

# 강화학습

## 강화학습 기초

고려대학교 세종캠퍼스 인공지능사이버보안학과  
구 자 훈

# 목차

1. 파이썬으로 MDP 프로그래밍
2. 랜덤 정책과 최적 정책의 기대 이득 비교
3. 정책과 가치 함수

## 2.2 파이썬으로 MDP 프로그래밍

- FrozenLake 과업을 활용하여 MDP 과정을 프로그래밍하는 실습
- FrozenLake는 단순하지만 강화 학습의 모든 요소 담고 있어 첫 실습에 적합

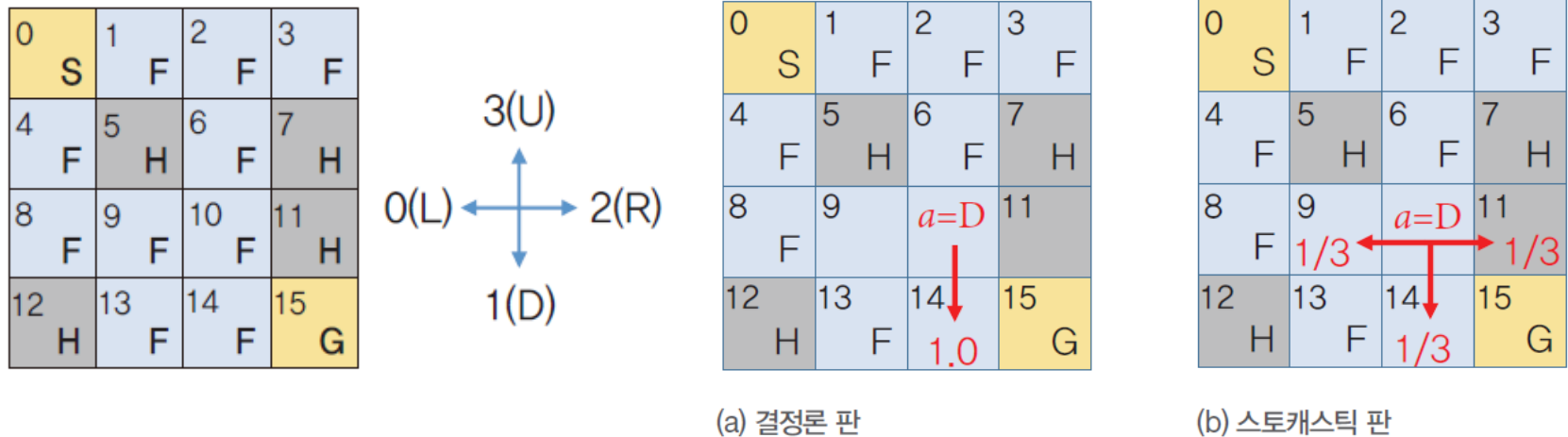


그림 2-5 FrozenLake 과업의 결정론 판과 스토캐스틱 판

## 2.2.1 Gymnasium 라이브러리

### ■ MDP 과정을 시뮬레이션하는 프로그램 필요

- 직접 작성하는 일은 매우 어려움(예를 들어, CartPole 경우 물리적 동역학 이해 필요)
- 오픈소스 라이브러리가 MDP를 구현하여 제공하고 있음
- 가장 널리 쓰이는 라이브러리는 Gymnasium

### ■ Gymnasium 라이브러리

- OpenAI 재단의 Gym으로 출발
- 2021년 Farama 재단으로 관리 권한 이관. Gymnasium으로 개명
- 관련 웹사이트
  - 공식 사이트: <https://farama.org>
  - 매뉴얼 사이트: <https://gymnasium.farama.org>
  - 소스코드 사이트: <https://github.com/Farama-Foundation/Gymnasium>

## 2.2.1 Gymnasium 라이브러리

### ■ 매뉴얼 사이트 자주 참조할 것 권장함

`class gymnasium.Env`

[\[source\]](#)

The main Gymnasium class for implementing Reinforcement Learning Agents environments.

The class encapsulates an environment with arbitrary behind-the-scenes dynamics through the `step()` and `reset()` functions. An environment can be partially or fully observed by single agents. For multi-agent environments, see PettingZoo.

The main API methods that users of this class need to know are:

- `step()` - Updates an environment with actions returning the next agent observation, the reward for taking that actions, if the environment has terminated or truncated due to the latest action and information from the environment about the step, i.e. metrics, debug info.
- `reset()` - Resets the environment to an initial state, required before calling step. Returns the first agent observation for an episode and information, i.e. metrics, debug info.
- `render()` - Renders the environments to help visualise what the agent see, examples modes are "human", "rgb\_array", "ansi" for text.
- `close()` - Closes the environment, important when external software is used, i.e. pygame for rendering, databases

그림 2-6 Gymnasium 라이브러리가 제공하는 문서의 예

## 2.2.1 Gymnasium 라이브러리

### ■ [External Environments] 메뉴에서 다양한 제3자 라이브러리 제공

- ① Autonomous driving(자율주행)
- ② Biological/medical(생물학/의학)
- ③ Economic/financial(경제/금융)
- ④ Electrical/energy(전기/에너지)
- ⑤ Game(게임)
- ⑥ Mathematics/computational(수학/컴퓨팅)
- ⑦ Robotics(로봇)
- ⑧ Telecommunication systems(통신 시스템)
- ⑨ Others(기타)

## 2.2.2 MDP 알고리즘

### ■ MDP 사이클

- 모든 강화 학습 알고리즘의 근간

#### 알고리즘 2-1 MDP 사이클

입력: MDP의 환경 env와 에이전트 agent

출력: 식 (2.1)의 에피소드  $e$

$$e = [-, s_0] a_0 [r_0, s_1] a_1 [r_1, s_2] a_2 \cdots [r_{T-2}, s_{T-1}] a_{T-1} [r_{T-1}, s_T]$$

```
1 state=env.reset() // 초기 상태 생성
2 while True
3     action=agent.policy(state) // 에이전트가 정책을 이용하여 행동을 선택
4     state1,reward,done=env.step(action) // 환경이 MDP 역학을 이용하여 다음 순간으로 전이
5     state=state1                                      $p(s_{t+1}, r_t | s_t, a_t)$ , 줄여서 쓰면  $p(s', r | s, a)$ 
6     if done, break
```

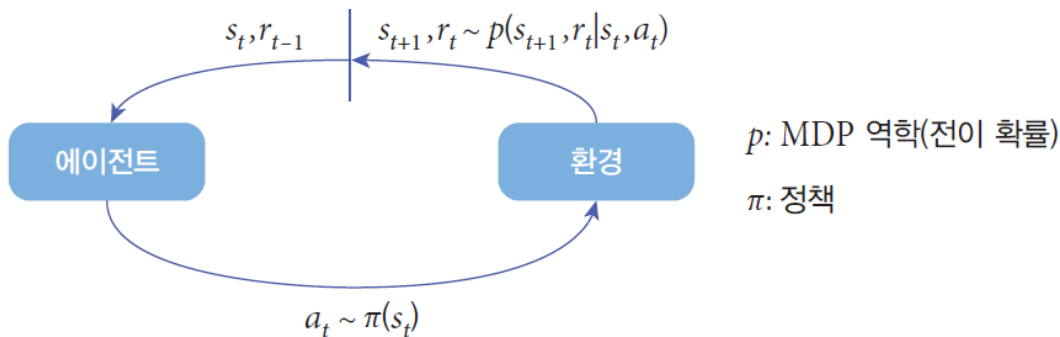
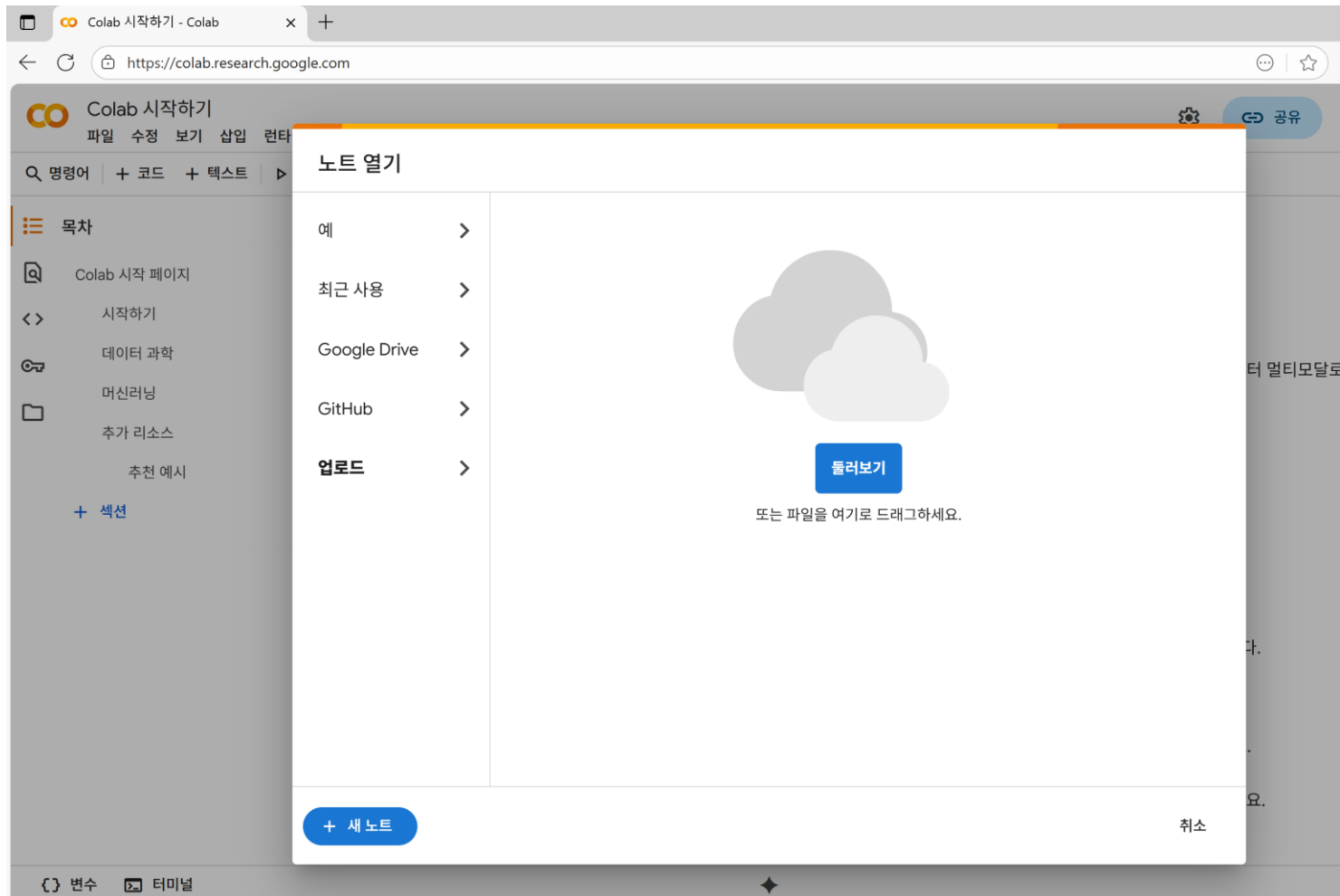


그림 2-7 MDP 사이클

## 2.2.3 MDP 프로그래밍

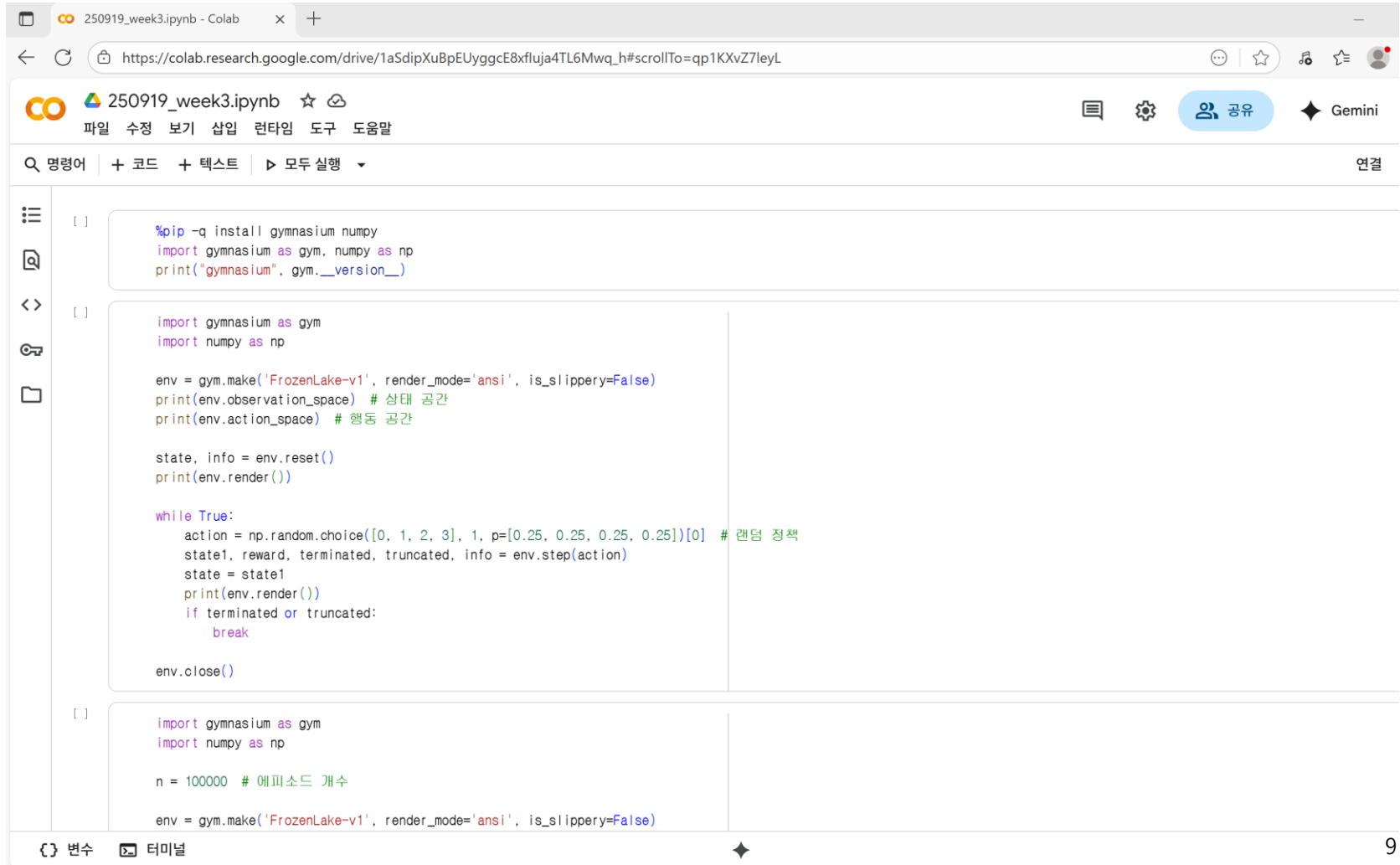
- 구글 Colab (<https://colab.research.google.com/>)
  - LMS를 통해 공유한 colab 노트북을 업로드하여 테스트





## 2.2.3 MDP 프로그래밍

- 구글 Colab (<https://colab.research.google.com/>)
  - LMS를 통해 공유한 colab 노트북을 업로드하여 테스트

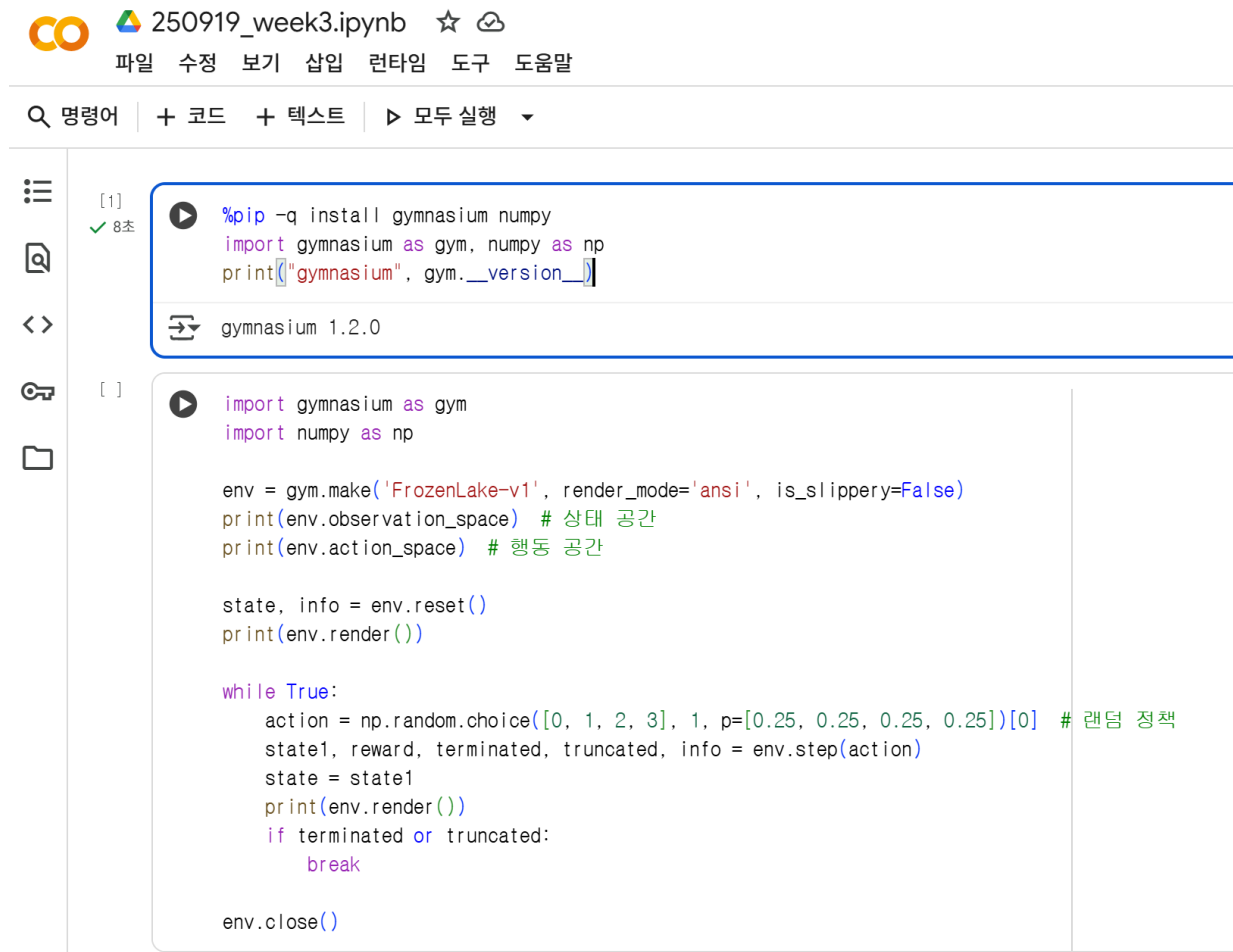


```
[ ]  
  
%pip -q install gymnasium numpy  
import gymnasium as gym, numpy as np  
print("gymnasium", gym.__version__)  
  
[ ]  
  
import gymnasium as gym  
import numpy as np  
  
env = gym.make('FrozenLake-v1', render_mode='ansi', is_slippery=False)  
print(env.observation_space) # 상태 공간  
print(env.action_space) # 행동 공간  
  
state, info = env.reset()  
print(env.render())  
  
while True:  
    action = np.random.choice([0, 1, 2, 3], 1, p=[0.25, 0.25, 0.25, 0.25])[0] # 랜덤 정책  
    state1, reward, terminated, truncated, info = env.step(action)  
    state = state1  
    print(env.render())  
    if terminated or truncated:  
        break  
  
env.close()  
  
[ ]  
  
import gymnasium as gym  
import numpy as np  
  
n = 100000 # 에피소드 개수  
  
env = gym.make('FrozenLake-v1', render_mode='ansi', is_slippery=False)
```

## 2.2.3 MDP 프로그래밍

### ■ 구글 Colab (<https://colab.research.google.com/>)

- 각 셀은 커서를 두고 shift+enter 시 동작
- 가장 처음으로 맨 위의 셀 (gymnasium)을 설치 후 다른 셀을 테스트



The screenshot shows a Google Colab notebook interface. At the top, the file name is '250919\_week3.ipynb'. Below the file name are tabs for '파일', '수정', '보기', '삽입', '런타임', '도구', and '도움말'. A search bar and a dropdown menu for '명령어' are also visible. The notebook contains two code cells. The first cell, labeled '[1]' and '8초', contains the following code: 

```
%pip -q install gymnasium numpy
import gymnasium as gym, numpy as np
print("gymnasium", gym.__version__)
```

 The output of this cell is 'gymnasium 1.2.0'. The second cell, labeled '[ ]', contains the following code: 

```
import gymnasium as gym
import numpy as np

env = gym.make('FrozenLake-v1', render_mode='ansi', is_slippery=False)
print(env.observation_space) # 상태 공간
print(env.action_space) # 행동 공간

state, info = env.reset()
print(env.render())

while True:
    action = np.random.choice([0, 1, 2, 3], 1, p=[0.25, 0.25, 0.25, 0.25])[0] # 랜덤 정책
    state1, reward, terminated, truncated, info = env.step(action)
    state = state1
    print(env.render())
    if terminated or truncated:
        break

env.close()
```

## 2.2.3 MDP 프로그래밍

### 프로그램 2-1

### FrozenLake 과업의 MDP 사이클 확인

```
1  import gymnasium as gym
2  import numpy as np
3
4  env=gym.make('FrozenLake-v1',render_mode='ansi',is_slippery=False)
5  print(env.observation_space)  # 상태 공간
6  print(env.action_space)      # 행동 공간
7
8  state,info=env.reset()
9  print(env.render())
10 while True:
11     action=np.random.choice([0,1,2,3],1,p=[0.25,0.25,0.25,0.25])[0]  # 랜덤 정책
12     state1,reward,terminated,truncated,info=env.step(action)
13     state=state1
14     print(env.render())
15     if terminated or truncated:
16         break
17
18 env.close()
```

## 2.2.3 MDP 프로그래밍

Discrete(16)

Discrete(4)

← 상태 공간과 행동 공간

	(Right)	(Up)	(Down)
SFFF	SFFF	SFFF	SFFF
FHFH	FHFH	FHFH	FHFH
FFFH	FFFH	FFFH	FFFH
HFPG	HFPG	HFPG	HFPG

식 (2.1)로 쓰면

$$e = [-,0]R[0,1]U[0,1]D[0,5]$$

\*\*\* 난수를 사용하기 때문에 실행할 때마다 다른 결과를 얻음

## 2.3 랜덤 정책과 최적 정책의 기대 이득 비교

- 강화 학습의 핵심 목표는 기대 이득 $\text{expected return}$ 을 최대화하는 것
- 이 절에서는 양 극단에 해당하는 두 정책의 기대 이득을 실험적으로 추정
  - 가장 비효율적인 랜덤 정책
  - 가장 효율적인 최적 정책
- 실험을 통해 기대 이득과 정책의 개념을 직관적으로 이해할 수 있음

## 2.3.1 랜덤 정책의 비효율

### ■ [프로그램 2-2]은 랜덤 정책의 효율 측정

- 10만개의 에피소드를 수집한 다음 목표 상태에 도달하여 보상 1을 받을 확률 계산
- 14행은 랜덤 정책  $\pi_1$ 을 사용

$$\text{랜덤 정책 } \pi_1 = \begin{cases} p(L|0) = 0.25, p(D|0) = 0.25, p(R|0) = 0.25, p(U|0) = 0.25 \\ p(L|1) = 0.25, p(D|1) = 0.25, p(R|1) = 0.25, p(U|1) = 0.25 \\ \vdots \\ p(L|5) = 0, p(D|5) = 0, p(R|5) = 0, p(U|5) = 0 \\ \vdots \\ p(L|14) = 0.25, p(D|14) = 0.25, p(R|14) = 0.25, p(U|14) = 0.25 \\ p(L|15) = 0, p(D|15) = 0, p(R|15) = 0, p(U|15) = 0 \end{cases}$$

#### 프로그램 2-2

#### FrozenLake에서 최적 정책의 기대 이득 추정

```
1 import gymnasium as gym
2 import numpy as np
3
4 n=100000 # 에피소드 개수
5
6 env=gym.make('FrozenLake-v1',render_mode='ansi',is_slippery=False)
7 env=gym.wrappers.TimeLimit(env,max_episode_steps=100)
```

## 2.3.1 랜덤 정책의 비효율

```
8  episodes=[]
9  for _ in range(n):
10     epi=[]
11     state,info=env.reset()
12     epi.append([None,state])
13     while True:
14         action=np.random.choice([0,1,2,3],1,p=[0.25,0.25,0.25,0.25])[0] # 랜덤 정책
15         state1,reward,terminated,truncated,info=env.step(action)
16         state=state1
17         epi.append([action,reward,state])
18         if terminated or truncated:
19             break
20     episodes.append(epi)
21
22 env.close()
23
24 expected_return=sum([e[-1][1] for e in episodes])/n # 기대 이득 계산
25 print('랜덤 정책의 기대 이득 =',expected_return)
```

랜덤 정책의 기대 이득 = 0.01412

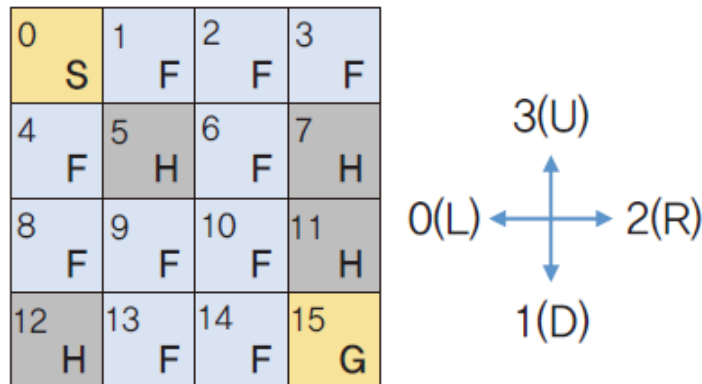
매우 낮은 기대 이득. 왜 이렇게 낮을까?

## 2.3.2 최적 정책의 효율

### ■ [프로그램 2-2]의 랜덤 정책은 왜 비효율적일까?

- 모든 상태에서 같은 확률로 네 가지 행동 중 하나를 선택
  - 예를 들어, 상태 4에서 오른쪽으로 가면 H에 빠지는데, 랜덤 정책은 네 번에 한번 꼴로 오른쪽 선택
  - 상태 4에서의 확률 분포를 아래처럼 설정하면 개선된 정책이 됨

$$p(a=L \mid s=4)=0, p(a=D \mid s=4)=1, p(a=R \mid s=4)=0, p(a=U \mid s=4)=0$$



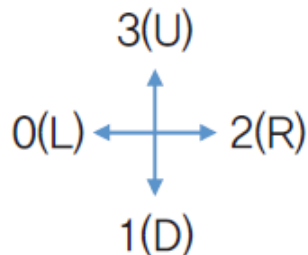


## 2.3.2 최적 정책의 효율

- 모든 상태에서 최적 행동을 선택하는 최적 정책  $\pi_2$

$$\text{최적 정책 } \pi_2 = \begin{cases} p(L|0)=0, p(D|0)=0.5, p(R|0)=0.5, p(U|0)=0 \\ p(L|1)=0, p(D|1)=0, p(R|1)=1, p(U|1)=0 \\ p(L|2)=0, p(D|2)=1, p(R|2)=0, p(U|2)=0 \\ p(L|3)=1, p(D|3)=0, p(R|3)=0, p(U|3)=0 \\ p(L|4)=0, p(D|4)=1, p(R|4)=0, p(U|4)=0 \\ p(L|5)=0, p(D|5)=0, p(R|5)=0, p(U|5)=0 \\ \vdots \\ p(L|14)=0, p(D|14)=0, p(R|14)=1, p(U|14)=0 \\ p(L|15)=0, p(D|15)=0, p(R|15)=0, p(U|15)=0 \end{cases}$$

0 S	1 F	2 F	3 F
4 F	5 H	6 F	7 H
8 F	9 F	10 F	11 H
12 H	13 F	14 F	15 G



## 2.3.2 최적 정책의 효율

프로그램 2-3

FrozenLake에서 최적 정책의 기대 이득 추정

```
1  import gymnasium as gym
2  import numpy as np
3
4  n=100000 # 에피소드 개수
5
6  pi2=np.array([[0,0.5,0.5,0],[0,0,1,0],[0,1,0,0],[1,0,0,0], # 최적 정책
7               [0,1,0,0],[0,0,0,0],[0,1,0,0],[0,0,0,0],
8               [0,0,1,0],[0,0.5,0.5,0],[0,1,0,0],[0,0,0,0],
9               [0,0,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,0]])
10
11 env=gym.make('FrozenLake-v1',render_mode='ansi',is_slippery=False)
12 env=gym.wrappers.TimeLimit(env,max_episode_steps=100)
```

## 2.3.2 최적 정책의 효율

```
13 episodes=[]
14 for _ in range(n):
15     epi=[]
16     state,info=env.reset()
17     epi.append([None,state])
18     while True:
19         action=np.random.choice([0,1,2,3],1,p=pi2[state])[0] # 최적 정책으로 행동 결정
20         state1,reward,terminated,truncated,info=env.step(action)
21         state=state1
22         epi.append([action,reward,state])
23         if terminated or truncated:
24             break
25     if epi[-1][1]!=1: print(epi); input('a')
26     episodes.append(epi)
27
28 env.close()
29
30 expected_return=sum([e[-1][1] for e in episodes])/n # 기대 이득 계산
31 print('최적 정책의 기대 이득 =',expected_return)
```

최적 정책의 기대 이득 = 1.0

기대 이득의 최대치 얻음

FrozenLake는 단순한 과업이기 때문에  
최적 정책  $\pi_2$ 를 사람이 알아낼 수 있음  
복잡한 과업은 학습 알고리즘으로 찾아야 함  
학습 알고리즘을 공부하는 긴 여정 시작

## 2.4 정책과 가치 함수

- 이 절에서는 정책과 가치 함수에 대한 엄밀한 수학적 정의
- 단순한 상황에 대한 가치 함수 계산 예제를 통해 좋은 정책과 가치 함수에 대한 직관적 이해
- 또한 좋은 정책을 찾는 실마리 제시

## 2.4.1 에이전트의 지능=좋은 정책

### ■ 정책 $\pi$ 는 어떤 상태에서 행동을 선택할 때 쓰는 확률 분포

- 상태  $s$ 에서 행동  $a$ 를 선택할 확률

$$\pi(a | s) = p(a | s), \forall s, \forall a \quad (2.6)$$

- 2.3절에서 제시한 랜덤 정책  $\pi_1$ 과 최적 정책  $\pi_2$ 는 정책의 사례
- 앞으로 최적 정책을 자동으로 찾는 학습 알고리즘에 대한 공부

### ■ 강화 학습 알고리즘의 일반 틀

#### 알고리즘 2-2 강화 학습 알고리즘의 일반 틀1

- 1 정책을 초기화한다.
- 2 while (not 만족스러운 성능)
- 3     좋은 행동은 강화하고 나쁜 행동은 억제하는 방향으로 정책을 개선한다.

## 2.4.1 에이전트의 지능=좋은 정책

### ■ 조금 구체화 하면,

- 시행착오를 반복함으로써 최적 정책에 점점 다가가는 전략 사용
- 강화 학습은 시행착오에 기반한 자율 학습을 함(자신이 생성한 데이터로 자신을 학습)

#### 알고리즘 2-3 강화 학습 알고리즘의 일반 틀2

입력: 과업의 MDP

출력: 최적 정책  $\pi_*$

```
1  정책  $\pi$ 를 난수로 초기화한다.
2  while (not 수렴 조건)
3       $\pi$ 로 에피소드를 생성한다. // 현재 정책으로 훈련 데이터 수집
4      에피소드로  $\pi$ 를 개선한다. // 훈련 데이터로 정책 개선
5   $\pi_* = \pi$ 
```

## 2.4.2 정책을 평가하는 가치 함수

### ■ [알고리즘 2-3]의 구현에서,

- 3행의 에피소드 생성은 [알고리즘 2-1]을 사용하면 됨
- 4행은 어떻게?
- [알고리즘 2-3]의 핵심은 4행을 어떻게 구현할지 구상하는 것

### ■ 정책 $\pi$ 를 개선한다는 말의 의미

- 두 정책  $\pi$ 와  $\pi'$ 가 식 (2.7)을 만족하면  $\pi'$ 는  $\pi$ 보다 우월

$$v_{\pi'}(s) \geq v_{\pi}(s), \forall s \in \mathcal{S} \quad (2.7)$$

- 식 (2.7)에 따르면 4행은 현재 정책  $\pi$ 를 개선하여  $\pi'$ 를 만들면 됨

### ■ 가치 함수<sup>value function</sup> $v_{\pi}(s)$

- 가치 함수는 정책  $\pi$ 가 얼마나 좋은지 평가
- 다시 말해,  $\pi$ 를 사용했을 때 상태  $s$ 에서 가치(기대 이득)를 평가해주는 함수
- '모든' 상태에 대해 평가할 수 있어야 함

## 2.4.2 정책을 평가하는 가치 함수

### ■ 최적 정책 $\pi_*$

- 발생 가능한 모든 정책으로 구성된 정책 공간에서 가장 우월한 정책

$$\pi_* = \operatorname{argmax}_{\pi} v_{\pi} \quad (2.8)$$

### ■ 강화 학습을 구현하려면 해야 하는 두가지 일

- 가치 함수를 어떻게 계산할 것인가?
- 무한하게 많은 정책 중에서 최적 정책  $\pi_*$ 를 어떻게 빠르게 찾을 것인가?

### ■ 상태 가치 함수state value function와 행동 가치 함수action value function

- 상태 가치 함수  $v_{\pi}(s)$ 와 행동 가치 함수  $q_{\pi}(s, a)$
- $q_{\pi}(s, a)$ 는 정책  $\pi$ 가 상태  $s$ 에서 행동  $a$ 를 취했을 때 가치(기대 이득)
- 식 (2.10)은 둘 사이의 관계

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a | s) q_{\pi}(s, a), \forall s \in \mathcal{S} \quad (2.10)$$



## 2.4.3 가치 함수의 계산

### ■ 이 절의 목적

- 단순한 상황에서 가치 함수를 계산하는 사례를 제시하여 개념과 원리 이해를 도움
- 실제로는 [알고리즘 2-3]처럼 에피소드 데이터를 샘플링하고 정책을 학습

### ■ 단순한 상황에서 가치 함수를 계산하는 분석적 방법

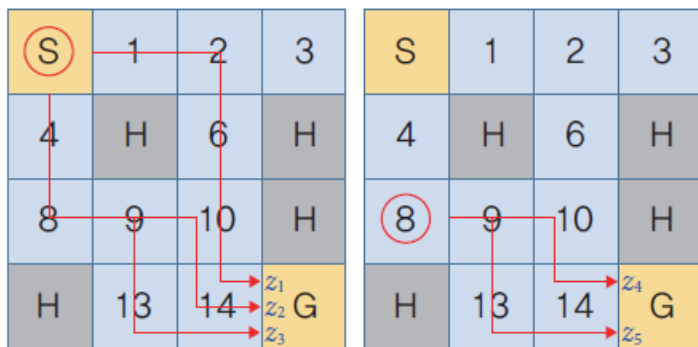
- 상태  $s$ 에서 출발하는 가능한 모든 궤적  $z$ 를 나열
- $z$ 의 발생 확률  $p(z)$ 를 가중치로 사용한 이득  $R(z)$ 의 가중합

$$v_{\pi}(s) = \mathbb{E}(R | s) = \sum_{s\text{에서 출발하는 궤적 } z} p(z)R(z), \forall s \in \mathcal{S} \quad (2.11)$$

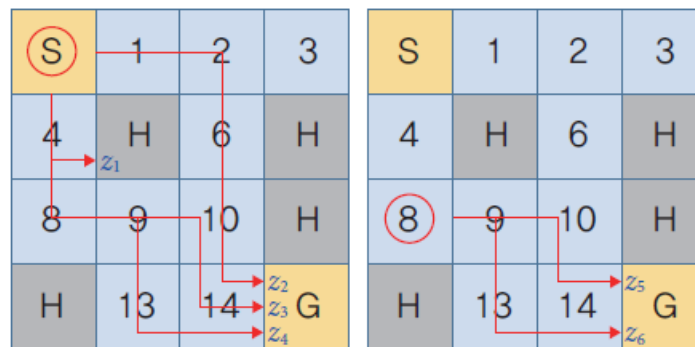
## 2.4.3 가치 함수의 계산

### ■ [예제 2-1] 식 (2.11)을 이용한 상태 가치 함수 계산

- 최적 정책  $\pi_2$ 는 유한 개의 궤적을 가짐



(a)  $\pi_2$ 에서 발생한 모든 궤적



(b)  $\pi_3$ 에서 발생한 모든 궤적

그림 2-8 정책  $\pi_2$ 와  $\pi_3$ 에서 발생한 모든 궤적

- 상태 0에서는 세 개 궤적이 있음. 식 (2.11)을 적용하면 상태 0의 가치(기대 이득)는 1

$$z_1 = [0,0]R[0,1]R[0,2]D[0,6]D[0,10]D[0,14]R[1,15] \quad p(z_1)=0.50 \quad R(z_1)=1$$

$$z_2 = [0,0]D[0,4]D[0,8]R[0,9]R[0,10]D[0,14]R[1,15] \quad p(z_2)=0.25 \quad R(z_2)=1$$

$$z_3 = [0,0]D[0,4]D[0,8]R[0,9]D[0,13]R[0,14]R[1,15] \quad p(z_3)=0.25 \quad R(z_3)=1$$

$$\rightarrow v_{\pi_2}(0) = p(z_1)R(z_1) + p(z_2)R(z_2) + p(z_3)R(z_3) = 1$$

## 2.4.3 가치 함수의 계산

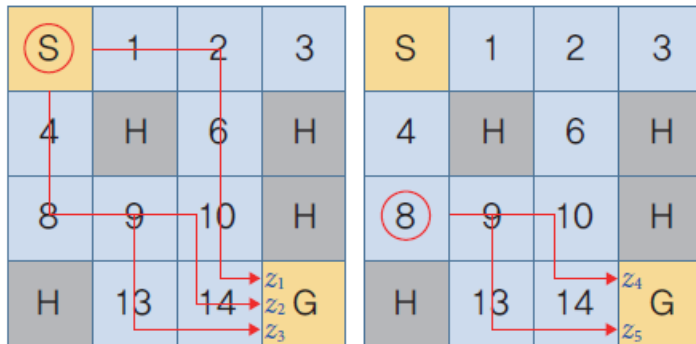
### ■ 가치 함수는 모든 상태의 기대 이득을 계산해야 완성

- 상태 8에 대해 계산하면, 상태 8의 가치는 1

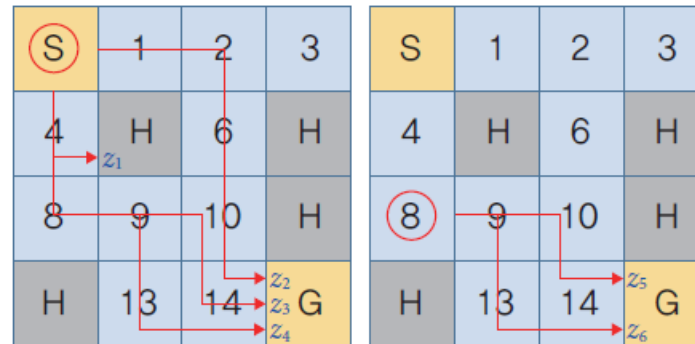
$$z_4 = [0,8]R[0,9]R[0,10]D[0,14]R[1,15] \quad p(z_4)=0.5 \quad R(z_4)=1$$

$$z_5 = [0,8]R[0,9]D[0,13]R[0,14]R[1,15] \quad p(z_5)=0.5 \quad R(z_5)=1$$

$$\rightarrow v_{\pi_2}(8) = p(z_4)R(z_4) + p(z_5)R(z_5) = 1$$



(a)  $\pi_2$ 에서 발생한 모든 궤적



(b)  $\pi_3$ 에서 발생한 모든 궤적

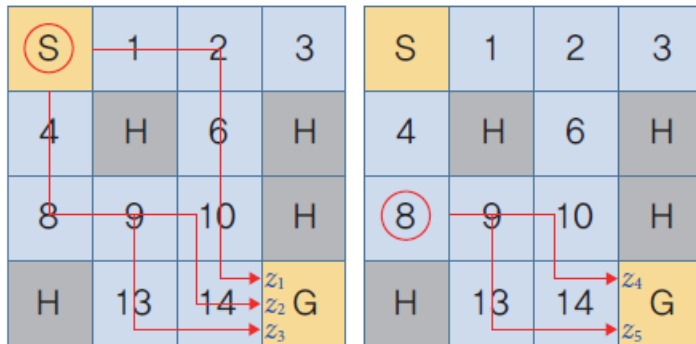
그림 2-8 정책  $\pi_2$ 와  $\pi_3$ 에서 발생한 모든 궤적

## 2.4.3 가치 함수의 계산

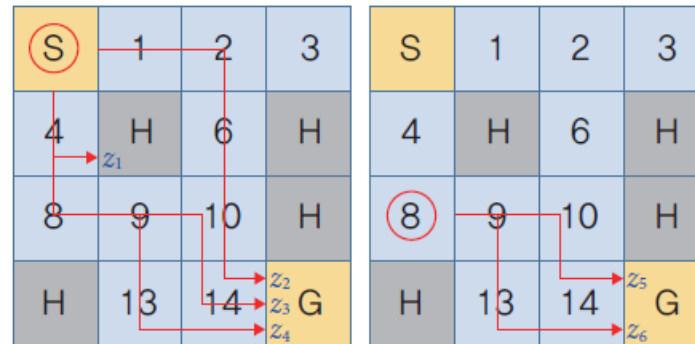
### ■ $\pi_2$ 를 약간 개조한 정책 $\pi_3$

- 상태 4에서 D와 R을 각각 0.5 확률로 선택 ([그림 2-8(b)])

$$\pi_3 = \begin{cases} p(L | 4)=0, p(D | 4)=0.5, p(R | 4)=0.5, p(U | 4)=0 \\ \text{나머지 상태는 } \pi_2 \text{와 동일} \end{cases}$$



(a)  $\pi_2$ 에서 발생한 모든 궤적



(b)  $\pi_3$ 에서 발생한 모든 궤적

그림 2-8 정책  $\pi_2$ 와  $\pi_3$ 에서 발생한 모든 궤적

## 2.4.3 가치 함수의 계산

### ■ $\pi_3$ 에서 가치 함수 계산

- 상태 0에 대해 계산하면,

$$z_1 = [0,0]D[0,4]R[0,5]$$

$$p(z_1) = 0.25 \quad R(z_1) = 0$$

$$z_2 = [0,0]R[0,1]R[0,2]D[0,6]D[0,10]D[0,14]R[1,15]$$

$$p(z_2) = 0.50 \quad R(z_2) = 1$$

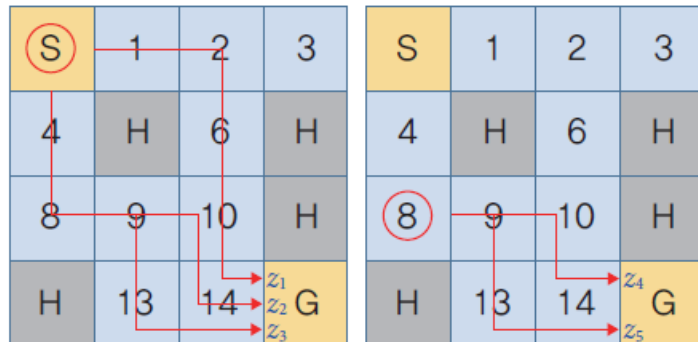
$$z_3 = [0,0]D[0,4]D[0,8]R[0,9]R[0,10]D[0,14]R[1,15]$$

$$p(z_3) = 0.125 \quad R(z_3) = 1$$

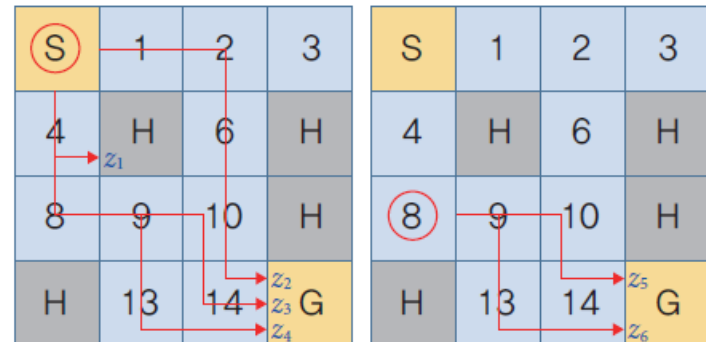
$$z_4 = [0,0]D[0,4]D[0,8]R[0,9]D[0,13]R[0,14]R[1,15]$$

$$p(z_4) = 0.125 \quad R(z_4) = 1$$

$$\rightarrow v_{\pi_3}(0) = p(z_1)R(z_1) + p(z_2)R(z_2) + p(z_3)R(z_3) + p(z_4)R(z_4) = 0.75$$



(a)  $\pi_2$ 에서 발생한 모든 궤적



(b)  $\pi_3$ 에서 발생한 모든 궤적

그림 2-8 정책  $\pi_2$ 와  $\pi_3$ 에서 발생한 모든 궤적

## 2.4.3 가치 함수의 계산

### ■ $\pi_3$ 에서 가치 함수 계산

- 상태 8에 대해 계산하면,

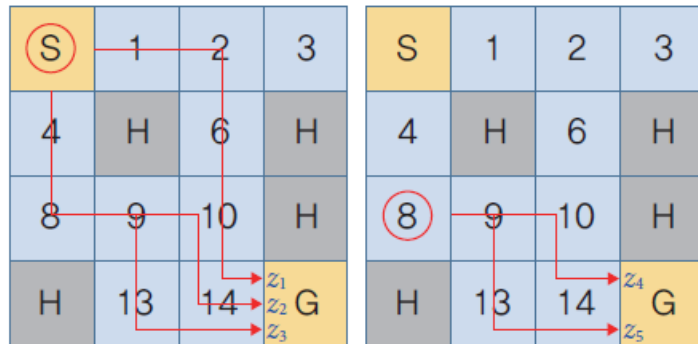
$$z_5 = [0,8]R[0,9]R[0,10]D[0,14]R[1,15]$$

$$p(z_5) = 0.5 \quad R(z_5) = 1$$

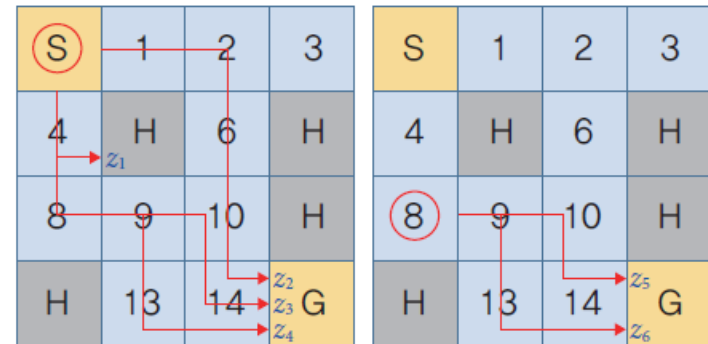
$$z_6 = [0,8]R[0,9]D[0,13]R[0,14]R[1,15]$$

$$p(z_6) = 0.5 \quad R(z_6) = 1$$

$$\rightarrow v_{\pi_3}(8) = p(z_5)R(z_5) + p(z_6)R(z_6) = 1$$



(a)  $\pi_2$ 에서 발생한 모든 궤적



(b)  $\pi_3$ 에서 발생한 모든 궤적

그림 2-8 정책  $\pi_2$ 와  $\pi_3$ 에서 발생한 모든 궤적

- $\pi_2$ 와  $\pi_3$ 은 상태 8에서는 가치가 같은데, 상태 0에서는 왜 다를까?

## 2.4.3 가치 함수의 계산

### ■ 행동 가치 함수 계산

- 상태  $s$ 에서 행동  $a$ 를 취했을 때 다음 상태  $s'$ 와 보상  $r$

$$q_{\pi}(s,a) = r + \mathbb{E}(R|s') = r + \sum_{s' \text{에서 출발하는 궤적 } z} p(z)R(z), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \quad (2.12)$$

### ■ [예제 2-2] 식 (2.12)를 이용한 행동 가치 함수 계산

- 정책  $\pi_2$ 의 사례
- 상태 0에서 행동 D를 취했을 때 가치 계산

$$z_1 = [0,4]D[0,8]R[0,9]R[0,10]D[0,14]R[1,15] \quad p(z_1)=0.5 \quad R(z_1)=1$$

$$z_2 = [0,4]D[0,8]R[0,9]D[0,13]R[0,14]R[1,15] \quad p(z_2)=0.5 \quad R(z_2)=1$$

$$\rightarrow q_{\pi_2}(0,D) = 0 + (p(z_1)R(z_1) + p(z_2)R(z_2)) = 1$$

## 2.4.4 에피소드 과업과 영구 과업

### ■ 유한한 시간에 끝나는 에피소드 과업

- FrozenLake, 바둑, 체스 등의 과업

### ■ 무한에 가까운 영구 과업

- 장시간 작업하는 로봇 과업 등
- 이득이 너무 커지는 경우를 방지하기 위해 이득 계산할 때 할인율(discount factor)  $\gamma$ 를 적용

$$R(e) = r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^{T-1} r_{T-1} = \sum_{i=0, T-1} \gamma^i r_i \quad (2.13)$$

$$R_t(z) = r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} = \sum_{i=t, T-1} \gamma^{i-t} r_i \quad (2.14)$$

- 할인율  $\gamma \leq 1$

### ■ 할인율은 신용 할당(credit assignment)에 유용

- 예를 들어  $\gamma = 0.99$ 라면, 바둑에서 승리했을 때  $a_{T-1}$ 은 1.0,  $a_{T-2}$ 는 0.99, ...,  $a_{T-3}$ 는 0.9801,  $a_{T-4}$ 는 0.970299만큼 보상을 부여. 종료 순간에서 멀수록 작아짐



*Thank you*

---

Jahoon Koo  
([sigmao@korea.ac.kr](mailto:sigmao@korea.ac.kr))