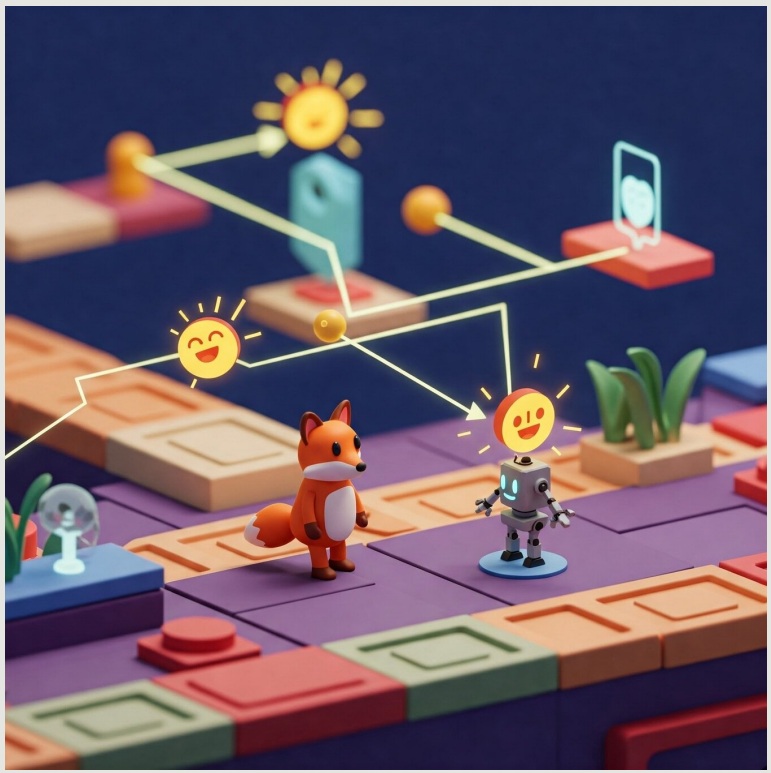
Reinforcement Learning

An Introduction to RL and the Q-value approach

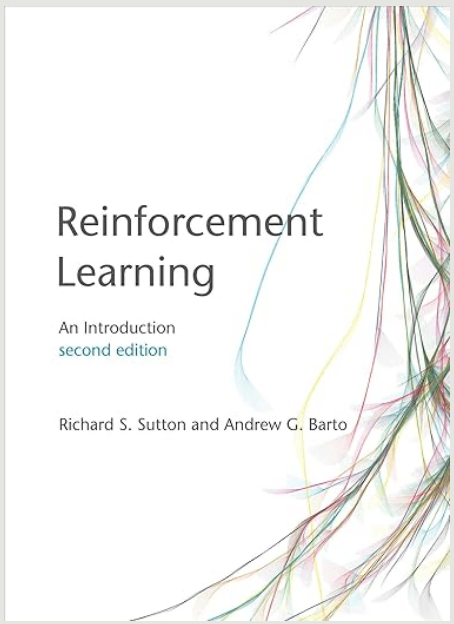


How can a robot learn?

Introduction

Motto : "Errare humanum est, perseverare autem diabolicum!"

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward. Unlike supervised learning, where the model learns from a dataset of labeled examples, RL learns from the consequences of its actions, using feedback from the environment to learn optimal behaviors over time.



Introduction

- If one wishes to design a robot or a machine capable of making autonomous decisions, how might this objective be realized?
- The Challenge: There exist numerous choices to be made (paths, courses of action, etc.); thus, which option merits pursuit, and in what manner can the machine be programmed to begin acquiring the ability to make judicious choices?
- One potential solution is to employ a Markov Decision Process framework (MDP).
- The learner and decision maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*.

Terminology

1. **Agent**: The learner or decision-maker.
2. **Environment**: Everything the agent interacts with. It provides feedback to the agent in the form of rewards.
3. **State**: A representation of the current situation or configuration of the environment.
4. **Action**: The set of all possible moves the agent can make.
5. **Reward**: Feedback from the environment to evaluate the action taken by the agent.
6. **Policy π**: A strategy used by the agent to determine the next action based on the current state.
7. **Value Function**: A function that estimates the expected cumulative reward from a given state or state-action pair.
8. **Q-Value (Action-Value)**: A function that estimates the expected cumulative reward for taking a specific action in a given state and following a certain policy thereafter.

Introduction

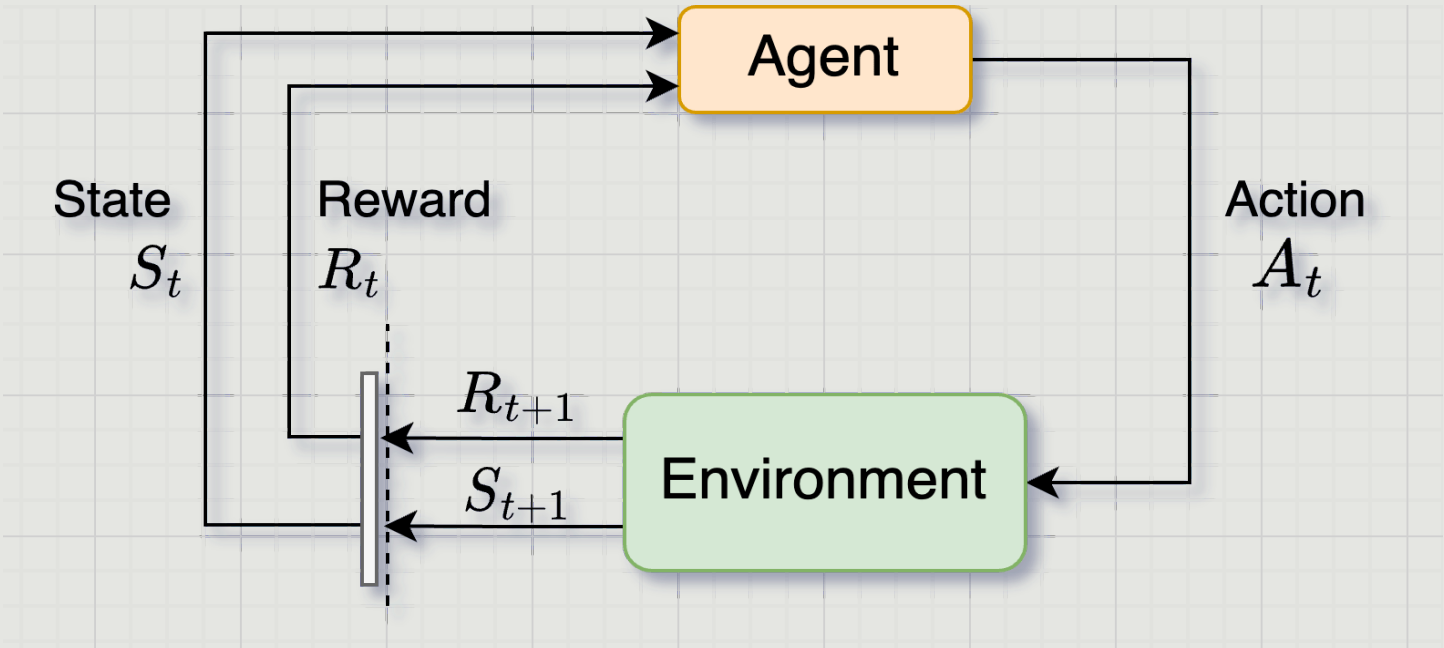
The setup can also be seen as a gambling problem. Suppose you have to choose among k different options or actions repeatedly. After each choice, you receive a numerical reward from a stationary probability distribution that depends on your selected action. Your objective is to maximize the expected total reward over some time, for example, over many action selections or time steps.

We want to determine the values of actions and use these estimates to make action selection decisions.

A natural way to estimate this is by averaging the rewards actually received:

$$Q_t(a) \triangleq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

A Markov Decision Process



The MDP and agent together give rise to a sequence or trajectory in time, such as:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_t, A_t, R_{t+1}, \ldots$$

In a finite MDP, the sets of states, actions, and rewards (S, A, and, R) all have a finite number of elements. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action.

Making a Policy

You may think that acting greedy would help create the best policy, but you are wrong!

The *greedy* action selection is defined as: $A_t \stackrel{\Delta}{=} \underset{a}{\operatorname{argmax}} Q_t(a)$

Greedy action-selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ϵ, instead select randomly from among all the actions with equal probability, independently of the action-value estimates. We call methods using this near-greedy action selection rule ϵ-greedy methods.

$\epsilon-$Greedy Logic

Bandit Algorithm Example

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \text{argmax}_a\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Two critical thinking questions:

1. How can you code this idea?
2. How can you simulate an environment to test it against a purely greedy policy?

Can the $\epsilon-$greedy method be improved?

Food for Thought: Central Limit Theorem and Confidence Intervals

If X is a random variable and we consider the simple random sampling of X with sample sizes greater than 30, then the distribution of the sample means is almost normally distributed.

$$\text{If the sample size } n, \text{increases, then } \bar{X} \sim N\left(\mu_X, \frac{\sigma_X}{\sqrt{n}}\right)$$

Big Idea: the random variable X may follow some unknown distribution; however, if we consider sampling from X with sample sizes that are big, then the distribution of the corresponding sample means is almost normal.

Recommended Instructional Video:
https://www.youtube.com/watch?v=oPQ4mNcqY7k&t=304

Can the $\epsilon-$greedy method be improved?

The Meaning of CLT in Probability

$$\mathrm{P}\left(\left|\bar{X}-\mu\right|<d\right)>1-\alpha$$

Here $1-\alpha$ is the confidence level.

We have that

$$\left|\bar{X}-\mu\right|<d$$

which means

$$\bar{X}-d<\mu<\bar{X}+d$$

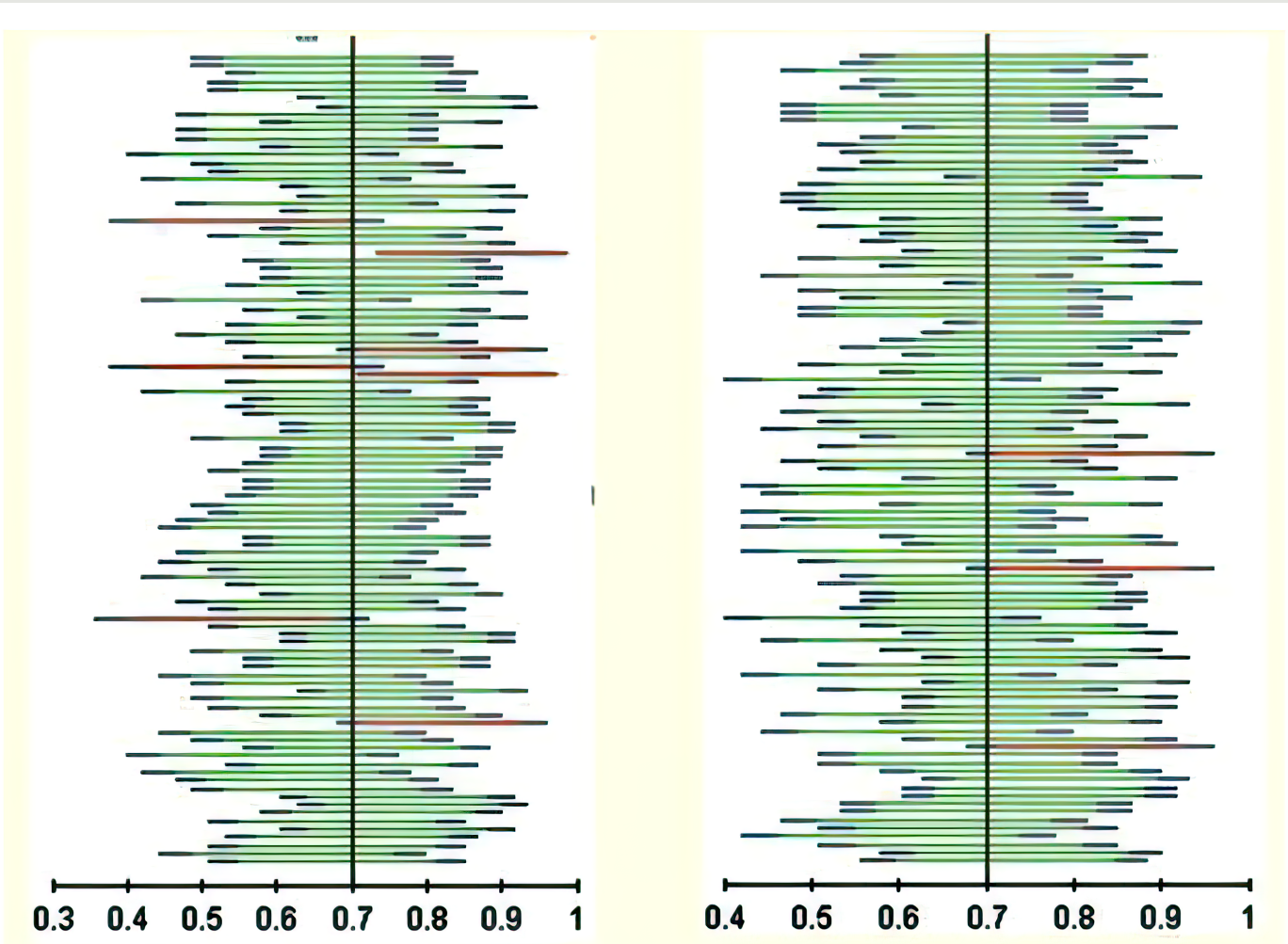
Here, d is called the margin of error and is sometimes denoted by E.

We call this distance d margin of error, and based on CLT, we compute d as follows:

$$d:=z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\text{ or }d:=t_{\alpha/2}\frac{s}{\sqrt{n}}$$

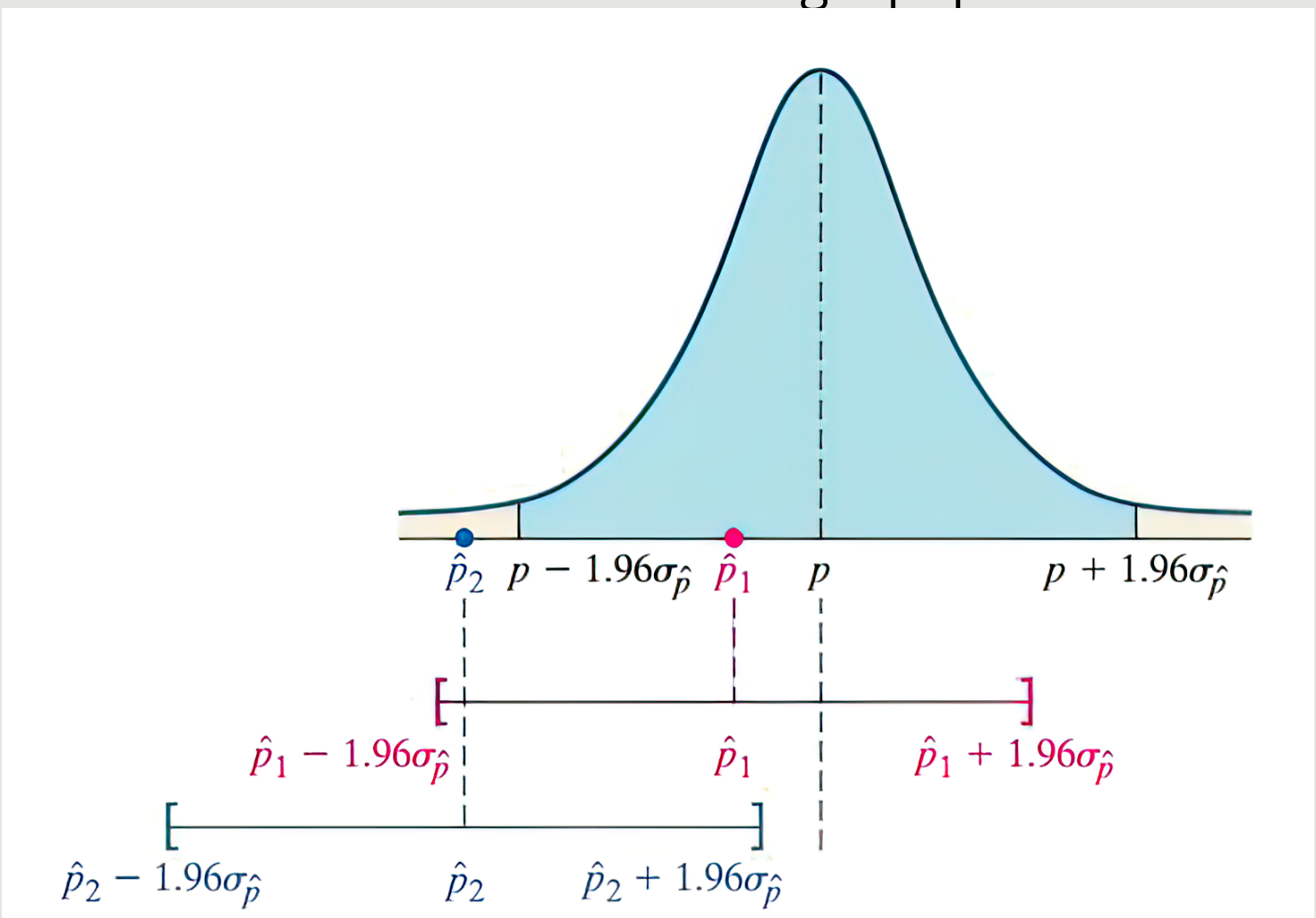
Confidence Intervals

The idea is to "trap" the value of a population mean inside an interval built around a sample mean.



Confidence Intervals

A picture representing this idea is the following, showing how a 95% confidence interval works for estimating a population proportion:



Chernoff Bounds

Theorem. Consider $X_1, \ldots, X_n$ defined by $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$. Assume all X_i are *independent.*, and that $X = \frac{1}{n} \sum_i X_i$, with $\mu = \frac{1}{n}(\sum_i \mu_i)$, then for all $t > 0$, we get the following bound

$$P(|X - \mu| > t) \leq 2e^{-2nt^2/(b-a)^2}$$

Theorem. Let $X_1, \ldots, X_n$ be independent random variables whose values are within some range $[a, b]$. Call $\mu_i = \mathbb{E}(X_i)$, $X = \sum_i X_i$, and $\mu = \mathbb{E}(X) = \sum_i \mu_i$. Then for all $t > 0$,

$$P(|X - \mu| > t) \leq 2e^{-2t^2/n(b-a)^2}$$

Critical Thinking: what would happen if we plug $t = c \cdot \sqrt{\frac{\log(k)}{n}}$ in the first equation?

Upper Confidence Bound Approximation

If we index the different stochastic games (machines) by i where $1 \leq i \leq d$ and we assume that the i^{th} machine was played n times, then the upper bound estimate for the confidence interval for the i^{th} machine is

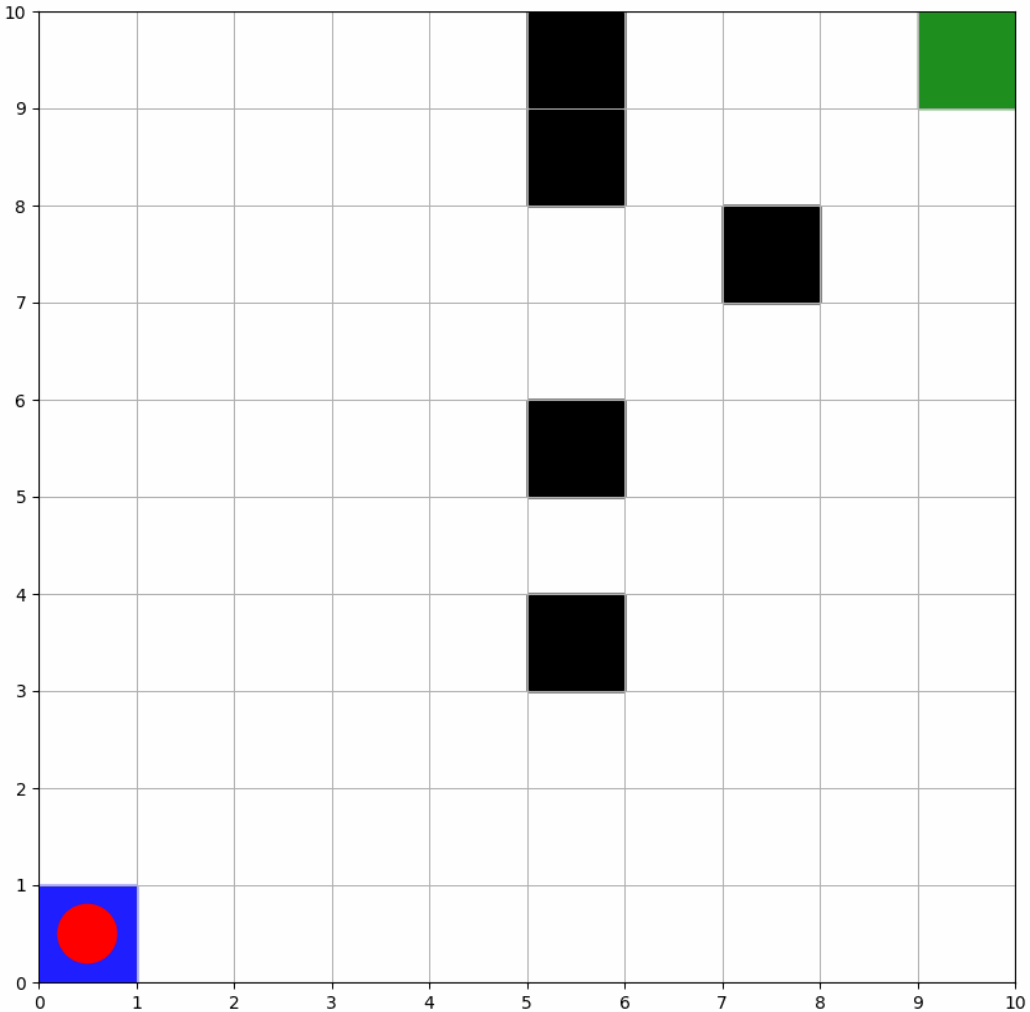
$$\bar{X}_i + c\sqrt{\frac{\ln(k)}{n}}$$

where k represents the total number of times any game was played so far.

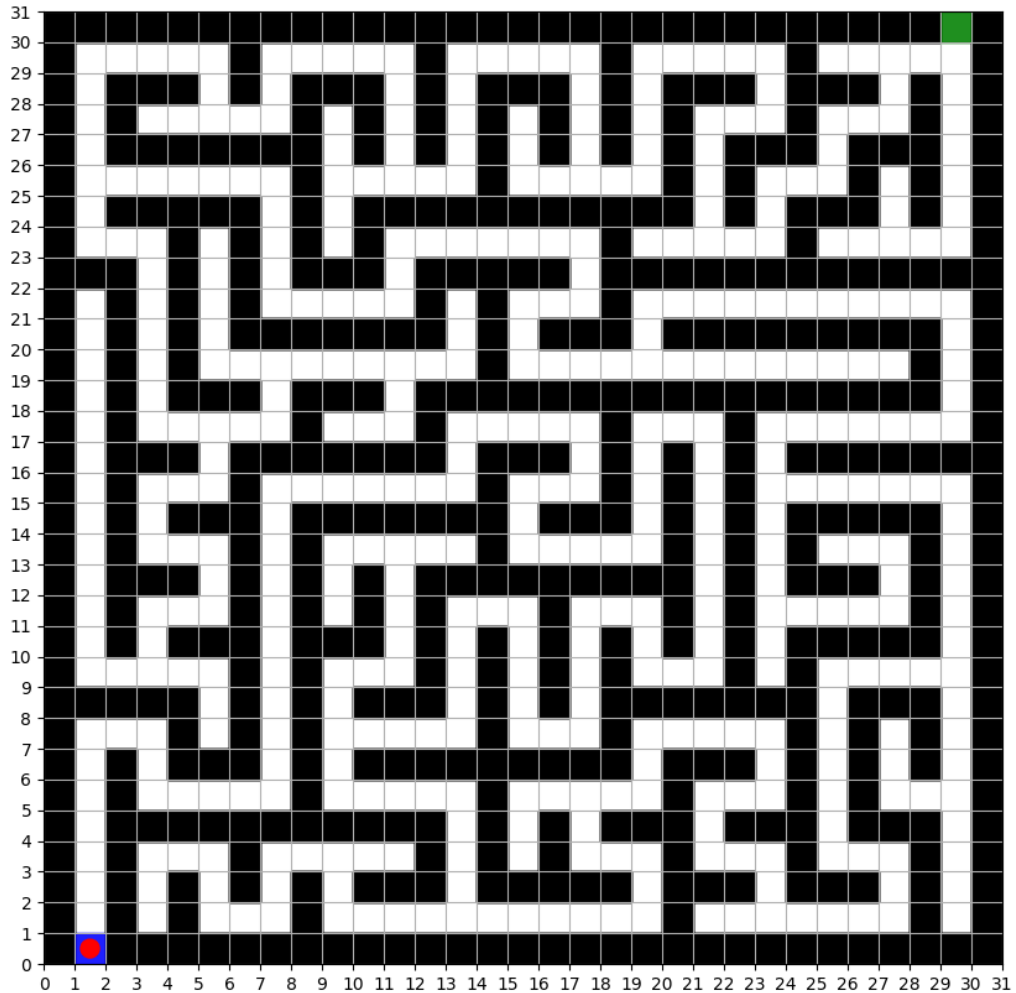
The more you play the same machine, the more the confidence level increases.

This method is sometimes referred as "The UCB - Bandit Algorithm"

What RL can do?



What RL can do?



How to update the Q-values

1. **Q-Learning**:
The agent learns a Q-value function, $Q(s, a)$, which represents the expected utility of taking action a in the state s and following the optimal policy thereafter.
The Q-value is updated using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

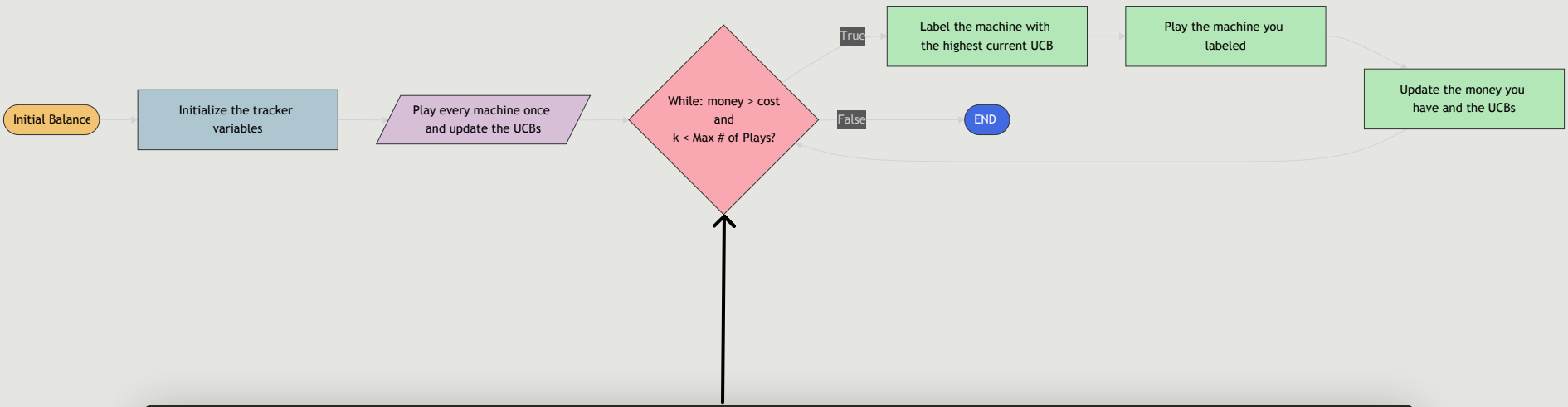
Here, α is the learning rate, γ is the discount factor, r is the reward, and s' is the next state.

2. **SARSA (State-Action-Reward-State-Action)**:
The Q-value is updated using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

The key difference from Q-Learning is that it uses the action a' chosen by the current policy for the next state.

Code Applications




```python
while (money > cost) & (k < max_number_of_plays):
    star = np.argmax(UCB)
    machine_record.append(star+1)
    n[star] = n[star] + 1
    k += 1
    reward = Casino(star+1) # sampling a reward from machine i+1
    money = money - cost + reward # this update the money you have
    R[star] = R[star] + reward # update the sum of the rewards
    UCB[star] = R[star]/n[star] + c*np.sqrt(np.log(k)/n[star]) # update th
```