

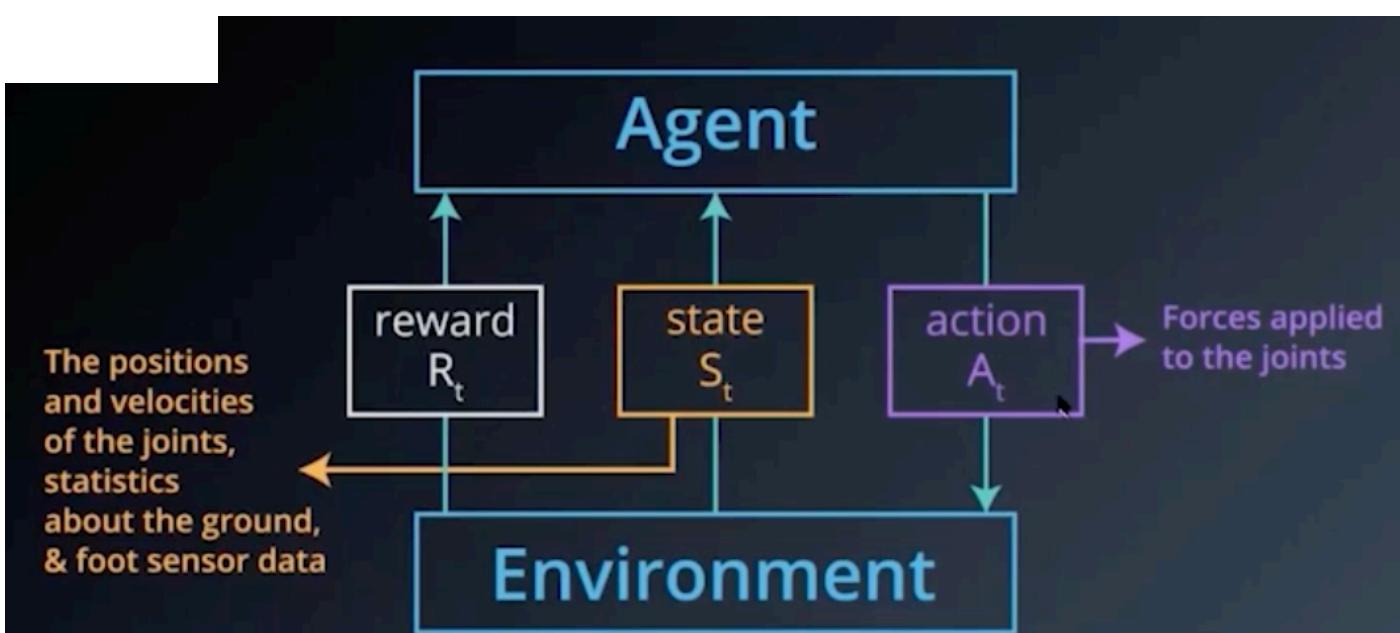
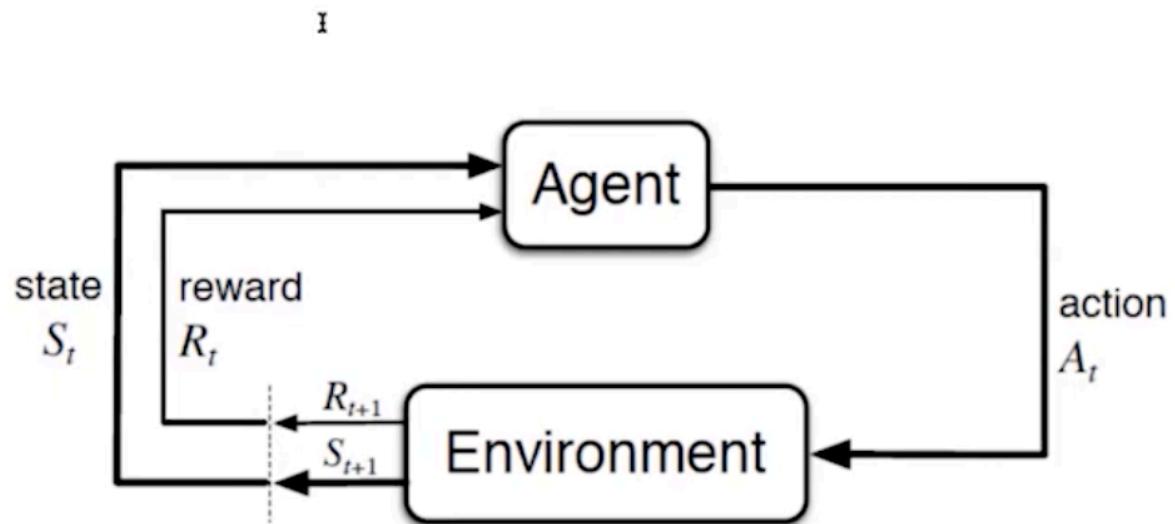
Reinforcement Learning using starcraft 2

(사내 강의 내용 정리)

주차	기간	내용	Project 과제
1	2020. 08. 12 ~ 08. 18	웨비나, Project 과제1 수행	Star2 환경설정
2	2020. 08. 19 ~ 08. 25	웨비나, Project 과제2 수행	기본 Agent만들기
3	2020. 08. 26 ~ 09. 01	웨비나, Project 과제3 수행	강화학습 Agent만들기
4	2020. 09. 02 ~ 09. 08	웨비나, Project 과제4 수행	심층강화학습 Agent만들기
5	2020. 09. 09 ~ 09. 15	Final Project 수행	나만의 심층강화학습 Agent만들기
6	2020. 09. 16 ~ 09. 22	Final Project 수행, 오프라인 코칭	
7	2020. 09. 23 ~ 09. 29	Final Project 오프라인 공유회	

RL Framework - Problem

RL is a learning paradigm
to solve **Sequential Decision Making Problem.**



Goal of the Agent:

Maximize expected **cumulative** reward

$$R_1 + \dots + R_t + R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \dots$$

in the past
(already decided) immediate reward
after A_t in the future

나중에 받을 보상의 가치를 낮게
평가하여 계산
Discounted Reward

현재 action 이후의 모든 보상의 합계

Definition

The return at time step t is

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \dots$$

At time step t , the agent picks A_t to maximize (expected) G_t

At time step t , the agent picks A_t to maximize (expected) G_t .

~~$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \dots$$~~

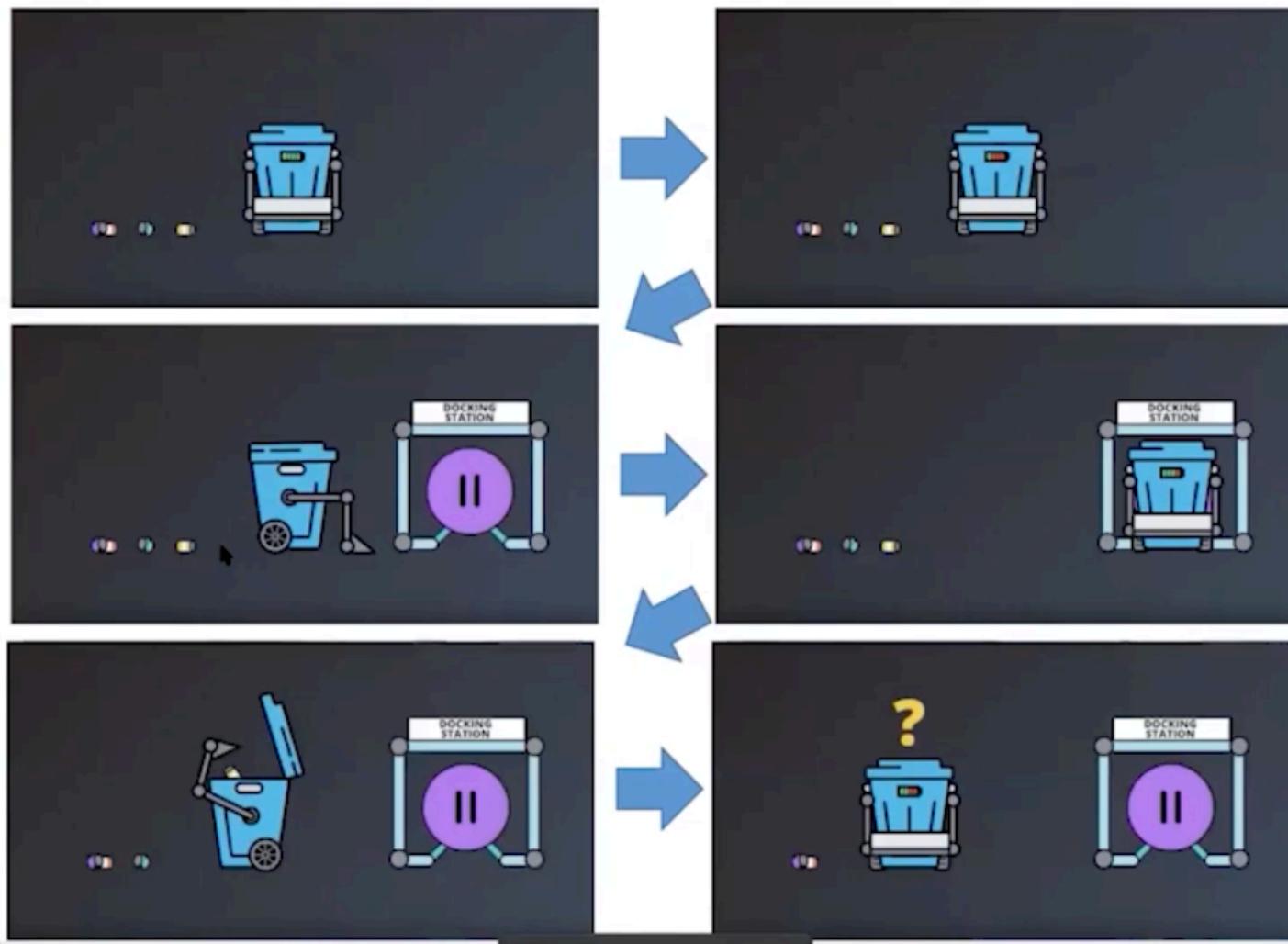
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots$$



discounted return

discount rate $\gamma \in [0, 1]$

One step dynamics



로봇의 배터리 양을 상태로 정의

What are the actions?

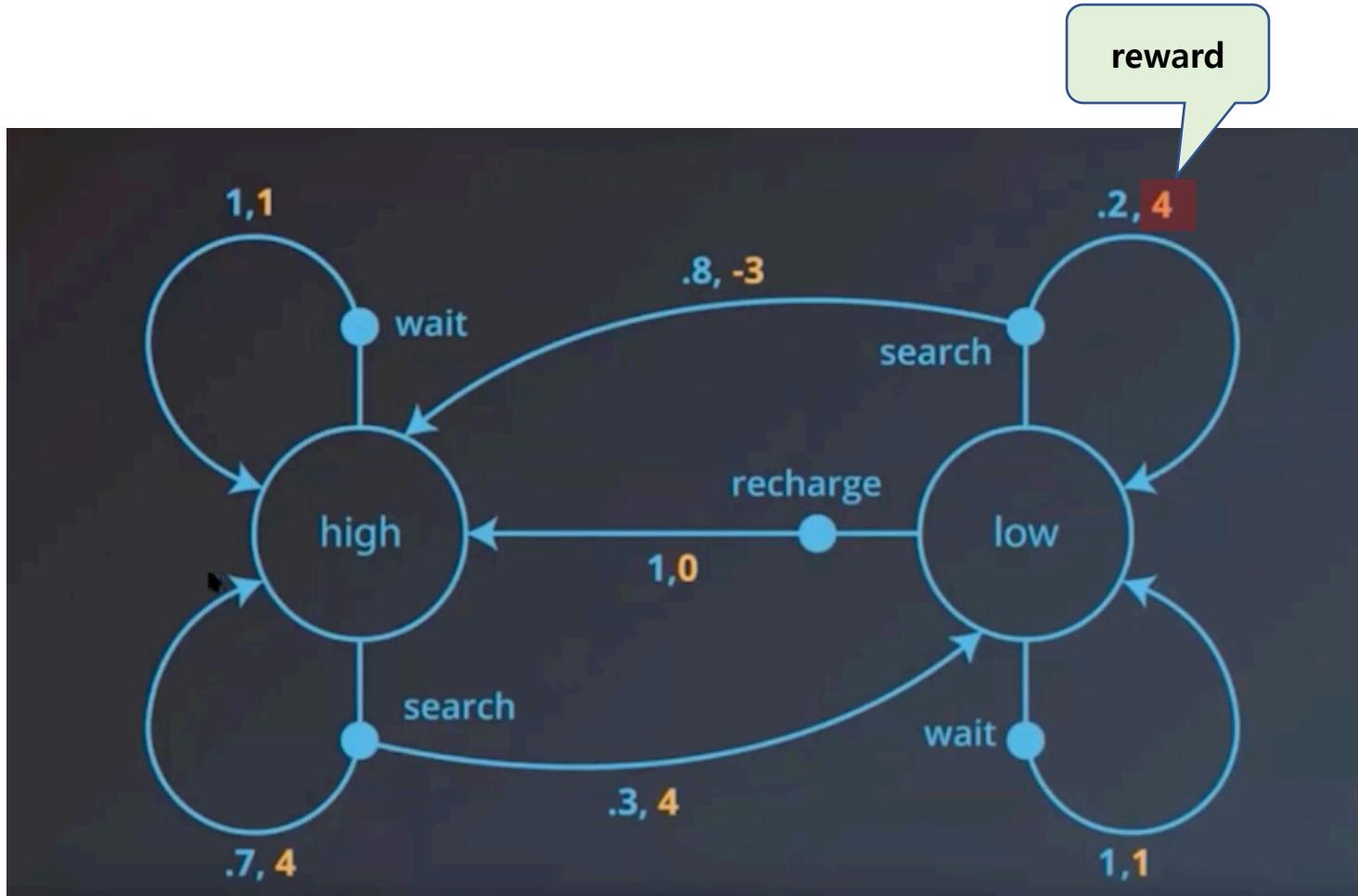
$$\mathcal{A} = \begin{bmatrix} \text{search} \\ \text{recharge} \\ \text{wait} \end{bmatrix}$$

What are the states?

$$\underline{\mathcal{S}} = \begin{bmatrix} \text{high} \\ \text{low} \end{bmatrix}$$

State





- 특정 상태(high)에서 특정 action을 했을 때,
- 다른 상태(high or low)로 변할 확률을 정의
 - 예를 들어 high \rightarrow wait action을 실행한 경우
 - 100%의 확률로 high 상태로 변함
 - Low 상태에서 "search" action은
 - 0.8 확률로 high 상태로 변하고,
 - 0.2 확률로 low 상태로 변하는 것이다.
- 하지만, Agent는 이렇게 정의된 환경을 알 수 없다.
 - 단지 action에 대한 reward를 통해서 판단

MDP : 문제를 풀기 위해 문제를 정의한 방법

현재 상태, 액션이 주어졌을 때,
다음 상태와 보상의 확률을
One-step dynamics에서 정의
(Agent는 미리 알 수 없음,
action을 실행해야 알 수 있음)

Definition

A (finite) Markov Decision Process (MDP) is defined by:

- ◆ a (finite) set of states \mathcal{S}
- ◆ a (finite) set of actions \mathcal{A}
- a (finite) set of rewards \mathcal{R}
- the one-step dynamics of the environment
 $p(s', r | s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$
- ◆ a discount rate $\gamma \in [0, 1]$ for all s, s', a and r

◆ - Agent knows ● - Agent does not know

MDP : 문제를 풀기 위한 방법

RL Framework - Solution

- Policy
- State Value Function
- Action Value Function
- Bellman Equation
- Optimal Value Function
- Optimal Policy

Policy (π)

현재 상태가 주어졌을 때, 최적의 action(보상이 큰)을 제공해 주는 함수

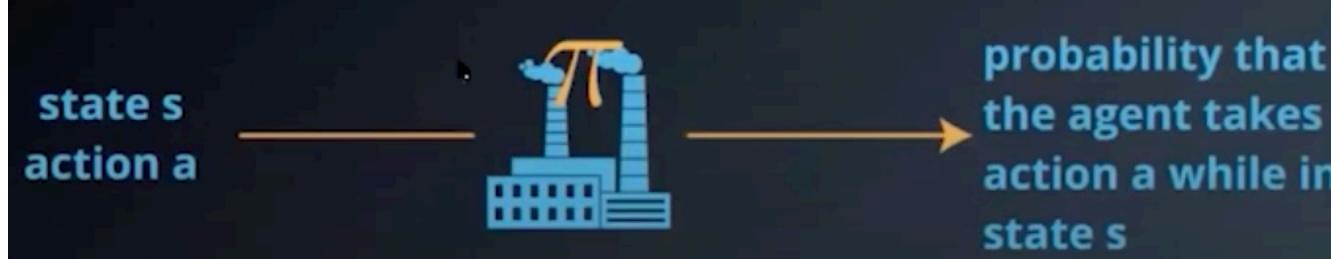
A deterministic policy is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$



- 예측 가능한 환경에서
- 특정 상태에서 특정한 action은
- 항상 동일한 보상을 제공하는 함수
- 즉, 특정 상태에서 최적의 action이 변하지 않음

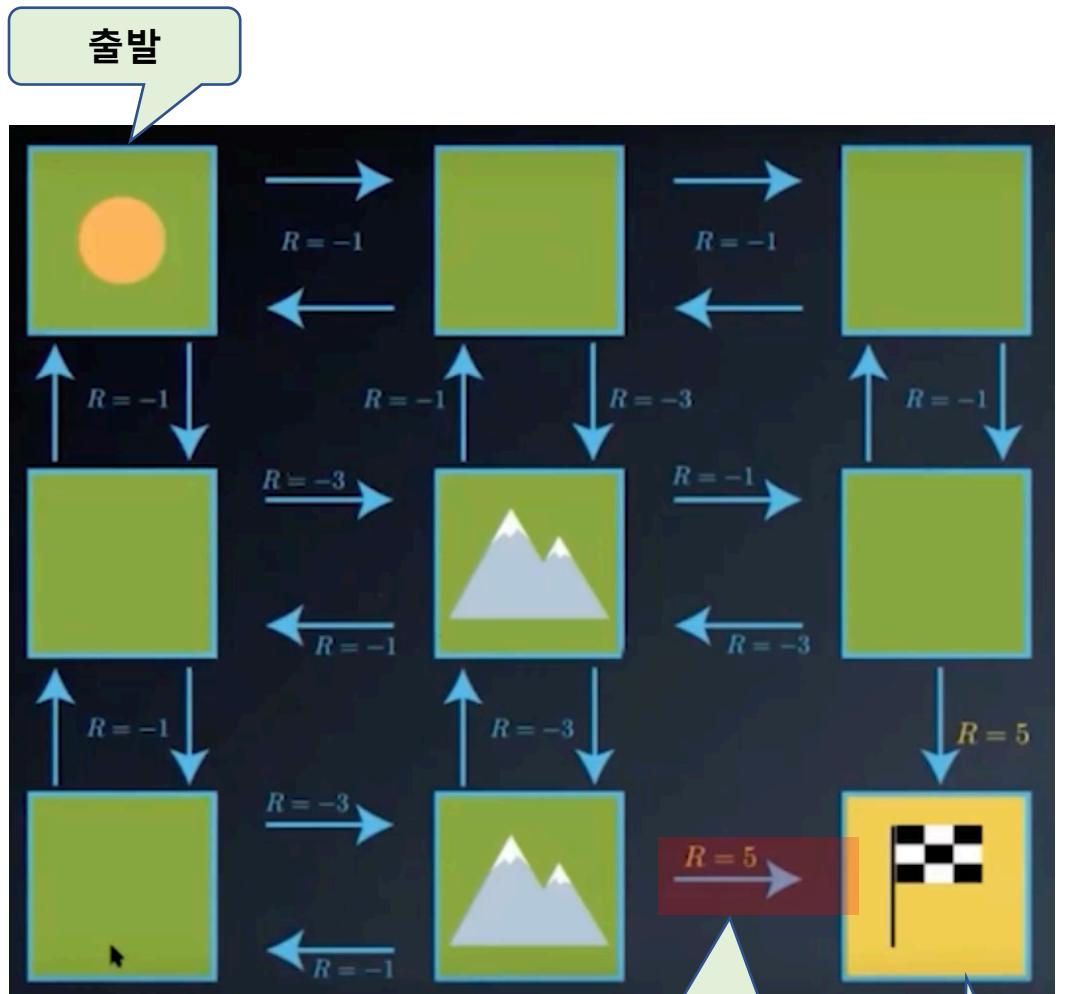
A stochastic policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$



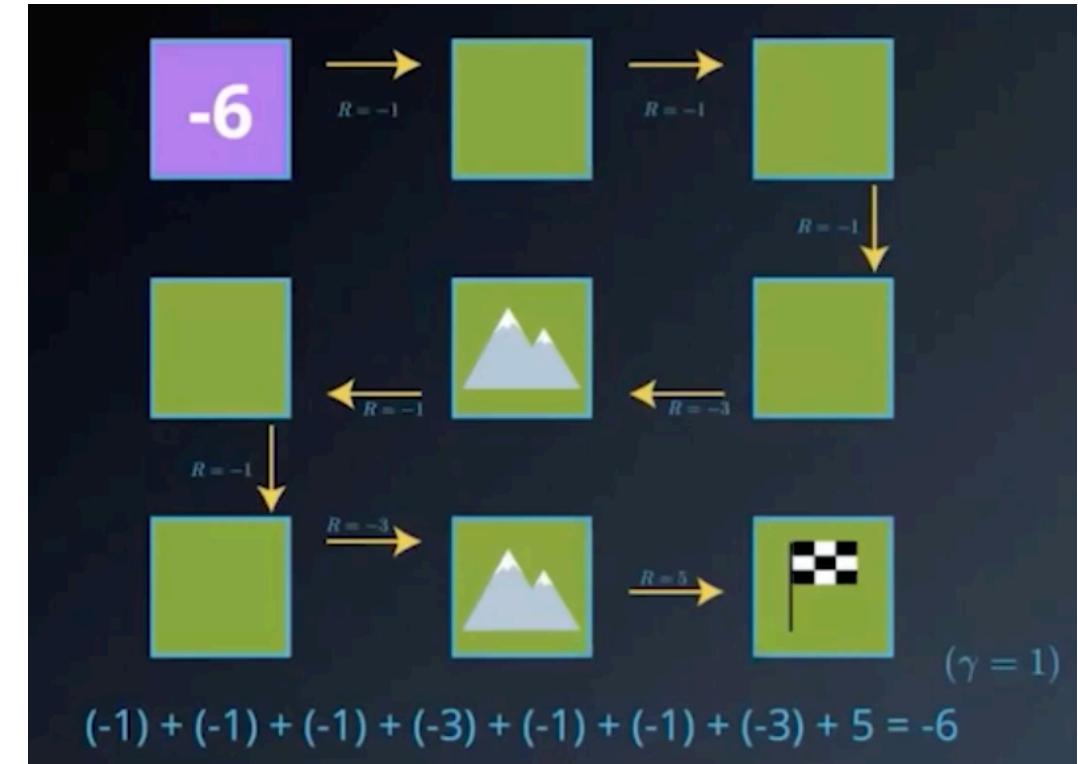
- 예측 불가능한 환경에서
- 특정 상태에서 취해야 할 action을 100% 확신 할 수 없는 환경
- 즉, 각 action의 확률로만 표현이 가능함

State-Value Function (목적지 까지 경로에 대한 값)



목적지에 도착해야만 양수의
reward가 부여됨.
그 전까지는 음수의 reward

도착



State-Value Function 계산 결과

현재 상태에서 Policy(π)에 따라
예상되는 목적지 까지 최종 reward 값을 계산

Definition

We call v_π the state-value function for policy π
The value of state s under a policy π is

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

For each state s
it yields the **expected return**
if the agent starts in state s
and then uses **the policy**
to choose its actions for all time steps

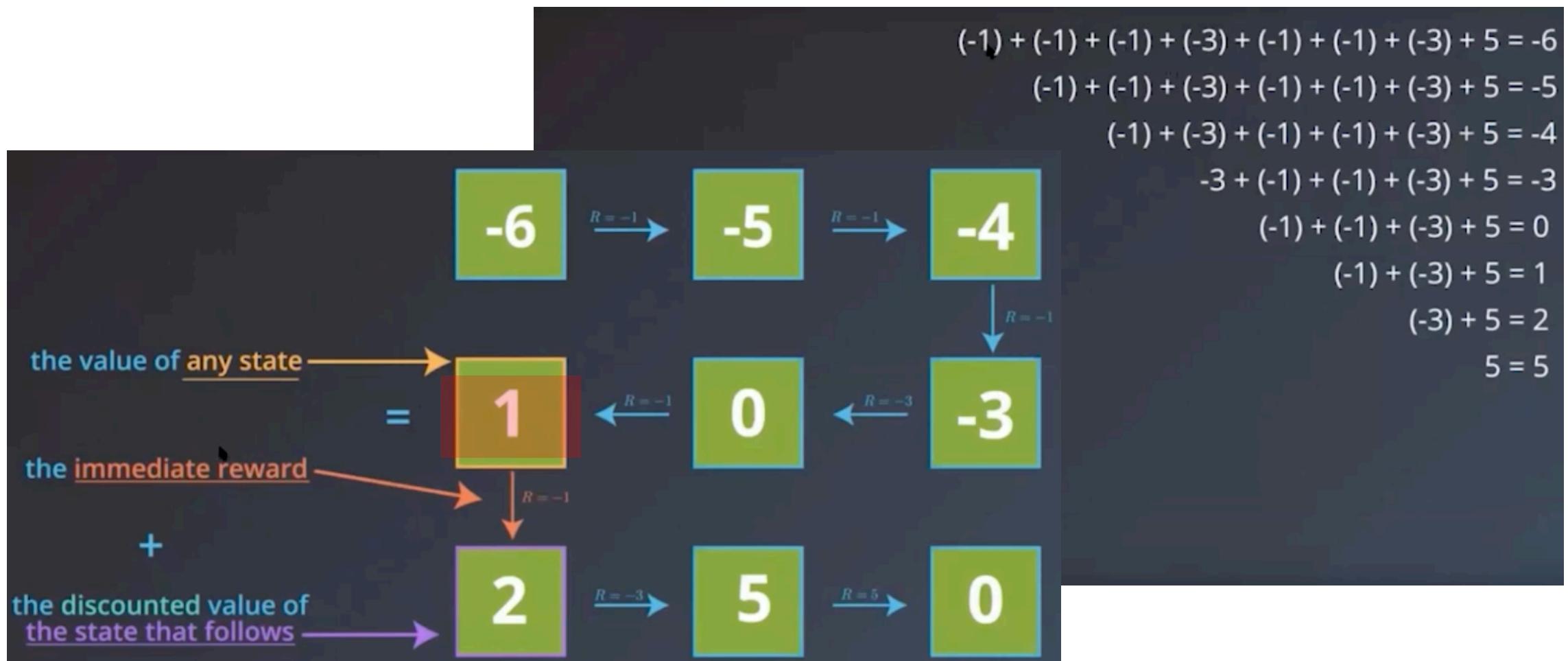
-6	-5	-4
1	0	-3
2	5	0

산맥에 가로막혀서,
더 이상 진행 불가

최종 목적지에 도착했으므로,
더 이상의 reward는 없음

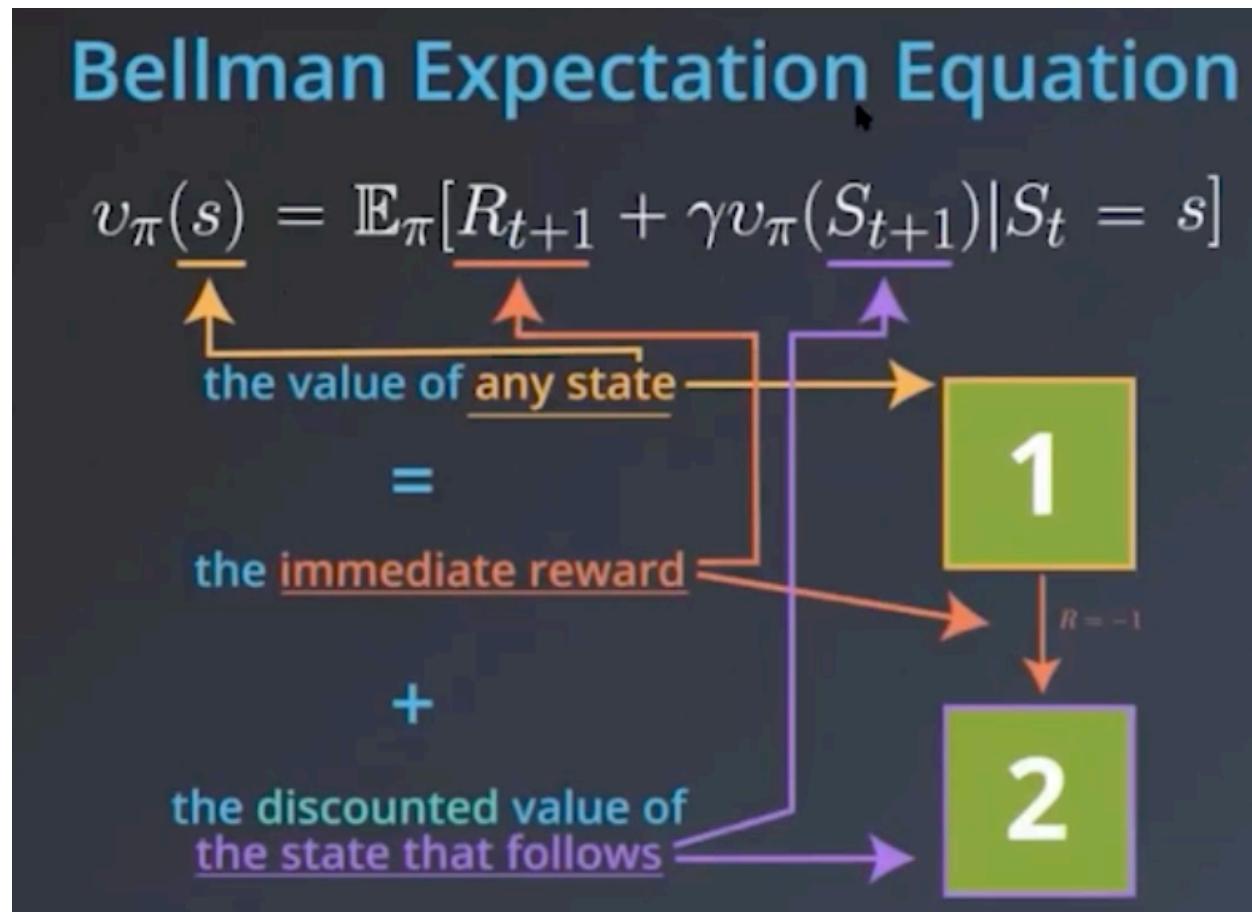
State-Value Function 계산 (Bellman Equation)

특정 상태의 value는 선택한 action의 보상 + 미래의 보상의 합계
($1 = -1 + 2$)



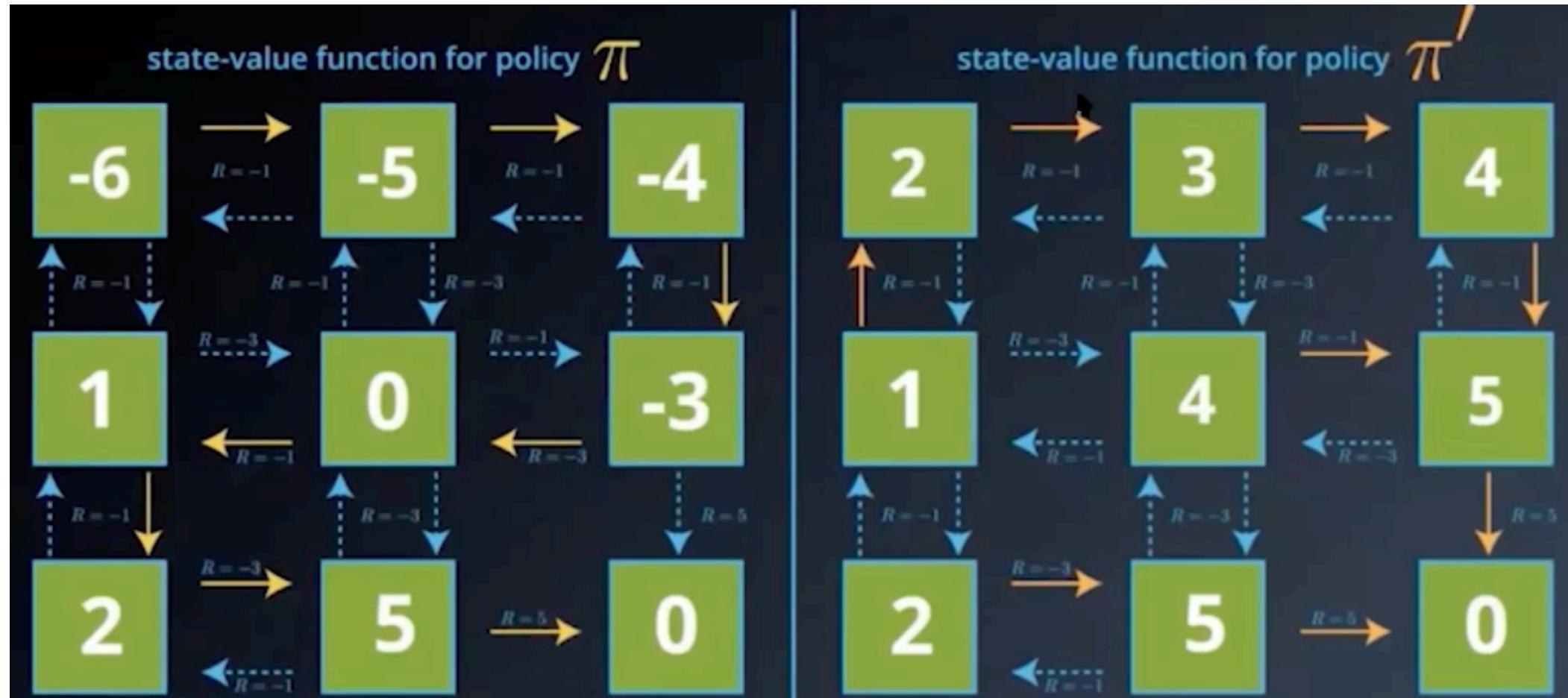
Bellman Equation

모든 상태를 계산하지 않아도, 이 공식을 이용하여 전체 보상 예측 가능



Optimal Policy

다수의 policy 중에서 각 state의 value가 다른 policy보다 큰 policy
(optimal policy는 여러 개 존재 가능)



Action-Value Function

State-Value

Definition

We call v_π the **state-value function** for policy π .

The value of state s under a policy π is

$$v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s]$$

For each state s,
it yields the expected return
if the agent starts in state s
and then uses the policy
to choose its actions for all time steps.

Action-Value

Definition

We call q_π the **action-value function** for policy π .

The value of taking action a in state s under a policy π is

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$$

For each state s and action a,
it yields the expected return
if the agent starts in state s
then chooses action a
and then uses the policy
to choose its actions for all time steps.

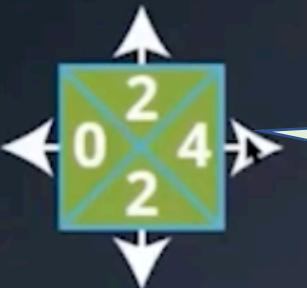
Action-Value Function

현재 state의 action 중에서 optimal policy를 제공하는 action 선택
즉, Action-value function만 미리 계산하면 쉽게 선택 가능

For each state, the state-value function yields the expected return,
if the agent starts in that state, and then follows the policy for all time steps.

4

For each state and action, the action-value function yields the expected
return, if the agent starts in that state, takes the action, and
then follows the policy for all future time steps.



여기서는 “오른쪽”
action 실행

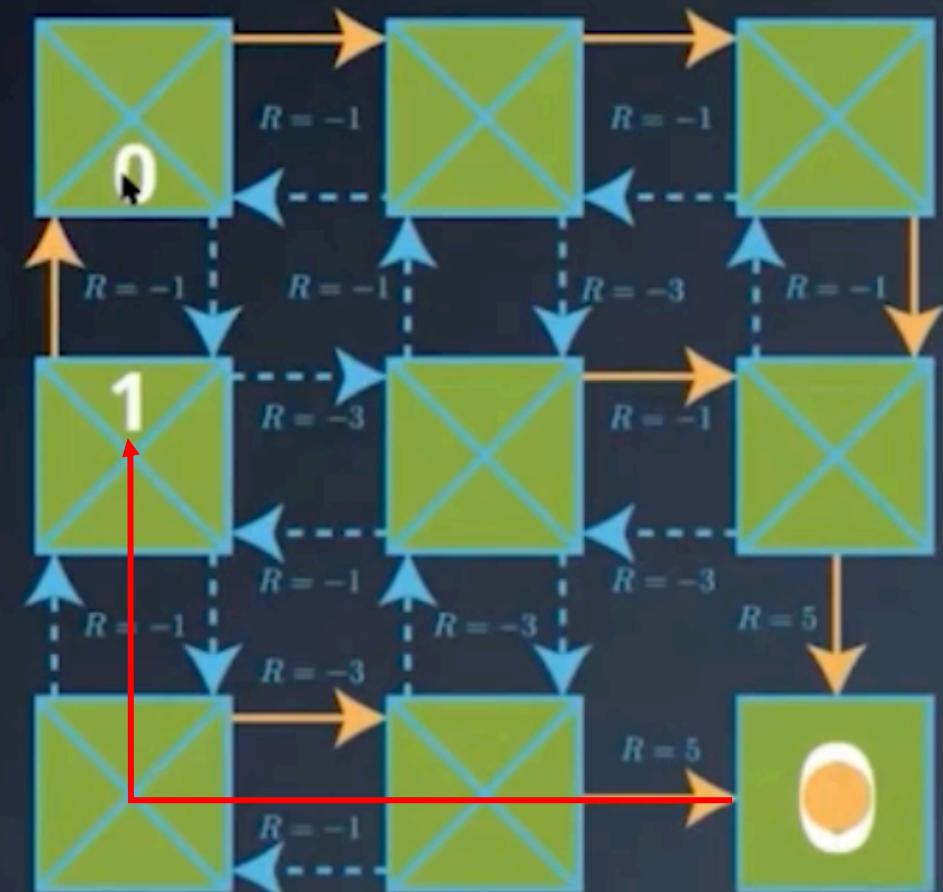
Action-Value Function 계산 방식

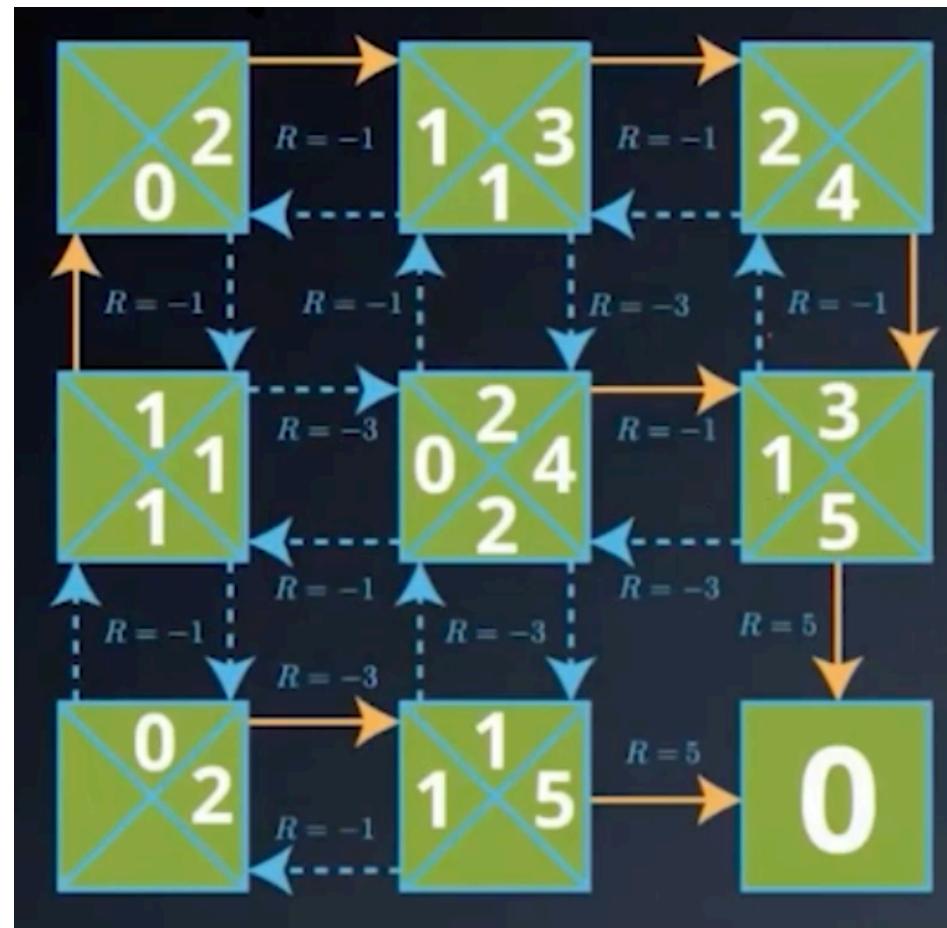
For each state and action, the action-value function yields the expected return, if the agent starts in that state, takes the action, and then follows the policy for all future time steps.

reward from
action "up"

$$(-1) + (-1) + (-1) + (-1) + 5 = \boxed{1}$$

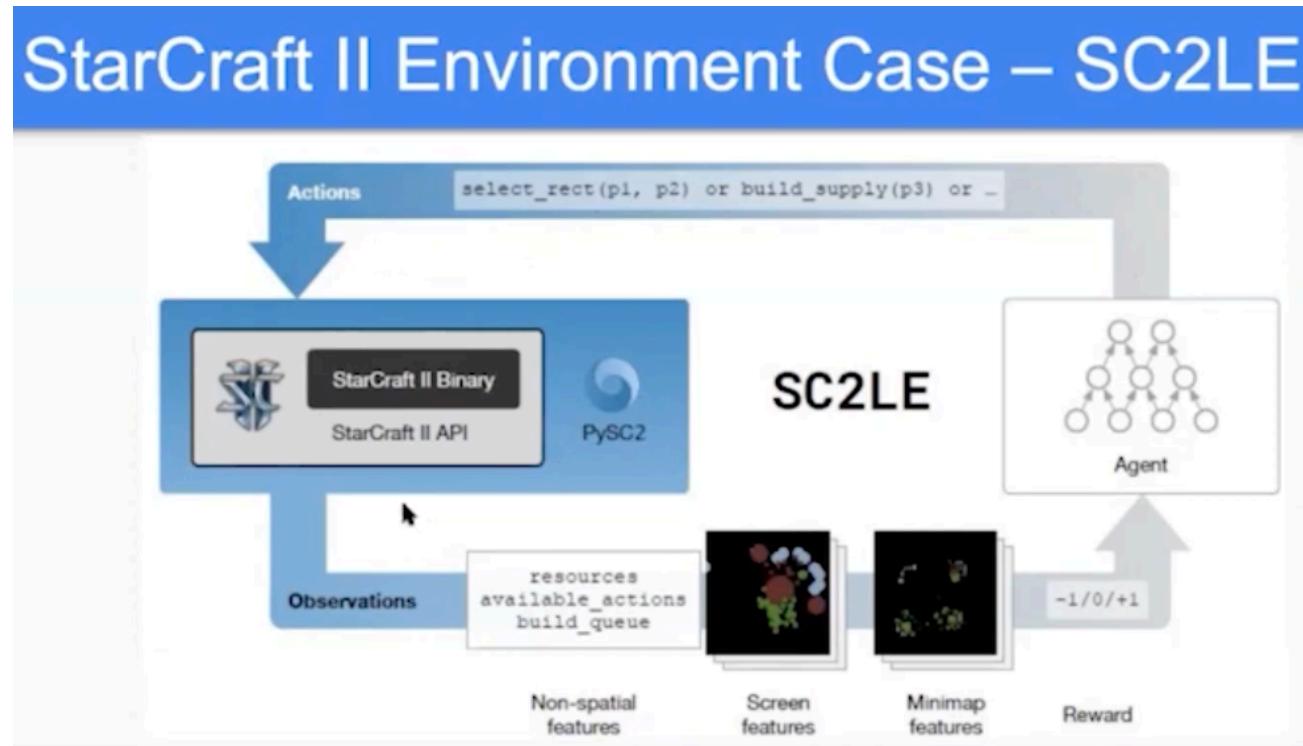
reward from following the policy
for all future time steps





- **S0**
 - $2 = -1 -1 -1 + 5$
 - $0 = -1 -1 -3 + 5$

Week2.



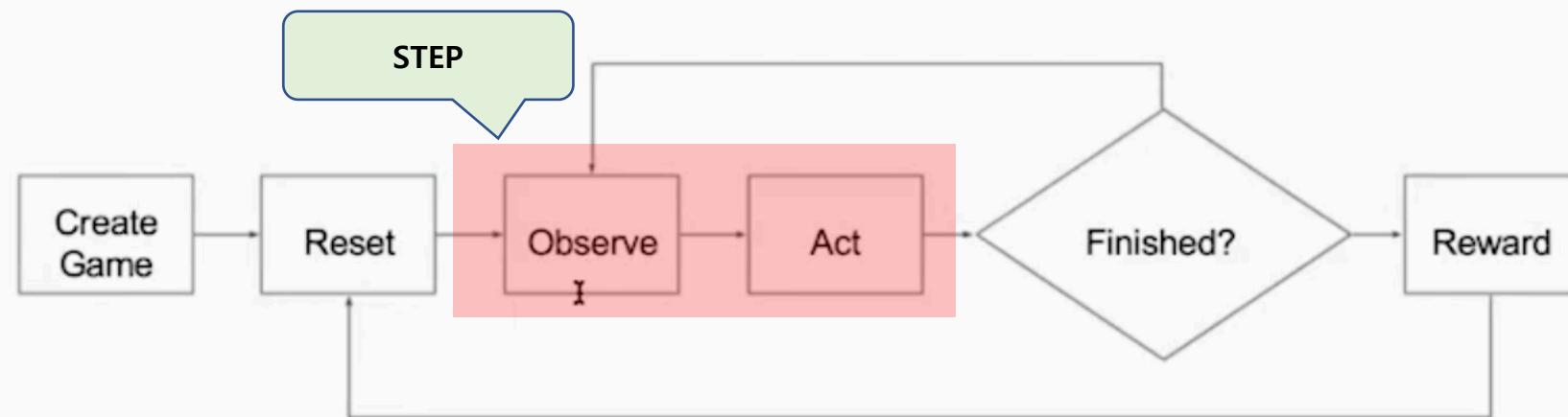
<https://deepmind.com/documents/110/sc2le.pdf>

What is Blizzard's StarCraft II API?

- Allows you to interact with the game via Protobuf
- Has limited Linux support
- Can play replays
- Provides the ability to investigate game state
- Provides the ability to perform player actions
- Supports 2 players
- Has some limitations
- Is still being developed

Ref : <https://blizzard.github.io/s2client-api/index.html>

Game Engine Flow



Main Game Options



Code Example of Game Options

```
with sc2_env.SC2Env(  
    map_name="Simple64",  
    players=[sc2_env.Agent(sc2_env.Race.terran),  
            sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],  
    agent_interface_format=features.AgentInterfaceFormat(  
        feature_dimensions=features.Dimensions(screen=84,  
                                                minimap=64),  
        action_space=actions.ActionSpace.FEATURES),  
) as env:
```

Map
Player
Screen
minimap

Non-Spatial
Feature를 리턴
받음

특정한 목적을 위해 작게 만들어진 Mini Game

Maps - Mini Games

마린을 최대한 빨리
생성

Simplified games designed for testing algorithms

- BuildMarines
- CollectMineralsAndGas
- CollectMineralShards
- DefeatRoaches
- DefeatZerglingsAndBanelings
- FindAndDefeatZerglings
- MoveToBeacon



Maps - Melee Maps

자원(가스)을 최대한
많이 수집

Full Game
(전체 게임)

- Designed for full game play
- Empty128
- Flat32/48/64/96/128 - No terrain variations
- Simple64/96/128 - Some terrain

Bot Difficulties

- very_easy
- easy
- medium
- medium_hard = Hard
- hard = Harder
- harder = Very hard
- very_hard = Elite
- cheat_vision
- cheat_money
- cheat_insane

Other Useful Game Options

step_mul

The number of steps to take before acting again, there are 22.4 steps per second for normal “faster” games, so a value of 8 = 168 APM, I have had issues below a step mul of 2

1초에 22.4 frame이 실행되며,
60초면 실행 가능한 action은 $22.4 * 60\text{초} = 1,334$
실제 사람은 1초에 24 action을 취할 수 없음
따라서 사람과 유사하게 조정하기 위한 값
8로 지정하면, $22.4/8 = 2.8$ (약 3번의 action 실행)
60초 기준으로 168($1334/8$) APM

game_steps_per_episode

The maximum steps to take before the game automatically ends

1 게임당 최대 action(step)
너무 게임이 길게 유지되지 않도록 최대 action을 제한

visualize

Shows a custom rendered version of the observations, good for debugging but slows the game considerably

게임이 학습되는 상황(Feature의 변화)을 UI로 보여줄 것인지? (성능을 높이려면 disable)

disable_fog

Disables the fog of war so everything is visible

자신이 탐색하고 있는 영역 외에 적군의 영역도 보여 줄 것인지?
(Map Hack 같은 효과)

Machine Learning Tips

- Start against very easy bots
- Play as a single race
- Play against a single race
- Disable the fog of war
- Use a single, simple map
- Human players can beat very easily with an APM <60 so consider a step_mul of 20-30

- Limit the games to half an hour or so (40,320 steps)
- Consider having your agent play against a simple scripted bot
- Consider self-play, you get twice the learning per game but it may not generalise
- Compare your bot against one that chooses completely random actions

초기 학습 시 권장 사항

Observation

```
state = (command_center_count,  
         supply_depot_count,  
         barracks_count,  
         scv_count,  
         marine_count,  
         base1_enemy_count,  
         base2_enemy_count,  
         base3_enemy_count,  
         base4_enemy_count,  
         base1_friendly_count,  
         base2_friendly_count,  
         base3_friendly_count,  
         base4_friendly_count)
```

상태 값을 정의
(Unit의 개수 정보)

현재의 Step
이 값을 22.4(step_mul)로 나누면 현재까지 시간
(초) 산출

State별로 가능한 action이 정해짐
(배력이 없으면 마린을 생성할 수 없음)

게임의 제일 처음 Step이 발생했을 때, 호출되는 함수
게임 시작 시 초기화 로직을 여기에 추가

Step "obs"

obs.first()

Whether or not this is the first step of the game, good for
doing things like position and race detection

obs.last()

Whether or not this is the last step, use this to learn,
save, and wrap things up

obs.observation.reward

Use this for sparse rewards, will be 0 for mid-steps or a
draw, -1 for a loss, 1 for a win

obs.observation.game_loop

The current step, you can work out the current second if
you divide by 22.4

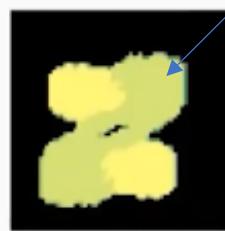
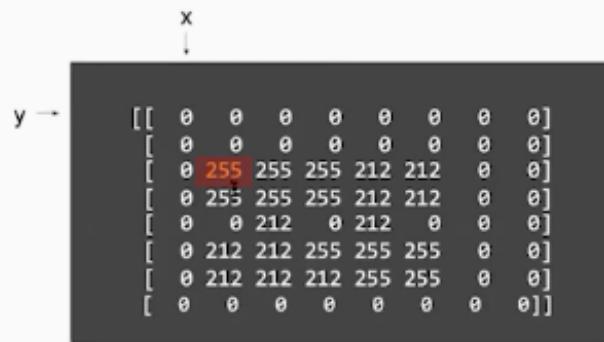
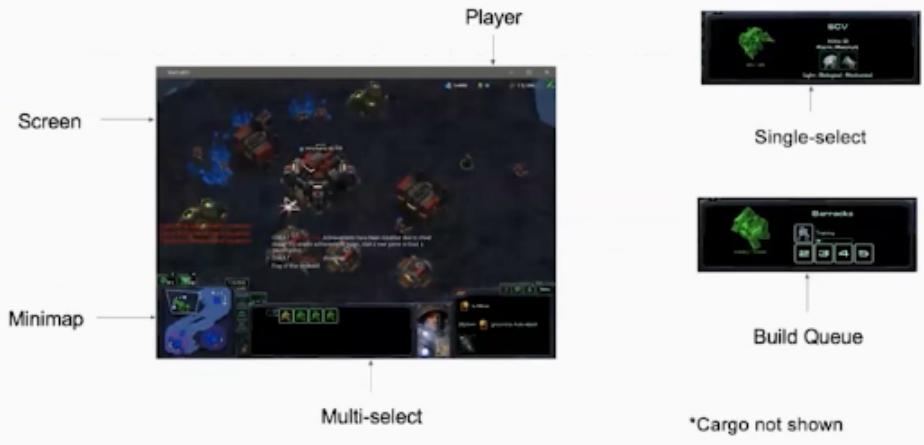
obs.observation.available_actions

Contains a list of actions that can be performed in the
current state

obs.observation
Everything else

Observation Features

Main "Features"



```
height_map = obs.observation.feature_minimap.height_map  
height_map[y][x]  
height_map[2][1] = 255
```

등고선 (높이 표시)

Some Screen Feature



Height map



Visibility

유닛의 유형



Unit type

Player relative

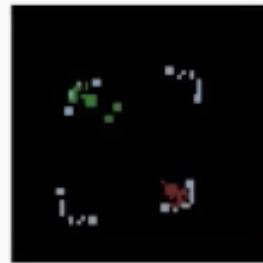
시야 (확인 가능한 영역)

Other important features include power, creep and effects

적, 중립, 우군 여부

Categorical 변수 예시

```
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 3 1 0 3 3 0 0]
 [0 3 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 3 0 0 0 3 0 0]
 [0 3 3 0 3 3 0 0]
 [0 0 0 0 0 0 0 0]]
```



```
player_relative = obs.observation.feature_minimap.player_relative
```

문자열 값을 숫자형으로 변환하기 위한 방법 중 하나로,
행 형태로 표현된 특성의 고유값을 열 형태로 차원을 변환하고 해당 컬럼만 1로 표시하고
나머지 컬럼은 0으로 표시하는 방식 (**Dummy Coding**)

원본데이터

상품분류
TV
냉장고
전자렌지
컴퓨터
선풍기
선풍기
믹서
믹서

One-Hot Encoding

상품분류_TV	상품분류_냉장고	상품분류_전자렌지	상품분류_컴퓨터	상품분류_선풍기	상품분류_믹서
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1

```
1 import pandas as pd
2
3 df = pd.DataFrame({'item':['TV', '냉장고', '전자렌지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']})
4 pd.get_dummies(df)
```

상태는 그대로, 원본 데이터를 더해온 형태

```
[[False False False False False False False]
 [False False False False False False False]
 [False False True False False False False]
 [False False True False False False False]
 [False False False False False False False]
 [False False False False False False False]
 [False False False False] False False False]
 [False False False False False False False False]
 [False False False False False False False False]]
```



```
player_relative = obs.observation.feature_minimap.player_relative
minimap_self = (player_relative == features.PlayerRelative.SELF)
```

```
([2, 2], # [x1, y1]
 [3, 2]) # [x2, y2]
```



```
player_relative = obs.observation.feature_minimap.player_relative
player_y, player_x = (player_relative == features.PlayerRelative.SELF).nonzero()
player_xy = zip(player_x, player_y)
```

Feature가 아닌 실제 이
미지를 가져올 때

Enable RGB Observations

```
with sc2_env.SC2Env(  
    map_name="Simple64",  
    players=[sc2_env.Agent(sc2_env.Race.terran),  
            sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],  
    agent_interface_format=features.AgentInterfaceFormat(  
        rgb_dimensions=features.Dimensions(screen=84,  
                                              minimap=64),  
        action_space=actions.ActionSpace.RGB),  
) as env:
```

RGB Observations

```
[[[255 255 255]  
 [  0   0   0]  
 [  0   0   0]  
 [  0   0   0]  
 [255 255 255]  
 [  0   0   0]  
 [241 191 126]  
 [241 191 126]  
 [241 191 126]  
 [  0   0   0]  
 [ 15  12  11]  
 [ 17  11  10]]]
```

B G R

*BGR instead of RGB may be a bug

obs.observation.rgb_minimap
obs.observation.rgb_screen

Player Observations

```
obs.observation.player.player_id  
obs.observation.player.minerals  
obs.observation.player.vespene  
obs.observation.player.food_used  
obs.observation.player.food_cap  
obs.observation.player.food_army  
obs.observation.player.food_workers  
obs.observation.player.idle_worker_count  
obs.observation.player.army_count  
obs.observation.player.warp_gate_count  
obs.observation.player.larva_count  
  
free_supply = food_cap - food_used
```

생산 가능한 unit 개수

Enable Feature Units

```
with sc2_env.SC2Env(  
    map_name="Simple64",  
    players=[sc2_env.Agent(sc2_env.Race.terran),  
            sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],  
    agent_interface_format=features.AgentInterfaceFormat(  
        feature_dimensions=features.Dimensions(screen=84,  
                                                minimap=64),  
        action_space=actions.ActionSpace.FEATURES,  
        use_feature_units=True),  
    ) as env:
```

게임 생성시 feature를 사용하기 위한 설정 코드

Feature Units

- Every visible unit on screen
- Exact unit location
- Build progress
- Assigned worker count
- Ideal worker count

```
marines = [unit for unit in obs.observation.feature_units  
          if unit.unit_type == units.Terran.Marine  
          and unit.alliance == features.PlayerRelative.SELF]
```

Marin인 모든 feature를 조회
(여기서 여러 조건을 명시하여 필요한 정보만 조회)

Feature Unit Properties

```
unit_type  
alliance  
health  
shield  
energy  
cargo_space_taken  
build_progress # 0-100  
health_ratio # 0-255  
shield_ratio # 0-255  
energy_ratio # 0-255  
display_type  
owner  
x  
y  
  
facing  
radius  
cloak  
is_selected  
is_bip  
isPowered  
mineral_contents  
vespene_contents  
cargo_space_max  
assigned_harvesters  
ideal_harvesters  
weapon_cooldown  
order_length  
addon_unit_type # soon?
```

조회된 Unit의 상세 속성들
상대 유닛이 fog로 들어가면 해당 unit feature가
삭제될 수도 있다.

Action

Select an Idle Worker

```
if obs.observation.player.idle_worker_count > 0:  
    return actions.FUNCTIONS.select_idle_worker("select")
```

Selecting idle workers will move the screen

You can also "select_all"

Autocast Repair

```
if (actions.FUNCTIONS.Effect_Repair_autocast.id in  
obs.observation.available_actions):  
    return actions.FUNCTIONS.Effect_Repair_autocast()
```

Harvest Minerals

```
if (actions.FUNCTIONS.Harvest_Gather_screen.id in  
obs.observation.available_actions):  
    minerals = [unit for unit in self.obs.observation.feature_units  
               if unit.unit_type in MINERAL_UNIT_TYPES]  
    if len(minerals) > 0:  
        mineral = random.choice(minerals)  
  
    return actions.FUNCTIONS.Harvest_Gather_screen(  
        "queued", (mineral.x, mineral.y))
```

I have not had a lot of success with queued mineral harvesting

Build a Barracks

```
Function.ability(42, "Build_Barracks_screen", cmd_screen, 321),
```

Screen의 위치

```
if (actions.FUNCTIONS.Build_Barracks_screen.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Build_Barracks_screen("now", (x, y))
```

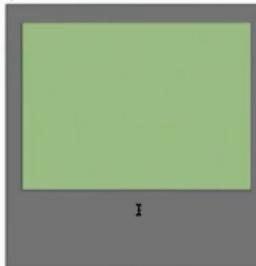
You can help your agent by preventing it from building in the mineral line

For Protoss or Zerg agents you can help by limiting to power or creep

Building Locations

It's best to add a margin to the left, top, and right, and a larger margin to the bottom

5-10%



건물을 지을 때 여유
공간 확보 필요

Selecting all Barracks

```
barracks = [unit for unit in self.obs.observation.feature_units  
            if unit.unit_type == units.Terran.Barracks  
            and unit.alliance == features.PlayerRelative.SELF]  
if len(barracks) > 0:  
    barrack = random.choice(barracks)  
  
    return actions.FUNCTIONS.select_point(  
        "select_all_type", (barrack.x, barrack.y))
```

Remember to clip the x and y coordinates to fit the screen resolution (e.g. 0-83)

Barrack is not the singular for barracks, I know

Set Rally Point

```
if (actions.FUNCTIONS.Rally_Units_screen.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Rally_Units_screen("now", (x, y))
```

You can also rally workers

You can also use the minimap

Add Barracks to Control Group

```
return actions.FUNCTIONS.select_control_group("append", 0)
```

You can also set and recall

I have had issues with this

Upgrade to Orbital Command

```
if (actions.FUNCTIONS.Morph_OrbitalCommand_quick.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Morph_OrbitalCommand_quick("now")
```

Scan

```
if (actions.FUNCTIONS.Effect_Scan_screen.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Effect_Scan_screen("now", (x, y))
```

You can also scan the minimap [I](#)

Research Stim

```
if (actions.FUNCTIONS.Research_Stimpack_quick.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Research_Stimpack_quick("now")
```

⋮

Train a Marine

```
if (actions.FUNCTIONS.Train_Marine_quick.id in  
    obs.observation.available_actions):  
    return actions.FUNCTIONS.Train_Marine_quick("now")
```

There seemed to be a bug currently that stops the correct distribution of build orders

In my experience when you have multiple units selected, all commands apply to all units even if normally they would not (e.g. SCVs and buildings)

It's hard to know if your research is done since research build queues don't exist (yet?), you may have to internally track how long it has been in progress, and check again if the research action is available (and you have enough resources)

Week 3

ACTION 및 State 정의

ACTION

```
# 아래 action 중 ACTION_ATTACK은 하나의 위치로 정할수 없기 때문에
# 4096개의 ACTION_ATTACK을 생성해야 함. (실습을 위해서는 16개로만 지정)
smart_actions = [
    ACTION_DO_NOTHING,
    ACTION_SELECT_SCV,
    ACTION_BUILD_SUPPLY_DEPOT,
    ACTION_BUILD_BARRACKS,
    ACTION_SELECT_BARRACKS,
    ACTION_BUILD_MARINE,
    ACTION_SELECT_ARMY,
    ACTION_BUILD_REFINARY,
]
```

- 게임 중 선택 가능한 ACTION 정의 (총 8개) + ATTACK_X_Y(16개) = 24개

```
['donothing', 'selectscv', 'buildsupplydepot', 'buildbarracks', 'selectbarracks', 'buildmarine', 'selectarmy', 'buildrefinary',
'attack_7_7', 'attack_7_23', 'attack_7_39', 'attack_7_55', 'attack_23_7', 'attack_23_23', 'attack_23_39', 'attack_23_55', 'attack_3
9_7', 'attack_39_23', 'attack_39_39', 'attack_39_55', 'attack_55_7', 'attack_55_23', 'attack_55_39', 'attack_55_55']
```

Len : 24

State (Action에 의해서 변하는 상태)

```
current_state = np.zeros(21)
current_state[0] = overload_count
current_state[1] = zergling_count
current_state[2] = supply_limit
current_state[3] = army_supply
```

4 + 16 (Attack) = 20 개

- 하나의 상태는 총 4개의 기본 상태와 16개의 적의 위치로 이루어짐.
 - 여기서 적의 위치는 변하는가?
 - 변하지 않는 정보를 상태로 관리해야 하나?
- 그리고 전체 상태를 21개로 정의했는데,
 - 왜 실제 상태는 20개로 구성했는가?
 - 나머지 1개의 값은 의미가 없는가?

```
'[ 0. 0. 15. 12. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]'
```

Week 4

Q-Table 생성

2. Adding 1st Step of Hierarchy Actions

< Hierarchy Actions >

Action Space를 줄여서 Q-table의 크기를 줄여 학습을 빠르게 하기 위함.

- * Do nothing – do nothing for 3 steps
- * Build supply depot – select SCV,
- * Build barracks – select SCV,
- * Build marine – select all barracks,
- * Attack (x, y) – select army,

STEP 1

STEP 2

STEP 3

2번째 action의 결과로 Q-Table을 업
데이트함

첫째, Ignoring Learning When State Does Not Change : 'Maximization Bias in RL' Problem 해결 방법

학습의 초기과정에서 같은 State에 자주 도착하는 경우 덜 가치있는 Action을 가장 가치있는 Action으로 밀어 붙이는 경향이 생깁니다. 이 것이 반복되면 시간이 지남에 따라 완전히 다른 State 대신 동일한 State에 더 자주 도착하면 모든 보상이 0에 가까워지는 문제가 생깁니다.

이를 해결하기 위해 아래와 같이 간단한 방법을 추가함.

```
def learn(self, s, a, r, s_):
    if s == s_:
        return
```

둘째, Preventing Invalid Actions

< Invalid Actions 의 예>

- supply depot 제한 갯수 2에 도달했거나 supply depot를 건설할 SCV가 없는 경우 에이전트가 supply depot를 건설 할 수 없도록 한다.
- supply depot가 없거나 barrack 제한 갯수 2에 도달했거나 barrack 제한 갯수를 지을 SCV가 없다면 에이전트가 barrack를 지을 수 없도록 한다.
- barrack가 없거나 supply limit에 도달한 경우 marine을 훈련시키지 않는다.

학습과정에서 에이전트가 Invalid Actions을 시도하는 것이 반복적으로 관찰됩니다. 사용가능한 Action의 수가 Available Actions 절반인 경우가 많기 때문에 에이전트는 불필요한 Action에서 학습하는 데 시간을 많이 보내게 됩니다.

이러한 Invalid Actions을 필터링하여 에이전트가 State 변경으로 이어지는 Action을 시도하는 데 집중하도록하여 Exploration 을 줄이고 학습 시간을 개선 할 수 있습니다.

이를 해결하기 위해 아래와 같이 self.disallowed_actions 와 excluded_action 을 활용하는 방법을 추가함.

셋째, Adding Our Unit Locations to the State

"아군이 어디에 있는지 모른다면 어떤 위치가 공격하기에 가장 좋은지 알 수 있지 않을까요?"

Week 5

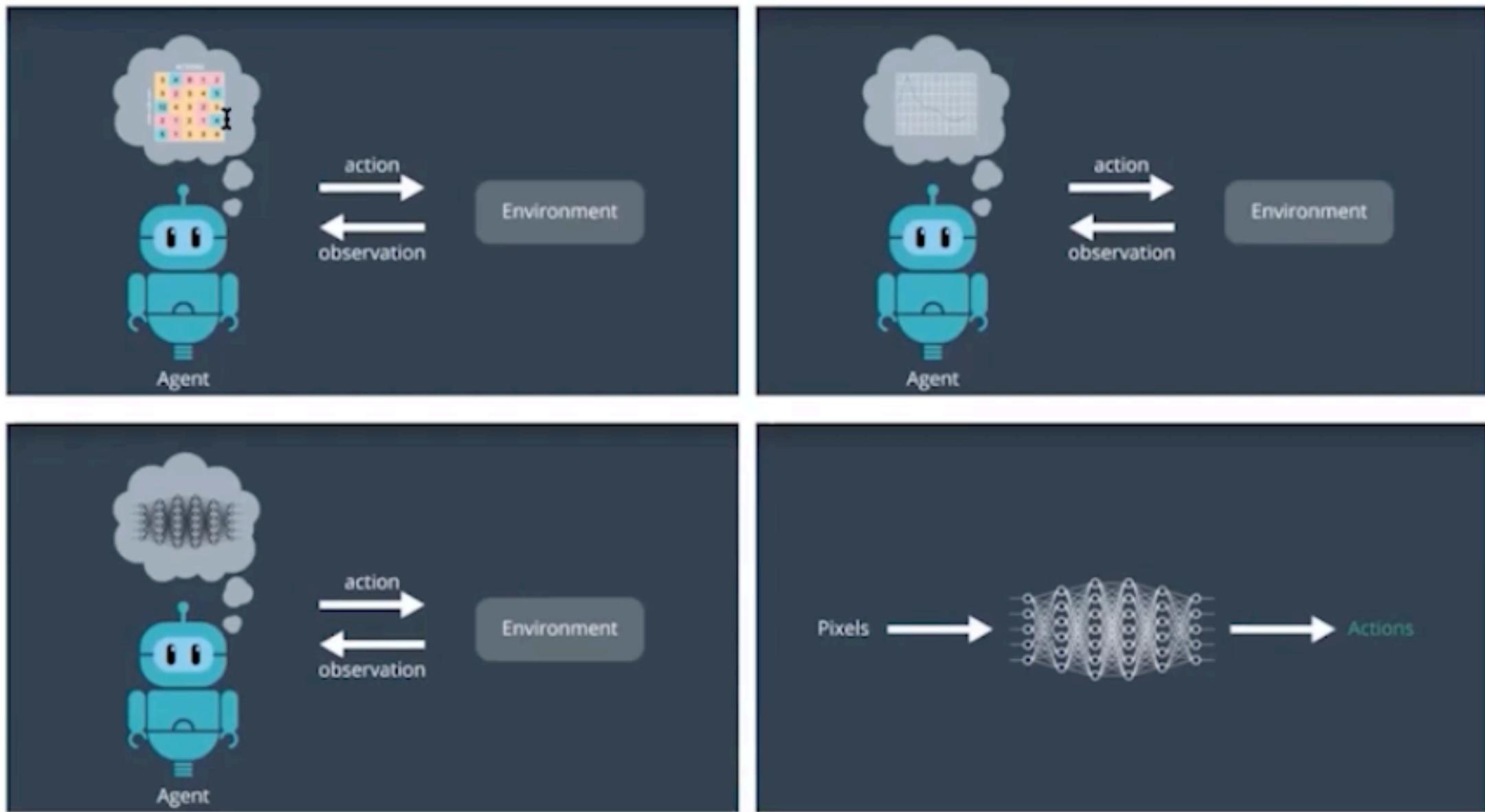
Model-based RL vs. Model-free RL

	Model-based RL	Model-free RL
특징	환경모델이 있음	환경모델이 없음
장점	<ul style="list-style-type: none">high 'Sample Efficiency'high 'Transferability'	<ul style="list-style-type: none">Useful under No Env. ModelUseful under Complex Task
단점	<ul style="list-style-type: none">high 'Computing Cost'high 'Model Error'	<ul style="list-style-type: none">Huge Training DataHard under Multi Task/Same Env.
	DP, Dyna-Q, Trajectory Sampling, RTDP, MBA, NVE, MBPO, GPS, iLQR, ...	SARSA, Q-learning, DQN, REINFORCE, PG, AC, PPO, DDPG, ...

Value-based RL vs. Policy-based RL vs. Actor-Critic RL

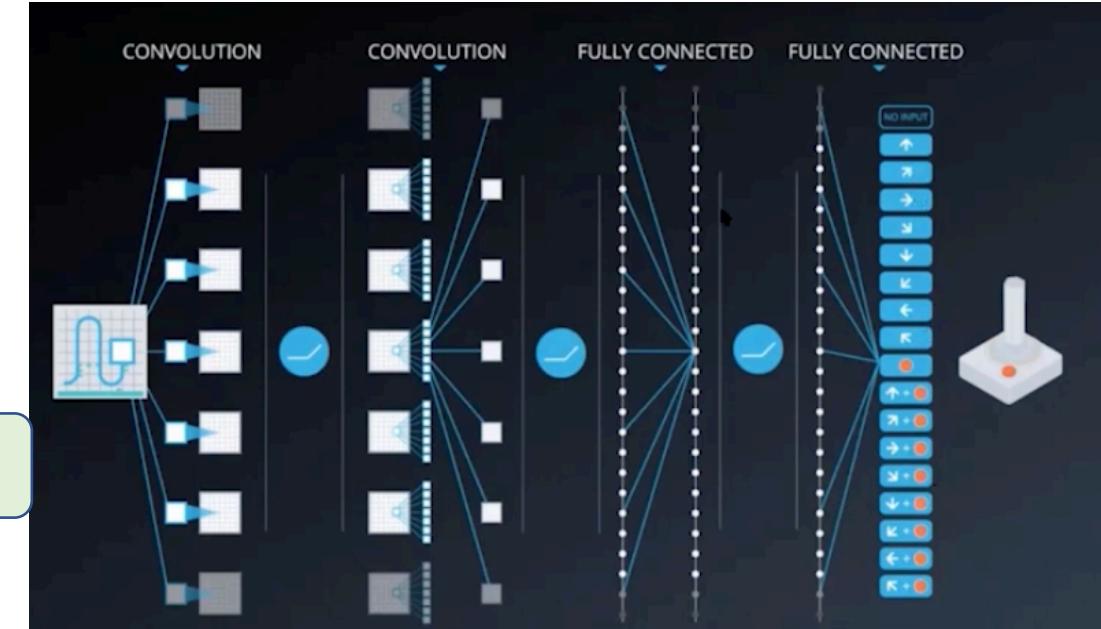
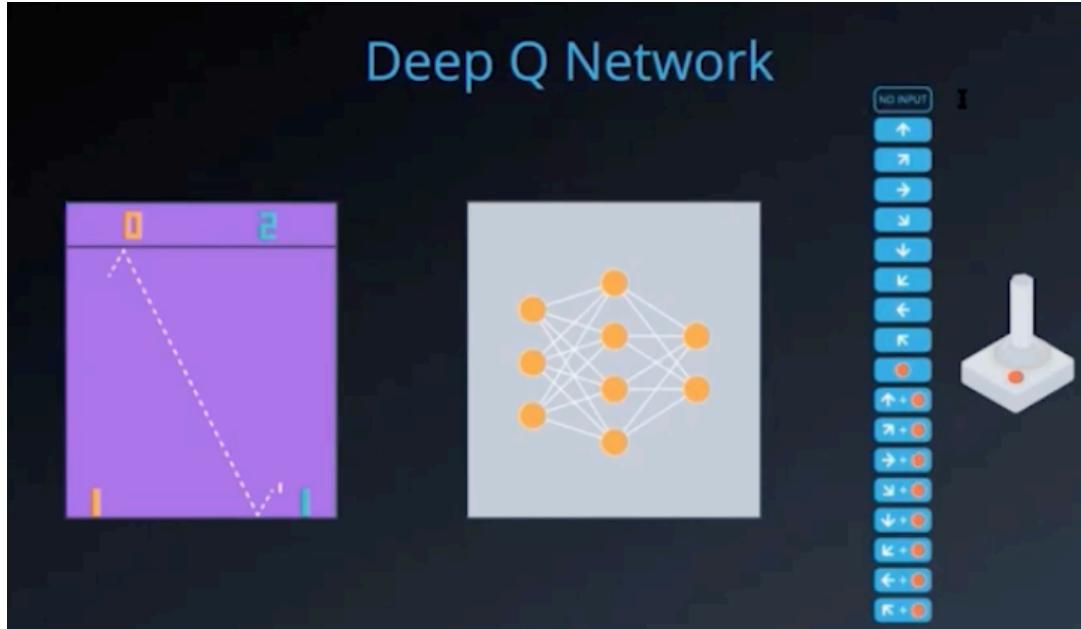
	Value-based RL	Policy-based RL	Actor-Critic RL
특징	Value 함수를 근사하여 Optimal Policy를 찾음.	Reward 함수를 직접 근사하여 Optimal Policy를 찾음.	Value-based, Policy-based 두 가지의 장점을 모두 취함.
장점	<ul style="list-style-type: none">Low Variance	<ul style="list-style-type: none">Low Bias	
단점	<ul style="list-style-type: none">High Bias	<ul style="list-style-type: none">High Variance	
	DQN, DDQN, PER, Dueling DQN, Rainbow, R2D2, ...	Hill Climbing, REINFORCE, PG, TRPO, PPO, ...	AC, A3C, A2C, GAE, DDPG, SAC ...

From RL to Deep RL

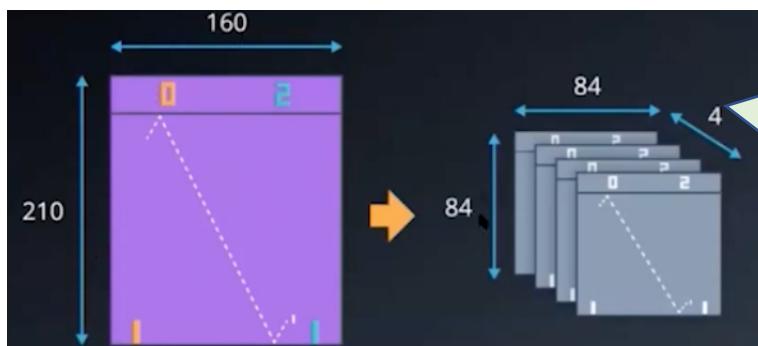


Deep Q Network

모든 상태를 계산하지 않아도, 이 공식을 이용하여 전체 보상 예측 가능



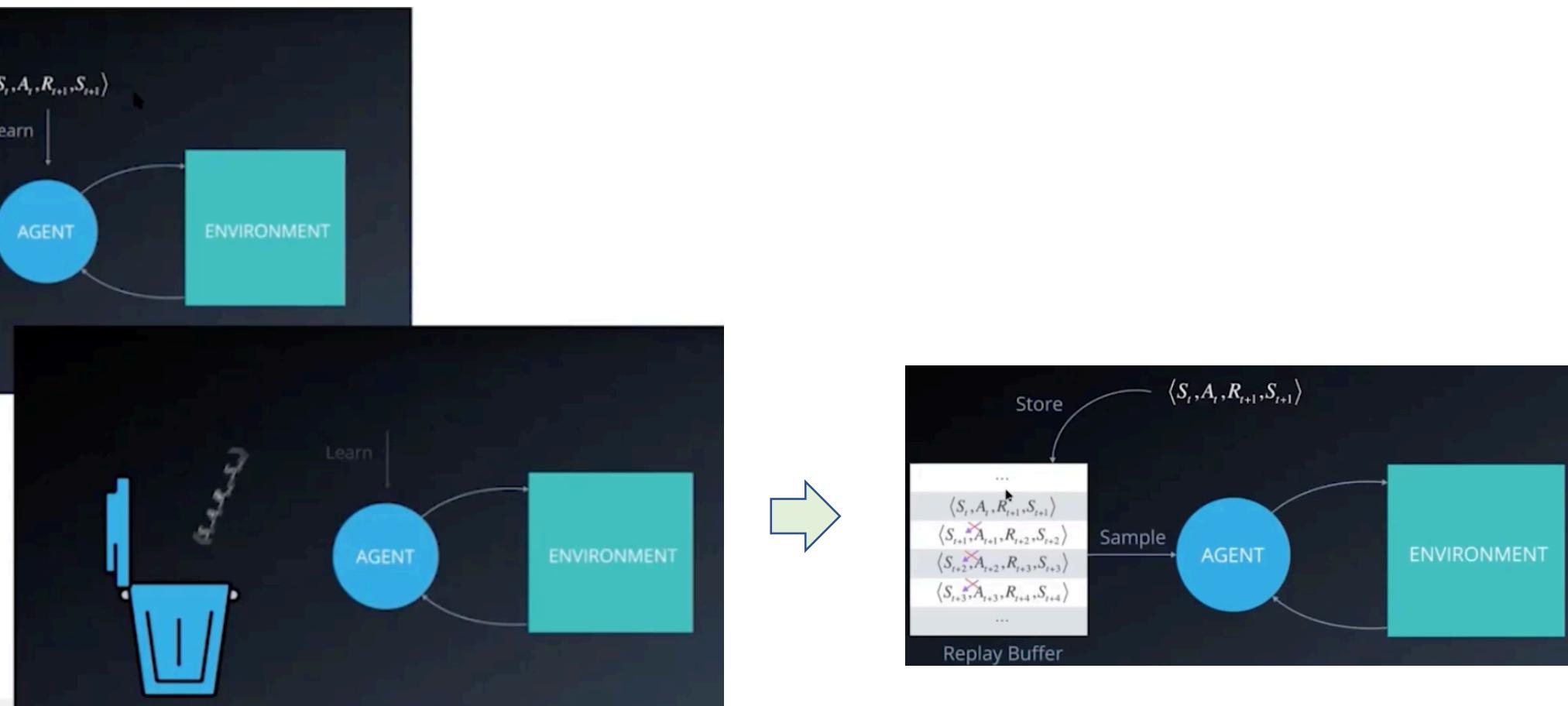
Input Data(상태, state)를
Neural Net의 입력 형태로 변환



현재 상태와 이전 3번의 상태를
하나로 합쳐서 하나의 상태로 결합
왜?
이전 상태와 결합된 현재 상태가
미래의 상태를 더 잘 예측할 수 있음

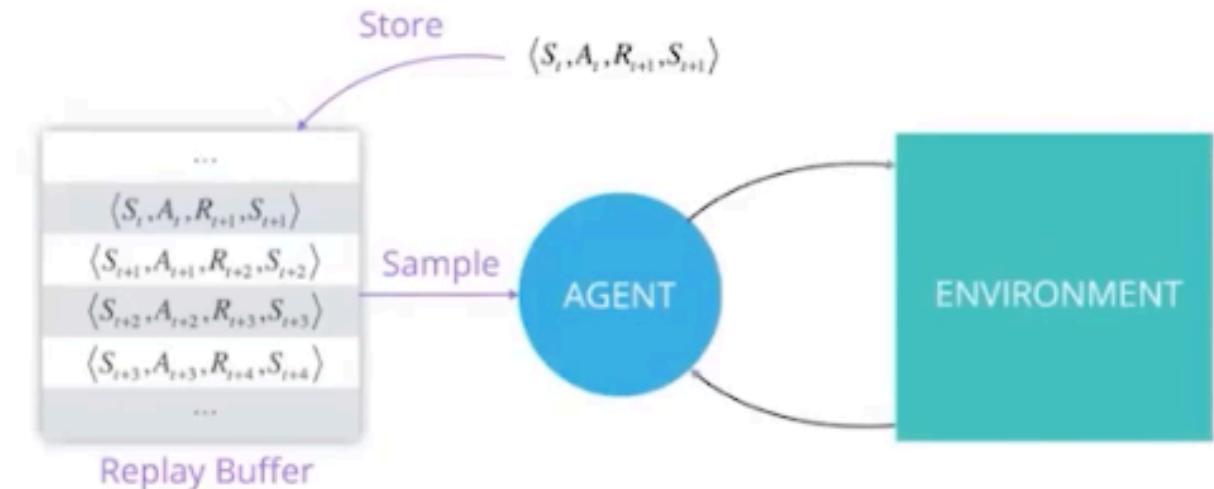
Experience Replay

이전에 경험했던 상태, 액션들을 기억(저장)했다가
나중에 (학습 데이터로) 활용하여 학습하자



Prioritized Experience Replay

기억을 저장할 때 희박한 액션(TD Error가 높을 수록)에 대해서 우선순위를 높여서 저장



Prioritized Experience Replay

TD Error

$$\delta_t = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{q}(S_t, A_t, w)$$

Priority

$$p_t = |\delta_t|$$

Sampling Probability

$$P(i) = \frac{p_i^a}{\sum_k p_k^a}$$

Modified Update Rule

$$\Delta w = \alpha \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^b \delta_i \nabla_w \hat{q}(S_t, A_t, w)$$

importance-sampling weight

Replay Buffer

Fixed Q-Targets

현재 상태를 학습하는 모델(Current)의 가중치와
현재까지 학습된 모델(Target)의 모델과 가중치를 분리하여 Target을 고정

Q-Learning Update

$$\Delta w = \alpha \left(\frac{R + \gamma \max_a \hat{q}(S', a, w)}{\text{TD target}} - \frac{\hat{q}(S, A, w)}{\text{current value}} \right) \nabla_w \hat{q}(S, A, w)$$

TD error

Fixed Target

$$\Delta w = \alpha \left(R + \gamma \max_a \hat{q}(S', a, w^-) - \hat{q}(S, A, w) \right) \nabla_w \hat{q}(S, A, w)$$

fixed

현재 모델(target)의 가중치(weight)

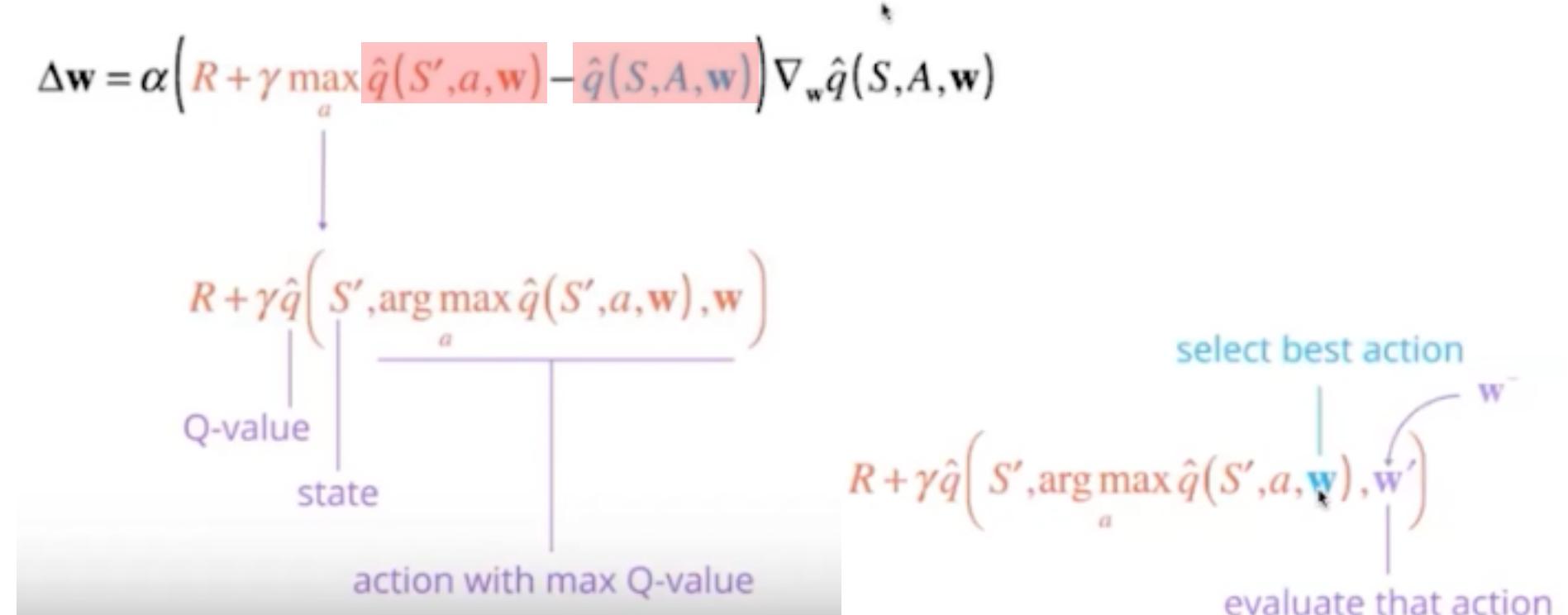
현재 상태로 학습한 모델의
가중치(weight)

Double DQN

동일한 NW를 사용하지 않고, NW 자체를 분리하여 학습을 함.
(Fixed Q-Target은 weight는 분리하지만, NW는 동일함)

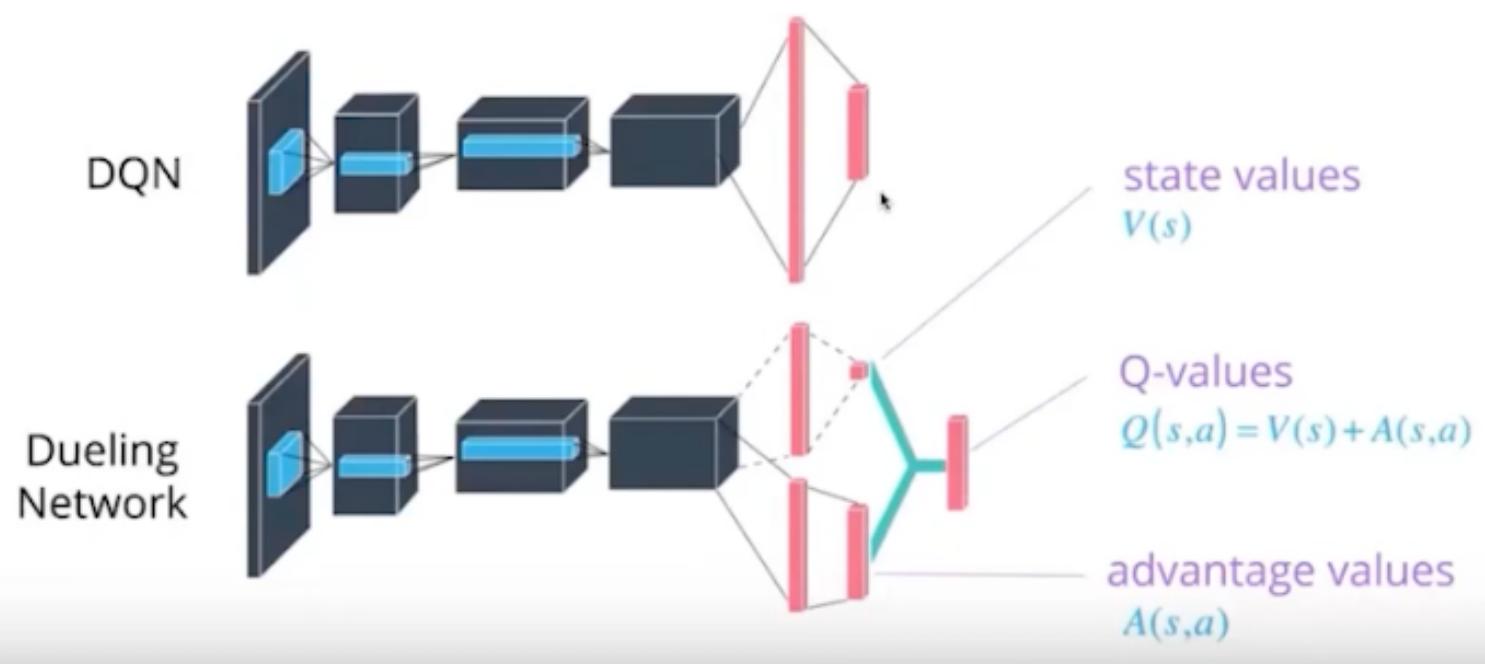
Overestimation of Q-values

$$\Delta \mathbf{w} = \alpha \left(R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}(S, A, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$



Dueling DQN

Q-Value에서 State를 제거한 advanced value를 별도로 추출



- State-Value
 - 현재 상태에서 가능한 모든 action을 고려한 평균값
 - 즉, 모든 액션을 다 고려한 Q-Value로 간주
- Q-Value
 - 현재 상태에서 모든 action의 평균 값(State-Value)과
 - 특정 action에서 계산된 값($A(s,a)$)을 더한 값
 - 즉, 특정 상태에서 action을 수행 했을 때 값
- Advantage values
 - Q-Value에서 특정 상태의 평균값을 뺀 값
 - 즉, 특정 상태에서 action이 평균적인 action에서 얼마나 벗어나 있는지 확인 가능한 값
 - 그 action이 얼마나 발생하기 어려운지? (클수록 발생 가능성이 낮음)

```
flags.DEFINE_bool("disable_fog", True, "Whether to disable Fog of War.")
```

```
flags.DEFINE_enum("agent2_race", "terran", sc2_env.Race._member_names_, # pylint: disable=protected-access  
                  "Agent 2's race.")
```

```
self.s_dim = 21  
self.a_dim = 6  
self.lr = 1e-4  
self.gamma = 1.0 # discount future reward (왜 1로 했을까? 미래의 보상을 그대로 반영하겠다는 의미?)  
self.epsilon = 1.0
```

```
if self.episode_count >= 150:  
    self.dqn.epsilon *= 0.999
```

Episode를 반복할 수록 exploration을 줄인다. (새로운 시도를 하지 않음)

학습 중 이상 상태(outlier)를 완화하기 위한 Huber Loss 사용

`nn.SmoothL1Loss()` = Huber loss 란?

Mean-squared Error (MSE) Loss 는 데이터의 outlier에 매우 취약하다. 어떤 이유로 타겟하는 레이블 y (이 경우는 q-learning target)이 noisy 할때를 가정하면, 잘못된 y 값을 맞추기 위해 파라미터들이 너무 sensitive 하게 움직이게 된다.

이런 현상은 q-learning 의 학습초기에 매우 빈번해 나타난다. 이러한 문제를 조금이라도 완화하기 위해서 outlier에 덜 민감한 Huber loss 함수를 사용한다.

SmoothL1Loss (aka Huber loss)

$$\text{loss}(x, y) = \frac{1}{n} \sum_i z_i$$

Loss가 일정 범위를 넘지 않으면,
L2 Loss(잘못된 예측에 대한 loss를 높여줌)

$|x_i - y_i| < 1$ 일때,

$$z_i = 0.5(x_i - y_i)^2$$

$|x_i - y_i| \geq 1$ 일때,

$$z_i = |x_i - y_i| - 0.5$$

ref : <https://pytorch.org/docs/master/generated/torch.nn.SmoothL1Loss.html>

Loss가 너무 크면,
Outlier(이상한 데이터)로 판단하고,
과도한 Loss가 발생하지 않도록
L1 Loss를 사용