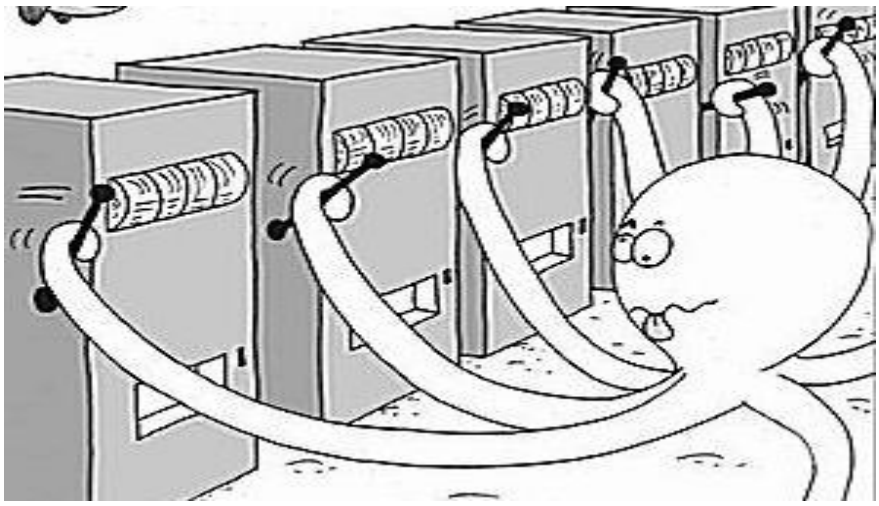
Explanation of the Above Concepts in Relation to Reinforcement Learning and the Associated Equations

This section provides a clear understanding of how the previously introduced concepts — such as the agent–environment interaction, states, actions, and rewards — form the foundation of Reinforcement Learning (RL). It also explains how these components are mathematically represented through the related equations that describe the agent's learning objective and behavior within the Markov Decision Process (MDP) framework.

1. Multi-Armed Bandit Problem(Exploration vs Exploitation)

Casino slot machines have a playful nickname - "one-armed bandit" - because of the single lever it has and our tendency to lose money when we play them. They also inspire the creativity of researchers.



Source: https://surl.li/fnitaf

Multi-Armed Bandit Problem

- **Meaning:**
  - A generalization of a slot machine ("one-armed bandit") to **multiple levers**, each with **different, unknown payout probabilities**.
- **Goal:**
  - Decide **which lever to pull** at each step to **maximize total reward** over time.
- **Challenge:**
  - You **don't know** which lever pays best without trying them.
  - Each pull gives limited information → must **balance learning and earning**.
- **Core Dilemma – Exploration vs. Exploitation:**
  - **Exploration:** Try different levers to discover payoffs.
  - **Exploitation:** Choose the lever that seems best so far.
  - Too much exploration = wasted pulls; too little = missed better rewards.

- **Historical Note:**
  - Originated during **World War II** in sequential decision research.
  - Considered so hard that scientists joked about **dropping it over Germany** to distract enemy researchers.
- **Modern Importance:**
  - Still a major research area (e.g., many papers at **NIPS 2015**).
  - Foundation of **Reinforcement Learning (RL)** — learning by **trial and error**, not labeled data.
- **Key Focus in RL:**
  - Addresses the **exploration–exploitation trade-off** — how to divide effort between discovering new options and maximizing known rewards.
- **Applications:**
  - Online ads, A/B testing, recommendation systems, and clinical trials.

Equations (action value & selection)

- **Sample-average estimate**:

$$Q_{t+1}(a) \leftarrow Q_t(a) + \frac{1}{N_t(a)}\left(R_t - Q_t(a)\right)$$

- **ε-greedy**: with prob. $1-\varepsilon$ choose $\arg\max_a Q_t(a)$, else random explore.
- **UCB1** (optimism via confidence term) (choose at time t):

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right].$$

(UCB family presented in the text as a principled exploration strategy.)

Variables

- k – number of arms (options)
- $a \in \{1, \ldots, k\}$ – an action/arm index.
- A_t – arm chosen at time t.
- R_t – reward received after pulling A_t.
- $Q_t(a)$ – current estimate of expected reward (mean payoff) for arm a.
- $N_t(a)$ – number of times arm a has been selected up to time t.

1 Sample-Average Update

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{N_t(a)}(R_t - Q_t(a))$$

- Updates the estimated value $Q_t(a)$ toward the newly observed reward.
- The step $1/N_t(a)$ ensures it becomes the **average** of all past rewards for that arm.
- Works best for **stationary** payoffs (means don't change).

2 ε-Greedy Action Selection

- With probability $1 - \varepsilon$: choose the **greedy arm** $a = \arg\max_a Q_t(a)$.
- With probability ε: pick a **random arm**.
- Balances **exploitation** (use best-known arm) and **exploration** (try others).
- Simple and widely used baseline policy.

3 UCB1 (Upper Confidence Bound)

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- Chooses arm with **highest optimistic estimate**.
- $Q_t(a)$: current mean reward (exploitation).
- $c\sqrt{\frac{\ln t}{N_t(a)}}$: exploration bonus — bigger for less-tried arms.
- c: controls exploration strength.
- Ensures **directed exploration** and gives **low regret** for stationary problems.

Summary:

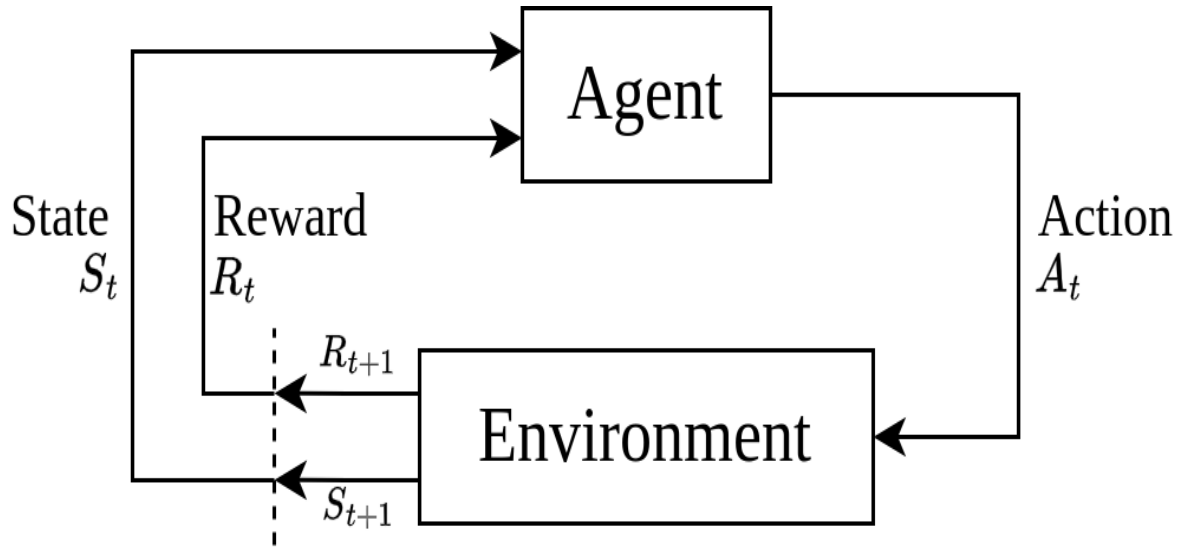
- $Q_t(a)$: learns expected payoff (by averaging rewards).
- **ε-greedy**: random exploration.
- **UCB**: confidence-based exploration.

  Together, they capture the **exploration–exploitation trade-off** central to reinforcement learning.

2. Finite Markov Decision Process(MDP)

Introduction – Finite Markov Decision Processes (MDPs)

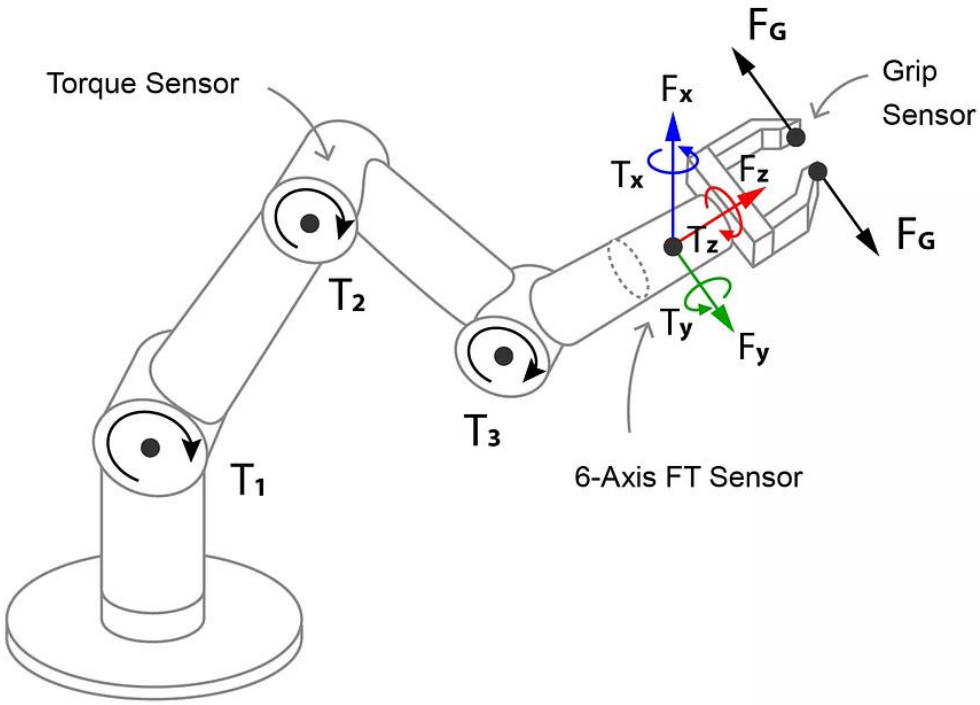
- **Concept:**
  - Extension of the **Multi-Armed Bandit** problem.
  - Decisions depend not only on actions but also on the **current state (context)**.
- **Purpose:**
  - Models **sequential decision-making**, where each action affects:
    - **Immediate rewards**, and
    - **Future states** and their rewards.
- **Core Idea:**
  - Agent must **balance immediate vs. delayed rewards** to achieve **optimal long-term performance**.
- **Difference from Bandits:**
  - Bandits estimate a single value per action → $Q(a)$.
  - MDPs estimate a value **per state–action pair** → $q^*(s, a)$.
  - Alternatively, they can estimate **state value** assuming optimal actions → $v^*(s)$.
- **Goal:**
  - Learn accurate **value estimates** ($v^*(s)$, $q^*(s, a)$) to assign **credit** for actions whose outcomes occur over time.



Source: The Agent-Environment interaction process in an MDP (Adapted from Sutton and Barto Fig 3.1)

The Agent–Environment Interface (MDP Framework)

- 1. **Agent:** The learner and decision-maker that selects actions to **maximize cumulative rewards** over time.
- 2. **Environment:** Everything external that **responds** to the agent's actions by providing **new states and rewards.**
- 3. **State** (S_t): The **current situation** or observation the agent receives from the environment at time t.
- 4. **Action** (A_t): The **decision** the agent makes based on the current state.
- 5. **Reward & Transition:** After action A_t, the agent receives a **reward** (R_{t+1}) and moves to a **new state** (S_{t+1}), continuing the interaction cycle.



Source : A robot arm with force and torque sensors forms an MDP where sensor readings represent the state, and actions control the arm's movement to accomplish tasks like grasping an object. ***(Adapted from the Reach Robotics Blog.)***

1 **Sensors and Measurements:**

The robotic arm is equipped with **torque sensors** (T_1, T_2, T_3) and a **6-axis force/torque (FT) sensor** that measures forces (F_x, F_y, F_z) and torques (T_x, T_y, T_z) at the end-effector, along with a **grip sensor** to detect gripping force (**F_G**).

2 **Function and Feedback:**

These sensors provide **real-time feedback** on the mechanical interaction between the arm and its environment—detecting load, contact pressure, and orientation during movement or object manipulation.

3 **Relevance to RL and MDPs:**

The sensor readings define the **state** of the system, while control commands (joint torques or motor inputs) represent **actions**. The goal (e.g., successful object grasp) serves as the **reward**, forming an **MDP framework** for learning optimal control policies.

MDP→

Core concept

Sequential decision problems where the **next state and reward depend only on current state and action** (Markov property). Defines the RL interface.

Variables

- $\mathcal{S}$: set of states, $s \in \mathcal{S}$
- $\mathcal{A}(s)$: actions available in state s
- $p(s', r \mid s, a)$: dynamics (transition/reward model)
- $\pi(a \mid s)$: policy (probability of action a in state s)
- $\gamma \in [0, 1)$: discount factor
- $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$: return
- $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$, $q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

Equations (Bellman)

- **Bellman expectation (state values):**

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma v_\pi(s') \big].$$

- **Bellman optimality:**

$$v_*(s) = \max_a \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma v_*(s') \big],$$

$$q_*(s, a) = \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma \max_{a'} q_*(s', a') \big].$$

(Uniqueness of solution for finite MDPs.)

Variables and Their Meaning (MDP Section)

- $\mathcal{S}$ – *Set of states*
  All possible situations the agent can be in.
  → Example: robot's location in a grid.
- $s \in \mathcal{S}$ – *Current state*
  Represents the environment's condition at time t.
- $\mathcal{A}(s)$ – *Set of actions available in state s*
  What the agent can do in that specific state.
  → Example: move left, right, up, or down.
- $p(s', r \mid s, a)$ – *Transition probability function*
  Probability of landing in next state s' and receiving reward r after taking action a in state s.
  Defines the **environment's dynamics**.
- $\pi(a \mid s)$ – *Policy*
  Probability of taking action a when in state s.
  Determines the agent's **behavior strategy**.
- $\gamma \in [0, 1)$ – *Discount factor*
  Controls importance of future rewards.
  - $\gamma = 0$: only immediate rewards matter.
  - $\gamma \to 1$: long-term rewards valued more.
- $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ – *Return*
  Total discounted reward starting at time t.
  → Measures long-term gain of the current policy

- $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$ – *State-value function*
  Expected return if the agent starts in state s and follows policy π.
  → Answers: "How good is it to be in state s?"
- $q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$ – *Action-value function*
  Expected return if the agent takes action a in state s and then follows policy π.
  → Answers: "How good is it to take action a in state s?"

Bellman Equations

Bellman Expectation Equation

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$

- Defines how the value of a state equals the expected immediate reward **plus** the discounted value of the next state.
- Expresses **recursive dependency** between state values.

Bellman Optimality Equations

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]$$

- Describe the **optimal value functions**, assuming the best possible actions are always chosen.
- v_* and q_* define the **foundation of optimal control** in reinforcement learning.

3. Dynamic Programming(DP) in RL

Dynamic Programming (DP) in Reinforcement Learning is a **model-based approach** used when the environment's transition dynamics $p(s', r \mid s, a)$ are fully known. It provides a systematic way to compute the **optimal policy** by iteratively evaluating and improving policies. The idea is to start with any policy π, compute its value function (policy evaluation), then generate a new improved policy (policy improvement) that acts greedily with respect to those values. Repeating this process until convergence yields the **optimal policy** π^* and **optimal value function** $v^*(s)$. This iterative combination of evaluation and improvement is called **policy iteration**, while a faster variant that combines both in a single step is known as **value iteration**.

The **policy evaluation equation**

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$

computes the expected value of each state under the current policy. The **policy improvement step**

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$

creates a new greedy policy that maximizes expected returns. **Value iteration** merges both steps, using

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$

to directly approach the optimal value function without separately evaluating a full policy. DP assumes a perfect model and a finite state–action space, making it computationally expensive for large problems but foundational for understanding later RL methods such as **Monte Carlo** and **Temporal-Difference (TD)** learning, which approximate these principles from experience rather than full models.