Smart Taxi

MUKESH KUMAR

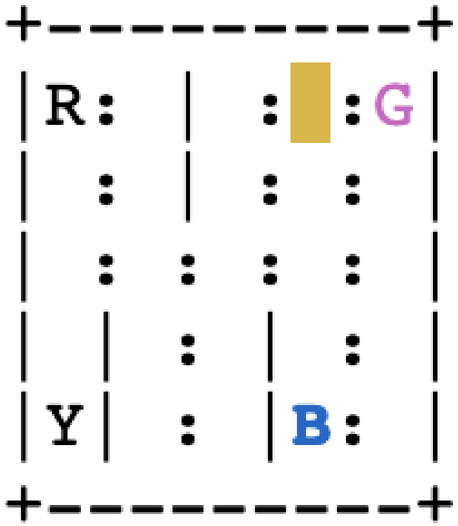
Reinforcement Learning with OpenAI

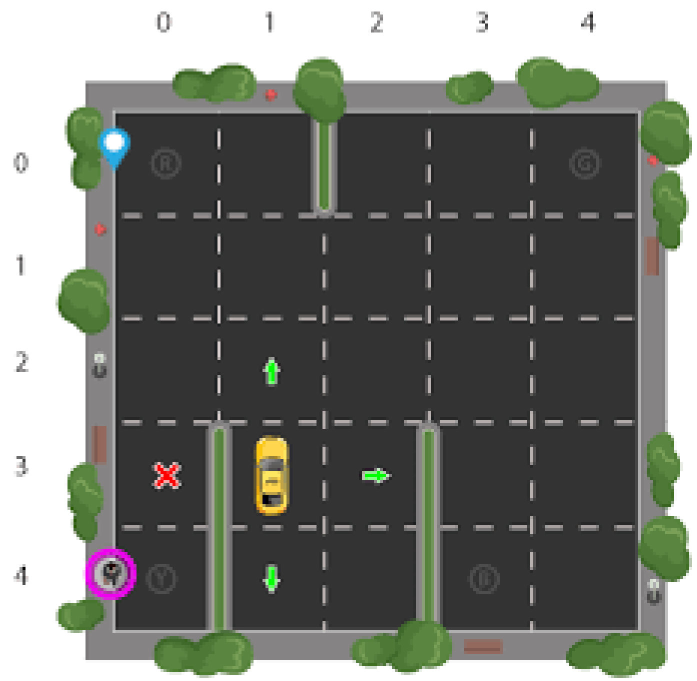
- OpenAI Gym: Simulated environments for RL experimentation
- Example Environments: CartPole, MountainCar, LunarLander, Atari

Smart Taxi

Smart Taxi

- There are four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5x5 grid world.
- https://gymnasium.farama.org/environments/toy_text/taxi/





How many states and actions??

- Please refer : Smart-Taxi-Observation-Space.ipynb

We will solve Taxi problem using:

- Q-Learning (off-policy)
- SARSA (on-Policy)

SARSA (State-Action-Reward-State-Action)

SARSA learns the value of the current state-action pair **based on the actual action the agent took next,** following its current policy (usually ε-greedy).
It updates the Q-value using the reward received and the Q-value of the **next state-action** pair.

Full Form:

- **S**tate, **A**ction, **R**eward, **S**tate (next), **A**ction (next)
  These 5 components are used in each update step.

SARSA Update Rule

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

Where:

- s = current state
- a = current action
- r = reward
- s′ = next state
- a′ = **next action actually taken (not best)**
- α = learning rate
- γ = discount factor

SARSA Intuition:

SARSA doesn't assume the agent will always behave optimally.

It **learns from what it actually did,** which may include mistakes or exploration.

This makes it more **realistic and safer** in unpredictable environments.

Common Use Cases:

- Cliff Walking (to avoid risky shortcuts)
- Flappy Bird (where one mistake = death)
- Driving/Racing Games (stochastic environments)
- Real-world robotics (with imperfect control)

Exploration vs Exploitation

Exploitation

- Choosing the action that has **highest known reward**
- Based on past experience
- Short-term gain

Exploration vs Exploitation

Exploration

- Trying new actions to **discover potentially better rewards**
- Risky, but essential for long-term improvement
- Long-term success

Exploration vs Exploitation

Balancing Act

- Too much exploration → inefficiency
- Too much exploitation → stuck in suboptimal policies

Common Strategy:

- **ε-Greedy:** With probability ε, explore; otherwise, exploit.

What is ε-Greedy?

ε-Greedy is a method that balances **exploration** (trying new things)

and **exploitation** (choosing what seems best) when selecting actions.

- **Why we need it:**
- If the agent always picks the **best known action** (pure exploitation),
- it might **miss out on better options**.
- If it always explores, it **never settles** on what’s best.
- ε-Greedy gives us a **controlled tradeoff** between the two.

How it works

- At each time step:

```
if random_number < ε:
    choose a random action    # 🎲 Explore
else:
    choose the best action    # 🧠 Exploit
```

- With probability ε (like 0.1), the agent picks a random action → **exploration**
- With probability 1 - ε, it picks the best known action (i.e., action with highest Q-value) → **exploitation**

Example

Suppose:

- Available actions: ['left', 'right', 'up', 'down']
- ε = 0.1

Then:

- 10% of the time, the agent will choose randomly (even if it’s not optimal).
- 90% of the time, it will choose the **action with the highest Q-value** in the current state.