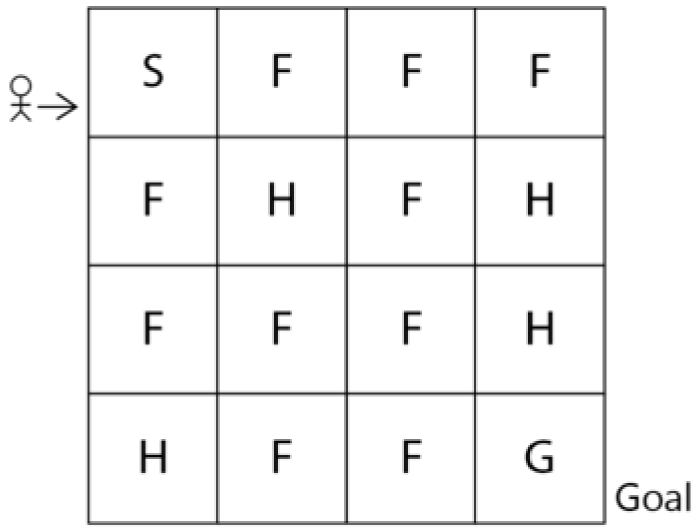
# The Frozen Lake Problem. An example of optimization policy.

Summary: We use the "Optimal Value Function Method" applied to the Frozen Lake Problem", but for a different purpose. Our goal is Oil Production Optimization. Despite the apparent differences, we are going to see that the two problems (Frozen Lake and Oil Production Optimization) can be faced with the same optimization approach.

Introduction: The "Frozen Lake" is a typical example of problem solved by a Reinforcement Learning approach. In this tutorial, we use the "Frozen Lake Problem" as a pragmatic analogy for simulating a scenario of oil production optimization, including the risk of water invasion (into the production well) caused by over-production. For that purpose, we modify a previous code-example about the general "Frozen Lake Problem" (our reference code example is re-adapted from the one included in the excellent book of Ravichandiran, 2020). In our analogy, we assume that the possible actions consists of increasing or decreasing oil production. If we increase such a production properly, we remain in a good state (F=frozen). If we increase too much, this can cause water invasion (H=hole). Such an analogy can be better understood after reminding the basics of the "Frozen Lake Problem", as following.

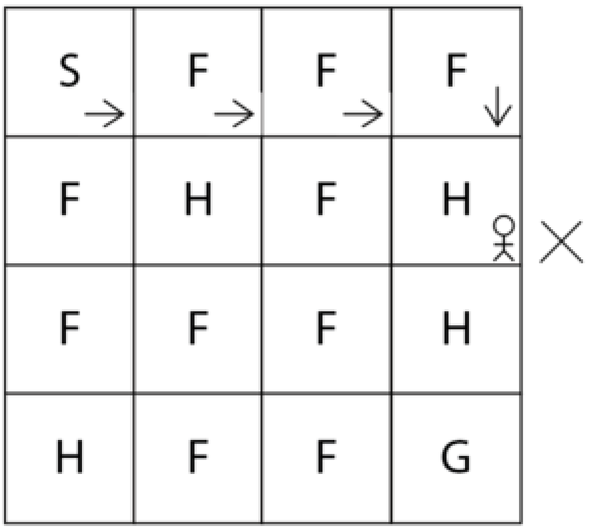
The frozen lake environment is graphycally shown below:



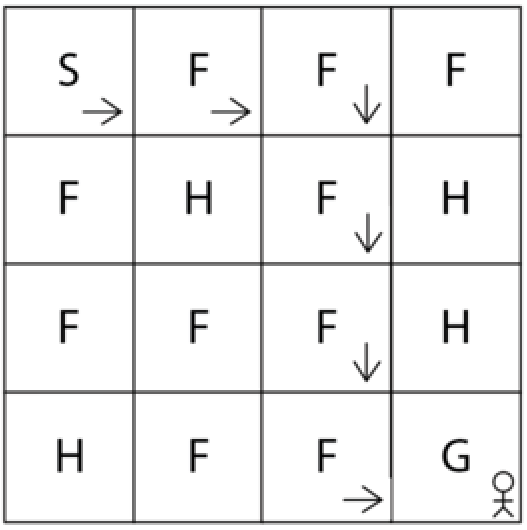
Let's recap the Frozen Lake environment. In the frozen lake environment shown above, we considet the following states and correspondent symbols:

- S is the starting state
- F is the frozen states
- H is the hole states
- G is the goal state

In the frozen lake environment, our goal is to reach the goal state G from the starting state S without visiting the hole states H. In fact, while trying to reach the goal state G from the starting state S, if the agent visits the hole state H then it will fall into the hole and dies as shown below:



So, the goal of the agent is to reach the state G starting from the state S without visiting the hole states H as shown below:



As anticipated at the beginning of this tutorial, we try to adapt the "Frozen Lake Problem" to an oil production scenario, including the risk of water invasion. This can be triggered by over-production. For every starting state (that is a certain rate of oil production), our goal is to find the optimal policy (the optimal production management) for producing the maximum amount of oil, without incurring into the undesired condition of water invasion into the well. In this analogy, we assume that the possible actions consist of increasing or decreasing oil production, with variable entity. If we increase production properly, we remain in a good state (F=frozen). If we increase too much, this can cause water invasion (H=hole). Our goal is to reach a final state (G) that, in our simulation-analogy, represents an optimal oil production, that is the "highest production for that well, without risk of water invasion".

In our analogy, the frozen lake states are reformulated as following:

S implies the starting state: this can be, for instance, a certain oil production rate, with the water front at a certain distance from the production well. Such a distance can be dectected, for instance, using a geophysical method, such as borehole Electric Resistivity Tomography (ERT) (see Dell'Aversana, 2021, for a detailed discussion about ERT combined with Reinforcement Learning). F implies the frozen states: in our analogy, this corresponds to a production state with a water front far away from the well. So it is still a safe state, without any significant risk of massive water invasion. H implies the hole states: in our analogy, this corresponds to a production rate where there is significant water invasion. G implies the goal state: in our analogy, this corresponds to an optimal production rate of oil, where we have high production and low risk of water invasion at the same time. In other words, our goal is to reach a stable state where we produce a lot of oil without incurring into the undesired situation of water invading the production well.

How can we achieve this goal? That is, how can we reach the state G from S without visiting H? In Reinforcement Learning, the optimal policy tells the agent to perform correct action in each state. So, if we find the optimal policy, then we can reach the state G from S without visiting the state H. In order to find the optimal policy, we can use the "Value Iteration Method" to find the optimal policy (this is well explained, for instance, by Ravichandiran, 2020).

All our states (S to G) will be encoded from 0 to 16 and all the four actions (left, down, up, right) will be encoded from 0 to 3 in the "gym toolkit" (that is a typical Reinforcement Learning tool kit). In our analogy, the actions above mentioned have the following meanings (imagine using a wheel that regulates production):

left = moderate production decrease down = high production decrease right = moderate production increase up = high production increase

CODING

```
The first step consists of importing all the necessary libraries:
```

In [34]:

```
import gym
import numpy as np
```

Next, we create the frozen lake environment using gym:

In [35]:

```
env = gym.make('FrozenLake-v0')
```

Let's look at the frozen lake environment using the render function:

In [36]:

```
env.render()
```

```
SFFF
FHFH
FFFH
HFFG
```

We can see that our agent is in the initial state S and it has to reach the state G without visiting the states H. Let's compute the optimal policy using the "Value Iteration Method". This is method of computing an optimal MDP policy and its value (MDP is the short notation for "Markov Decision Process").

First, let's learn how to compute the optimal value function and then we will see how to extract the optimal policy from the computed optimal value function.

## Computing optimal value function

We will define a function called `value_iteration` where we compute the optimal value function iteratively by taking maximum over Q function. Remind that Q-Learning is a value-based Reinforcement Learning algorithm which is used to find the optimal action-selection policy using a Q function. Our goal is to maximize the value function Q. The Q table helps us to find the best action for each state. A Q-value function is a function that maps an observation-action pair to a scalar value representing the expected total long-term rewards that the agent is expected to accumulate when it starts from the given observation and executes the given action. Next, we define `value_iteration` function which takes the environment as a parameter:

In [37]:

```
def value_iteration(env):

    #set the number of iterations
    num_iterations = 1000

    #set the threshold number for checking the convergence of the value function
    threshold = 1e-20

    #set the discount factor
    gamma = 1.0

    #REMINDER. The discount factor essentially determines how much the reinforcement learni
    #cares about rewards in the distant future relative to those in the near future.
    #If γ=0, the agent will be completely myopic and only learn about actions that produce
    #If γ=1, the agent will evaluate each of its actions based on the sum total of all of i

    #now, we will initialize the value table, with the value of all states to zero
    value_table = np.zeros(env.observation_space.n)

    #for every iteration
    for i in range(num_iterations):

        #update the value table: every iteration, we use the updated value
        #table (state values) from the previous iteration
        updated_value_table = np.copy(value_table)

        #next, we compute the value function (state value) by taking the maximum of Q value

        #thus, for each state, we compute the Q values of all the actions in the state and
        #we update the value of the state as the one which has maximum Q value as shown bel
        for s in range(env.observation_space.n):

            Q_values = [sum([prob*(r + gamma * updated_value_table[s_])
                             for prob, s_, r, _ in env.P[s][a]])
                                   for a in range(env.action_space.n)]

            value_table[s] = max(Q_values)

        #after computing the value table, that is, value of all the states, we check whethe
        #difference between value table obtained in the current iteration and previous iter
        #less than or equal to a threshold value. If that conditon is true, then we break t
        #value table as our optimal value function as shown below:

        if (np.sum(np.fabs(updated_value_table - value_table)) <= threshold):
             break

    return value_table
```

Going through the previous step, we have computed the optimal value function by taking the maximum over Q values. The next step is to extract the optimal policy from the optimal value function.

## Extracting optimal policy from the optimal value function

Next, let see how to extract the optimal policy from the computed optimal value function.

First, we define a function called `extract_policy` which takes the `value_table` as a parameter:

In [86]:

```
def extract_policy(value_table):

    #set the discount factor.

    gamma = 1

    #first, we initialize the policy with zeros, that is, first, we set the actions for all
    #be zero
    policy = np.zeros(env.observation_space.n)

    #Next, we compute the Q function using the optimal value function obtained from the
    #previous step. After computing the Q function, we can extract policy by selecting acti
    #maximum Q value. Since we are computing the Q function using the optimal value
    #function, the policy extracted from the Q function will be the optimal policy.

    #As shown below, for each state, we compute the Q values for all the actions in the sta
    #then we extract policy by selecting the action which has maximum Q value.

    #for each state
    for s in range(env.observation_space.n):

        #compute the Q value of all the actions in the state (again, we apply one of the Be
        Q_values = [sum([prob*(r + gamma * value_table[s_])
                             for prob, s_, r, _ in env.P[s][a]])
                                   for a in range(env.action_space.n)]

        #extract policy by selecting the action which has maximum Q value
        policy[s] = np.argmax(np.array(Q_values))

    return policy
```

Now, we will see how to extract the optimal policy in our frozen lake environment.

## Putting it all together

We remind that in the frozen lake environment our goal is to find the optimal policy which selects the correct action in each state so that we can reach the state G from the state A without visiting the hole states.

First, we compute the optimal value function using our `value_iteration` function by passing our frozen lake environment as the parameter:

In [87]:

```
optimal_value_function = value_iteration(env=env)
```

Next, we extract the optimal policy from the optimal value function using our extract_policy function as shown below:

In [88]:

```
optimal_policy = extract_policy(optimal_value_function)
```

We can print the obtained optimal policy:

In [89]:

```
print(optimal_policy)
```

```
[0. 3. 3. 3. 0. 0. 0. 0. 3. 1. 0. 0. 0. 2. 1. 0.]
```

As we can observe, our optimal policy tells us to perform the correct action in each state. Following our analogy, it means to perform correct regulation of oil production, by regulating it through our production wheel. In order to translate the above numbers into a practical decisional policy aimed at optimizing oil production, we need to apply the relationships between the above numbers and actions: in our analogy, 0 means "moderate production decrease"; 1 means "high production decrease"; 2 means "moderate production increase"; 3 means "high production increase". Of course, in more realistic scenarios, those qualitative definitions can be replaced by quantitative production values.

## CONCLUSIONS AND FINAL REMARKS

Using the solution of "Frozen Lake Game", we were able to set an efficient policy for regulating the oil production of our well in the optimal way. That means: producing at the maximum possible rate without incurring into the undesired situation of massive water in the well itself. Our example is extremely simple, because it has demonstrative purposes. It is just a trivial simulation. However, the approach has a general validity and the same method can be applied to very complex scenarios. In order to apply it to real cases, we need to have real production data and real measurements of the water front distance with respect to the well. That is possible using, for instance, borehole ERT (Electric Resistivity Tomography). We have described this method in detail in previous works (Dell'Aversana, 2021).

## REFERENCES

Paolo Dell'Aversana, 2021. Reservoir prescriptive management combining electric resistivity tomography and machine learning[J]. AIMS Geosciences, 2021, 7(2): 138-161. doi: 10.3934/geosci.2021009.

Sudharsan Ravichandiran, 2020. Deep Reinforcement Learning with Python: Master classic RL, deep RL, distributional RL, inverse RL, and more with OpenAI Gym and TensorFlow, 2nd Edition (English Edition). Packt Editor.

In [ ]:

```

```