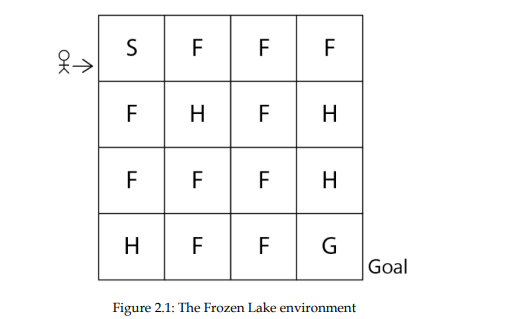
**PGM1 : IMPLEMENT THE FROZEN LAKE ENVIRONMENT USING A RANDOM POLICY**

Gym provides a variety of environments to train a RL agent. Frozen Lake is one of the simplest Gym environment. It is a stochastic environment.

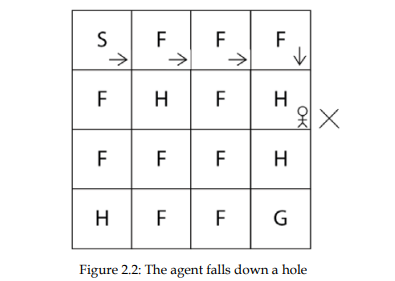


Each grid is a state, so we have totally 16 states (from S to G).

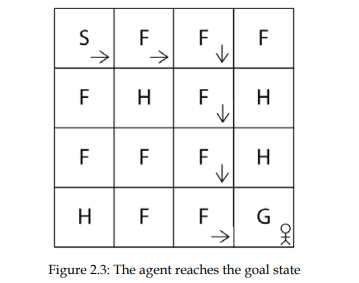
**States :**  S ------ Starting state; F ---- frozen state; H----- hole state; G ---- goal state

**Goal of the agent is to start from the initial state S and reach the goal state G without visiting states H.**

If the agent visits state H, then the agent will fall into the hole and die as shown below:



So, ensure that the agent starts from S and reaches G, without visiting H, as shown below:



**Actions** : 4 possible actions: up, down, left, right.

**Rewards** : Since the agent has to reach state G, we assign +1 reward, if the agent takes an action that moves it to state G; else the reward is -1.

**Implementation in Python :**

1. Import necessary libraries : import gym (or) import gymnasium as gym

import time

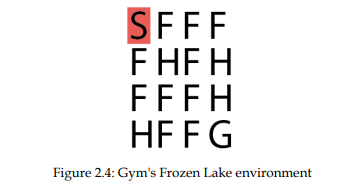
1. Create the environment :

env = gym.make("FrozenLake-v1", render\_mode = "human")

1. Visualize the environment :

env.render()

This gives an output as below. State S is highlighted, since it is the initial state. Whenever the environment is created, the agent will be in the initial state. We can always reset the environment using env.reset()



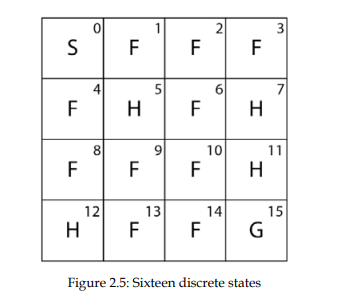
1. Exploring the environment :

#print the states

print(env.observation\_space)

Output : Discrete(16)

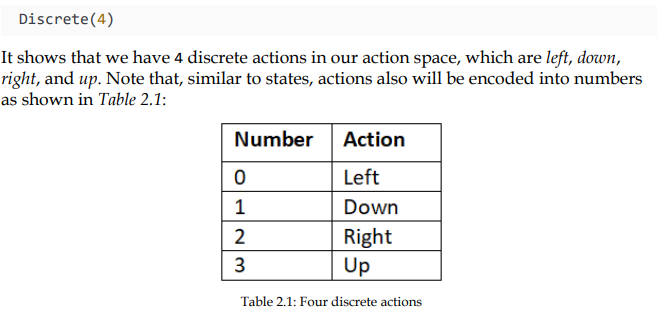
It implies that we have 16 discrete states in our state space starting from state S to G. The states are encoded as numbers as shown below :



#print the action space left =0 down = 1, right = 2 and up=3

print(env.action\_space)

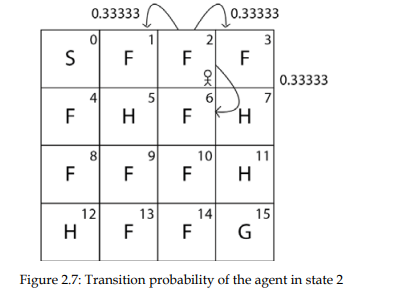
Output :



**Transition probablility and reward function:**

Since the envt is stochastic, we cannot always say that by performing an action ‘a’ in state ‘s’, the agent will not always reach state ‘s’. This sue to the randomness associated with the stochastic envt. So by perfoming an action ‘a’ in state ‘s’, the agent reaches the next state s’ with some probability.

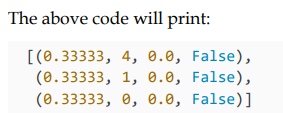
Ex: In state 2(F), if we perform action 1(down), the next states are {1,2,6} with probilities {0.33, 0.33, 0.33} respectively as shown :

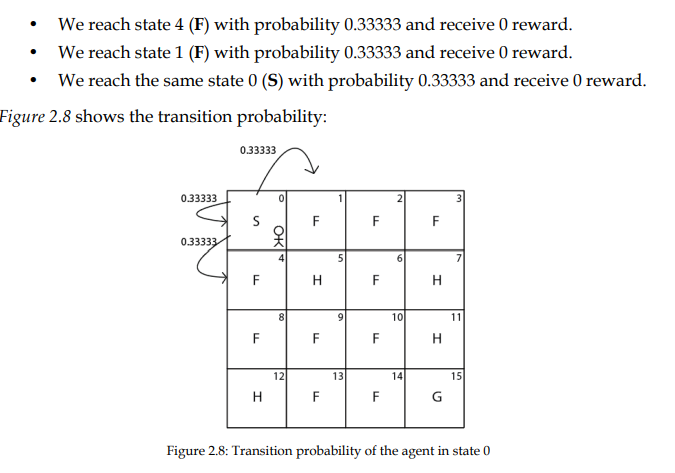


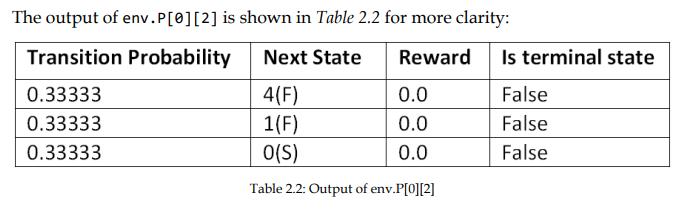
**How to obtain these transition probabilities** ? env.P[state][action]

print(env.P[0][2]) # this gives the transition prob of state ‘s’ and action ‘right’

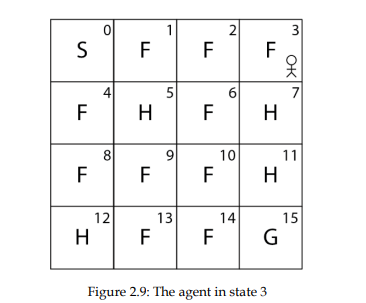
Output is in the form of **[(transition probability, next state, reward, Is terminal state?)].** :



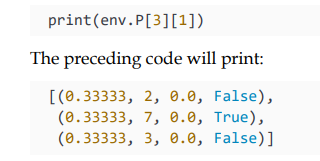


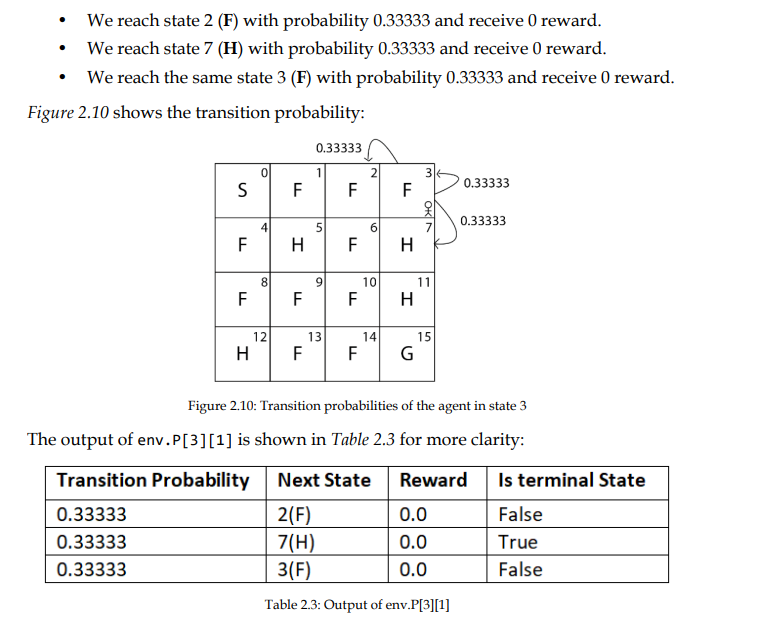


**Ex 2 : Assume the agent is in state 3 (F) as in**



**Find the transition probability of being in state 3(F) and performing action 1 (down) by :**





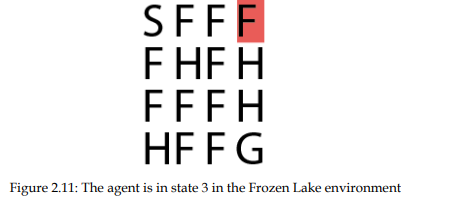
**So, we have learnt how to obtain the state space, action space, transition probability and the reward function.**

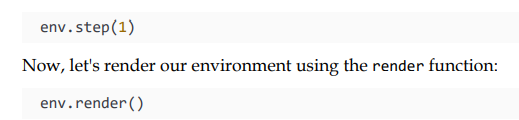
1. **Generate an episode:**

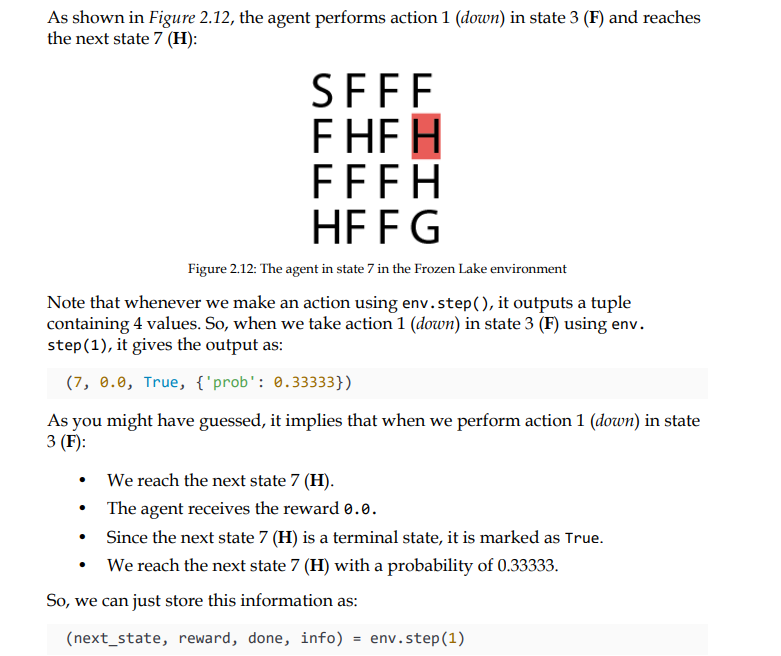
The agent-environment interface starting from the initial state to the final state is an episode. First initialise the state using

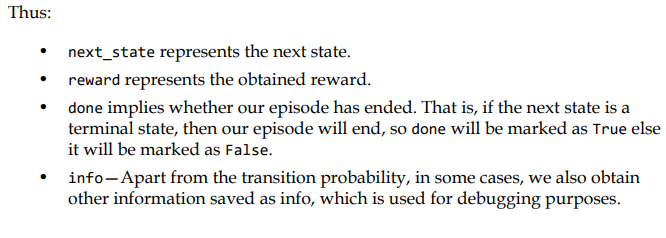
state = env.reset()

* 1. **Selecting an action**. Assume the agent is in state 3 (F) and performs action 1(down) and moves to the next state 7 (H). Perform an action using the env.step(action code).



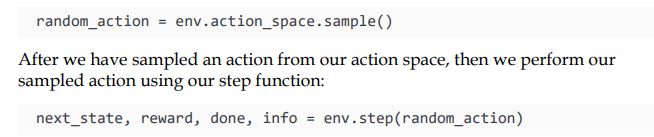






**How to perform a random action from an action space?**

Ans: sample an action from the action space and then perform the action.



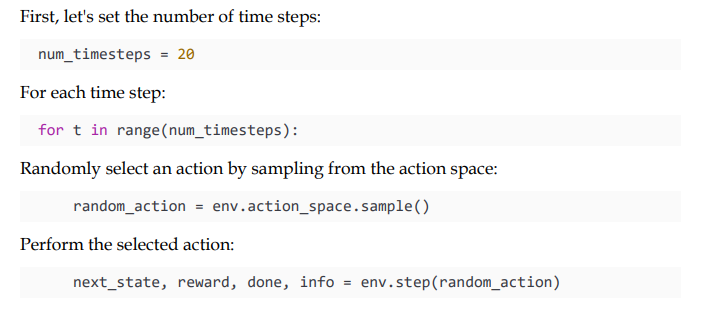
**How to generate an episode?**

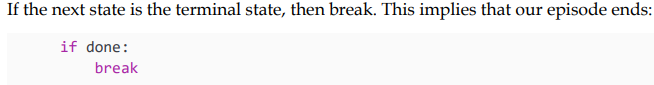
The episode is the agent environment interaction starting from the initial state to the terminal state. The agent interacts with the environment by performing some action in each state. An episode ends if the agent reaches the terminal state.

So, in the Frozen Lake environment, the episode will end if the agent reaches the terminal state, which is either the hole state (H) or goal state (G).

**How to generate an episode with the random policy?**

We learned that the random policy selects a random action in each state. So, we will generate an episode by taking random actions in each state. So for each time step in the episode, we take a random action in each state and our episode will end if the agent reaches the terminal state





**Generating Multiple Episodes:**

for i in range(10):

num\_timesteps = 20

env.reset()

for t in range(num\_timesteps):

rnd\_action = env.action\_space.sample()

(next\_state, reward, done,trans\_prob, info) = env.step(rnd\_action)

env.render()

time.sleep(5)

if done:

break