19 - Reinforcement Learning

Ludwig Krippahl

Reinforcement Learning

Summary

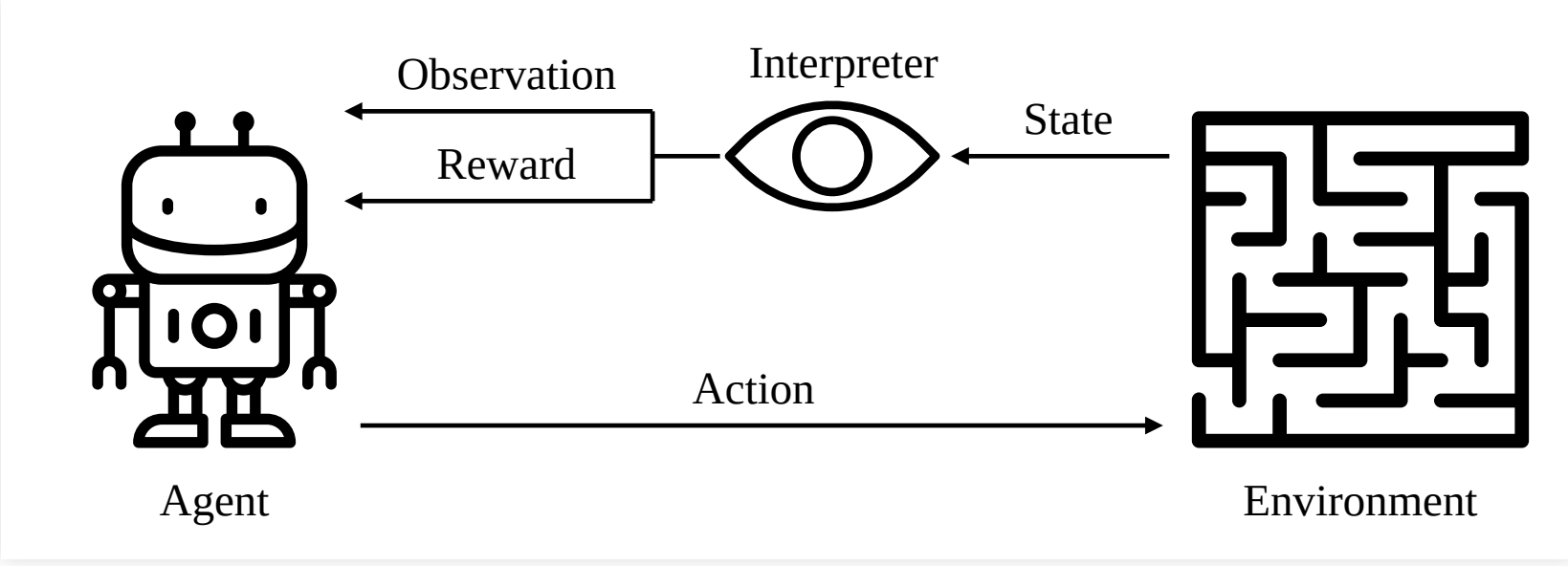
- Introduction to Reinforcement Learning
- Markov Decision Process
- Policies and how to evaluate them
- Improving policies

Introduction

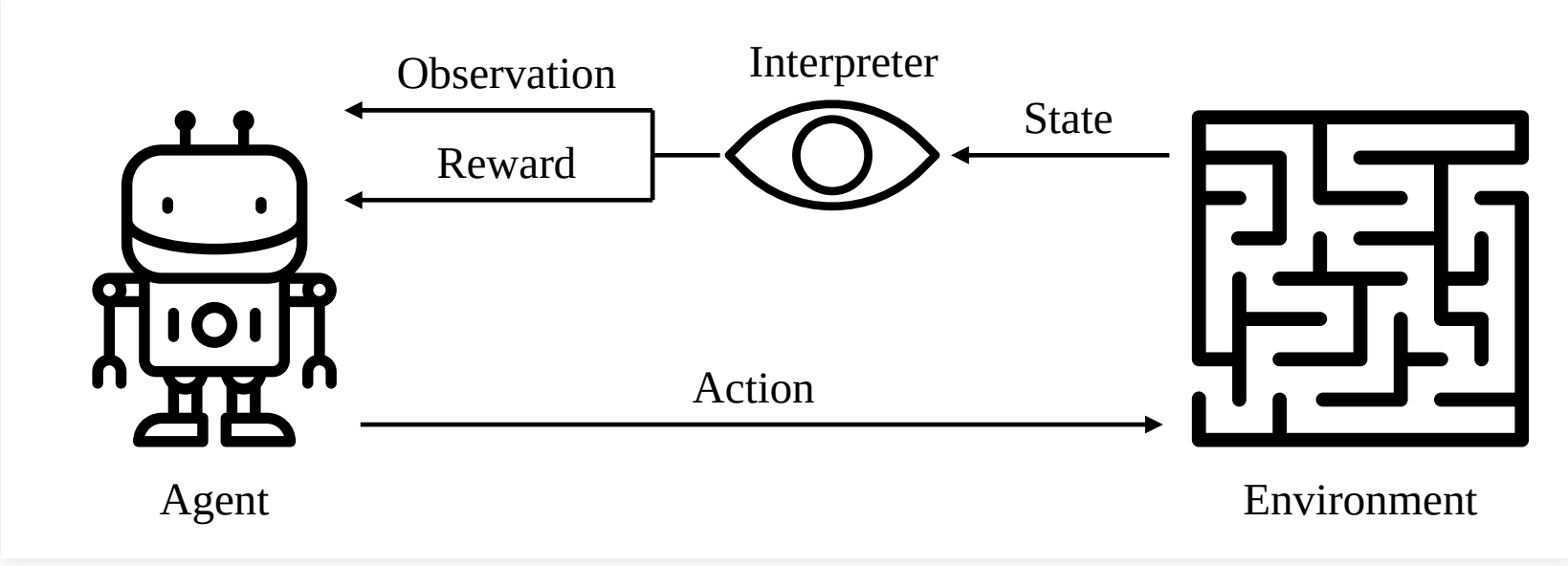
Introduction to RL

- Supervised learning
  - Fit model to minimize error to target
- Unsupervised learning
  - Similar, but without requiring labels

Reinforcement Learning



Reinforcement Learning

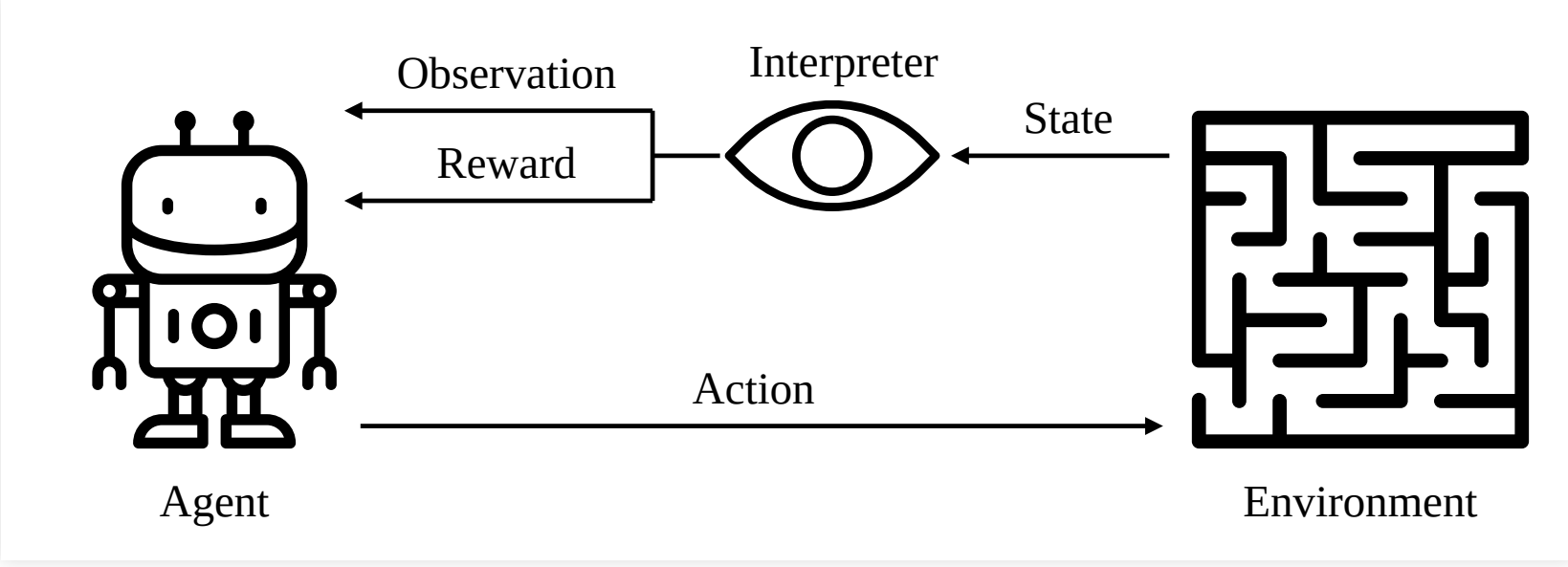


- Agent decides actions
- Environment changes state
- Something gives agent observations and indicates the reward

Introduction to RL

- The action cycle:

- Agent receives observation and, some times, feedback (reward)
- Agent adjusts its decision algorithm based on the observations and rewards (reinforcement learning)
- Agent chooses an action.
- Action affects the environment leading to new observation and feedback



Markov Decision Process

Markov Property: no memory beyond last state

$$P(X_t = x_t \mid X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \ldots, X_1 = x_1) =$$

$$P(X_t = x_t \mid X_{t-1} = x_{t-1})$$

- This may depend on how we represent the process

- Bag with blue and white marbles; probability of white at t will depend on previous draws
- But if we know the state of the bag, previous states are irrelevant.

- This is a useful propert in reinforcement learning

- Without it we would have to keep track of too much data.

Markov Decision Process

$$(S, A, P_a, R, S_\theta, \gamma, H)$$

- S set of states that describe the environment.
- A set of actions the agent can take.
- $P_a(s, s')$ probability of going from s to s' if action a

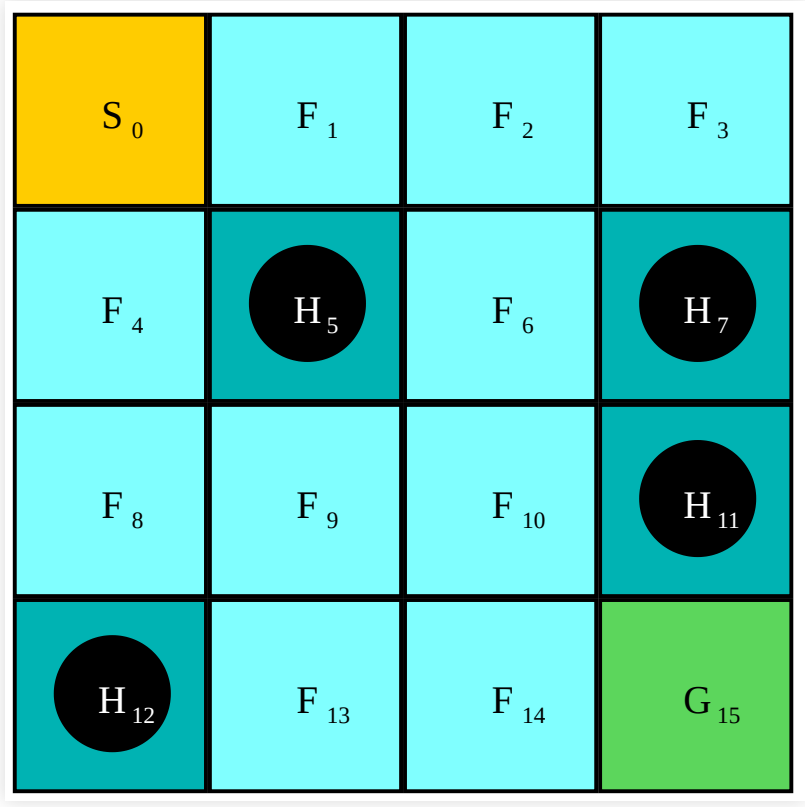
$$P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$$

.

- $R(s, a, s')$ reward given by transitioning from s to s' with action a.
- S_θ is the distribution of initial states.
- γ is the discount factor for future rewards:
  - if r at time 0 then $\gamma^t r$ at time t.
- H is the planning horizon the agent will plan for
  - This can be infinite or can specify a maximum number of steps.

MDP

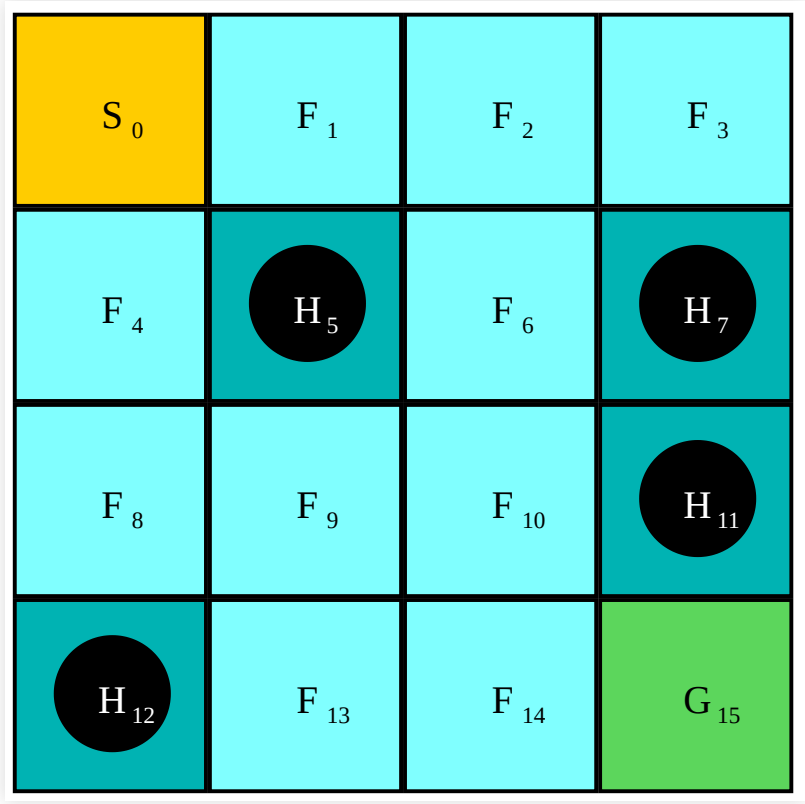
- A simple example: navigate a frozen lake
  - F: frozen, agent can walk over
  - H: holes, terminal states
  - G: goal, terminal state



MDP

- A simple example: navigate a frozen lake
  - 4 actions: N,E,S,W
  - But ice slippery

```
0  N   N  0.333  0
0  N   E  0.333  1
0  N   W  0.333  0
0  S   S  0.333  4
0  S   E  0.333  1
0  S   W  0.333  0
...
1  N   N  0.333  1
1  N   E  0.333  2
1  N   W  0.333  0
...
5  E   E  0.333  5
5  E   N  0.333  5
5  E   S  0.333  5
5  W   W  0.333  5
5  W   N  0.333  5
5  W   S  0.333  5
```



MDP

- A simple example: navigate a frozen lake

- Reward: 1 for goal
- Start at S
- H of 15 steps

- Discount factor

- Favour obtaining the reward sooner rather than later
- The return at time t is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \gamma R_T = \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$$

- Or, recursively

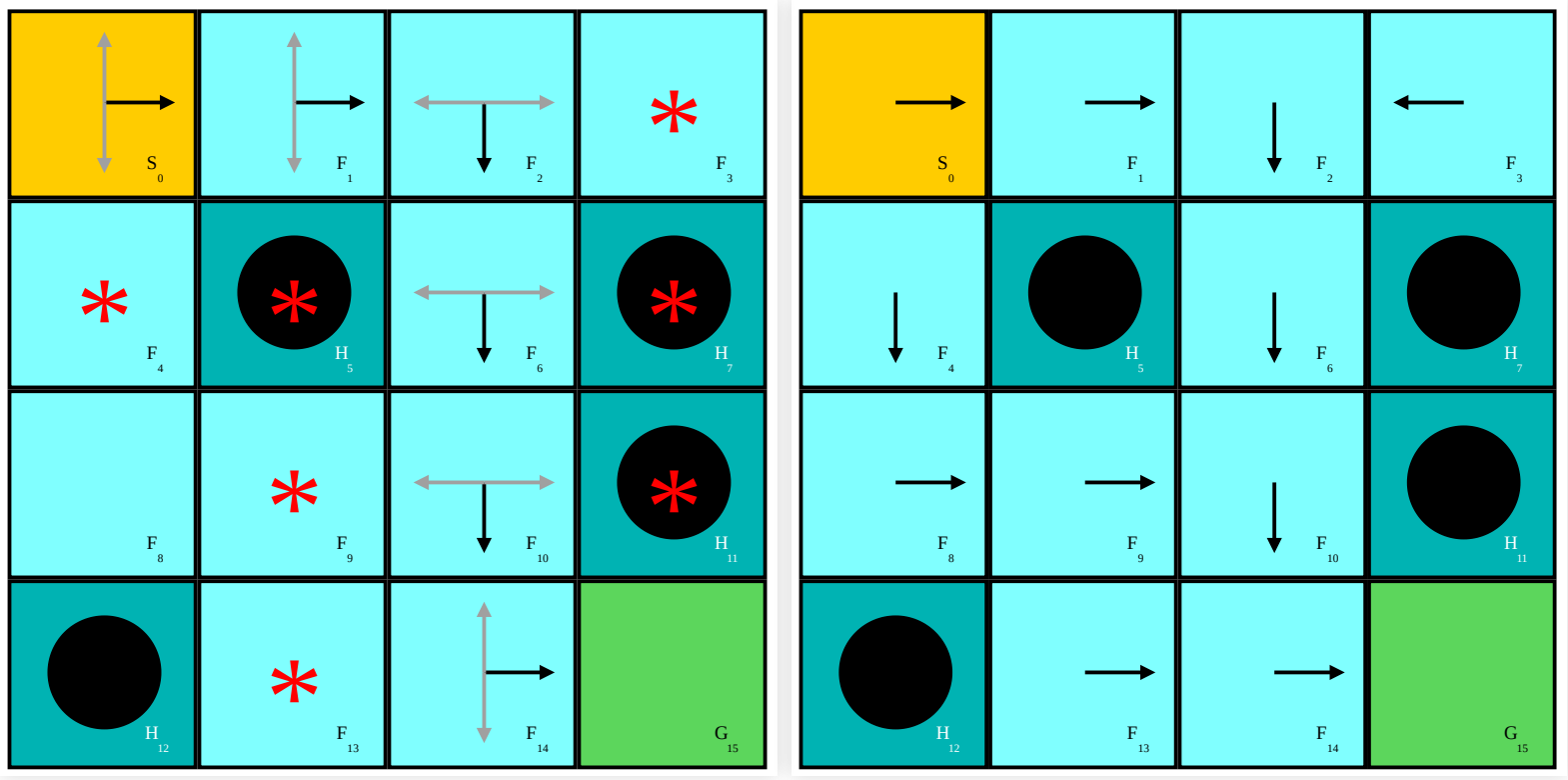
$$G_t = R_{t+1} + \gamma G_{t+1}$$

Plans and Policies

Plans and Policies

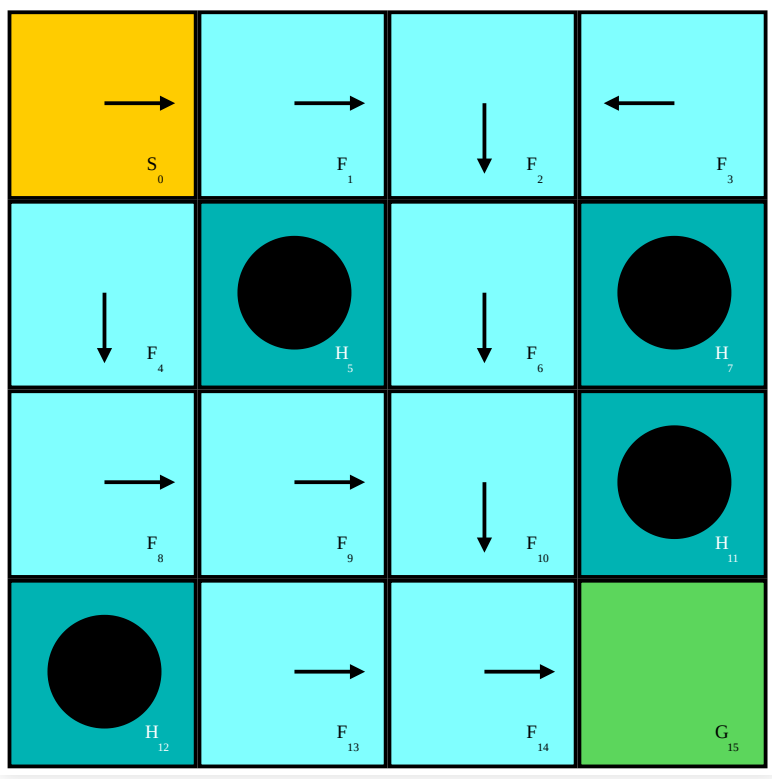
- A plan to navigate the frozen lake: a sequence of actions
- But ice is slippery, so we need a policy

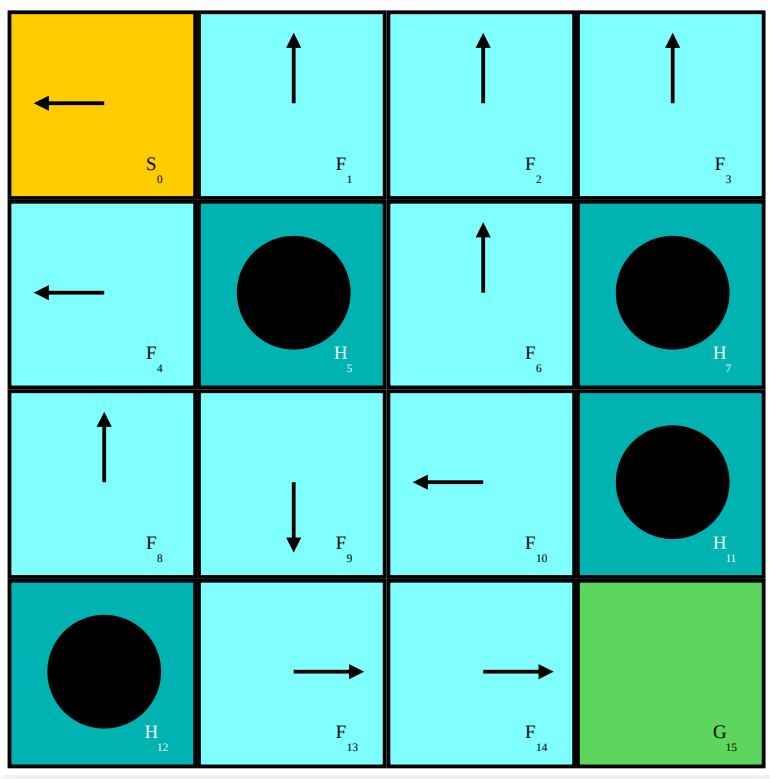
- For each state, tells us the probability of each action



Plans and Policies

- Greedy may cause agent to fall in a hole
- (ice is slippery)
- Which one is better?





State-Value function

- Expected return from state s following policy π

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

$$= \mathbb{E}_\pi\left(R_t + \gamma R_{t+1} + \ldots \mid S_t = s\right)$$

$$= \mathbb{E}_\pi\left(R_t + \gamma G_{t+1} \mid S_t = s\right)$$

- Bellman equation:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a)\left(r_{s,a,s'} + \gamma v_\pi(s')\right)$$

- Bellman's principle of optimality:

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

Plans and Policies

Action-Value function

- Expected return from choosing a in state s following policy π

$$q_\pi(s,a) = \mathbb{E}_\pi \left(G_t \mid S_t = s, A_t = a \right)$$

$$= \mathbb{E}_\pi \left(R_t + \gamma G_{t+1} \mid S_t = s, A_t = a \right)$$

- Bellman equation:

$$q_\pi(s,a) = \sum_{s'} P(s' \mid s,a) \left(r_{s,a,s'} + \gamma v_\pi(s') \right)$$

Action-advantage function

- How much better action a is than what is expected from policy π

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

Plans and Policies

Optimizing policies:

- Improve by replacing action with alternative that gives an advantage
- Find policy that maximizes V and Q:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S$$

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in S, \forall a \in A$$

- Bellman equations:

$$v_*(s) = \max_a \sum_{s'} P(s' \mid s,a)\left(r_{s,a,s'} + \gamma v_*(s')\right)$$

$$q_*(s,a) = \sum_{s'} P(s' \mid s,a)\left(r_{s,a,s'} + \gamma \max_{a'} q_*(s',a')\right)$$

Iterative Policy Evaluation

How do we find v_π?

- Iterative policy evaluation:

- Initialize $v_\pi(s)$ arbitrarily, but with 0 for terminal states
- Apply policy-evaluation equation until convergence:

$$v_{\pi,k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \left(r_{s,a,s'} + \gamma v_{\pi,k}(s')\right)$$

Iterative Policy Evaluation

- Example, frozen lake (needs Open-AI gym)

```
pip install gym
```

- Get environment and MDP from the environment:

```
In : import gym
In : P = gym.make('FrozenLake-v0').env.P
{0:
  {0:[(0.3333333333333333, 0, 0.0, False),
      (0.3333333333333333, 0, 0.0, False),
      (0.3333333333333333, 4, 0.0, False)],
  1: [(0.3333333333333333, 0, 0.0, False),
      (0.3333333333333333, 4, 0.0, False),
      (0.3333333333333333, 1, 0.0, False)],
...
 14:
  {0: [(0.3333333333333333, 10, 0.0, False),
         ...
   1: [(0.3333333333333333, 13, 0.0, False),
       (0.3333333333333333, 14, 0.0, False),
       (0.3333333333333333, 15, 1.0, True)],
...
```

Iterative Policy Evaluation

- We need a dictionary `pi` with policy

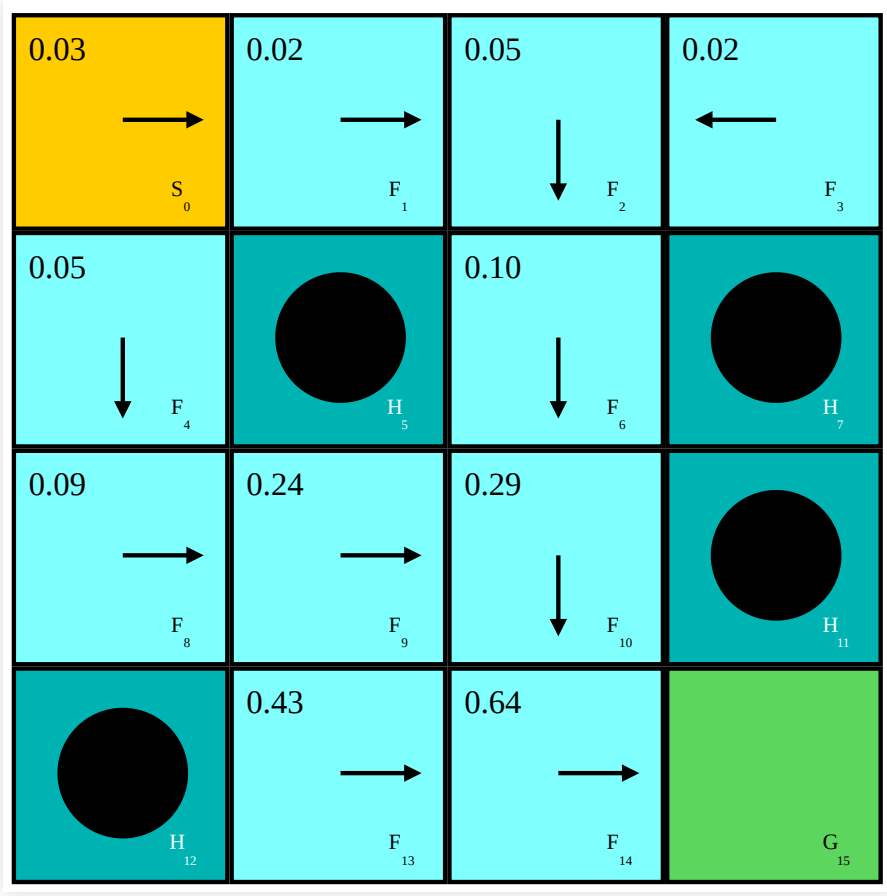
- We assume a deterministic policy

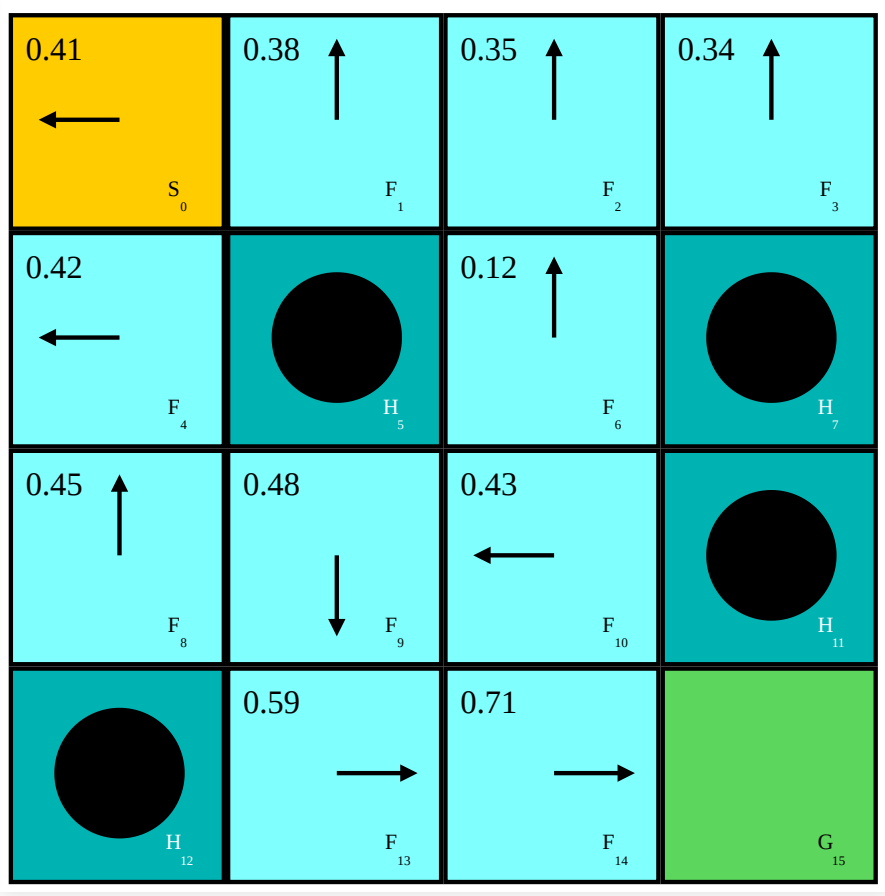
```
def policy_evaluation(pi, P, gamma=1.0, theta=1e-10):
    prev_V = np.zeros(len(P))
    while True:
        V = np.zeros(len(P))
        for s in range(len(P)):
            for prob, next_state, reward, done in P[s][pi[s]]:
                V[s] += prob*(reward+gamma*prev_V[next_state]*(not done))
        if np.max(np.abs(prev_V - V)) < theta:
            break
        prev_V = V.copy()
    return V
```

M. Morales, Grokking Deep Reinforcement Learning, 2020

Iterative Policy Evaluation

- Comparing greedy and safe policies ($\gamma = 0.99$):





Generalized Policy iteration

- With V-function v_{π_n} we can compare policies

- But random generation and testing not eficient

- Better to use the Q-function, which we can compute from V-function

$$q_\pi(s,a) = \sum_{s'} P(s' \mid s,a)\left(r_{s,a,s'} + \gamma v_\pi(s')\right)$$

- and then adjust policy to choose best action

- (more Bellman equations)

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s'} P(s' \mid s,a)\left(r_{s,a,s'} + \gamma v_\pi(s')\right)$$

Generalized Policy iteration

```
def policy_improvement(V, P, gamma=1.0):
    Q = np.zeros((len(P), len(P[0])), dtype=np.float64)
    for s in range(len(P)):
        for a in range(len(P[s])):
            for prob, next_state, reward, done in P[s][a]:
                Q[s,a] += prob * (reward + gamma * V[next_state] * (not done))
    new_pi = {s:a for s, a in enumerate(np.argmax(Q, axis=1))}
    return new_pi
```

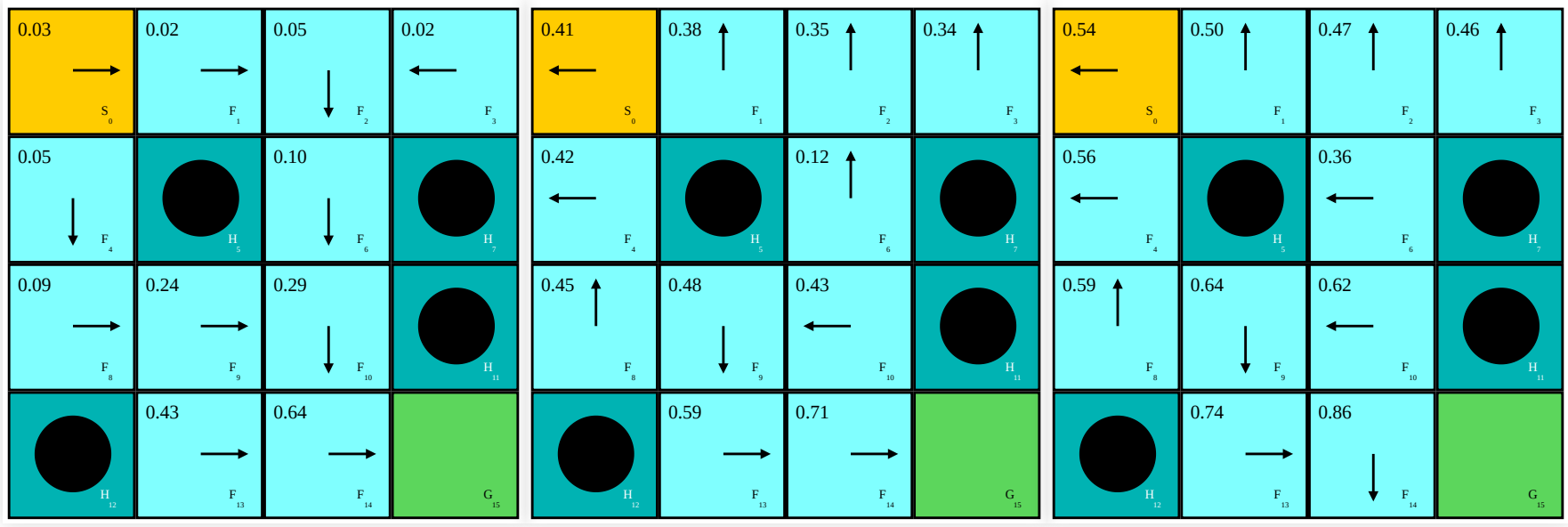
M. Morales, Grokking Deep Reinforcement Learning, 2020

- Doing this iteratively, we get the optimal policy:

```
def policy_iteration(P, gamma=1.0, theta=1e-10):
    random_actions = np.random.choice(tuple(P[0].keys()), len(P))
    pi ={s:a for s, a in enumerate(random_actions)}
    while True:
        old_pi = pi
        V = policy_evaluation(pi, P, gamma, theta)
        pi = policy_improvement(V, P, gamma)
        if old_pi == pi:
            break
    return V, pi
```

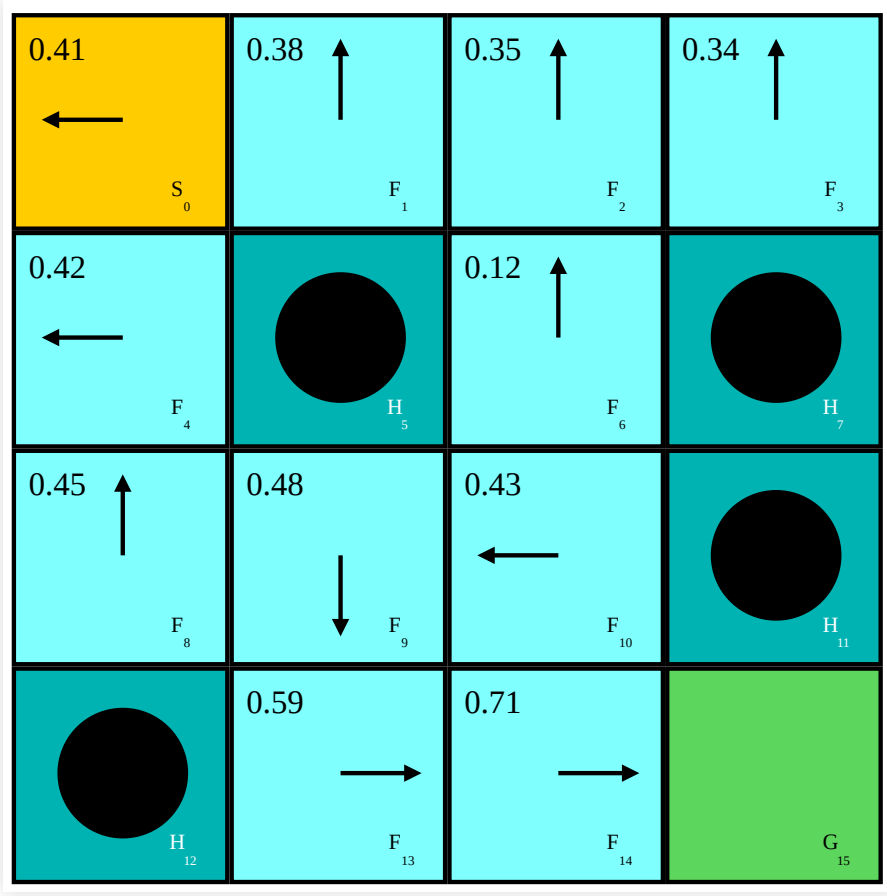
Generalized Policy iteration

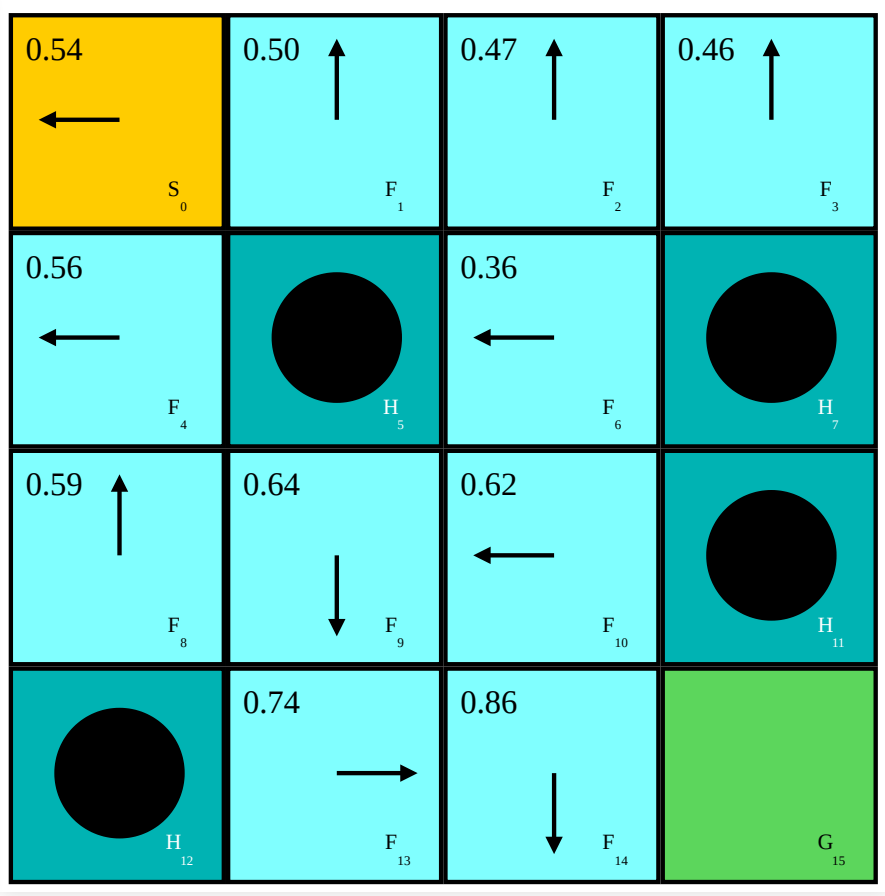
- Three policies: greedy, safe and optimal



Generalized Policy iteration

- Safe vs optimal





Summary

Reinforcement Learning

Summary

- Reinforcement Learning:
  - Action, state, observation, reward
- Markov Decision Process
  - Next step depends only on current state
- Bellman equations, dynamic programming
  - Reward for next step plus discounted estimate of what comes after
- Evaluating policies
- Improving policies

Further reading (Optional)

- Morales, Grokking Deep Reinforcement Learning, 2020, Chp. 1-3