

# **Lecture 5**

## **Model-Free Methods**

### **Monte Carlo / TD**

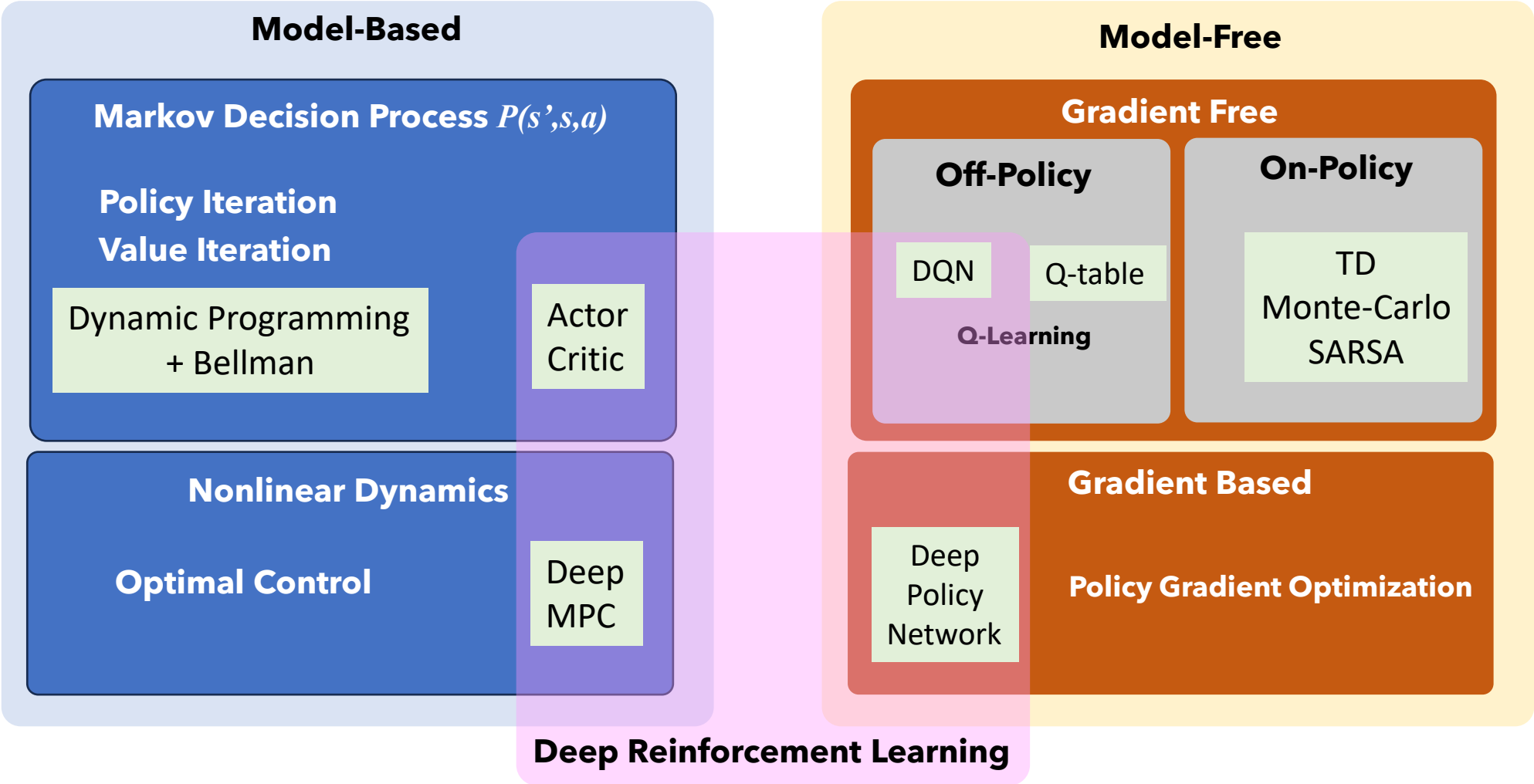
[jmanero@faculty.ie.edu](mailto:jmanero@faculty.ie.edu)

- **Classification of RL methods**
- **Monte-Carlo**
- **Monte-Carlo method for RL**
- **Temporal Difference (TD) methods**
- **Wrap-up**

# **Classification of RL methods**

# Where we are

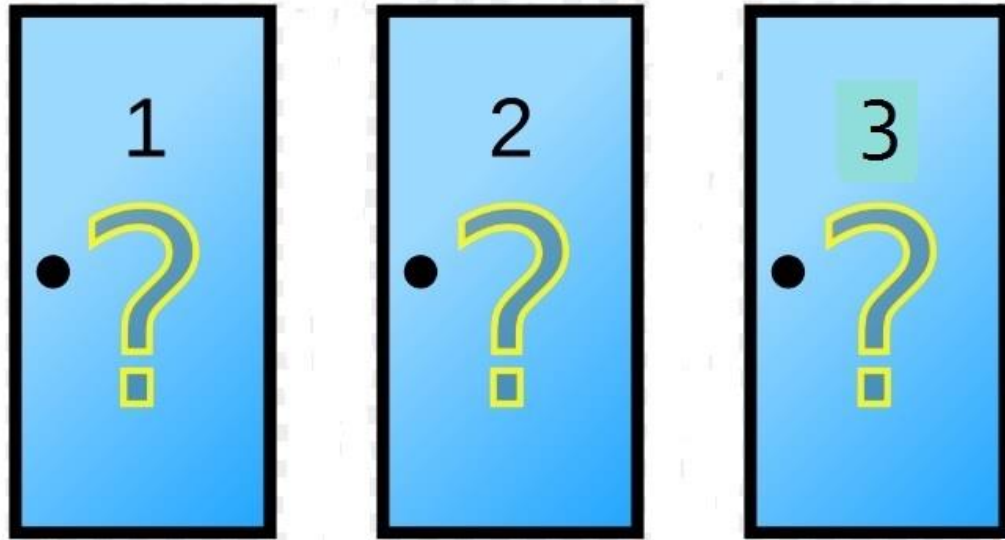
## Classification of RL Methods



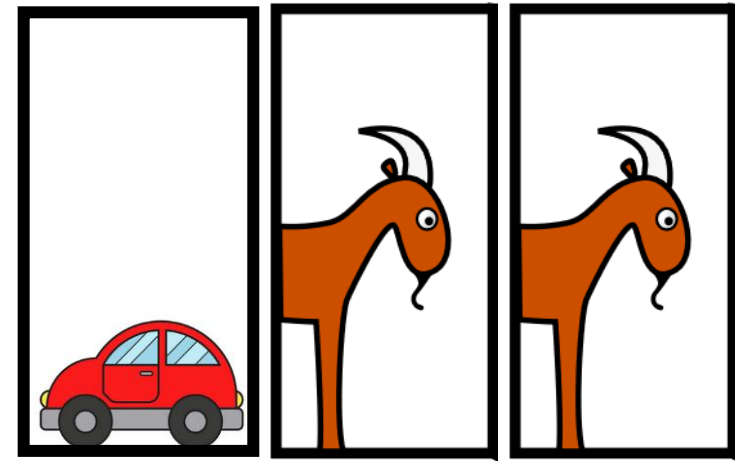
# Game time

# Monte-Carlo method

## Monty Hall problem



Suppose you're on a game show  
You're given the choice of three doors

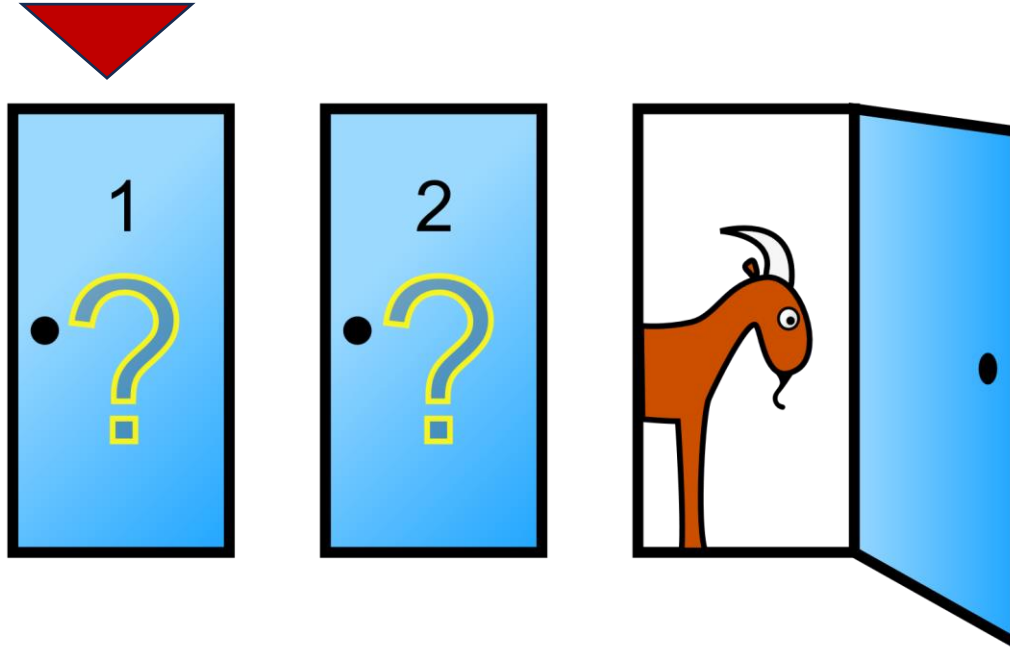


Behind the doors there are 2 goats  
and 1 car

# Monte-Carlo method

## Monty Hall problem

Door Chosen



In search of a new car, the player chooses a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player switch from door 1 to door 2.

Switch or Stay?



# What is better. To stay in 1 or switch to 2?

# Monte-Carlo method

## Monty Hall problem

- Steve Selvin wrote a letter to the American Statistician in 1975, describing a problem based on the game show Let's Make a Deal, dubbing it the "Monty Hall problem" in a subsequent letter. The problem is equivalent mathematically to the Three Prisoners problem described in Martin Gardner's "Mathematical Games" column in Scientific American in 1959 and the Three Shells Problem described in Gardner's book Aha Gotcha
- Marilyn vos Savant's proposed and demonstrated that switching is better in the Parade magazine
- But many people refused to believe switching is beneficial and rejected her explanation. After the problem appeared in *Parade*, approximately 10,000 readers, including nearly 1,000 with PhDs, wrote to the magazine, most of them calling Savant wrong. Even when given explanations, simulations, and formal mathematical proofs, many people still did not accept that switching is the best strategy.
- Paul Erdős, one of the most prolific mathematicians in history, remained unconvinced until he was shown a computer simulation demonstrating Savant's predicted result

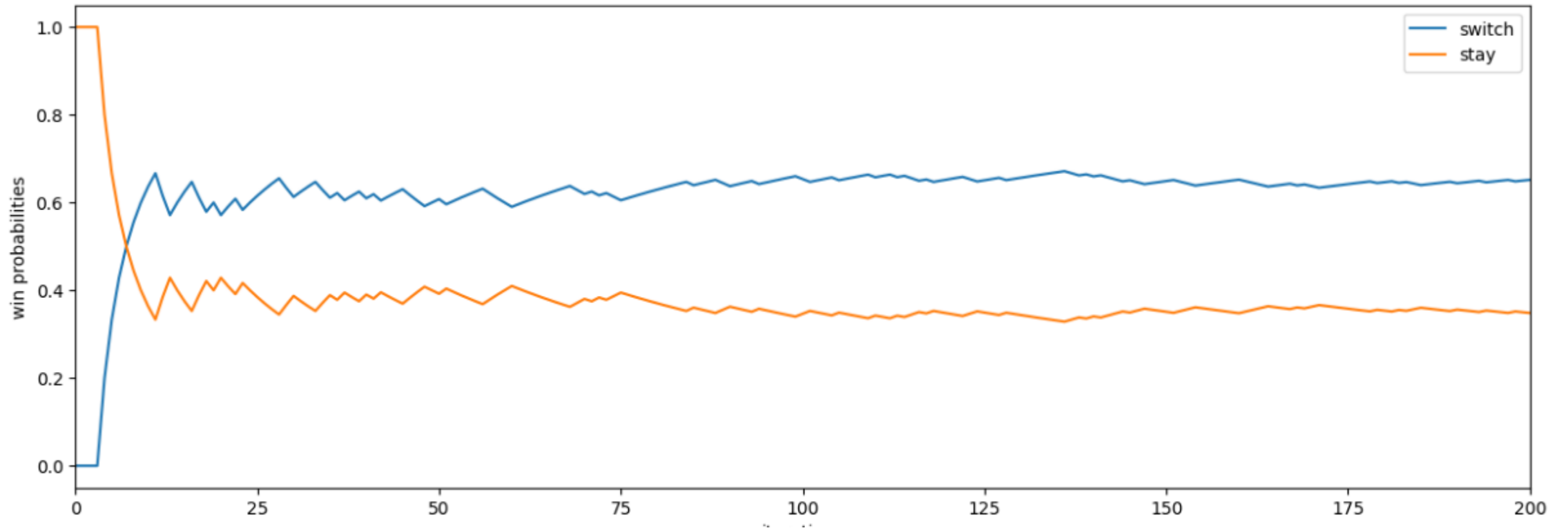
*Behind Monty Hall's Doors: Puzzle, Debate and Answer?- New York Times 21 July 1991*



# Monte-Carlo method

## Simulating the choices

018\_MC\_simulation\_MontyHall.ipynb



# Monte-Carlo

- **What are the Problems with Dynamic Programming methods?**
  - Sweep of full steps or random steps
  - **Need to know the model**

We'll see now methods that do not require a model but only *experiences* to build evaluations of policies and also to find optimal policies

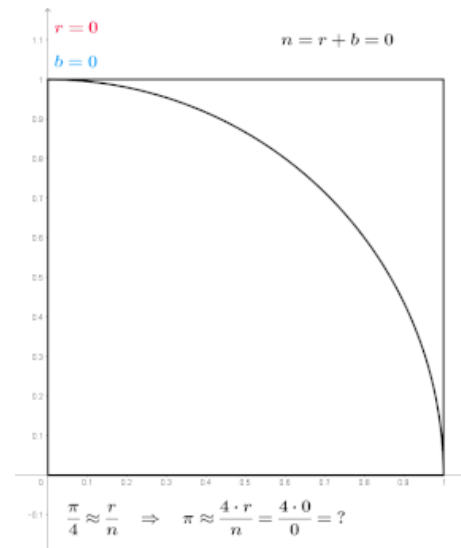
- **Methods we will see:**
  - Monte-Carlo
  - Temporal differences; n-steps and TD( $\lambda$ )
  - Sarsa, Expected Sarsa
  - Q-learning

# Monte-Carlo method

## A sampling method



- Monte Carlo methods were created by Stanislaw Ulam inspired by his uncle's gambling issues
- Are used extensively in computer simulations and optimization methods
- They are based in the following process
  1. Define possible inputs
  2. Generate inputs randomly
  3. Use inputs to generate the outputs



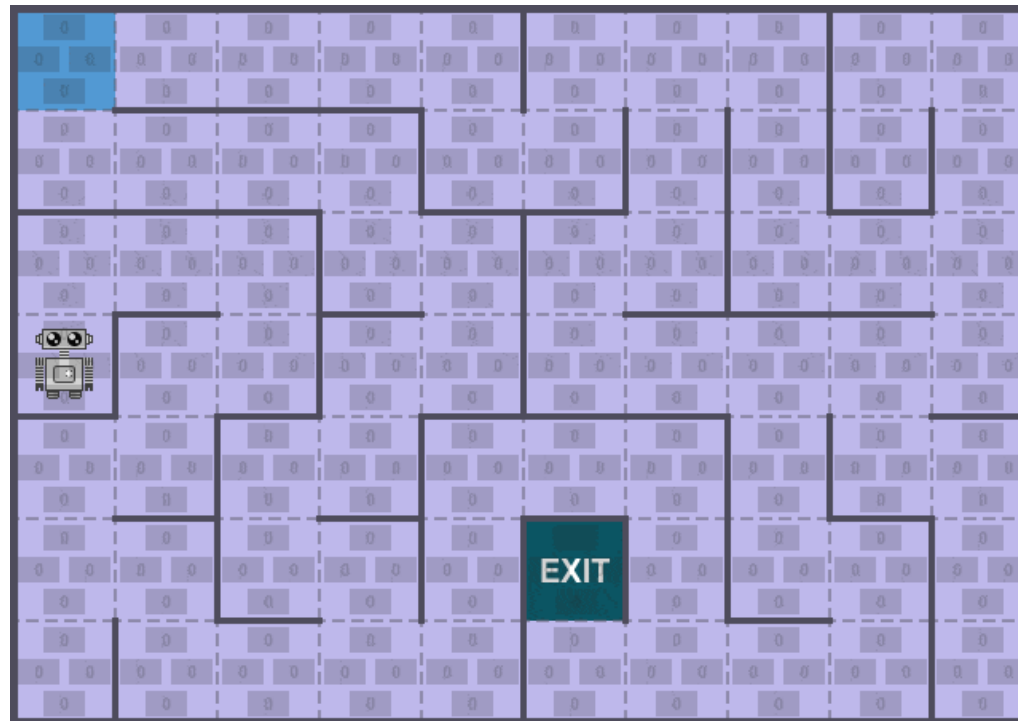
### Calculate pi using a Monte-Carlo Method

Count the number of points inside the quadrant, i.e. having a distance from the origin of less than 1  
The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas,  $\pi/4$ . Multiply the result by 4 to estimate  $\pi$ .

# Monte-Carlo method for RL

## What defines a Monte-Carlo method

- Learns directly from episodes of experience
- It is **MODEL-FREE** There is no understanding of transitions or rewards
- MC learns from complete episodes
- The idea is very basic  $V(s) = \text{mean\_return}$
- Must be applied only to MDP episodic, ie: episodes must finish
- No model is given the agent must discover the properties of the environment through exploration...



# Monte-Carlo method

## Policy Evaluation in Monte-Carlo

- Goal: learn  $v_\pi$  from episodes of experience under policy  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

# Monte-Carlo method

## Policy Evaluation in Monte-Carlo FIRST VISIT

- To evaluate state  $s$
- The **first** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$



# Monte-Carlo method

## Policy Evaluation in Monte-Carlo EVERY VISIT

- To evaluate state  $s$
- **Every** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- Again,  $V(s) \rightarrow v_{\pi}(s)$  as  $N(s) \rightarrow \infty$

# Monte Carlo Method

## Monte-Carlo Prediction algorithm FIRST VISIT

### Monte-Carlo Prediction Algorithm First Visit (Sutton)

Initialize:

$\pi \leftarrow$  policy to be evaluated

$V \leftarrow$  an arbitrary state-value function

$Returns(s) \leftarrow$  an empty list, for all  $s \in S$

Repeat Forever:

Generate an episode using  $\pi$

**for each** state  $s$  appearing in the episode:

$G \leftarrow$  return following the first occurrence of  $s$

Append  $G$  to  $Returns(s)$

$V(s) \leftarrow$  average ( $Returns(s)$ )

An obvious way to estimate it from experience, then, is simply to average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value. This idea underlies all Monte Carlo methods.

The first-visit MC method estimates  $v(s)$  as the average of the returns following first visits to  $s$ , whereas the every-visit MC method averages the returns following all visits to  $s$ .

# Monte Carlo RL Methods

## Monte-Carlo Prediction algorithm EVERY VISIT

### Monte-Carlo Prediction Algorithm Each Visit (Sutton)

Initialize:

$\pi \leftarrow$  policy to be evaluated

$V \leftarrow$  an arbitrary state-value function

$Returns(s) \leftarrow$  an empty list, for all  $s \in S$

Repeat Forever:

Generate an episode using  $\pi$

**for each** state  $s$  appearing in the episode **do**

$G \leftarrow$  return following each occurrence of  $s$

Append  $G$  to  $Return(s)$

$V(s) \leftarrow$  average ( $Returns(s)$ )

# Monte Carlo RL Methods

## Comparison FIRST VISIT and EVERY VISIT

Properties:

- First-visit Monte Carlo
  - $V^\pi$  estimator is an unbiased estimator of true  $\mathbb{E}_\pi[G_t | s_t = s]$
  - By law of large numbers, as  $N(s) \rightarrow \infty$ ,  $V^\pi(s) \rightarrow \mathbb{E}_\pi[G_t | s_t = s]$
- Every-visit Monte Carlo
  - $V^\pi$  every-visit MC estimator is a **biased** estimator of  $V^\pi$
  - But consistent estimator and often has better MSE

# Monte Carlo RL Method

## Monte-Carlo Prediction algorithm EVERY VISIT

- Consistency: with enough data, does the estimate converge to the true value of the policy?
- Computational complexity: as get more data, computational cost of updating estimate
- Memory requirements
- Statistical efficiency (intuitively, how does the accuracy of the estimate change with the amount of data)
- Empirical accuracy, often evaluated by mean squared error

# Monte-Carlo method

## Monte-Carlo on $V(s)$ and $Q(s,a)$

If a model is not available, then it is particularly useful to estimate action values (the values of state{action pairs) rather than state values. With a model, state values alone are sufficient to determine a policy; one simply looks ahead one step and chooses whichever action leads to the best combination of reward and next state, as we did in the chapter on DP. Without a model, however, state values alone are not sufficient.

We use a state, action table, a Q-Table

# Monte-Carlo method

## Monte-Carlo on $V(s)$ and $Q(s,a)$

- Monte-Carlo can optimize  $V(s)$  : Value Function or  $Q(s,a)$ : state-action value function

### Monte-Carlo Control - Q Table - (Sutton)

Initialize, for all  $s \in S, a \in A(s)$

$Q(s, a) \leftarrow$  arbitrary

$\pi(s) \leftarrow$  arbitrary

$Returns(s, a) \leftarrow$  empty list

Repeat Forever:

Choose  $S_0 \in \delta$  and  $A_0 \in A(S_0)$  s.t. all pairs have probability  $> 0$

Generate an episode starting from  $S_0, A_0$ , following  $\pi$

**for each** pair  $s, a$  appearing in the episode:

$G \leftarrow$  return following the first occurrence of  $s, a$

Append  $G$  to  $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

**for each**  $s$  in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

<https://medium.com/intro-to-artificial-intelligence/relationship-between-state-v-and-action-q-value-function-in-reinforcement-learning-bb9a988c0127>

<https://twice22.github.io/rl-part4/>

# Monte-Carlo method

## Main Loop

[https://github.com/castorgit/RL\\_course/blob/main/018\\_MC\\_Frozenlake.ipynb](https://github.com/castorgit/RL_course/blob/main/018_MC_Frozenlake.ipynb)

```
for episode in range(n_episodes):
    state, _ = env.reset()
    epsilon = max(epsilon_min, epsilon_max - (epsilon_max - epsilon_min) * (episode / n_episodes))
    episode_states, episode_actions, episode_rewards = [], [], []

    for step in range(max_steps):
        action = epsilon_greedy_policy(state, epsilon)

        next_state, reward, done, _, _ = env.step(action)

        episode_states.append(state)
        episode_actions.append(action)
        episode_rewards.append(reward)

        if done:
            break

    state = next_state
```



# Monte-Carlo method

## Replay and update Q-Table

[https://github.com/castorgit/RL\\_course/blob/main/018\\_MC\\_Frozenlake.ipynb](https://github.com/castorgit/RL_course/blob/main/018_MC_Frozenlake.ipynb)

```
# Calculate returns and update Q-table
G = 0
for t in range(len(episode_states) - 1, -1, -1):
    state = episode_states[t]
    action = episode_actions[t]
    G = gamma * G + episode_rewards[t]

    N[state, action] += 1
    Q[state, action] += (alpha * (G - Q[state, action]))

total_rewards.append(sum(episode_rewards))

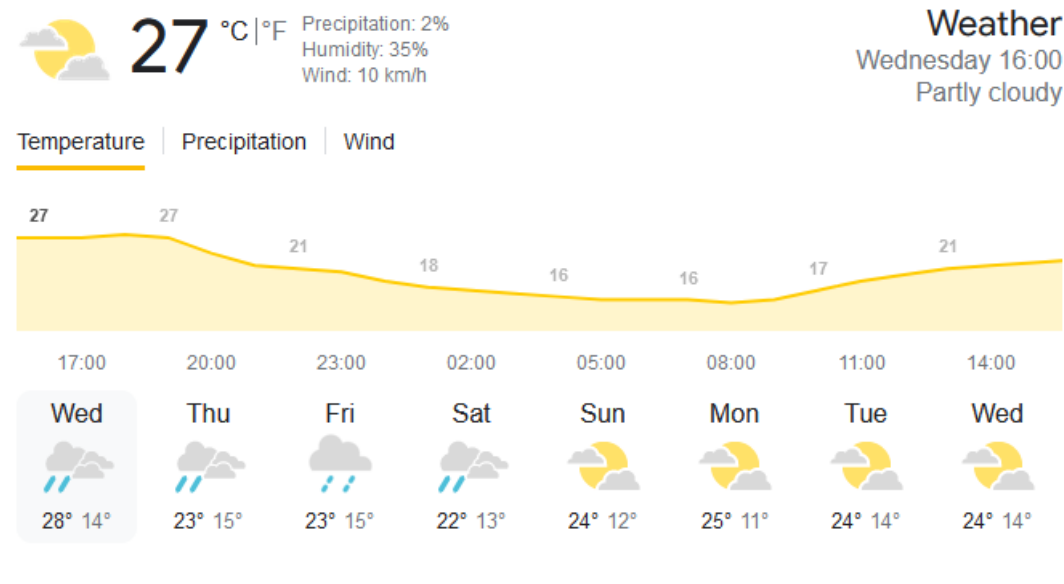
# Print progress
if (episode + 1) % 1000 == 0:
    avg_reward = np.mean(total_rewards[-1000:])
    print(f"Episode: {episode + 1}, Average Reward (last 1000 episodes): {avg_reward:.2f}")
```

# Temporal Difference Methods

# Monte-Carlo method

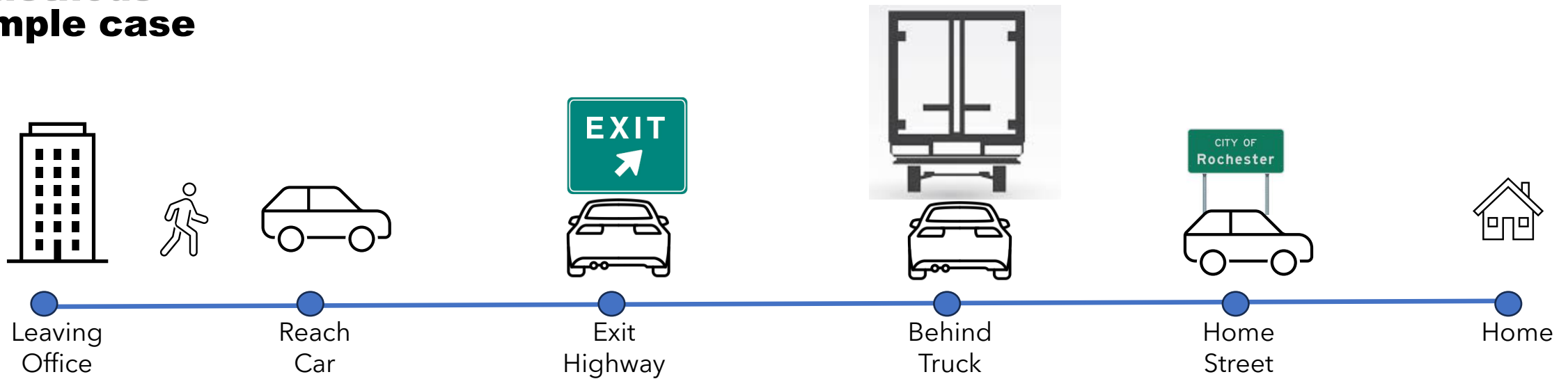
## Temporal Difference – naïve idea

Suppose you wish to predict the weather for Saturday, and you have some model that predicts Saturday's weather, given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your models. However, when it is, for example, Friday, you should have a pretty good idea of what the weather would be on Saturday – and thus be able to change, say, Saturday's model before Saturday arrives.



# TD methods

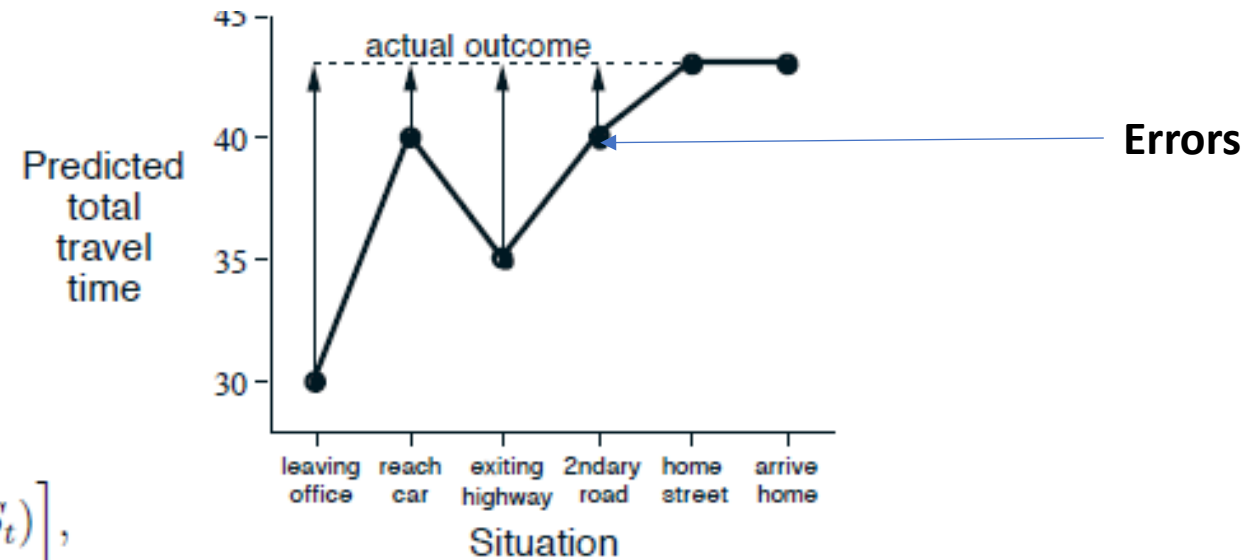
## A simple case



- When you leave the office the estimate says 30 minutes trip
- Starts to rain, the estimate when you reach car and start driving is 35 minutes
- You go very well, and then when you exit highway is again 35 minutes
- Then you must follow a slow truck and prediction was 40 min
- Entering home street prediction is 43
- Home at 43 minutes

| <i>State</i>                | <i>Elapsed Time<br/>(minutes)</i> | <i>Predicted<br/>Time to Go</i> | <i>Predicted<br/>Total Time</i> |
|-----------------------------|-----------------------------------|---------------------------------|---------------------------------|
| leaving office, friday at 6 | 0                                 | 30                              | 30                              |
| reach car, raining          | 5                                 | 35                              | 40                              |
| exiting highway             | 20                                | 15                              | 35                              |
| 2ndary road, behind truck   | 30                                | 10                              | 40                              |
| entering home street        | 40                                | 3                               | 43                              |
| arrive home                 | 43                                | 0                               | 43                              |

## What would Monte-Carlo do?

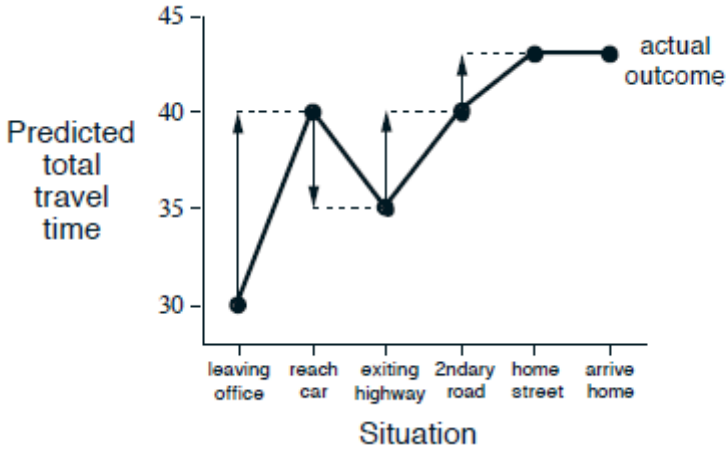


$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

| <i>State</i>                | <i>Elapsed Time<br/>(minutes)</i> | <i>Predicted<br/>Time to Go</i> | <i>Predicted<br/>Total Time</i> |
|-----------------------------|-----------------------------------|---------------------------------|---------------------------------|
| leaving office, friday at 6 | 0                                 | 30                              | 30                              |
| reach car, raining          | 5                                 | 35                              | 40                              |
| exiting highway             | 20                                | 15                              | 35                              |
| 2ndary road, behind truck   | 30                                | 10                              | 40                              |
| entering home street        | 40                                | 3                               | 43                              |
| arrive home                 | 43                                | 0                               | 43                              |

# Monte-Carlo method

## Driving Home Example TD METHOD

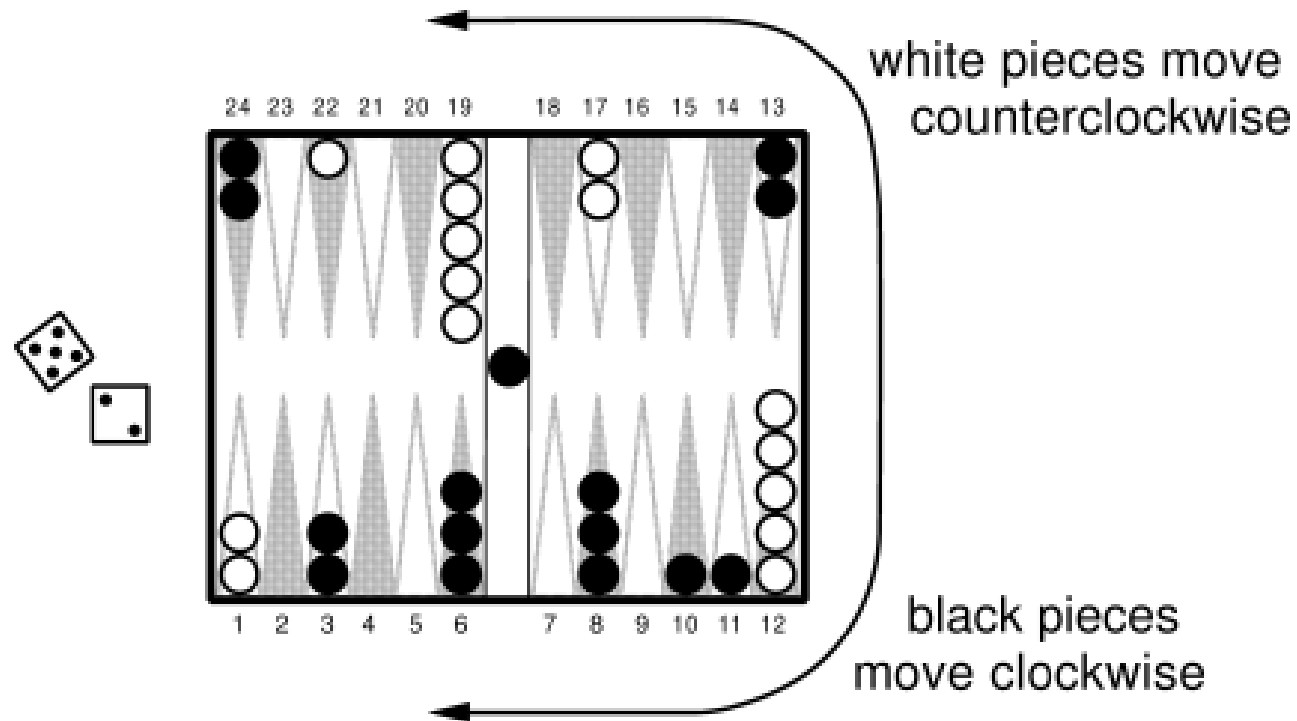


| State   | Elapsed Time<br>(minutes) | Predicted<br>Time to Go | Predicted<br>Total Time |
|---|---------------------------|-------------------------|-------------------------|
| leaving office                                    | 0                         | 30                      | 30                      |
| reach car, raining <sup><math>\sigma</math></sup> | 5                         | 35                      | 40                      |
| exit highway                                      | 20                        | 15                      | 35                      |
| behind truck                                      | 30                        | 10                      | 40                      |
| home street                                       | 40                        | 3                       | 43                      |
| arrive home                                       | 43                        | 0                       | 43                      |

- Combination of Monte Carlo & dynamic programming methods
- Model-free
- **Bootstraps and samples**
- Can be used in episodic or infinite-horizon non-episodic settings
- Immediately updates estimate of  $V$  after each  $(s, a, r, s')$  tuple
- Biased estimator (early on will be influenced by initialization, and won't be unbiased estimator)
- Generally lower variance than Monte Carlo policy evaluation
- Consistent estimator if learning rate  $\alpha$  satisfies same conditions specified for incremental MC policy evaluation to converge
- **Note: algorithm I introduced is TD(0). In general can have approaches that interpolate between TD(0) and Monte Carlo approach**

# TD Methods

## TD-gammon (1991)



This program, TD-Gammon, required little backgammon knowledge, yet learned to play extremely well, near the level of the world's strongest grandmasters. The learning algorithm in TD-Gammon was a straightforward combination of the TD( $\lambda$ ) algorithm and nonlinear function approximation using a multilayer neural network trained by backpropagating TD errors.



# Monte-Carlo method

## Temporal Difference

- Temporal-Difference puts together the ideas of dynamic Learning and MC methods
- TD methods use, as MC the experience on the environment to create  $V(s)$  and/or  $Q(s,a)$
- TD does bootstrapping (does not need to wait the end of the episode) to update value of the state
- TD update the state value in the next time step, at the next time step  $t+1$  they immediately form a target and make a useful update using the observed reward.

# Function Approximation

## Temporal Difference

### Temporal Difference (TD) Prediction - (Sutton)

Input, the policy  $\pi$  to be evaluated  
Initialize  $V(s)$  arbitrarily (e.g.  $V(s) = 0, \forall s \in S^+$ )  
Repeat (for each episode):  
    Initialize  $S$   
    Repeat (for each step of episode):  
         $A \leftarrow$  action given by  $\pi$  for  $S$   
        Take action  $A$ ; observe reward,  $R$  and next state  $S'$   
         $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$   
         $S \leftarrow S'$   
    until  $S$  is terminal

The DP target

is an estimate not because of the expected values, which are assumed to be completely provided by a model of the environment, but because  $v(S_{t+1})$  is not known and the current estimate,  $V(S_{t+1})$ , is used instead. The TD target is an estimate for both reasons: it samples the expected values in (6.4) and it uses the current estimate  $V$  instead of the true  $v$ . Thus, TD methods combine the sampling of Monte Carlo with the bootstrapping of DP. As we shall see, with care and imagination this can take us a long way toward obtaining the advantages of both Monte Carlo and DP methods.

# Function Approximation

## Exploration-Exploitation

### TD(0) Prediction - Exploitation-Exploration

```
Input, the policy  $\pi$  to be evaluated
 $\epsilon \leftarrow$  Exploration Rate
Initialize  $V(s)$  arbitrarily (e.g.  $V(s) = 0, \forall s \in S^+$ )
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):

    if random  $< \epsilon$  then
       $A \leftarrow$  Random possible action
    else
       $A \leftarrow$  action given by  $\pi$  for  $S$ 

     $A \leftarrow$  action given by  $\pi$  for  $S$ 
    Take action  $A$ ; observe reward,  $R$  and next state  $S'$ 
     $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
     $S \leftarrow S'$ 

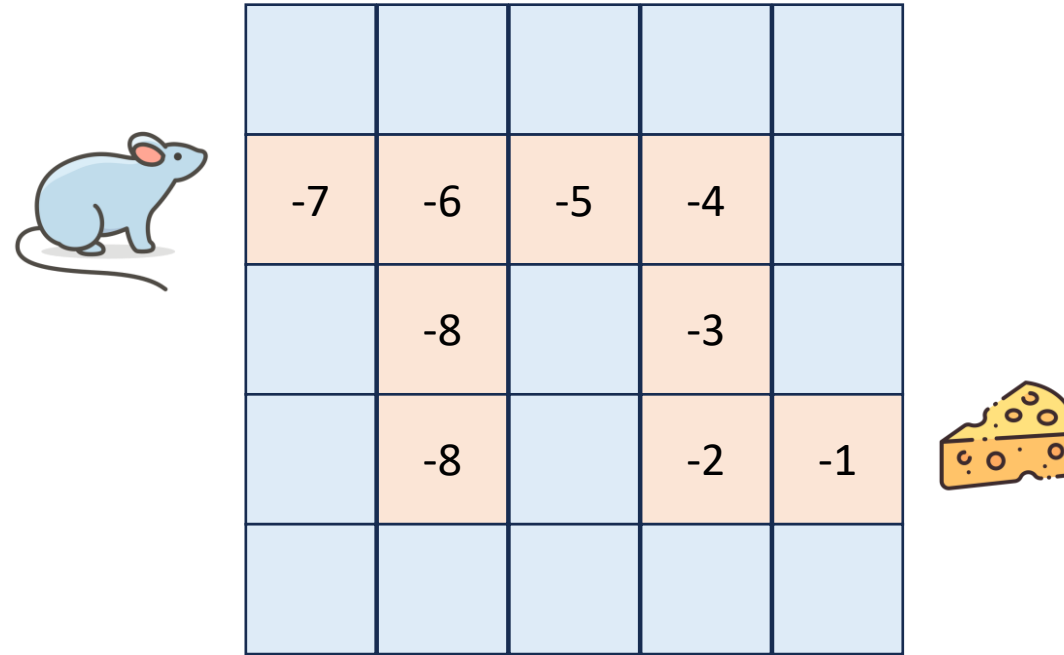
     $\epsilon \leftarrow$  decay

  until  $S$  is terminal
```

# Value Based Function

# Monte-Carlo method

## State-Value Function



$$\underline{V_{\pi}(s)} = \underline{\mathbf{E}_{\pi}}[\underline{G_t} | \underline{S_t = s}]$$

Value of state  $s$

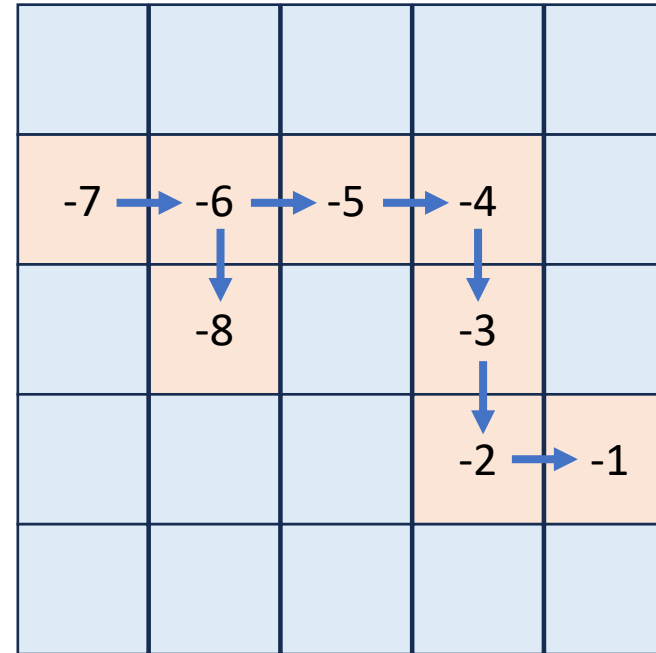
Expected return

If the agent starts at state  $s$

And uses the policy to choose its actions for all time steps

# Monte-Carlo method

## Action-Value Function



$$Q((1,1), \text{right}) = -6$$

$$Q((1,2), \text{right}) = -5$$

$$Q((1,2), \text{down}) = -8$$

$$Q_{\pi}(s, a) = \mathbf{E}_{\pi}[G_t | S_t = s, A_t = a]$$

Value of state-action pair  $s, a$

Expected return

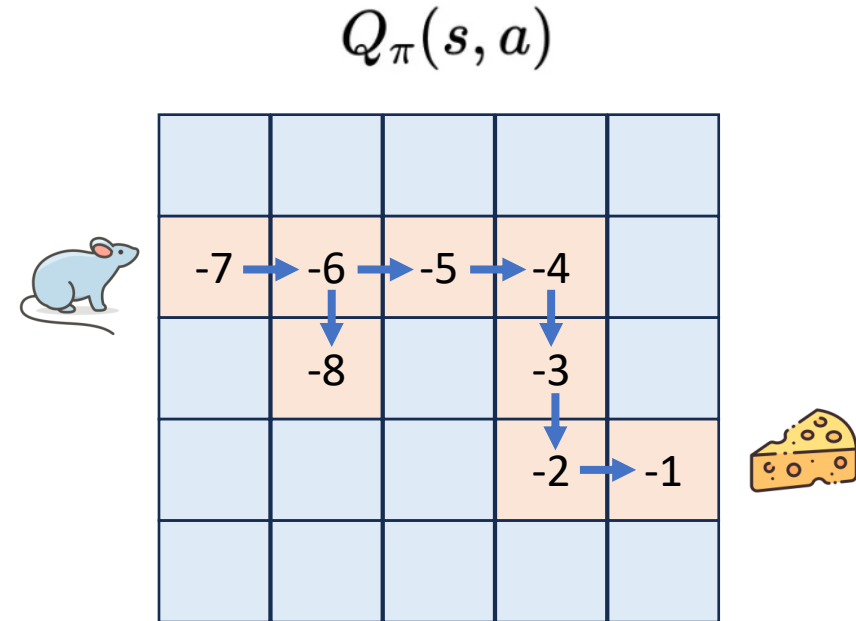
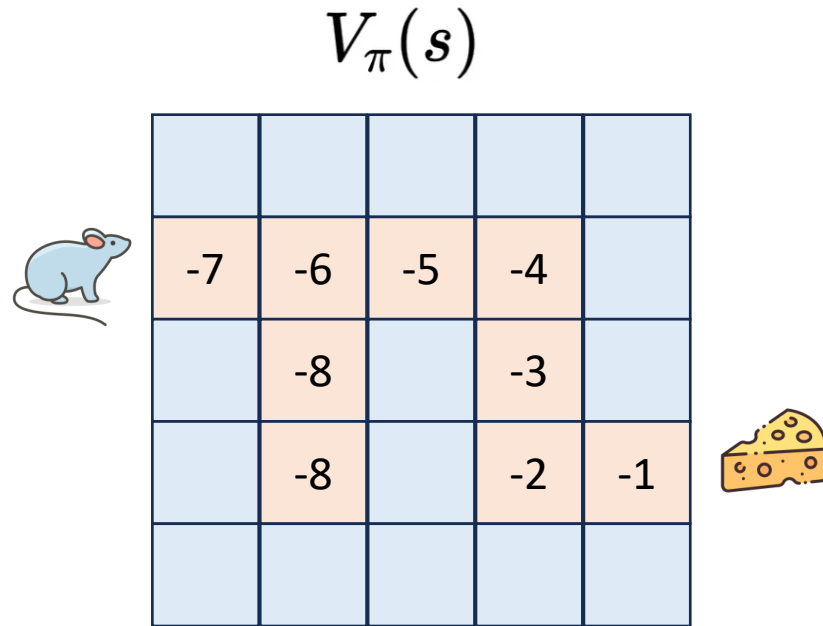
If the agent starts at state  $s$

and chooses action  $a$

And then uses the policy to choose its actions for all time steps

# Monte-Carlo method

## Comparison of Value Based Methods



- In state-value function, we calculate **the value of a state ( $S_t$ )**.
- In action-value function, we calculate **the value of state-action pair ( $S_t, A_t$ )** hence the **value of taking that action at that state**.

# Monte-Carlo method

## MC at the end of the episode

$$\underline{V(S_t)} \leftarrow \underline{V(S_t)} + \alpha [\underline{G_t} - \underline{V(S_t)}]$$

New value of state t

Former estimation of value of state t  
(= Expected return starting at that state)

Learning Rate

Return at timestep t

Former estimation of value of state t  
(= Expected return starting at that state)



# Monte-Carlo method

## TD update every step

$$\underbrace{V(S_t)}_{\text{New value of state t}} \leftarrow \underbrace{V(S_t)}_{\text{Former estimation of value of state t}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R_{t+1}}_{\text{Reward}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value of next state}} - \underbrace{V(S_t)}_{\text{Former estimation of value of state t}}]$$

TD Target

# Monte-Carlo method

## How it uses the policy to create the value function

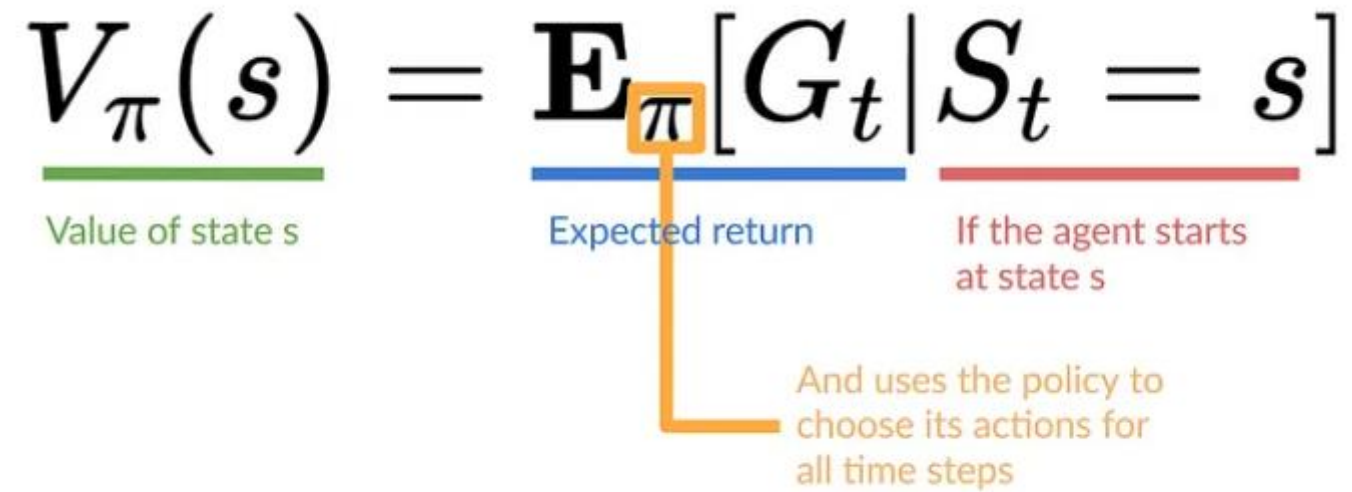
$$\underline{V_{\pi}(s)} = \underline{\mathbf{E}_{\pi}}[\underline{G_t} | \underline{S_t = s}]$$

Value of state  $s$

Expected return

If the agent starts at state  $s$

And uses the policy to choose its actions for all time steps

The diagram shows the equation  $V_{\pi}(s) = \mathbf{E}_{\pi}[G_t | S_t = s]$ . The term  $V_{\pi}(s)$  is underlined in green and labeled 'Value of state s'. The term  $\mathbf{E}_{\pi}$  is underlined in blue and labeled 'Expected return'. The term  $G_t$  is underlined in red and labeled 'If the agent starts at state s'. The term  $S_t = s$  is underlined in red and labeled 'If the agent starts at state s'. An orange box highlights the  $\pi$  in  $\mathbf{E}_{\pi}$ , with an orange line pointing to the text 'And uses the policy to choose its actions for all time steps'.

<https://thomassimonini.medium.com/q-learning-lets-create-an-autonomous-taxi-part-1-2-3e8f5e764358>

# Exploration - Exploitation

# Function Approximation

## Exploration-Exploitation

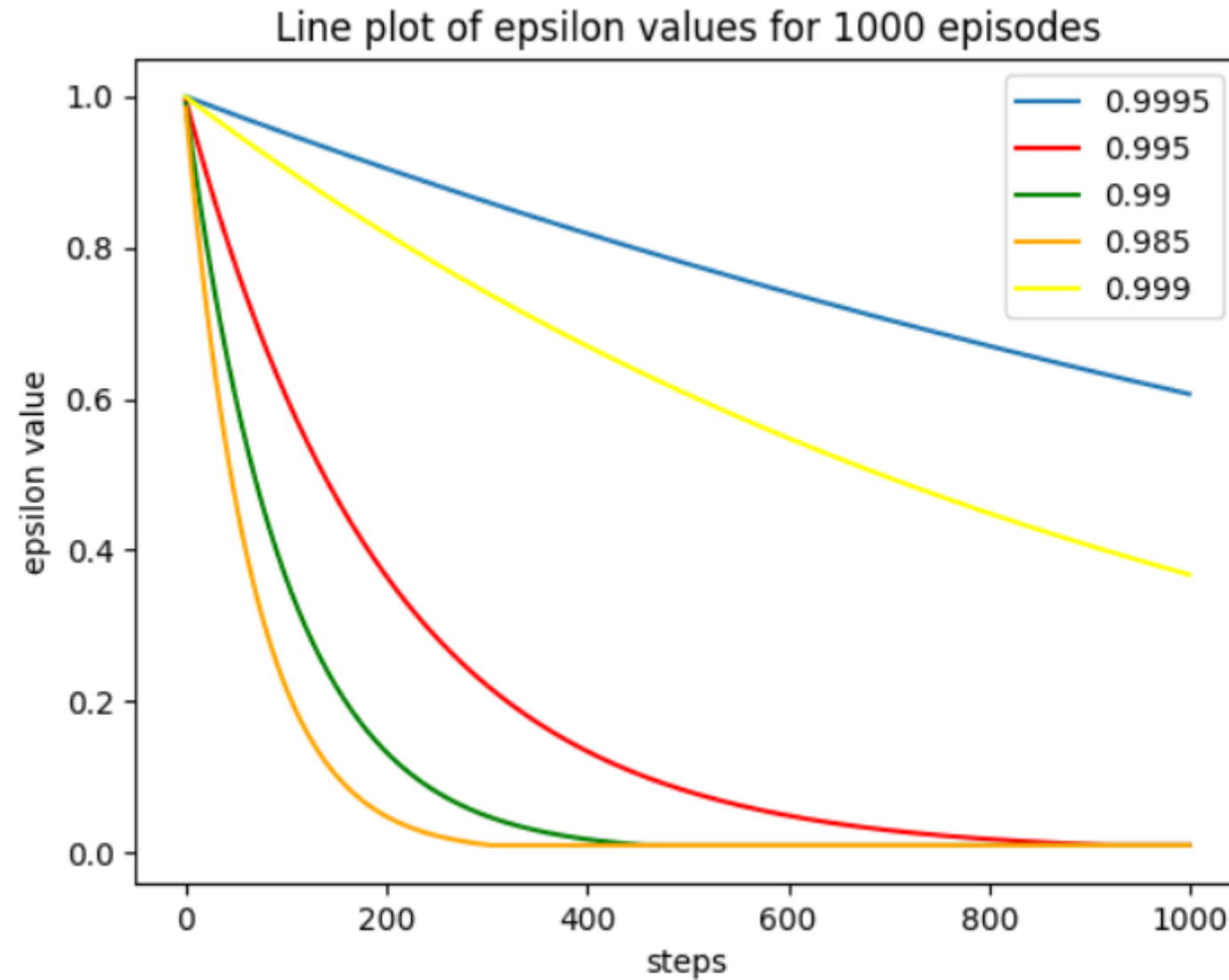
```
if np.random.rand() <= epsilon:
    action = random.action()           # explore
else:
    action = action.from.policy()     # exploit
```

.....

decay-epsilon-every-step

# Function Approximation

## Decaying $\epsilon$



$$\text{epsilon} = \text{epsilon} \max(\text{epsilon\_min}, \text{epsilon} * \text{epsilon\_decay})$$

# Monte-Carlo method

## Advantages of TD Methods

- Are Model-Free
- They don't need to finish episodes, as they are built in an incremental fashion
- Some applications have very long episodes, this becoming a critical difference vs MC
- It assures convergence to  $V_{\pi}$  optimal value function
- TD updates a guess towards a guess

# Monte-Carlo method

## Temporal Difference and MC

- Goal: learn  $v_\pi$  online from experience under policy  $\pi$
- Incremental every-visit Monte-Carlo

- Update value  $V(S_t)$  toward *actual* return  $G_t$

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(0)

- Update value  $V(S_t)$  toward *estimated* return  $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*
    - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the *TD error*

# Wrap-up



## Temporal Difference & Monte-Carlo (MC)

- **Learning from Experience vs Predictions**
  - Monte Carlo methods learn directly from complete episodes of experience, waiting until the end of an episode to update value estimates
  - TD learning learns from incomplete episodes by bootstrapping - it updates estimates based on other learned estimates without waiting for final outcomes
- **Update Timing**
  - MC: Updates occur only after the episode ends, using actual returns
  - TD: Updates happen at each step using estimated returns (TD(0) looks just one step ahead)
- **Bias vs Variance Trade-off**
  - MC methods have higher variance but zero bias (they use actual returns)
  - TD methods have lower variance but some bias (due to bootstrapping from estimates)
- **Online vs Offline Learning**
  - TD can learn online, updating estimates during the episode
  - MC must wait until episode completion, making it less suitable for continuous tasks
- **Handling Incomplete Sequences**
  - TD can learn from incomplete sequences and works in continuing environments
  - MC requires complete episodes and terminal states, limiting its application in some scenarios

# END

## Session 4

