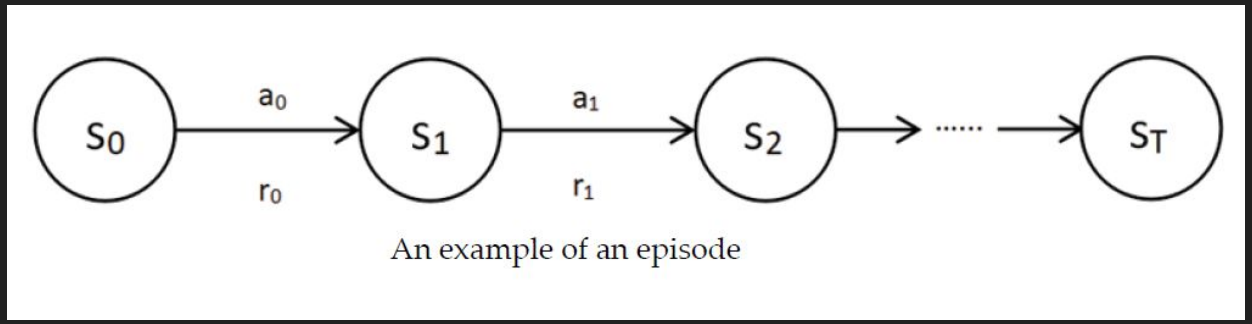
Episode

- The agent interacts with the environment by performing some actions, starting from the initial state and reaches the final state. This **agent-environment interaction starting from the initial state until the final state is called an episode**.
- For instance, in a car racing video game, the agent plays the game by starting from the initial state (the starting point of the race) and reaches the final state (the endpoint of the race). This is considered an episode. An episode is also often called a trajectory (the path taken by the agent) and it is denoted by τ
- An agent can play the game for any number of episodes, and each episode is independent of the others. **What is the use of playing the game for multiple episodes?** In order to learn the optimal policy, that is, the policy that tells the agent to perform the correct action in each state, the agent plays the game for many episodes.
- Say we begin the game from an initial state at a time step t = 0 and reach the final state at a time step T, then the episode information consists of the agent-environment interaction, such as state, action, and reward, starting from the initial state until the final state, that is, (s0, a0, r0, s1, a1, r1,...,sT).

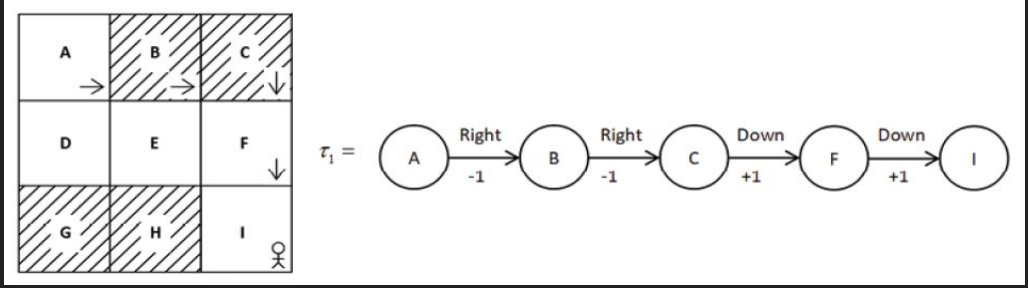


An example of an episode

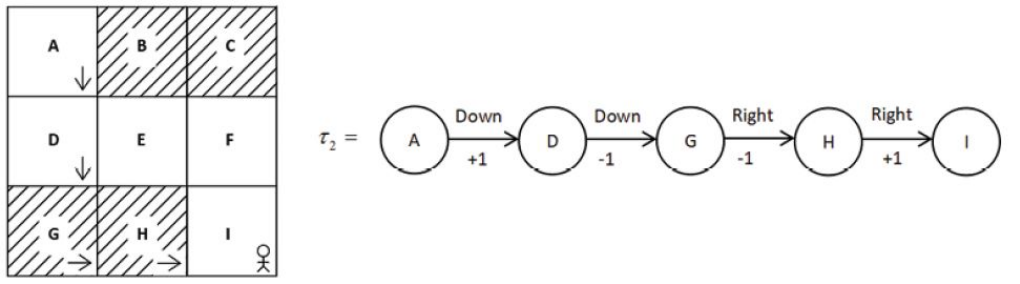
- When we say generate an episode, it means going from the initial state to the final state. The agent generates the first episode using a random policy and explores the environment and over several episodes, it will learn the optimal policy.

Consider Grid World Example :

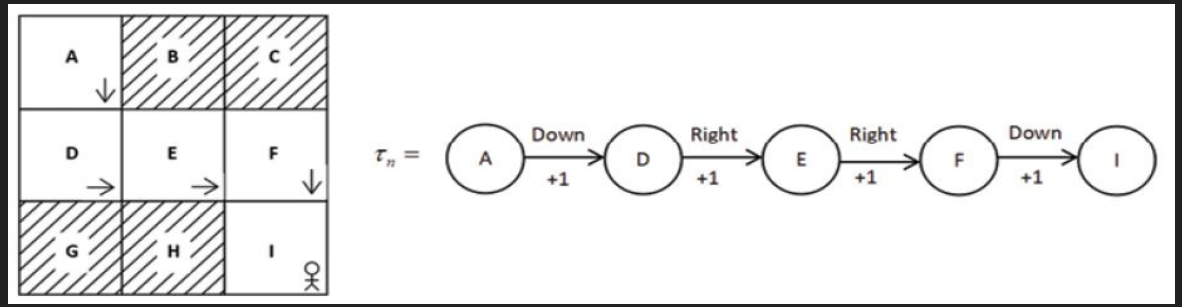
Episode 1: The agent uses a random policy and selects a random action in each state from the initial state until the final state and observes the reward.



Episode 2: The agent tries a different policy to avoid the negative rewards it received in the previous episode. For instance, as we can observe in the previous episode, the agent selected the action right in state A and received a negative reward, so in this episode, instead of selecting the action right in state A, it tries a different action, say down.



Episode n : Thus, over a series of episodes, the agent learns the optimal policy, that is, the policy that takes the agent to the final state I from state A without visiting the shaded states.



Episodic and continuous tasks:

Episodic task:

- As the name suggests, an episodic task is one that has a **terminal/final state.** That is, episodic tasks are tasks made up of episodes and thus they have a terminal state. For example, in a car racing game, we start from the starting point (initial state) and reach the destination (terminal state).

Continuous task:

- Unlike episodic tasks, continuous tasks do not contain any episodes and so they don't have any terminal state. For example, a personal assistance robot does not have a terminal state..

Horizon (Time_Step)

Horizon is the time step until which the agent interacts with the environment. We can classify the horizon into two categories:

Finite horizon
Infinite horizon

Finite horizon:

- **If the agent-environment interaction stops at a particular time step, then the horizon is called a finite horizon.** For instance, in episodic tasks, an agent interacts with the environment by starting from the initial state at time step t = 0 and reaches the final state at time step T. Since the agent-environment interaction stops at time step T, it is considered a finite horizon.

Infinite horizon:

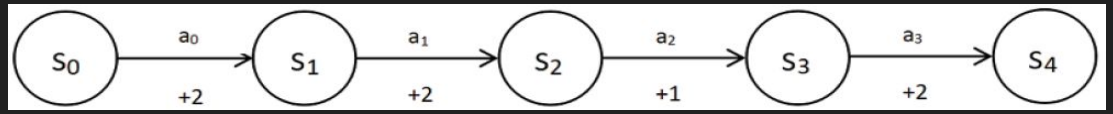
- **If the agent-environment interaction never stops, then it is called an infinite horizon.** For instance, we learned that a continuous task has no terminal states. This means the agent environment interaction will never stop in a continuous task and so it is considered an infinite horizon.

Return and discount factor

- **A return can be defined as the sum of the rewards obtained by the agent in an episode.** The return is often denoted by R or G. Say the agent starts from the initial state at time step t = 0 and reaches the final state at time step T, then the return obtained by the agent is given as:

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_T \qquad R(\tau) = \sum_{t=0}^{T} r_t$$

Consider the trajectory,



The return of the trajectory is the sum of the rewards, that is, $R(\tau) = 2 + 2 + 1 + 2 = 7$

- Thus, we can say that the goal of our agent is to maximize the return, that is, maximize the sum of rewards (cumulative rewards) obtained over the episode.

How can we maximize the return?

- We can maximize the return if we perform the correct action in each state.

how can we perform the correct action in each state?

- We can perform the correct action in each state by using the optimal policy. Thus, we can maximize the return using the optimal policy. Thus, the optimal policy is the policy that gets our agent the maximum return (sum of rewards) by performing the correct action in each state.

how can we define the return for continuous tasks?

- We learned that in continuous tasks there are no terminal states, so we can define the return as a sum of rewards up to infinity.

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_\infty$$

how can we maximize the return that just sums to infinity?

- We introduce a new term called discount factor and rewrite our return as,

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

$$R(\tau) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \ldots + \gamma^n r_\infty$$

how is this discount factor helping us?

- It helps us in **preventing the return from reaching infinity** by deciding how much importance we give to future rewards and immediate rewards. The value of the discount factor r**anges from 0 to 1.**
- When we set the discount factor to a **small value (close to 0)**, it implies that we give more importance to immediate rewards than to future rewards.
- When we set the discount factor to **a high value (close to 1)**, it implies that we give more importance to future rewards than to immediate rewards.

Small discount factor

- Let's set the discount factor to a small value, say 0.2, that is, let's set, then we can write: $\gamma = 0.2$

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\ &= (0.2)^0 r_0 + (0.2)^1 r_1 + (0.2)^2 r_2 + \ldots \\ &= (1) r_0 + (0.2) r_1 + (0.04) r_2 + \ldots \end{aligned}$$

- We can observe that the reward at each time step is weighted by a discount factor. As the time steps increase, the discount factor (weight) decreases and thus the importance of rewards at future time steps also decreases.
    - At time step 0, the reward r0 is weighted by a discount factor of 1.
    - At time step 1, the reward r1 is weighted by a heavily decreased discount factor of 0.2.
    - At time step 2, the reward r2 is weighted by a heavily decreased discount factor of 0.04.

- The discount factor is heavily decreased for the subsequent time steps and more importance is given to the immediate reward r0 than the rewards obtained at the future time steps. Thus, when we set the discount factor to a small value, we give more importance to the immediate reward than future rewards.

Large Discount Factor

Let's set the discount factor to a high value, say 0.9

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (0.9)^0 r_0 + (0.9)^1 r_1 + (0.9)^2 r_2 + \ldots \\
&= (1) r_0 + (0.9) r_1 + (0.81) r_2 + \ldots
\end{aligned}
$$

From this equation, we can infer that as the time step increases the discount factor (weight) decreases; however, it is not decreasing heavily (unlike the previous case) since here we started off with 0.9. So, in this case, we can say that we give more importance to future rewards. That is, from the equation, we can observe that:

- At time step 0, the reward r0 is weighted by a discount factor of 1.
- At time step 1, the reward r1 is weighted by a slightly decreased discount factor of 0.9.
- At time step 2, the reward r2 is weighted by a slightly decreased discount factor of 0.81.

As we can observe, the discount factor is decreased for subsequent time steps but unlike the previous case, the discount factor is not decreased heavily. Thus, when we set the discount factor to a high value, we give more importance to future rewards than the immediate reward.

What happens when we set the discount factor to 0 ?

When we set the discount factor to 0, it implies that we consider only the immediate reward r0 and not the reward obtained from the future time steps. Thus, when we set the discount factor to 0, then the agent will never learn as it will consider only the immediate reward r0, as shown here:

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (0)^0 r_0 + (0)^1 r_1 + (0)^2 r_2 + \ldots \\
&= (1) r_0 + (0) r_1 + (0) r_2 + \ldots \\
&= r_0
\end{aligned}
$$

As we can observe, when we set , our return will be just the immediate reward r0.

What happens when we set the discount factor to 1?

When we set the discount factor to 1,it implies that we consider all the future rewards. Thus, when we set the discount factor to 1, then the agent will learn forever, looking for all the future rewards, which may lead to infinity, as shown here.

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (1)^0 r_0 + (1)^1 r_1 + (1)^2 r_2 + \ldots \\
&= r_0 + r_1 + r_2 + \ldots
\end{aligned}
$$

As we can observe, when we set , then our return will be the sum of rewards up to infinity. Thus, we have learned that when we set the discount factor to 0, the agent will never learn, considering only the immediate reward, and when we set the discount factor to 1 the agent will learn forever, looking for the future rewards that lead to infinity. So, the optimal value of the discount factor lies between 0.2 and 0.8.

why should we care about immediate and future rewards?

- We give importance to immediate and future rewards depending on the tasks. In some tasks, future rewards are more desirable than immediate rewards, and vice versa.

- In a chess game, the goal is to defeat the opponent's king. If we give more importance to the immediate reward, which is acquired by actions such as our pawn defeating any opposing chessman, then the agent will learn to perform this sub-goal instead of learning the actual goal. So, in this case, we give greater importance to future rewards than the immediate reward.

- whereas in some cases, we prefer immediate rewards over future rewards.

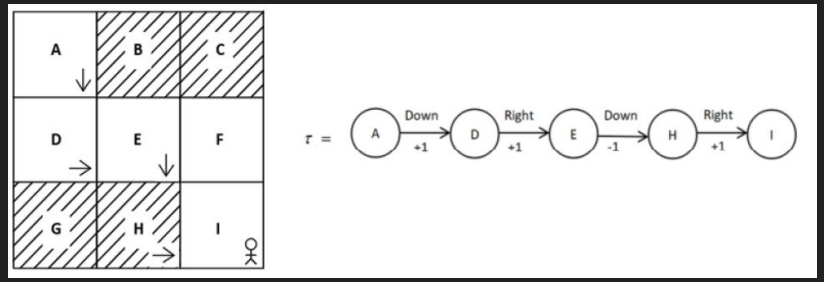
The value function

The value function, also called the state value function, denotes the value of the state. The **value of a state is the return an agent would obtain starting from that state following policy** π. The value of a state or value function is usually denoted by V(s) and it can be expressed as:

$$V^{\pi}(s) = [R(\tau)|s_0 = s]$$

where s0 = s implies that the starting state is s. The value of a state is called the state value.

Let's understand the value function with an example. Let's suppose we generate the trajectory following some policy in grid world environment.



How do we compute the value of all the states in our trajectory?

- The value of state A is the return of the trajectory starting from state A. Thus, V(A) = 1+1+ -1+1 = 2.
- The value of state D is the return of the trajectory starting from state D. Thus, V(D) = 1-1+1= 1.
- The value of state E is the return of the trajectory starting from state E. Thus, V(E) = -1+1 = 0.
- The value of state H is the return of the trajectory starting from state H. Thus, V(H) = 1.

Note that the **value function also depends on the policy,** that is, the value of the state varies based on the policy we choose. There can be many different value functions according to different policies. The optimal value function V*(s) yields the maximum value compared to all the other value functions. It can be expressed as, $V^*(s) = \max_{\pi} V^{\pi}(s)$

For example, let's say we have two policies π_1 and π_2. Let the value of state s using policy π_1 be $V^{\pi_1}(s) = 13$ and the value of state s using policy π_2 be . $V^{\pi_2}(s) = 11$ Then the optimal value of state s will be $V^{\pi_1}(s) = 13$ as it is the maximum. The policy that gives the maximum state value is called the optimal policy . Thus, in this case, π_1 is the optimal policy as it gives the maximum state value.

We can view the value function in a table called a value table. Let's say we have two states s0 and s1, then the value function can be represented as,

State	Value
s_0	7
s_1	11

Value table

From the value table, we can tell that it is be□er to be in state s1 than state s0 as s1 has a higher value. Thus, we can say that state **s1 is the optimal state.**

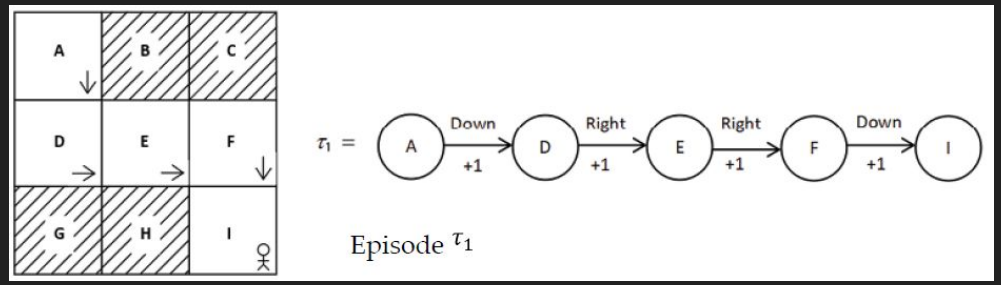
Expected return

Instead of taking the return directly as a value of a state, we will use the expected return. Thus, the value function or the value of state s can be defined as the expected return that the agent would obtain starting from state s following policy π. It can be expressed as

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)|s_0 = s]$$

Let's suppose in grid environment we are in state A and the stochastic policy returns the probability distribution over the action space as [0.0,0.80,0.00,0.20]. It implies that with the stochastic policy, in state A, we perform the action down 80% of the time, that is, $\pi(\text{down}|A) = 0.8$, and the action right 20% of the time, that is $\pi(\text{right}|A) = 0.20$

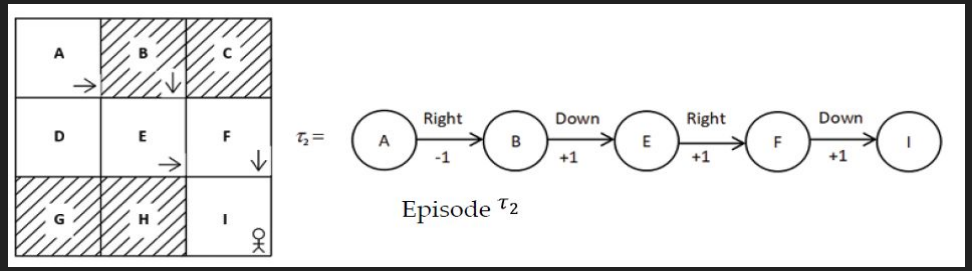
- Generate an episode τ_1 using our stochastic policy π



The value of state A is the return (sum of rewards) of the trajectory starting from state A

$$V(A) = R(\tau_1) = 1 + 1 + 1 + 1 = 4$$

- We generate another episode τ_2 using the same given stochastic policy π



The value of state A is the return (sum of rewards) of the trajectory from state A.

$$V(A) = R(\tau_2) = -1 + 1 + 1 + 1 = 2.$$

As you may observe, although we use the same policy, the values of state A in trajectories and are different. This is because our policy is a stochastic policy and it performs the action down in state A 80% of the time and the action right in state A 20% of the time.**The expected return is basically the weighted average, that is, the sum of the return multiplied by their probability.**

$$V^{\pi}(A) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = A]$$

$$= \sum_i R(\tau_i)\pi(a_i|A)$$

$$= R(\tau_1)\pi(\text{down}|A) + R(\tau_2)\pi(\text{right}|A)$$
$$= 4(0.8) + 2(0.2)$$
$$= 3.6$$