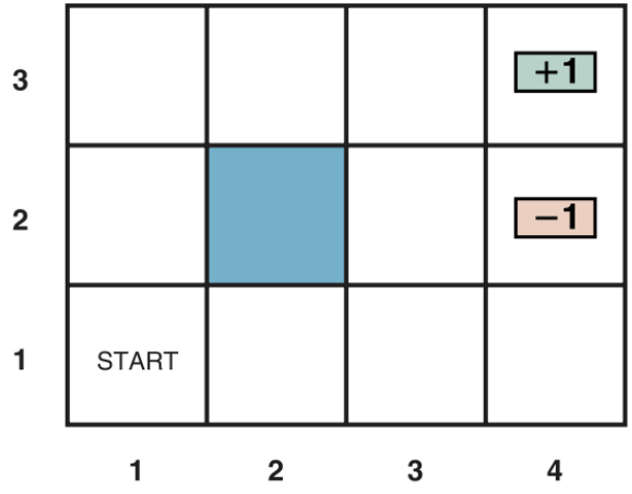
If robot lands on the -1(red) cell, it loses and gain a -1 reward.

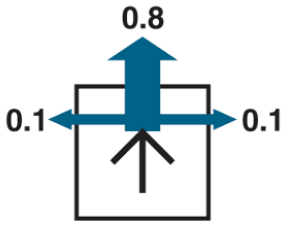




A robot movement, however, is unreliable. Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction. Furthermore, if the agent bumps into a wall, it stays in the same square.

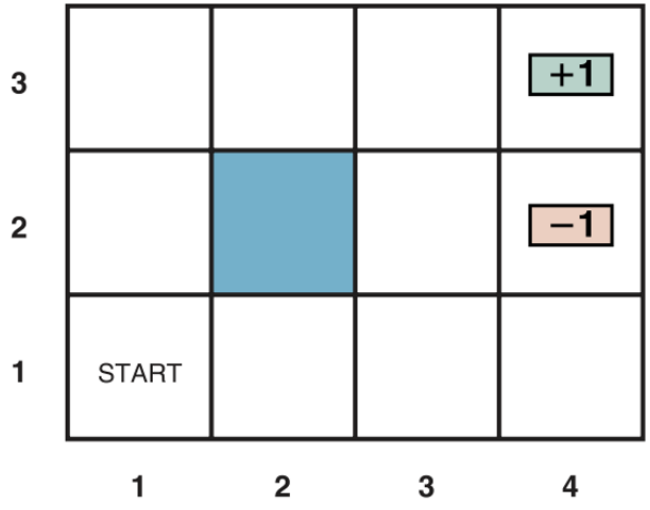
If a robot intends to go right, for example, it goes right 80% of the time, goes up 10% of the time and goes down 10% of the time.

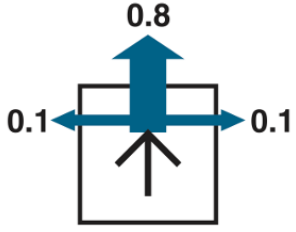




For our particular example, the reward is −0.04 for all transitions except those entering terminal states (which have rewards +1 and –1). The negative reward of –0.04 gives the agent an incentive to reach (4,3) quickly.

Everytime the robot successful exit, it follows a certain path to get there. It receives a reward of +1. But the robot will calculate the **utility** of that path by taking into account how many steps it took to get there.

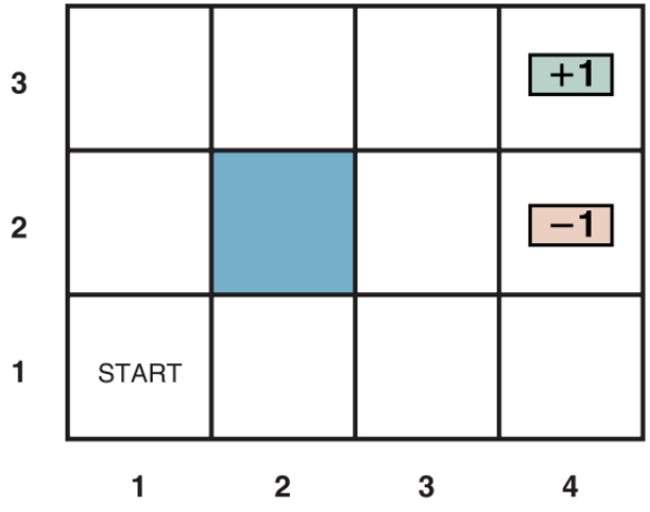


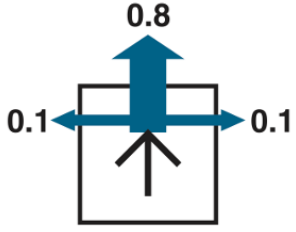


Because the robot does not know what the task is, early in the learning process, it's possible for the robot to initially go in random directions, including backtracking.

For example, if the agent reaches the +1 state after 9 steps: [up, up, down, up, right, right, down, up, right], then its total utility will be $(8 \times -0.04) + 1 = 0.68$.

Note that this utility function is a function of the starting state, in this case, (1, 1).

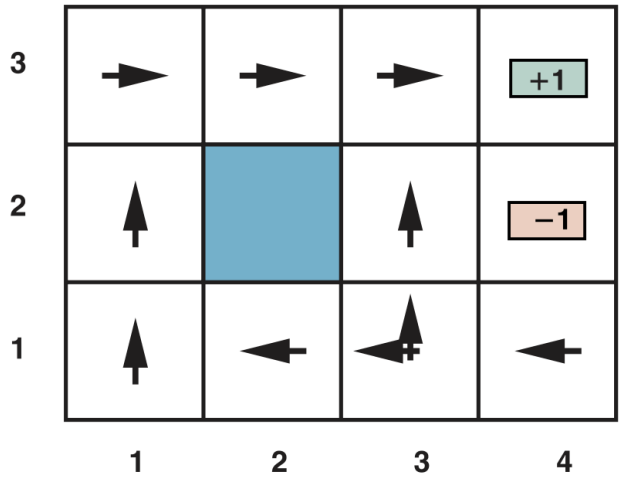




The next question is, what does a solution to the problem look like? No fixed action sequence can solve the problem, because the agent might end up in a state other than the goal.

Therefore, a solution must specify what the agent should do for any state that the agent might reach. A solution of this kind is called a **policy**.

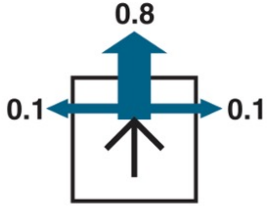
It is traditional to denote a policy by π, and $\pi(s)$ is the action recommended by the policy π for state s. No matter what the outcome of the action, the resulting state will be in the policy, and the agent will know what to do next.



(a)

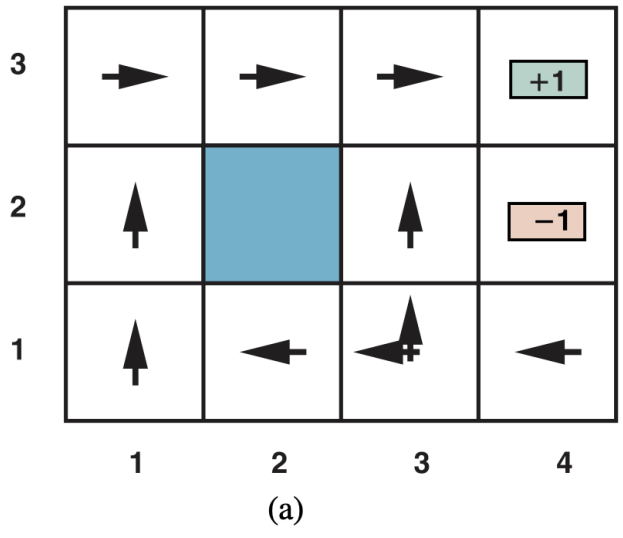
3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	−1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

(b)

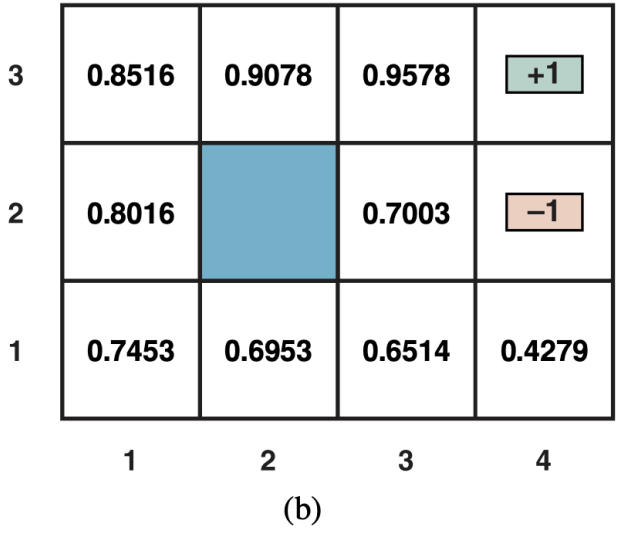


Each time a given policy is executed starting from the initial state, the **stochastic** (probabilistic) nature of the environment may lead to a different environment history.

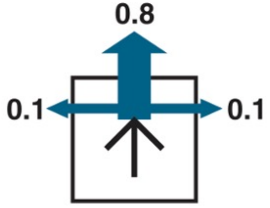
Given a fixed policy, we can calculate the **expected value(or expected utility)** of the possible environment histories generated by that policy. For example, the table b) on the right is the utilities of states V(s) calculated from the policy on the left table a).



(a)



(b)



An **optimal policy** is a policy that yields the highest expected utility. We use $\pi*$ to denote an optimal policy. Given $\pi*$, the agent decides what to do executing the action given by $\pi*(s)$.

Note that in the diagram above, there are TWO policies because in state (3,1) both Left and Up are optimal. Why?

Markov Decision Process

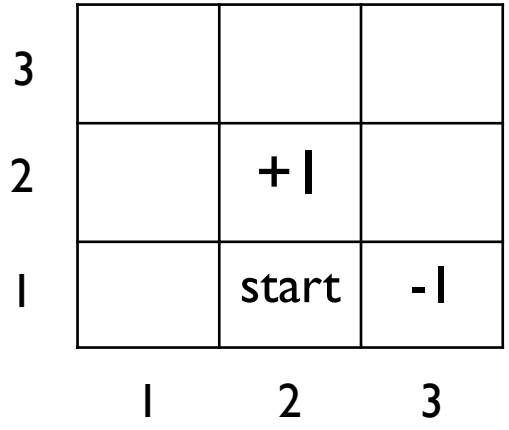
A problem of this kind is called a **Markov Decision Process(MDP)**: a sequential decision problem for a fully observable, stochastic environment and additive rewards.

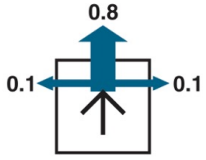
An MDP can be represented as a graph. The nodes in this graph include both states and chance nodes.

Edges coming out of states are the possible actions from that state, which lead to chance nodes. Edges coming out of a chance nodes are the possible random outcomes of that action, which end up back in states.

Our convention is to label these chance-to-state edges with the probability of a particular transition and the associated reward for traversing that edge.

Markov Decision Process



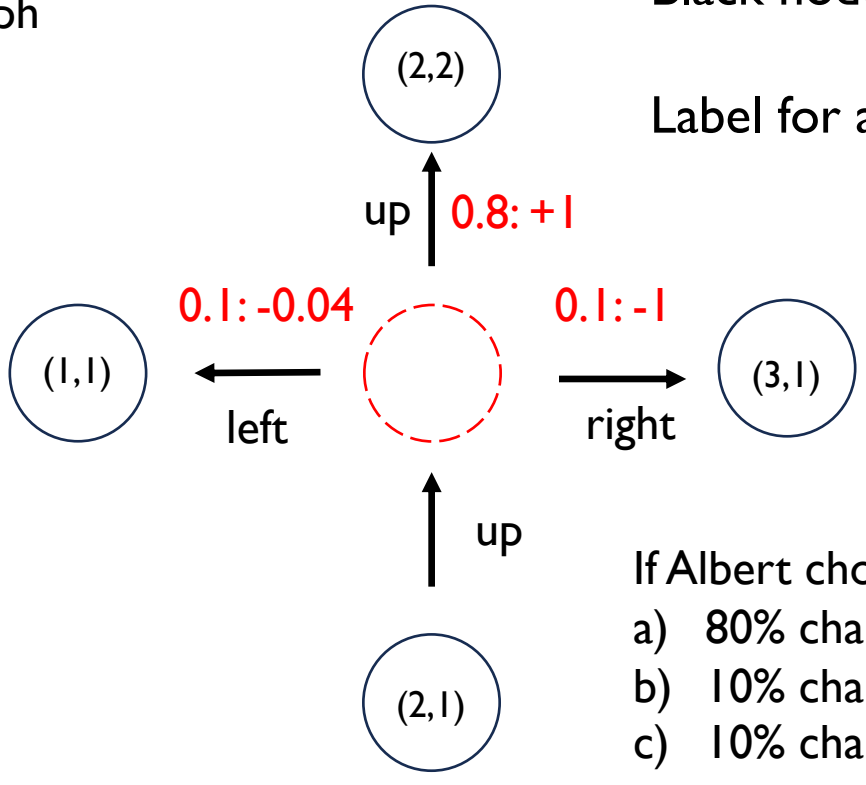


Same setup as the previous problem but with different map.

We can represent this MDP as a graph as shown in the next slide.

Representing MDP as a Graph

Only partial graph starting at (2,1).



Red node is chance node.
Black nodes are state nodes.

Label for actions: probability:reward

If Albert chooses to go up:

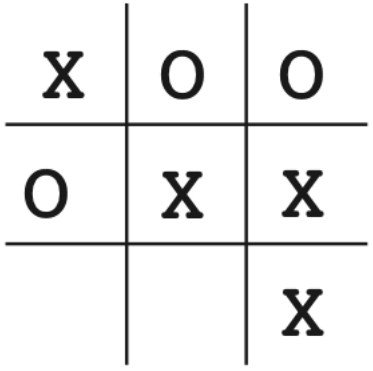
a) 80% chance, he does go up for +1 reward
b) 10% chance, he goes left for -0.04 reward
c) 10% chance, he goes right for -1 reward.

Exploitation vs Exploration

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation.

To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before.

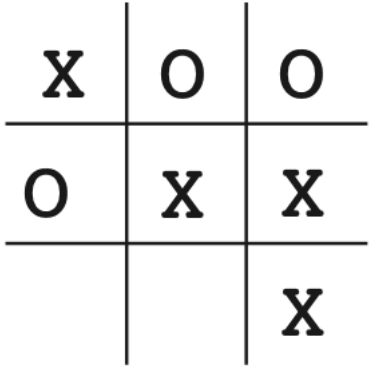
The agent has to **exploit** what it has already experienced in order to obtain reward, but it also has to **explore** in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best.



Here is how the tic-tac-toe problem would be approached with a method making use of a value function.

Similar to the robot problem, we like to find the expected value of every configuration(state) of the board relative to some player(X or O).

First we would set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state.



We treat this estimate as the state's value, and the whole table is the learned value function.

State A has higher value than state B, or is considered "better" than state B, if the current estimate of the probability of our winning from A is higher than it is from B.

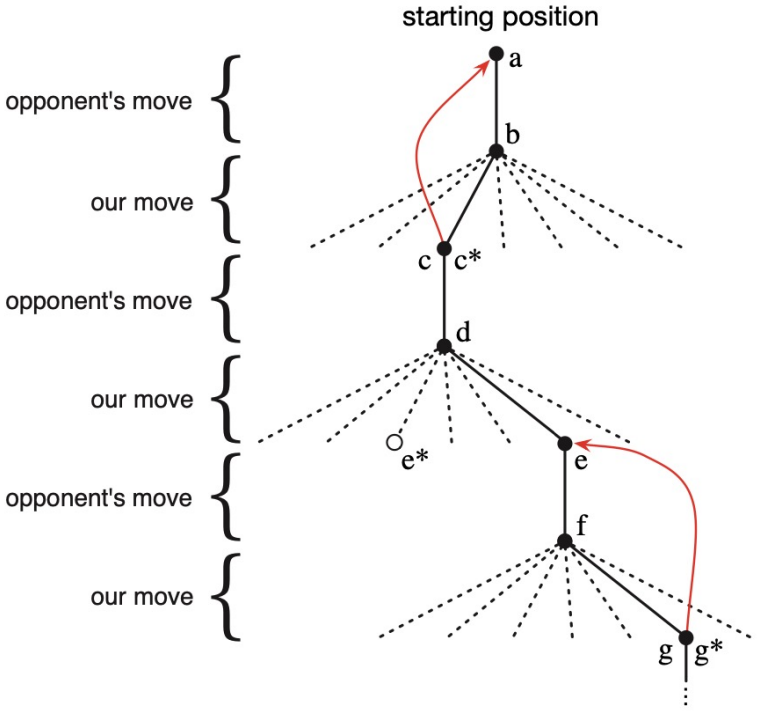
We set the initial values of states where X already won to 1, states where O already won to 0 and all the other states to 0.5, representing a guess that we have a 50% chance of winning.

To train the AI(first player), we let it play many games(e.g., 100k) against another AI opponent(second player).

To select our moves we examine the states that would result from each of our possible moves (one for each blank space on the board) and look up their current values in the table.

Most of the time we move **greedily**, selecting the move that leads to the state with greatest value, that is, with the highest estimated probability of winning.

Occasionally, however, we select randomly from among the other moves instead. These are called **exploratory** moves because they cause us to experience states that we might otherwise never see.



A sequence of tic-tac-toe moves. The solid black lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make.

The * indicates the move currently estimated to be the best. Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e*, was ranked higher.

We attempt to make them more accurate estimates of the probabilities of winning.

To do this, we "back up" the value of the state after each greedy move to the state before the move, as suggested by the arrows in Figure 1.1.

More precisely, the current value of the earlier state is updated to be closer to the value of the later state. This can be done by moving the earlier state's value a fraction of the way toward the value of the later state.

If we let S_t denote the state before the greedy move, and S_{t+1} the state after that move, then the update to the estimated value of S_t, denoted $V(S_t)$,

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[V(S_{t+1}) - V(S_t)\Big],$$

where α is a small positive fraction called the step-size parameter, which influences the rate of learning.

References

1) Artificial Intelligence: A Modern Approach by Peter Norvig and Stuart J. Russell

2) Reinforcement Learning: An Introduction by Richard Sutton and Andrew Barto

3) CS221: Machine Learning course at Stanford. https://stanford-cs221.github.io/