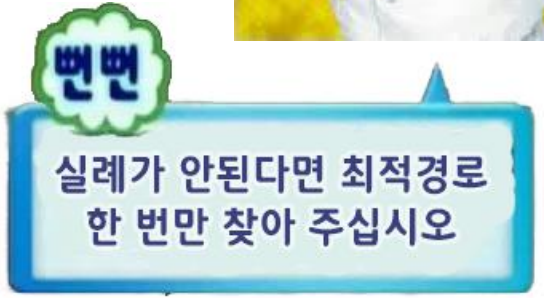
애자일소다 강화학습 1조

오강자 길찾기 프로젝트

유상민(퍼실), 신휘정(팀장), 장진구, 박흥선, 한태양, 노현호

목차

- **프로젝트 주제**
  - 문제 설명
  - 수행 일정
- **아이디어**
  - PQ
  - MQQ
  - DQN
- **Further Study**
- **팀 소개 및 R&R**
- **References**
- **Appendices**



애자일소다 과제 소개

1. 과제의 목적

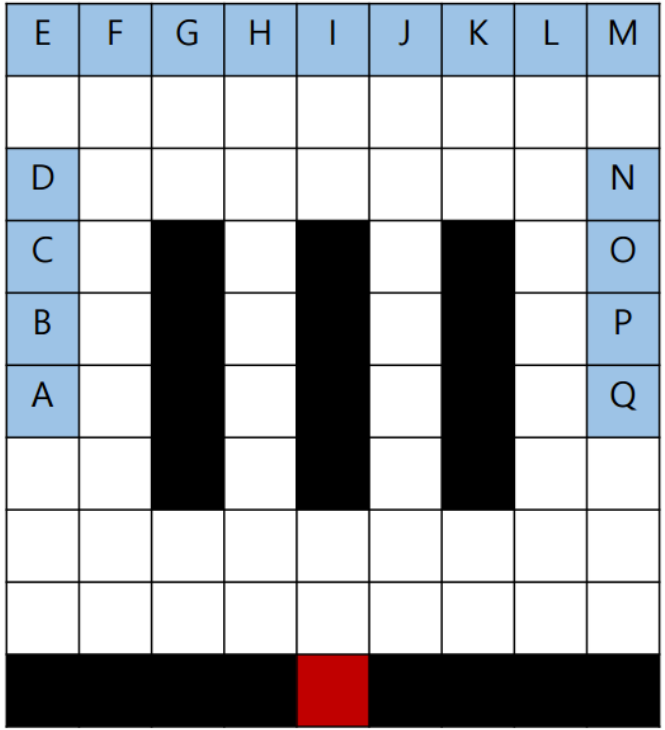
- 참여자들 대상 강화학습에 대한 핵심 개념의 이해
- 미니 프로젝트 수행을 통한 강화학습 실무 경험 제공
- 수강자들의 채용 경쟁력 제고

2. 애자일소다의 바램

- 국내 AI 시장 강화학습 분석가들의 확대
- 프로젝트를 통한 인재 채용

프로젝트 주제 : 강화학습을 이용한 최단 루트 찾기

과제 목표 : (가제) 강화학습을 활용해 최단경로로 아이템 찾아 목적지로 돌아오는 Agent 생성



item의 위치

Start Point & End Point

장애물

과제 설명

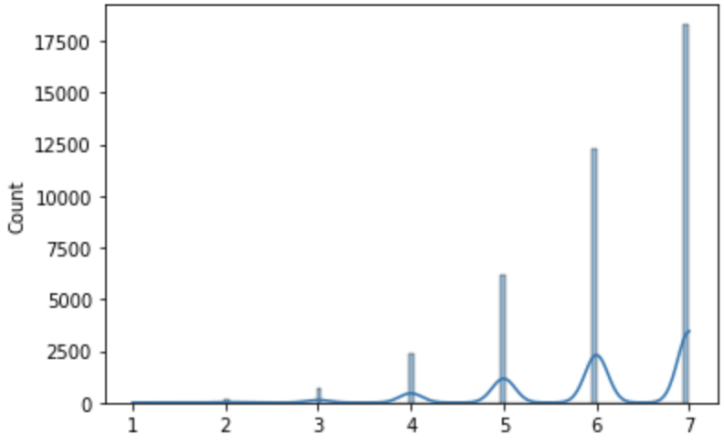
예를 들어, 물류 시스템에서 배달을 위한 제품을 Pick up 하는 효율적인 경로 제공
Start point에서 출발하여 주문된 item list 내 모든 items을 챙겨 End point에 돌아오는 것을 목표로 함
(들어오는 item list의 size는 8로 제한함)

수행 일정

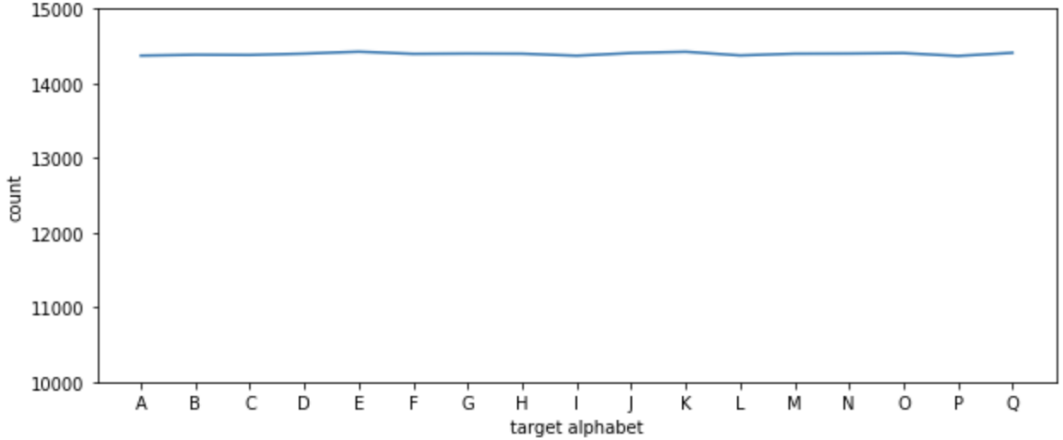
	4/25~5/1	5/2~5/8	5/9~5/15	5/16~5/22	5/23~5/29	5/30~6/5	6/6~6/9
	1주차	2주차	3주차	4주차	5주차	6주차	7주차
강화학습 스터디							
베이스라인 코드 분석 및 개선			◆ 기업 Baseline 코드 수령				
모델 및 참고문헌 연구							
PQ모델 개발 및 테스팅				PQ 결과 나옴 ◆			
MQQ모델 개발 및 테스팅					MQQ 결과 나옴 ◆		
DQN모델 개발 및 테스팅						DQN 결과 나옴 ◆	
프로젝트 마무리, 보고서 작성							

학습 데이터 분석

Train data의 길이 분포

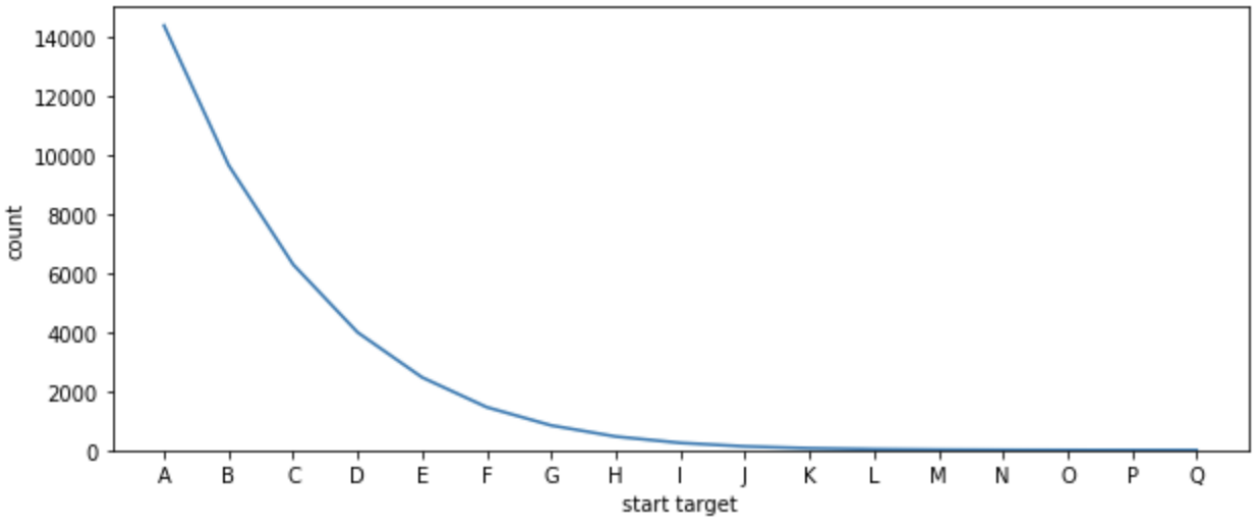


Target 아이템의 분포



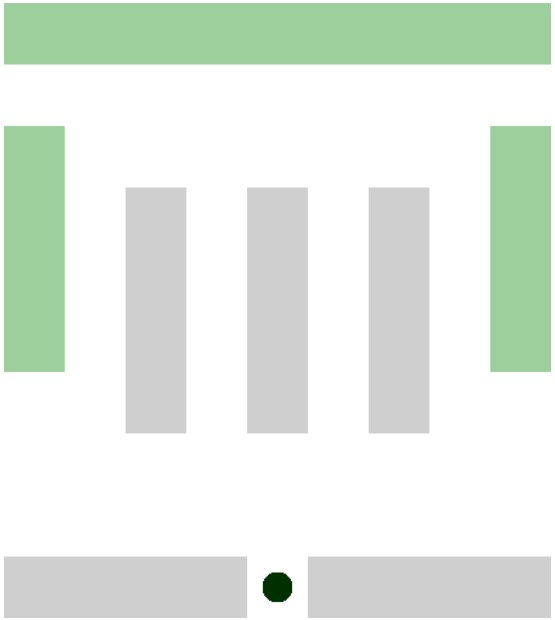
- 모든 아이템에 대하여 골고루 가져올 수 있도록 구성되어있다.

학습 데이터 분석

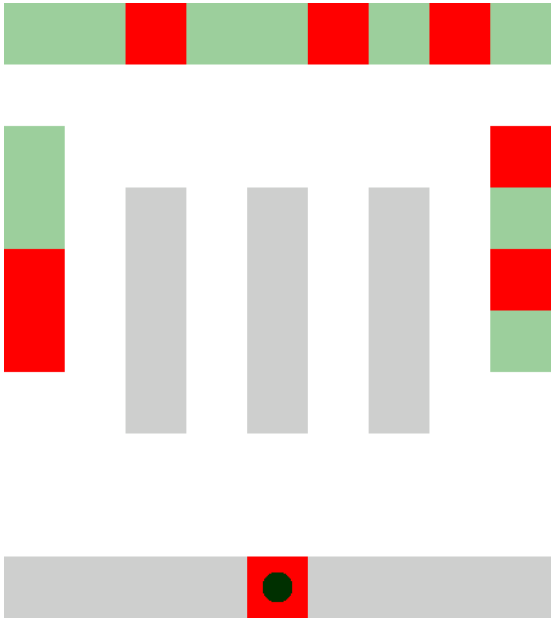


- A로 시작하는 트레인 데이터가 많으므로, K-Q로 시작하는 데이터의 수는 작다

베이스라인 코드 분석 및 개선



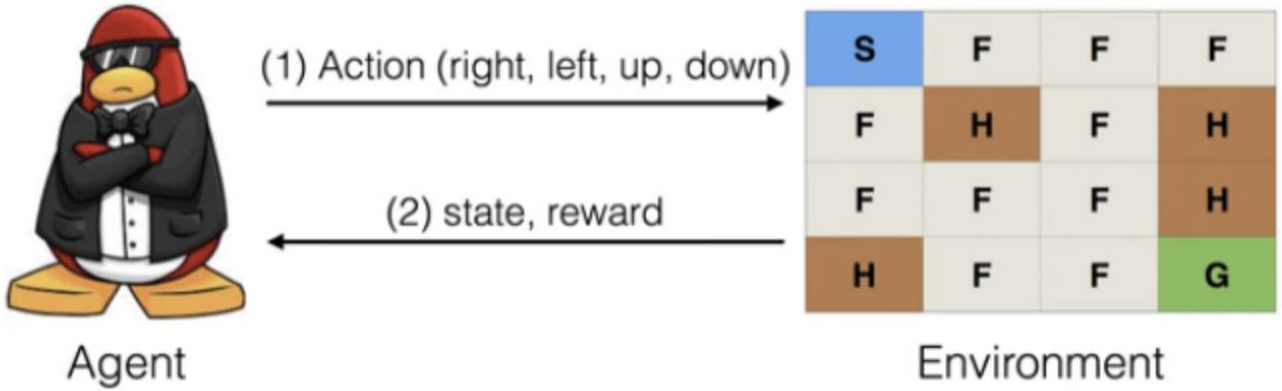
수정



타겟 아이템 위치를 알 수 없음

타겟 아이템 위치를 표시함

강화학습이란?



에이전트가 한 행동이 좋았다면 좋은 보상(“+”)

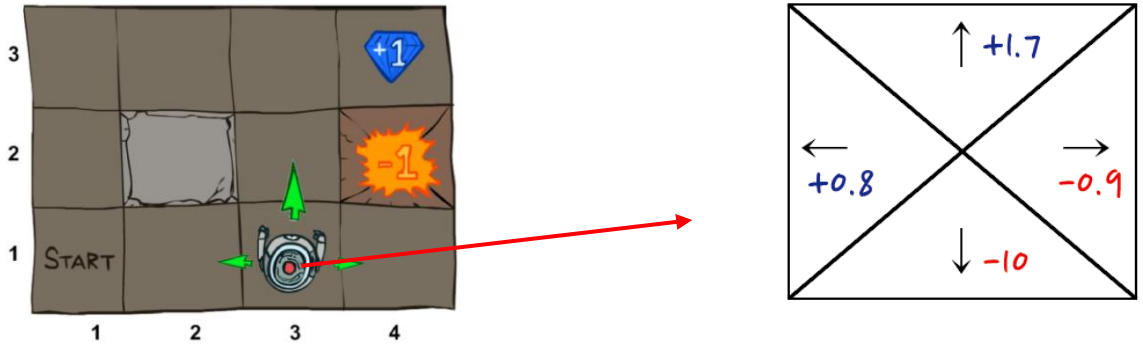
나쁜 행동이었다면 나쁜 보상(“-”)를 주면서

에이전트의 행동을 강화시키는 방법

Q Learning 이란?

Q러닝은 각 State마다 선택할 수 있는 Action의 가치를 측정합니다.

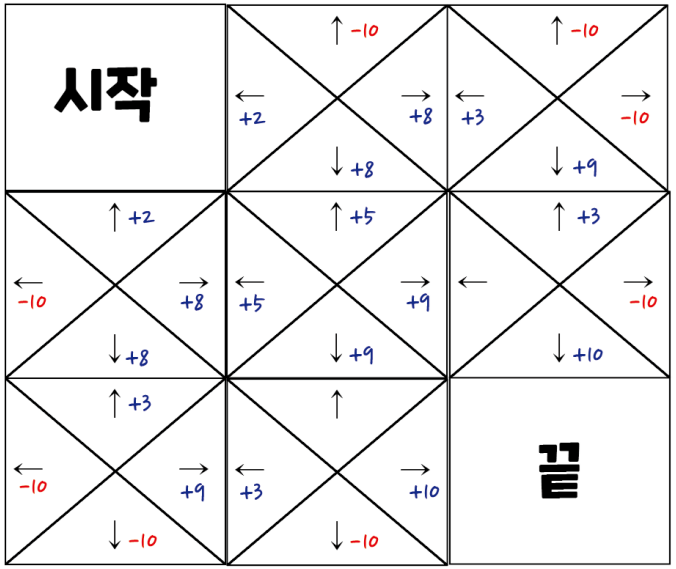
Q 테이블이란, 모든 State에 대한 Action 가치를 저장한 테이블입니다.



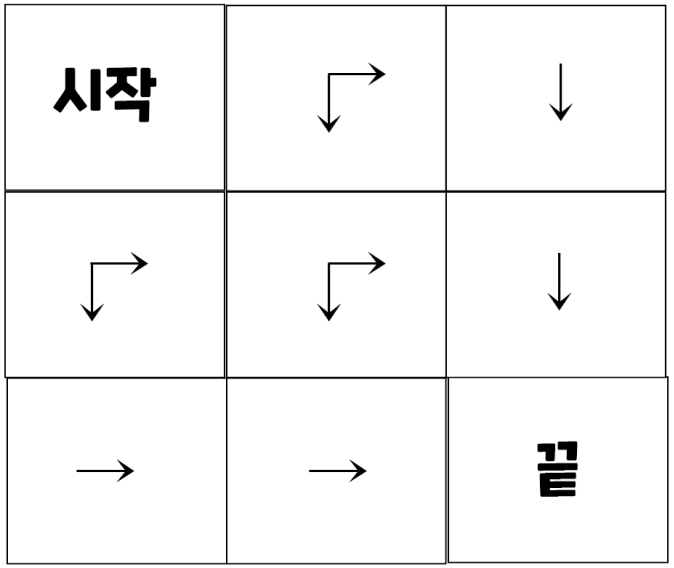
시행착오를 거치다보면 Action이 가지는 가치를 정확히 판단 가능하고

로봇은 함정을 피해 목적지에 도달할 수 있음

Q Learning 이란?



argmax

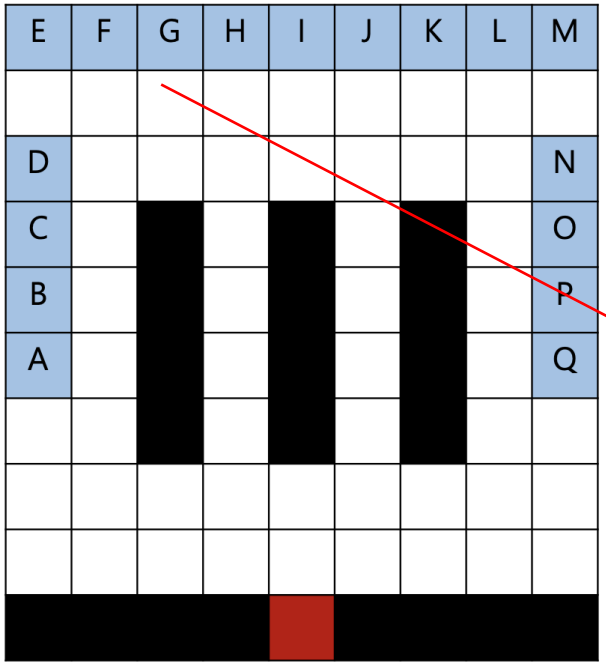


수렴된 Q-Table

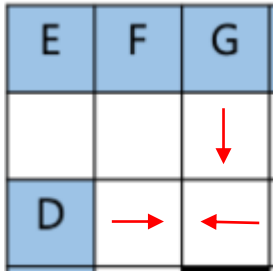
수렴된 정책 테이블

Q Learning 적용 아이디어

일반적인 Q-Learning은 출발지에서 하나의 목적지를 가지고 학습을 수행하지만, 우리의 주제는 연속적으로 여러 개의 Target을 가져와야 한다.

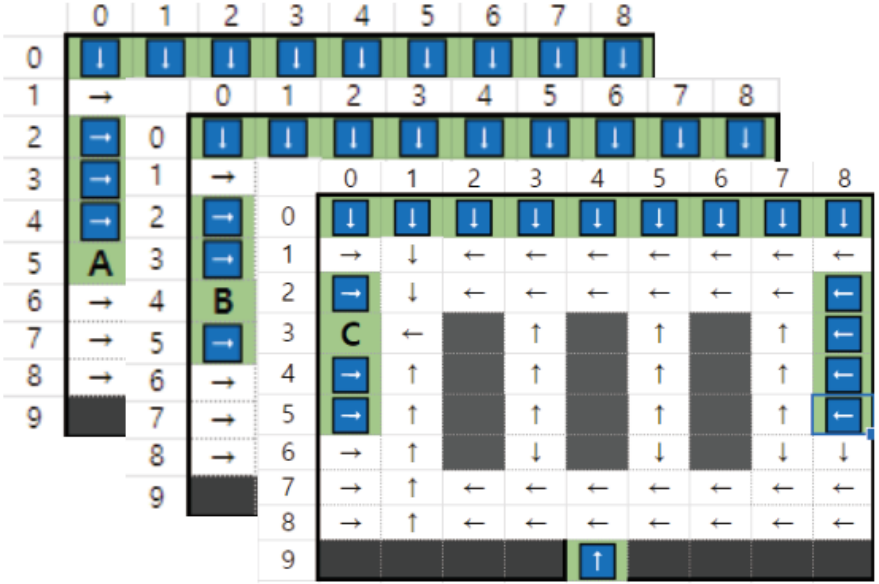


주어진 Target이 D와 G라고 한다면



D로 수렴하는 Q_value와 G로 수렴하는 Q_value가 충돌해 방황하게 된다.

Q Learning 적용 아이디어



Item의 위치는 고정되어 있다.

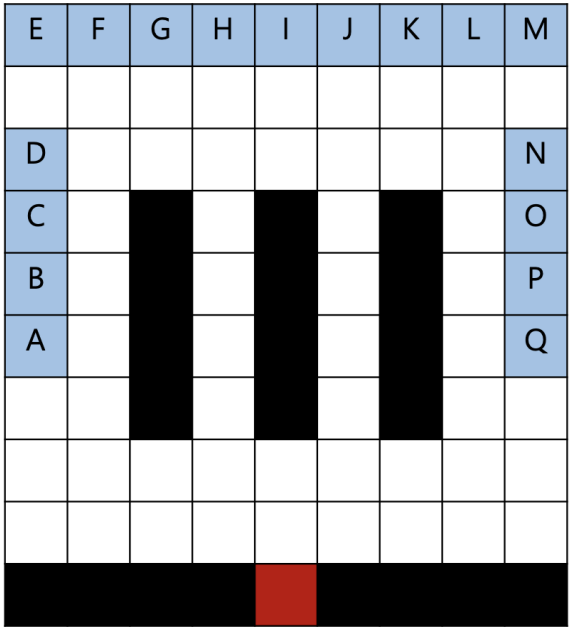
Q테이블은 Target Item지점으로 향하는 최적 경로를 가지고 있다.

각 Item에 대한 Q테이블을 **모두** 가지고 있으면 (Ready Made)

이 문제를 해결할 수 있지 않을까?

Pre Q Learning

```
Q_tables = np.zeros((18, 10, 9, 4))
```

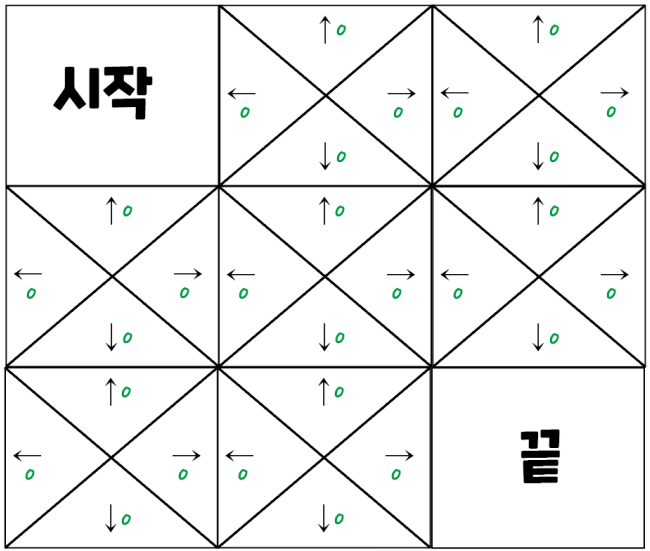


각 아이템 별 Q_table의 집합인 Q_tables를 만들어 각각의 타겟에 수렴하는 Q-Table을 미리 학습시켜 준비해놓는다.

Prepare Q Learning을 줄여서

Pre Q Learning 이라 이름 붙였다.

ε-greedy 란?



모든 행동들의 가치를 알기 위해서는,

에이전트가 직접 그 행동을 해보고 보상을

받아봐야 한다.

때문에 ε-Greedy라는 정책을 통해

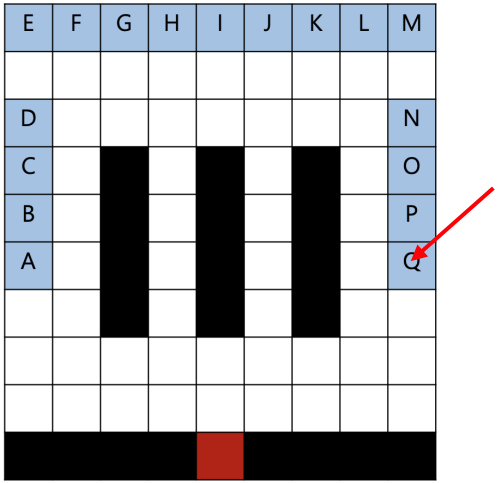
일정 확률로 랜덤한 이동으로 탐색을 하고

일정 확률로는 greedy한 행동을 한다.

처음에는 100%로 랜덤하게 움직이면서 시행착오를 계속하다가

이후 점점 적은 확률로 탐색을 하며 최적 루트를 찾게 된다.

Pre Q Learning 구현 방법?



ε-greedy 는 확률에 의존한 탐색을 하므로,

시작위치에서 E로 랜덤선택으로 갈 확률은

$$\frac{1}{4^{13}} \simeq \frac{1}{70,000,000}$$

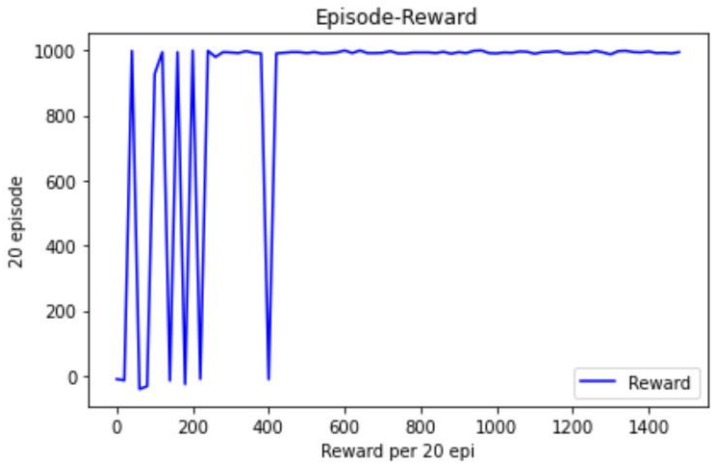
위와 같은 이유로 시작 위치를 **무작위로 하는 탐색 전략**을 사용했다.

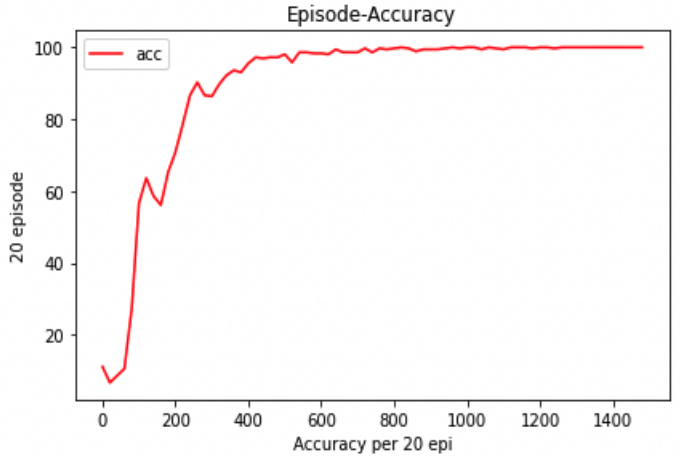
```
self.curloc = [np.random.randint(10), np.random.randint(9)] #시작 위치 초기화 (랜덤 위치)
```

Pre Q Learning

학습 결과

```
에피소드 :9940, 액션 수: 8, 리턴: 993, 성공율 : 100.00 %
에피소드 :9960, 액션 수: 7, 리턴: 994, 성공율 : 100.00 %
에피소드 :9980, 액션 수: 13, 리턴: 988, 성공율 : 100.00 %
time : 1493.94
```



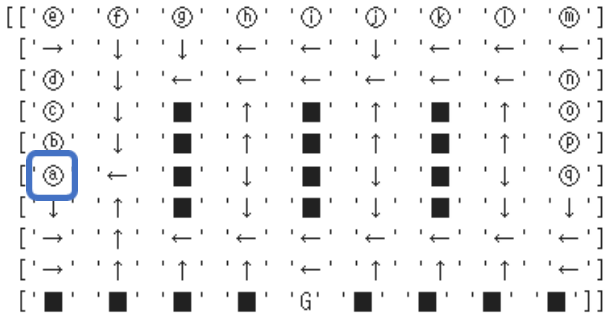


Pre Q Learning

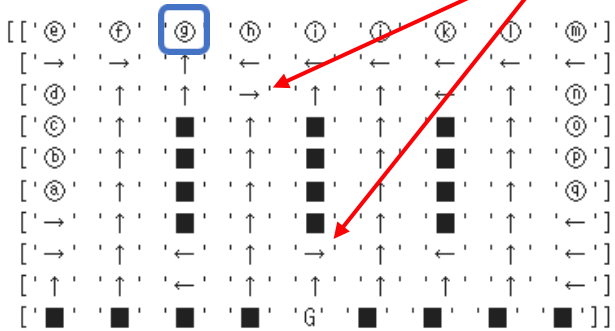
A-Q, 종료위치 T를 End state로 하는 Q-Table을 수렴할 때까지 학습시킨 뒤 합쳐서 종합적인 Q-Tables를 만들어낸다.
완성된 테이블에 argmax를 하면 정책 테이블을 만들 수 있다.

Miss

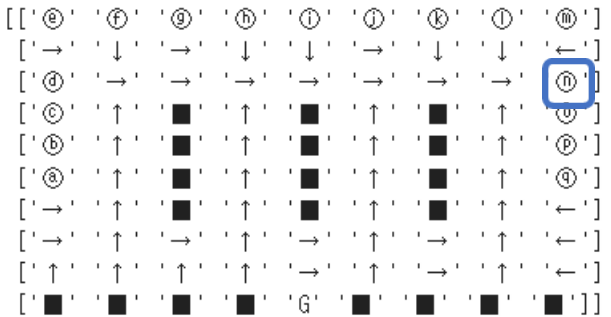
A



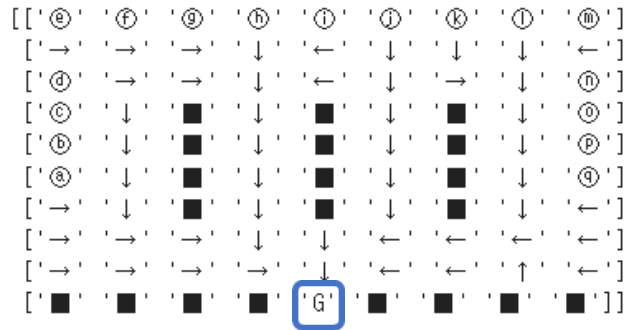
G



N



T



Pre Q Learning

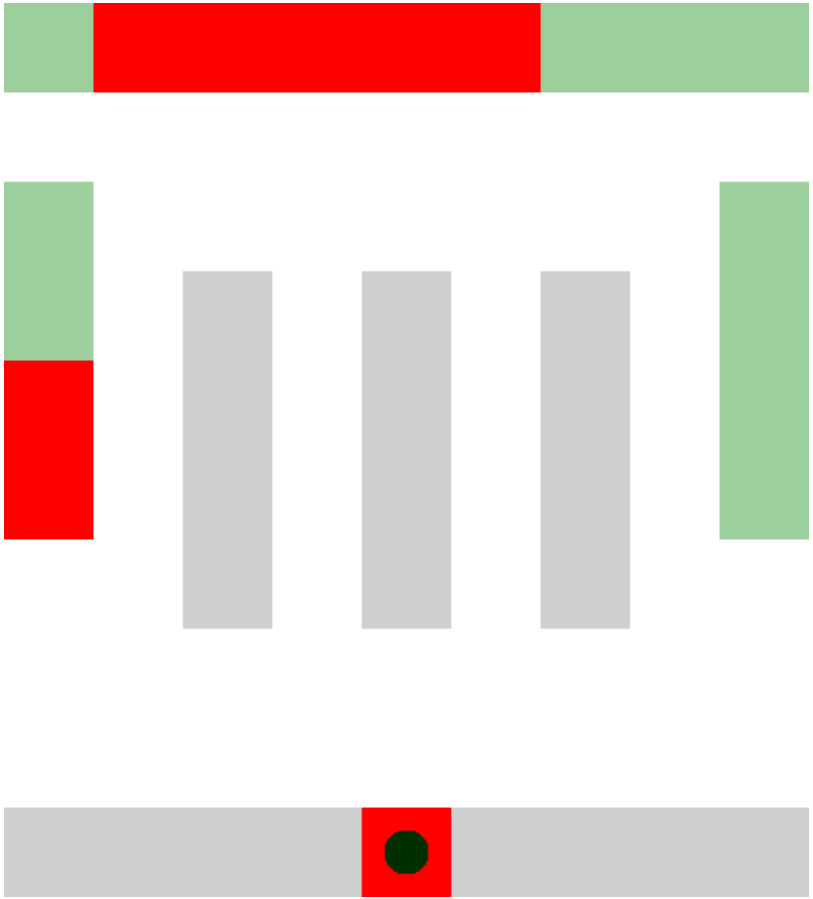
테스트 결과

각 Target item에 따른 수렴된 Q-Table을 보고 greedy로 이동한다.

```
episode :1221, Steps: 42, Return: 7966, Finish Rate : 100.00 %
episode :1222, Steps: 44, Return: 7964, Finish Rate : 100.00 %
episode :1223, Steps: 44, Return: 7964, Finish Rate : 100.00 %
episode :1224, Steps: 42, Return: 7966, Finish Rate : 100.00 %
episode :1225, Steps: 44, Return: 7964, Finish Rate : 100.00 %
평균 Step : 43.4
```

만들어진 Q-Table 따라 이동만 하면 되므로, 1분이내의 적은 시간만이 걸렸다.

Pre Q Learning



Episode 270 결과

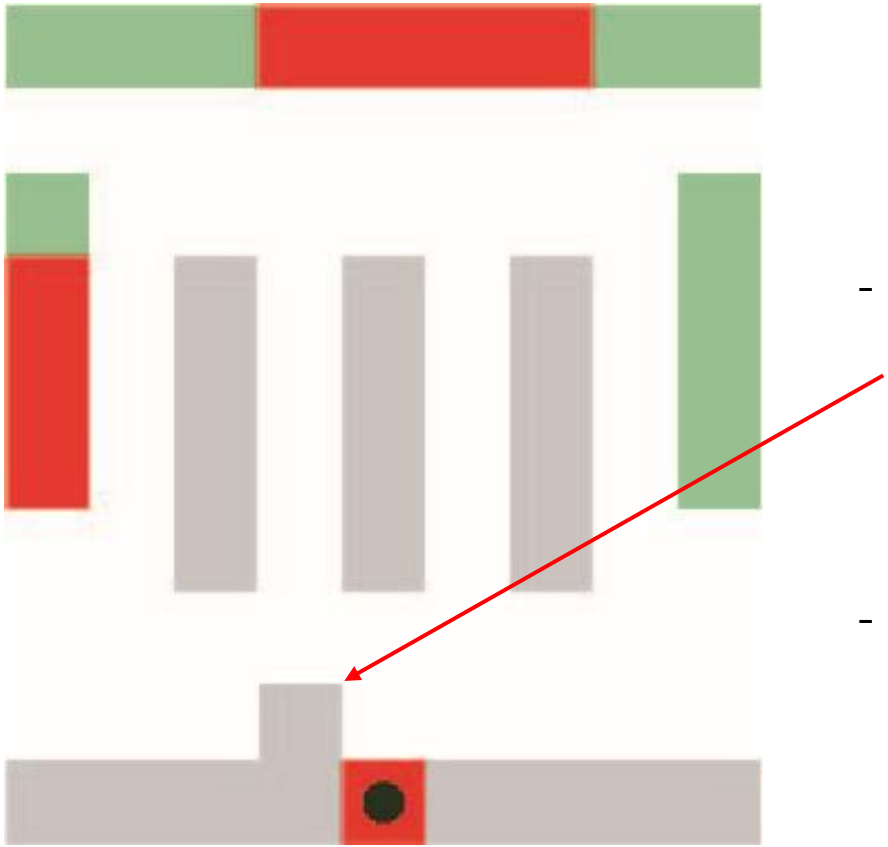
- 목표 : [A, B, F, G, H, I, J]
- 최적 step : 38
- 실제 step : 38
- Item A로 수렴하는 Table을 따라 이동

  → Item B로 수렴하는 Table 따라 이동

  …

  → 시작점으로 수렴하는 Table을 따라 이동

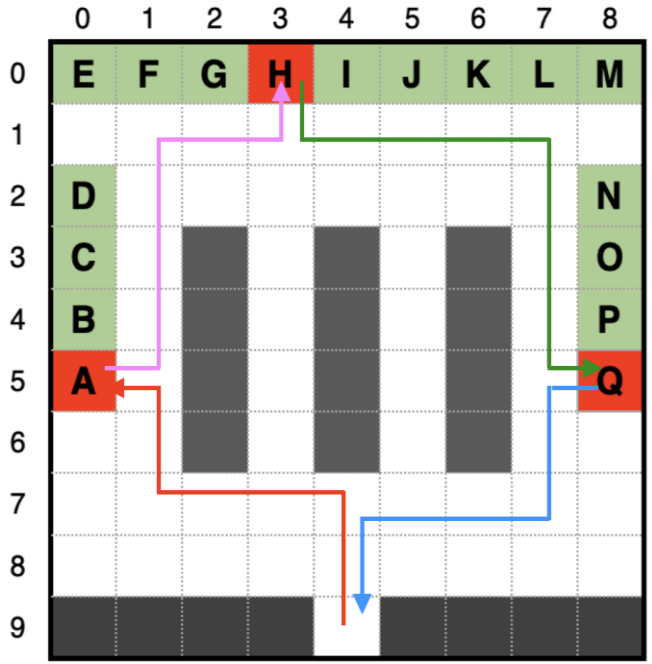
Pre Q Learning의 한계



- Q-Table은 정책이 고정되어 있고, 정해진 Q값만 가지고 이동하게 되므로, 환경이 바뀌면, 모든 타겟에 대한 Q-Table을 다시 학습해야 한다.

- 타겟이 많아질수록 더 많은 Q-Table을 만들어야 한다.

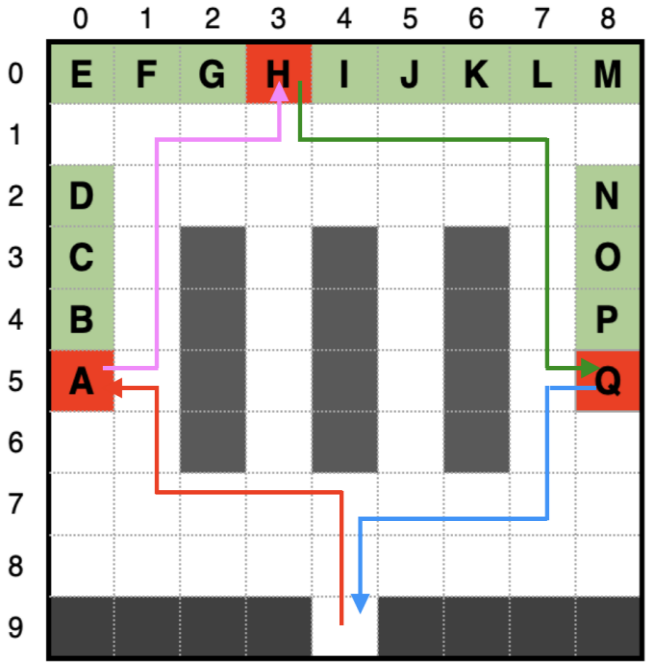
Multi Q-table Q-learning Algorithm (MQQ)



- PQ는 아이템에 대한 Q테이블을 학습시키는 방법

- MQQ는 S → A → H → Q라는 경로를 찾아야 할 때,
  [S → A], [A → H], [H → Q]
  해당 경로에 대한 Q테이블을 업데이트 하는 방식

MQQ 방식



- [S → A → H → Q → S]의 문제를
- [S → A] 경로의 Q-table을 만들어 수렴하면 이동
- [A → H] 경로의 Q-table 업데이트 후 수렴하면 이동
- [H → Q] 경로의 Q-table 업데이트 후 수렴하면 이동
- [Q → S] 경로의 Q-table 업데이트 후 수렴하면 이동

MQQ 결과

목적지 좌표: [[5, 0], [0, 3], [5, 8], [9, 4]]

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

현재 에피소드에서 들러야 할 전체 좌표 리스트

시작지점 [9, 4] → A [5, 0]으로 가는 최적 Q_Table Data

0 (상⬆) 1 (하⬇) 2 (좌⬅) 3 (우 ➡)

```
[['ⓔ' 'ⓕ' 'ⓖ' 'ⓗ' 'ⓘ' 'ⓙ' 'ⓚ' 'ⓛ' 'ⓜ']
 ['→' '↓' '←' '←' '→' '→' '←' '↓' '↑']
 ['ⓓ' '↓' '←' '↑' '→' '←' '→' '→' 'ⓝ']
 ['ⓒ' '↓' '■' '↑' '■' '→' '■' '→' 'ⓞ']
 ['ⓑ' '↓' '■' '↓' '■' '→' '■' '→' 'ⓟ']
 ['ⓐ' '←' '■' '↓' '■' '↑' '■' '↑' 'ⓠ']
 ['→' '↑' '■' '↓' '■' '↓' '■' '↑' '→']
 ['→' '↑' '←' '←' '↓' '↓' '↓' '↑' '→']
 ['↑' '↑' '←' '←' '←' '←' '←' '↓' '↑']
 ['■' '■' '■' '■' 'S' '■' '■' '■' '■']]
```

Q Table 시각화

[[9, 4], [8, 4], [8, 3], [8, 2], [8, 1], [7, 1], [6, 1], [5, 1], [5, 0]]

시작지점 [9, 4] → A [5, 0] 으로 가는 최적의 경로

MQQ 결과

```
[[2. 1. 1. 0. 1. 2. 2. 2. 2.]
 [3. 3. 3. 0. 2. 2. 3. 0. 0.]
 [3. 3. 3. 0. 0. 3. 2. 1. 0.]
 [3. 0. 0. 0. 0. 2. 0. 3. 0.]
 [3. 0. 0. 0. 0. 1. 0. 3. 0.]
 [3. 0. 0. 2. 0. 1. 0. 2. 0.]
 [3. 0. 0. 1. 0. 1. 0. 2. 1.]
 [1. 0. 3. 3. 0. 2. 0. 1. 0.]
 [2. 1. 3. 1. 2. 3. 0. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 0. 0.]]
[['ⓔ' 'ⓕ' 'ⓖ' 'ⓗ' 'ⓘ' 'ⓙ' 'ⓚ' 'ⓛ' 'ⓜ']
 ['→' '→' '→' '↑' '←' '←' '→' '↑' '↑']
 ['ⓓ' '→' '→' '↑' '↑' '→' '←' '↓' 'ⓝ']
 ['ⓒ' '↑' '■' '↑' '■' '←' '■' '→' 'ⓞ']
 ['ⓑ' '↑' '■' '↑' '■' '↓' '■' '→' 'ⓟ']
 ['ⓐ' '↑' '■' '←' '■' '↓' '■' '←' 'ⓠ']
 ['→' '↑' '■' '↓' '■' '↓' '■' '←' '↓']
 ['↓' '↑' '→' '→' '↑' '←' '↑' '↓' '↑']
 ['←' '↓' '→' '↓' '←' '→' '↑' '→' '↑']
 ['■' '■' '■' '■' 'S' '■' '■' '■' '■']]
[[5, 0], [5, 1], [4, 1], [3, 1], [2, 1], [2, 2], [2, 3], [1, 3], [0, 3]]
```

A(5, 0) → H(0, 3)

```
[[0. 2. 0. 1. 1. 1. 1. 0. 0.]
 [0. 3. 3. 3. 3. 1. 1. 1. 2.]
 [0. 2. 3. 3. 3. 3. 3. 1. 2.]
 [1. 2. 0. 2. 0. 0. 0. 1. 2.]
 [2. 3. 0. 3. 0. 0. 0. 1. 2.]
 [0. 2. 0. 2. 0. 2. 0. 3. 0.]
 [3. 3. 0. 0. 0. 2. 0. 0. 2.]
 [2. 0. 0. 2. 0. 3. 3. 0. 0.]
 [3. 1. 2. 2. 2. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]]
[['ⓔ' 'ⓕ' 'ⓖ' 'ⓗ' 'ⓘ' 'ⓙ' 'ⓚ' 'ⓛ' 'ⓜ']
 ['↑' '→' '→' '→' '→' '↓' '↓' '↓' '←']
 ['ⓓ' '←' '→' '→' '→' '→' '→' '↓' 'ⓝ']
 ['ⓒ' '←' '■' '←' '■' '↑' '■' '↓' 'ⓞ']
 ['ⓑ' '→' '■' '→' '■' '↑' '■' '↓' 'ⓟ']
 ['ⓐ' '←' '■' '←' '■' '←' '■' '→' 'ⓠ']
 ['→' '→' '■' '↑' '■' '←' '■' '↑' '←']
 ['←' '↑' '↑' '←' '↑' '→' '→' '↑' '↑']
 ['→' '↓' '←' '←' '←' '↓' '↑' '↑' '↑']
 ['■' '■' '■' '■' 'S' '■' '■' '■' '■']]
[[0, 3], [1, 3], [1, 4], [1, 5], [2, 5], [2, 6], [2, 7], [3, 7], [4, 7], [5, 7], [5, 8]]
```

H(0, 3) → Q(5, 8)

MQQ 결과

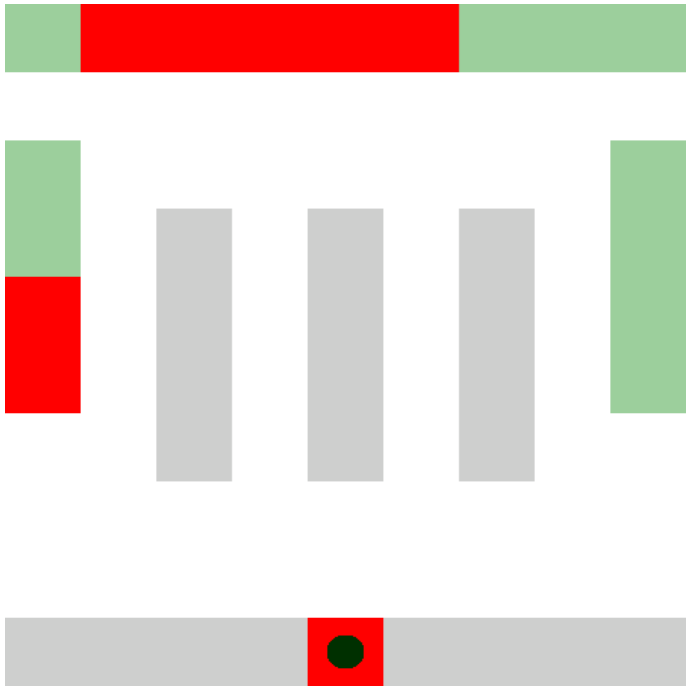
```
[[2. 2. 0. 2. 0. 0. 0. 2. 0.]
 [1. 2. 3. 0. 0. 0. 2. 1. 1.]
 [0. 2. 1. 0. 3. 0. 0. 3. 0.]
 [1. 2. 0. 0. 0. 1. 0. 1. 1.]
 [0. 2. 0. 0. 0. 2. 0. 1. 2.]
 [0. 3. 0. 0. 0. 2. 0. 1. 2.]
 [0. 2. 0. 1. 0. 1. 0. 1. 2.]
 [1. 0. 3. 3. 1. 2. 1. 1. 1.]
 [0. 3. 3. 3. 1. 2. 2. 2. 2.]
 [0. 0. 0. 0. 2. 0. 0. 0. 0.]]
[['ⓔ' 'ⓕ' 'ⓖ' 'ⓗ' 'ⓘ' 'ⓙ' 'ⓚ' 'ⓛ' 'ⓜ']
 ['↓' '←' '→' '↑' '↑' '↑' '←' '↓' '↓']
 ['ⓓ' '←' '↓' '↑' '→' '↑' '↑' '→' 'ⓝ']
 ['ⓒ' '←' '■' '↑' '■' '↓' '■' '↓' 'ⓞ']
 ['ⓑ' '←' '■' '↑' '■' '←' '■' '↓' 'Ⓟ']
 ['ⓐ' '→' '■' '↑' '■' '←' '■' '↓' 'ⓠ']
 ['↑' '←' '■' '↓' '■' '↓' '■' '↓' '←']
 ['↓' '↑' '→' '→' '↓' '←' '↓' '↓' '↓']
 ['↑' '→' '→' '→' '↓' '←' '←' '←' '←']
 ['■' '■' '■' '■' 'S' '■' '■' '■' '■']]
[[5, 8], [5, 7], [6, 7], [7, 7], [8, 7], [8, 6], [8, 5], [8, 4], [9, 4]]
```

FINAL PATH =

```
[[9, 4], [8, 4], [8, 3], [8, 2], [8, 1], [7, 1], [6, 1],
[5, 1], [5, 0],

[5, 0], [5, 1], [4, 1], [3, 1], [2, 1], [2, 2],
[2, 3], [1, 3], [0, 3],

[0, 3], [1, 3], [1, 4], [1, 5], [2, 5], [2, 6], [2, 7],
[3, 7], [4, 7], [5, 7], [5, 8],

[5, 8], [5, 7], [6, 7], [7, 7], [8, 7], [8, 6], [8, 5],
[8, 4], [9, 4]]
```

Q(5, 8) → S(9, 4)

MQQ 최종 결과



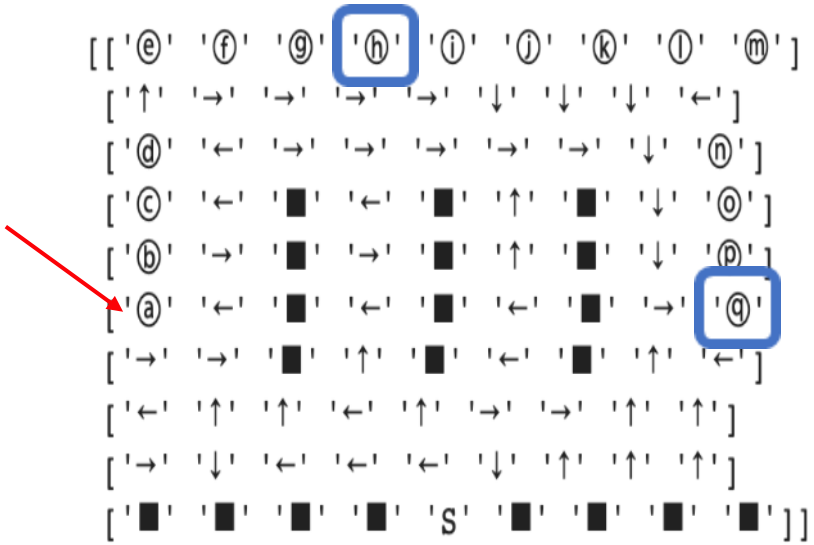
TEST EPI 270

Episode 270 결과

- 목표: [A, B, F, G, H, I, J]
- 최적 step: 38
- 실제 step: 38

- 아이템을 다 찾고 최적의 경로 직선으로 이동
- 전체 에피소드에 대하여…
  - 평균 step수: 43.31342577487765

MQQ에서의 시도



다른 방안으로 Q테이블을 저장해 사용하는 것을 시도

- H → Q 로 가는 Q테이블이 있다면 학습한 Q테이블이 모두 Q로 가는 방향으로 학습하지 않았을까?
- PQ와 다르게 MQQ는 모든 경로의 테이블을 저장해야 되어 PQ의 방법으로 사용하는 것이 효율적

PQ 와 MQQ의 비교

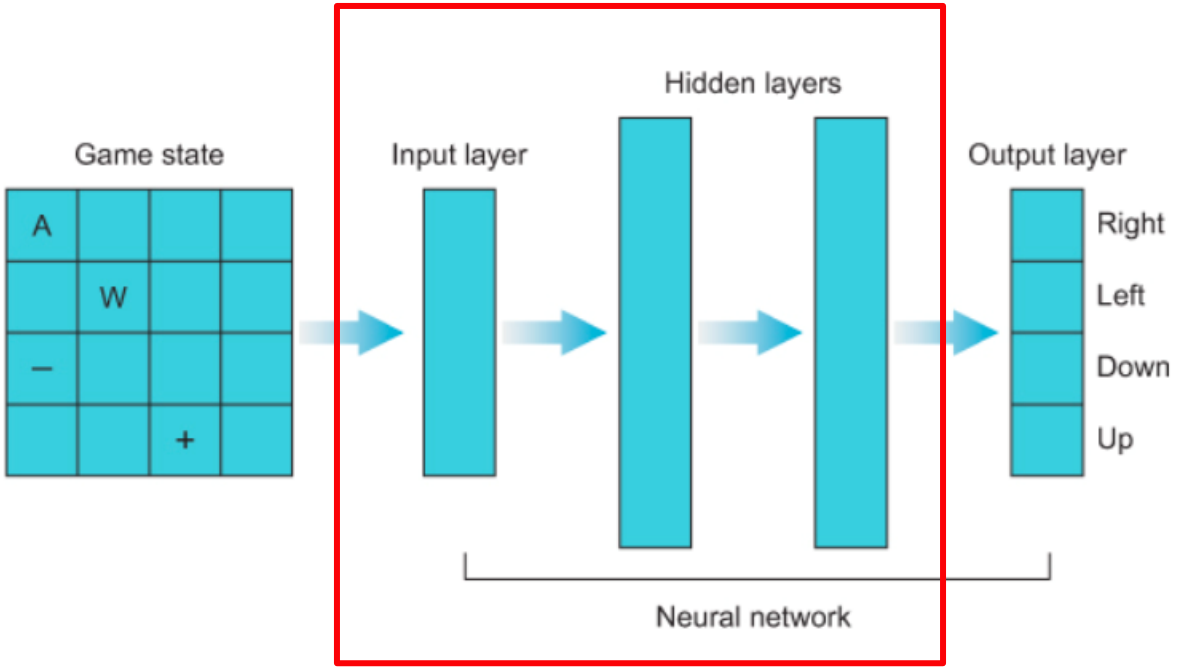
	Pre Q learning (PQ)	Multi Q-table Q-learning (MQQ)
장점	Q테이블을 미리 저장하므로 탐색시간 단축	환경 변화에 유리
공통 장점	학습에 필요한 시간이 (20분 이내?) 적은 환경에서는 매우 효율적	
단점	환경이 변하면 Q테이블 다시 만들어야 함	경로마다 탐색과정을 거쳐서 상대적으로 시간이 더 소요
공통 단점	상태나 액션이 다양해지면 신경망을 사용해야 함	

강화학습 모델 분류

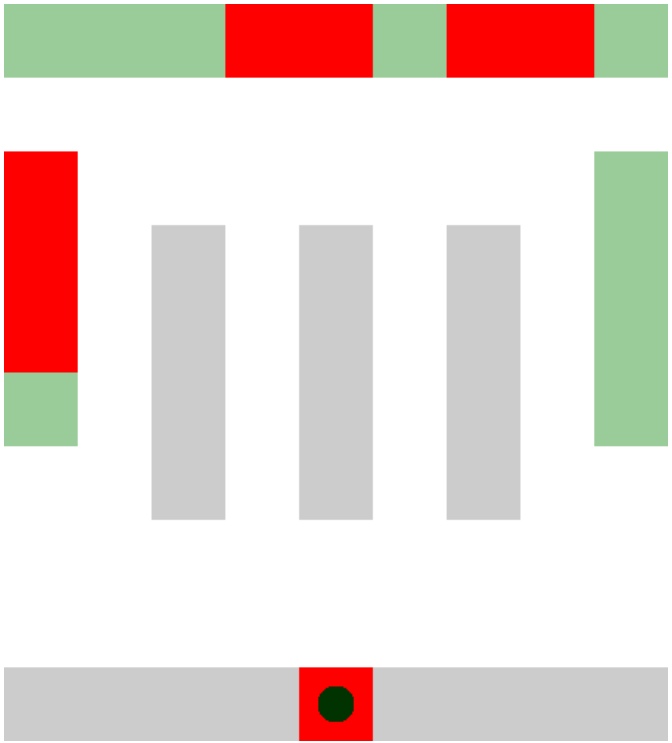
	Value Based	Policy Based	Actor-Critic
On-Policy	• Monte Carlo Learning (MC) • TD(0) • SARSA • Expected SARSA • n-Step TD/SARSA • TD(λ)	• REINFORCE ☑ • REINFORCE with Advantage	• A3C • A2C ☑ • TRPO • PPO
Off-Policy	• Q-Learning • DQN ☑ • Double DQN ☑ • Dueling DQN		• DDPG • TD3 • SAC • IMPALA

기본적인 모델부터 시도...

DQN (Deep Q Networks)



오강자의 초기 DQN



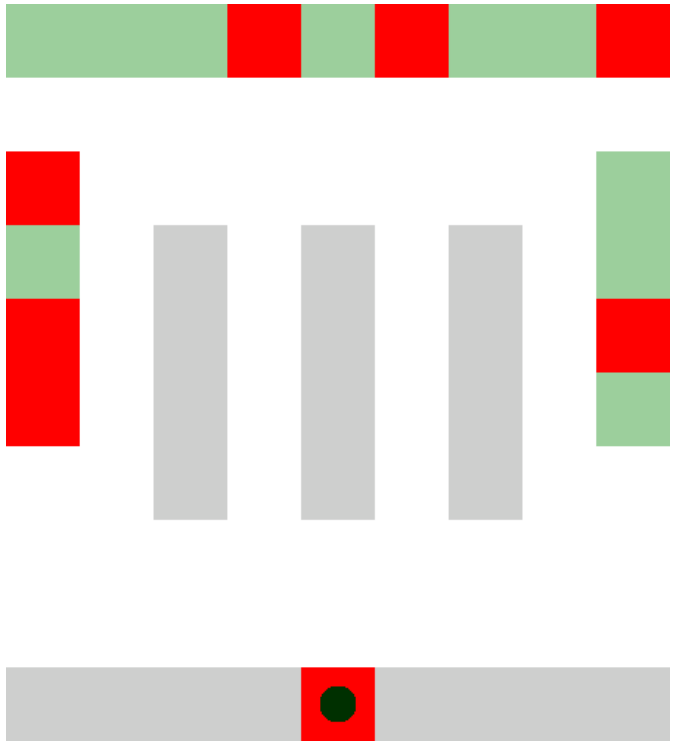
- **Input**
  - 시작 좌표 [9, 4], 아이템 좌표[5, 0]
- **Environment**
  - 기업에서 제공한 baseline 코드
  - 장애물에 닿거나 밖으로 나가면 종료
- **Agent**
  - Network: FCN

    (Fully Connected Network, Linear)
- **Hyperparameter**
  - 문헌 참고

오늘의 강자...

.....

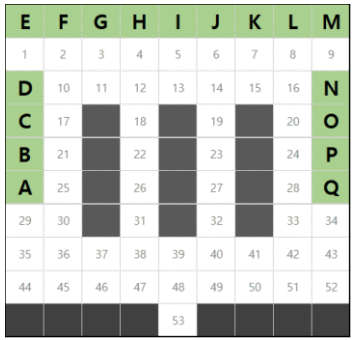
```
Episode: 1220   Timestep: 52992   Moves: 42
Average Reward: 79.21   Finish Rate: 1.00

Episode: 1221   Timestep: 53034   Moves: 42
Average Reward: 79.22   Finish Rate: 1.00

Episode: 1222   Timestep: 53078   Moves: 44
Average Reward: 79.22   Finish Rate: 1.00

Episode: 1223   Timestep: 53122   Moves: 44
Average Reward: 79.22   Finish Rate: 1.00

Episode: 1224   Timestep: 53164   Moves: 42
Average Reward: 79.22   Finish Rate: 1.00

Episode: 1225   Timestep: 53208   Moves: 44
Average Reward: 79.23   Finish Rate: 1.00

Test performed: 1226, # of wins: 1226

Win percentage: 100.0%

Average step: 43.39967373572594

1.0
```



개선한 사항

1. (Env.) Grid Map
2. (Input) Observations
3. (Env.) Distance-to-Target Reward
4. (Env.) Rules of the Game
5. (Main) Action masking & (Env.) Move block
6. (Agent) Network
7. (Main) Hyperparameters

개선사항 1. (Env.) Grid map



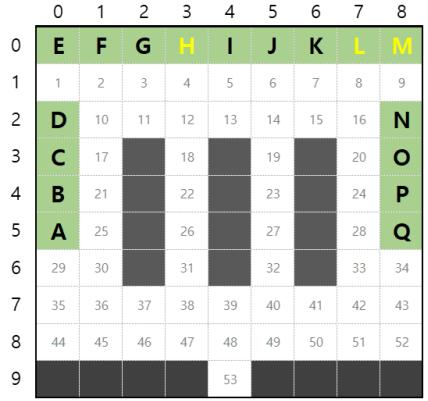
100	100	100	-100	100	100	100	-100	-100
1	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	100
100	1	0	1	0	1	0	1	100
100	1	0	1	0	1	0	1	100
100	1	0	1	0	1	1	1	100
1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	0	0	0	-5	0	0	0	0

-1	-1	-1	10	-1	-1	-1	9	8
0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	-1
-1	0	-10	0	-10	0	-10	0	-1
-1	0	-10	0	-10	0	-10	0	-1
-1	0	-10	0	-10	0	-10	0	-1
0	0	-10	0	-10	0	-10	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-10	-10	-10	-10	1	-10	-10	-10	-10

- 빈 선반: 100 → -1 — 빈 선반에 부딪히면 종료하지 않음
- **목적지: -100 → 10, 9, 8, …** — **목적지 그리드 값을 차등화**
- 장애물: 0 → -10 — 장애물 값을 목적지 값과 같은 수준으로 조정
- 현 위치: -5 → 1 — 현 위치값 조정
- 그 외: 1 → 0 — 그 외의 기본값 조정

✓ **그리드 값 간의 편차 축소**

개선사항 2. (Input) Observations

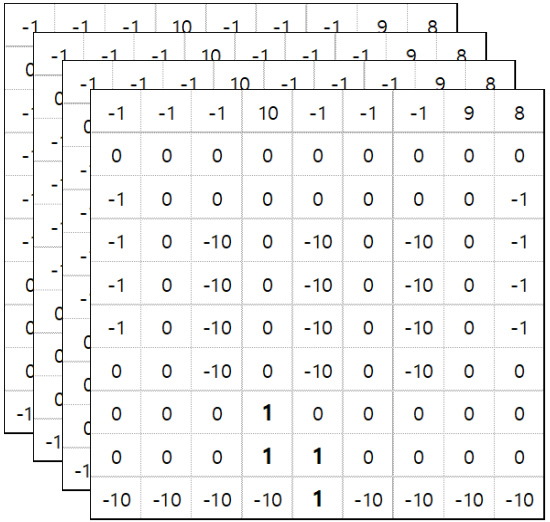


출발지 → 목적지

[9, 4] → [0, 3]
[0, 3] → [0, 7]
[0, 7] → [0, 8]
[0, 8] → [9, 4]

100	100	100	-100	100	100	100	-100	-100
1	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	100
100	1	0	1	0	1	0	1	100
100	1	0	1	0	1	0	1	100
100	1	0	1	0	1	1	1	100
1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	0	0	0	-5	0	0	0	0

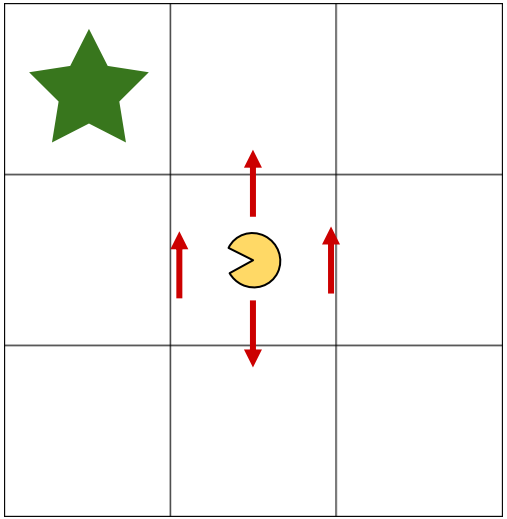
그리드맵 1장

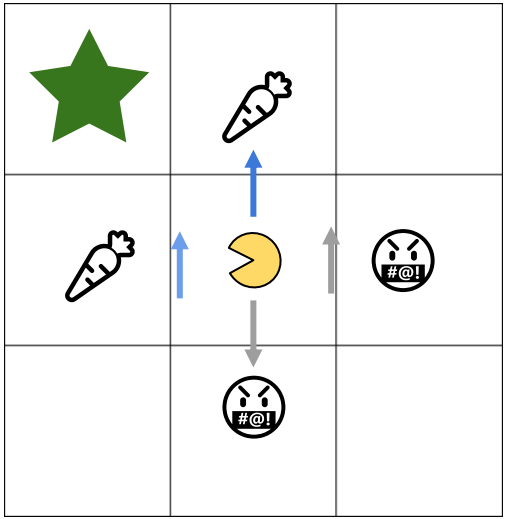


그리드맵 4장

✓ Input 정보 (observations) 의 확대

개선사항 3. (Env.) Distance-to-Target Reward





- 어디로 가던지 동일한 보상
- **에이전트 방황**

- 목적지와 가까울수록 “+”, 멀어질수록 더 큰 “-” 보상
- **에이전트를 목적지로 유도**

✓ 목적지까지의 거리에 비례하는 확실한 보상

개선사항 4. (Env.) Rules of the Game

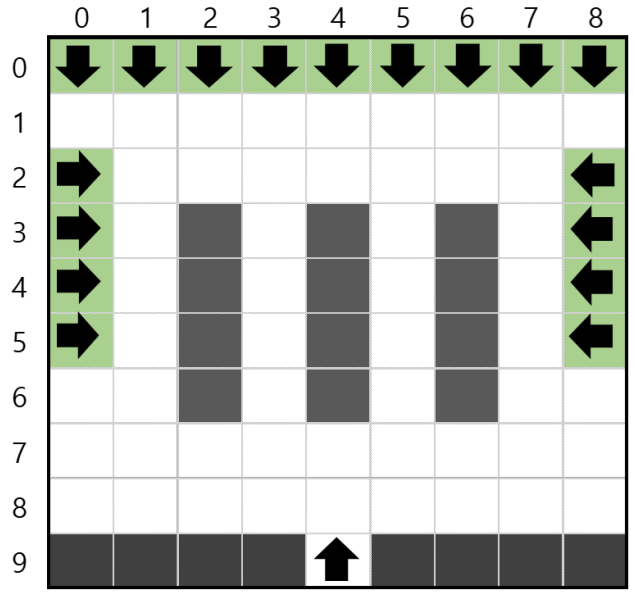
	기존 모델	DQN 모델
그리드 월드 벗어날 시	Game over	Continue, 들어갈 수 없음
장애물에 도착시	Game over	Game over
빈 선반(rack) 도착시	들어갈 수 있음	들어갈 수 없음
최대 step수	적용 X	200
Action Masking	적용 X	적용 O

✓ 강자가 죽지 않게 Rule을 조정하여 더 많은 학습 경험을 갖도록 유도

✓ 끝나지 않고 계속 돌아다니는 현상을 막기 위해 최대 step 수 추가

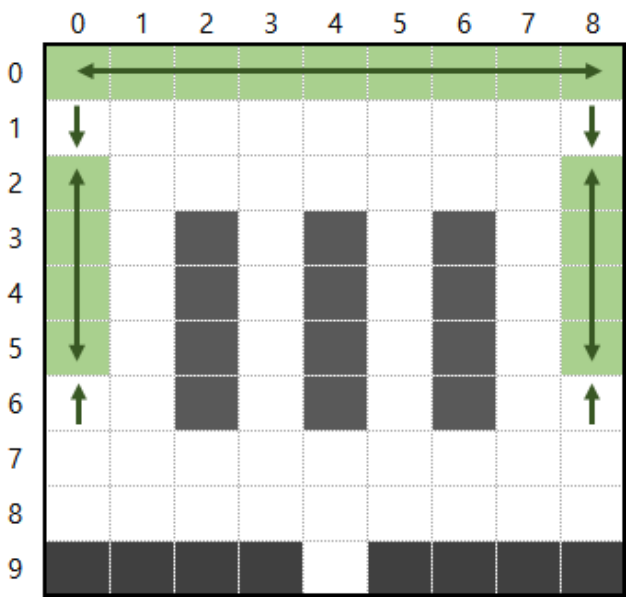
(그리드월드내 최대 이동 가능 수 53의 약 4배로 지정)

개선사항 5. (Main) Action Masking & (Env.) Move Block



Action Masking

무조건 이동하는 방향



Move Block

이동할 수 없는 방향

개선사항 6. (Agent.) Network

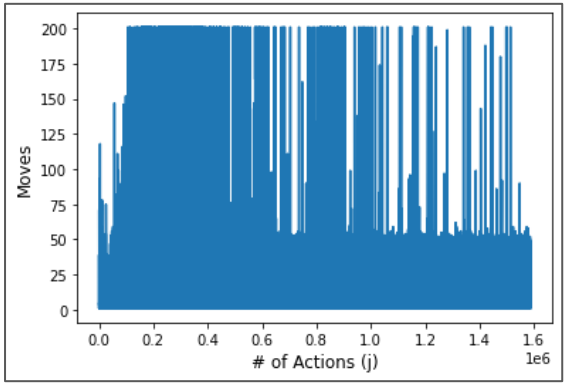
Linear (FCN)		Convolution
input 크기 : 4 현재 x, y 좌표, 타겟 x, y 좌표 작고 간단한 Input 데이터	→	Input 크기: 10*9*4 연속으로 변화하는 Grid Map 4장 이미지 지역 정보 학습

DQN		DDQN
특정 조건에서 Action Value의 Overestimation 발생	→	Overestimation 현상을 줄여주며, 다양한 환경에서 더 나은 성능을 보여줌

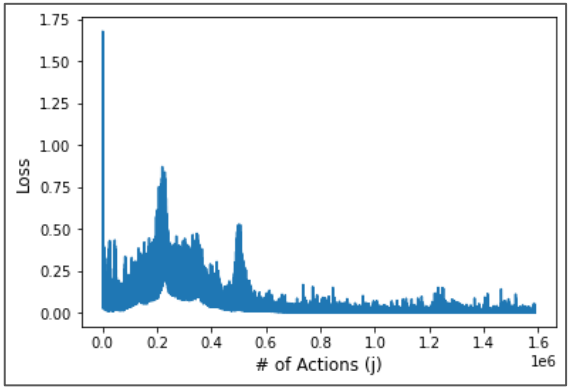
개선사항 7. Hyperparameters

	Trial #1x	Trial #2x	Trial #3x	Finally...
buffer limit	200,000	100,000	100,000	100,000
batch size	200	400	200	200
learning rate	0.00025	0.00025	0.0025	0.00025
sync freq	500	500	500	500
train start	50,000	50,000	50,000	50,000
epochs	80,000	80,000	80,000	50,000
max moves	200	200	200	200
epsilon	0.3	max(0.05, 1.0-0.1*(epi/2000))	max(0.05, 1.0-0.1*(epi/2000))	max(0.05, 1.0-0.1*(epi/2000))

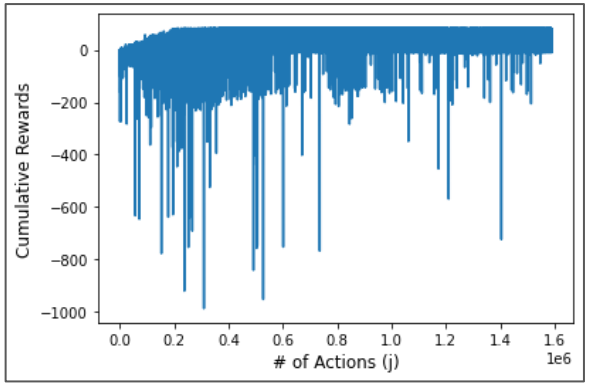
DQN 학습 결과



Moves



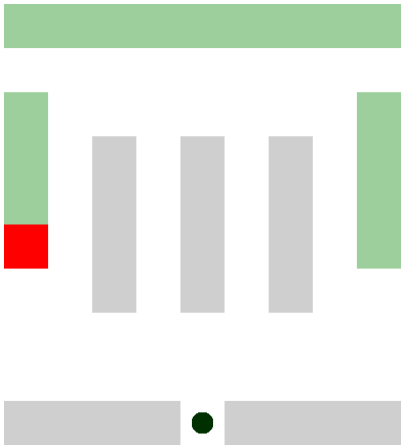
Loss



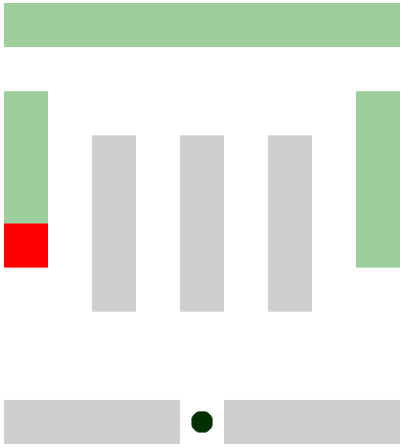
Cum Rewards

Problem Solving Approach: Divide & Conquer!

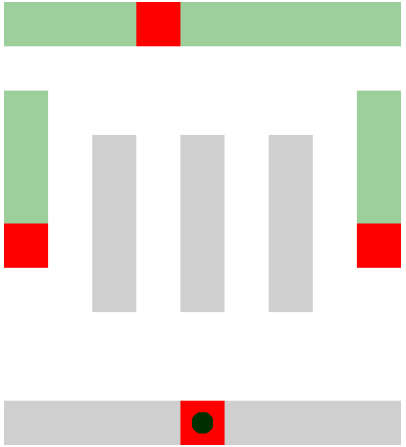
1 Target 1 Way
시작 → A



2 1 Target Roundtrip
시작 → A → 종료



3 Multi Target Path
시작 → A → H → Q→ 종료

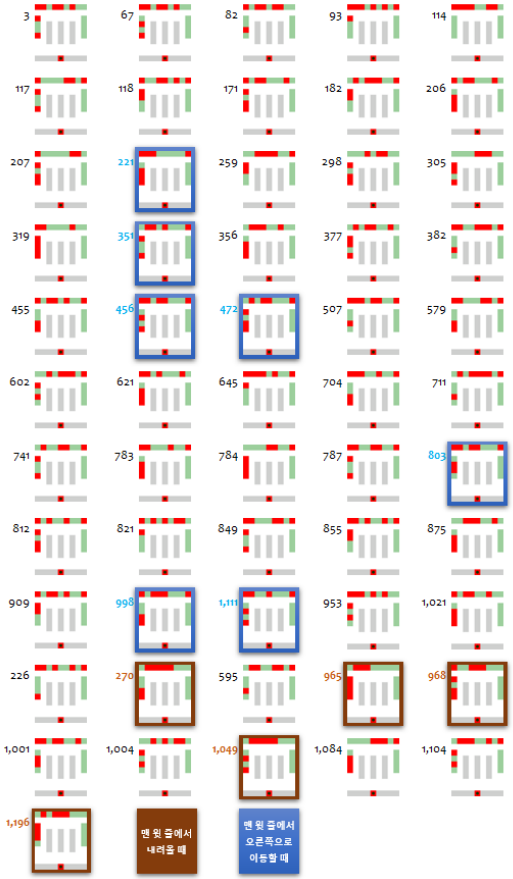


각자 다양하게 시도한 모델들에서 좋은 결과를 얻지 못함

문제를 단순화하여 **문제 원인 파악 및 하이퍼파라미터 튜닝 등 다양한 해결 방안 시도**

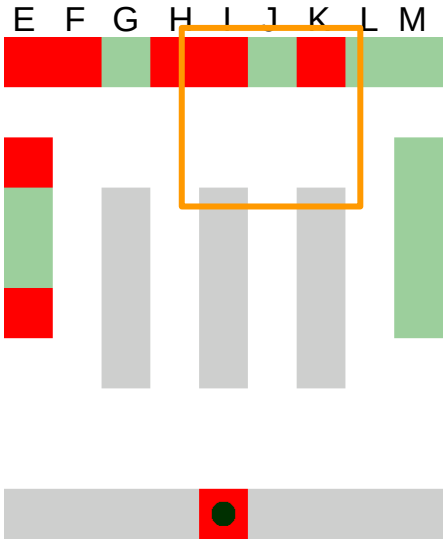
(사례) 1을 통하여 17개 목적지 중 일부 목적지에 대해서만 DQN이 최적화되는 현상 발견: 17개 중 6개, 10개 등

DQN test 결과



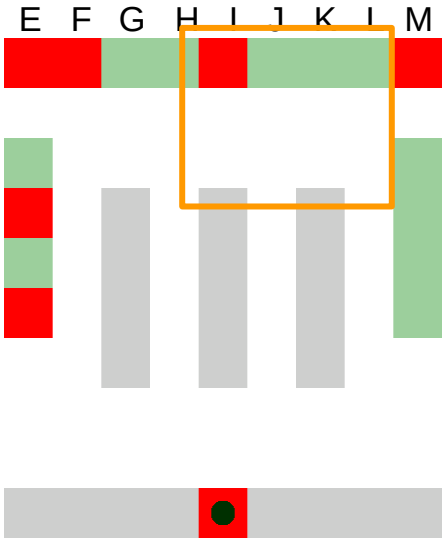
Move 수	Episode 개수	비최적 개수	비최적 비율	Test Episode 번호
38	7	1	2%	965
40	64	5	9%	270, 298, 305, 1049, 1196
42	422	4	7%	82, 356, 968, 1004
44	554	12	21%	3, 67, 114, 207, 298, 305, 319, 377, 595, 741, 821, 875, 953, 1021
46	159	14	25%	117, 171, 182, 206, 226, 259, 351, 602, 621, 711, 784, 1001, 1084, 1104
48	16	16	29%	93, 118, 382, 455, 472, 507, 579, 645, 704, 783, 787, 812, 849, 855, 909, 1111
50	4	4	7%	221, 456, 803, 998
합계	**1,226**	**56**	**4.57%**	

DQN 최적이 아닌 경로 유형



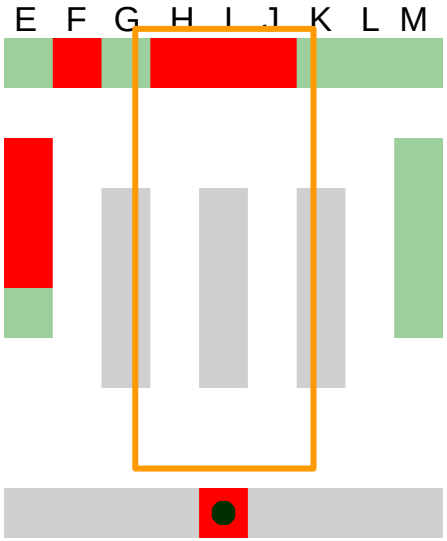
에피소드 3번

오른쪽으로 이동 시
아래로 한번 더 이동하는 유형



에피소드 1,111번

오른쪽으로 이동시
아래로 두 번 더 이동하는 유형



에피소드 1,196번

왼쪽으로 돌아 나가는 유형

모델 비교

	Optimum	PQ	MQQ	DQN
평균 Move	**43.2969**	43.4029	**43.3100**	43.3997
비최적 개수	(전체 1,226)	65	**8**	56
비최적 개수 비율		5.30%	0.65%	4.57%
Move 오차 합		130	**16**	126
(최적 대비 초과율)		0.24%	0.03%	0.24%
Move 최대 오차		2	**2**	4
(최적 대비 초과율)	(43.2969 대비)	4.62%	4.62%	9.24%
비고		이론적으로 무한 번 반복시 최적과 같아야 함		

마치며 ...

신경망 초기 모델인 DQN : 우수한 성능

모델을 더 개선하고, 최신 모델을 적용한다면

이 프로젝트는 얼마나 더 무궁무진해질까?

Further Study

- **직선으로 움직이도록 움직임 개선**
- **현재 모델의 성능 향상을 위한 시도**
  - 인풋 데이터 정보량 늘이기: 4 → 8개
- **MQQ 아이디어의 네트워크 버전 시도**
- **보다 큰 그리드월드에 대한 성능 향상을 위한 시도**
  - Deep Network 활용: ResNet, DenseNet 등
- **최신 (정책 기반) 강화학습 알고리즘 적용**
  - PPO (Proximal Policy Optimization),
    TRPO (Trust Region Policy Optimization) 등

팀원 소개 및 역할

신휘정

업 앤 다운이 확실한 팀장

TD-AC, REINFORCE 모델 구현 및 연구

보상체계 개발

DQN 모델 개선

프로젝트 관리

노현호

성격은 순둥, 열정은 맹렬!

DQN 모델 개선

MQQ 모델 구현 및 연구

MQQ, DQN 논문 분석 및 발표

노션관리

팀원 소개 및 역할

박흥선

내용 총정리의 달인이자 논리킹

MQQ 모델 구현 및 연구

DQN 모델 구현 및 개선

DQN 논문 분석

프로젝트 관리

장진구

퐁퐁 솟는 아이디어의 샘물

PQ 모델 구현 및 연구

좌수법, A* 휴리스틱 알고리즘 개발

DQN 모델 연구

Draw_utils 모듈 개선

팀원 소개 및 역할

한태양

막힐 땐 그를 찾아라. 그가 해결해줄지니..

MQQ 모델 개선, DQN 모델 연구

Draw_utils 모듈 개선

MQQ 논문 분석

노션 및 패들릿 관리

공통

강화학습 스터디

베이스라인 코드 분석

프로젝트 아이디어 제시 및 구현

PPT, 보고서 작성

Speatial thank to

퍼실 : 유 상민

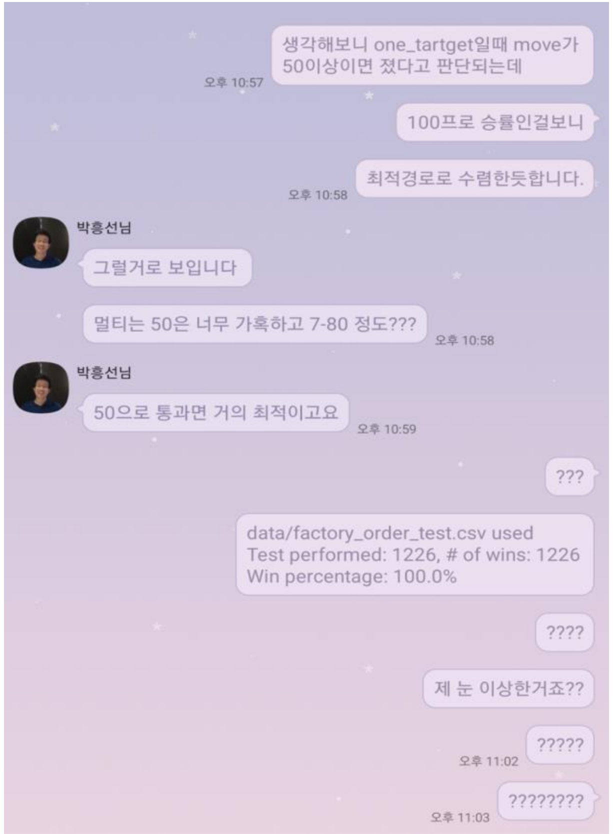
멘토 : 채 지훈

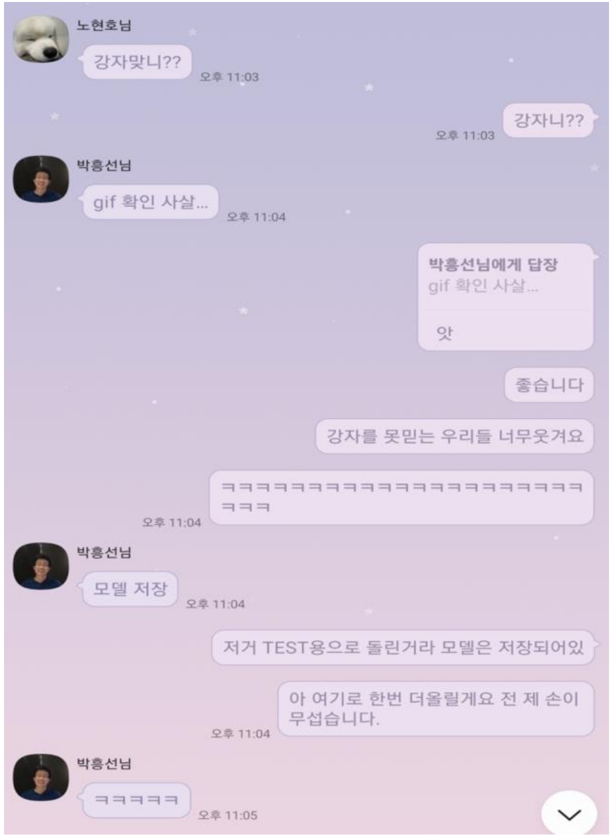
민 예린

References

1. “강화학습-Double-DQN,” 최호승, *https://velog.io/@d2h10s/강화학습-Double-DQN.*
2. “모두를 위한 머신러닝과 딥러닝의 강의 - 시즌 RL - Deep Reinforcement Learning,” 김성훈, *https://hunkim.github.io/ml/.*
3. *바닥부터 배우는 강화학습*, 노승은, 영진닷컴, 2021.
4. *파이썬과 케라스로 배우는 강화학습*, 이웅원 외, 2018.
5. “Part 1 — Building a deep Q-network to play Gridworld — DeepMind’s deep Q-networks,” N. Joshi, *https://towardsdatascience.com/part-1-building-a-deep-q-network-to-play-gridworld-deepminds-deep-q-networks-78842007c631*, 2021.
6. “Part 2 — Building a deep Q-network to play Gridworld — Catastrophic Forgetting and Experience Replay,” N. Joshi, *https://towardsdatascience.com/part-2-building-a-deep-q-network-to-play-gridworld-catastrophic-forgetting- and-experience-6b2b000910d7*, 2021.
7. “Part 3— Building a deep Q-network to play Gridworld — Learning Instability and Target Networks,” N. Joshi, *https://towardsdatascience.com/part-3-building-a-deep-q-network-to-play-gridworld-learning-instability-and- target-networks-fb399cb42616*, 2021.
8. “Multi Q-Table Q-Learning,” N. Kantasewi, S. Marukatat, S. Thainimit, and O. Manabu, *2019 10th IC-ICTES*, 2019.
9. “Path Planning via an Improved DQN-Based Learning Policy,” L. LV, S. Zhang, D. Ding, and Y. Wang, *IEEE*, vol. 7, 2019.

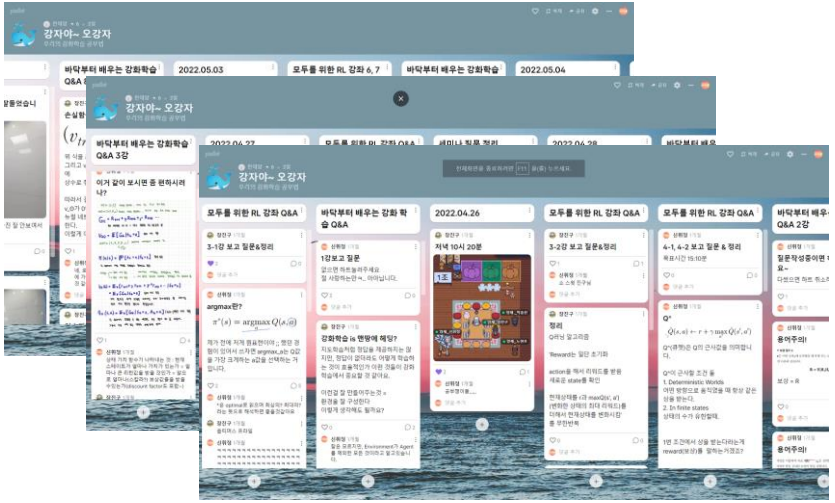
발표 들어주셔서 감사합니다.





Appendix #0. 우리들의 일대기

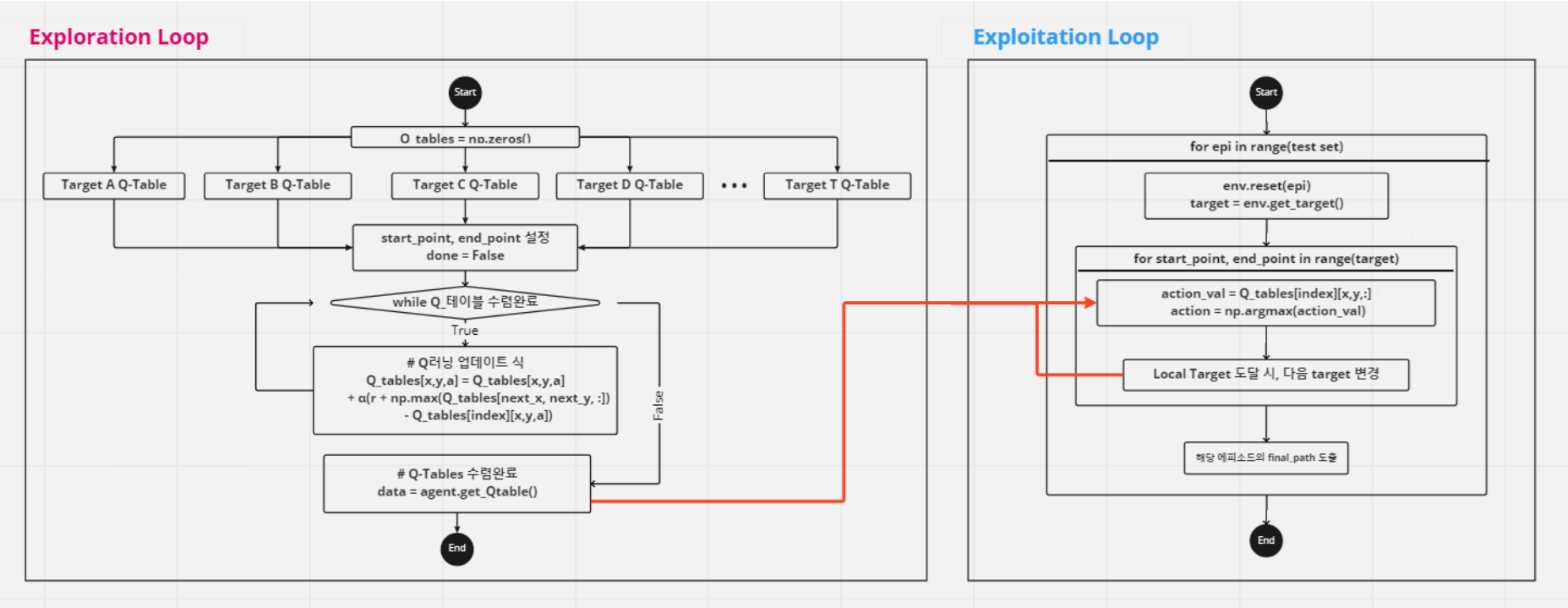
1. 우리들의 공부 일지
https://padlet.com/tmsk0711/ogangza



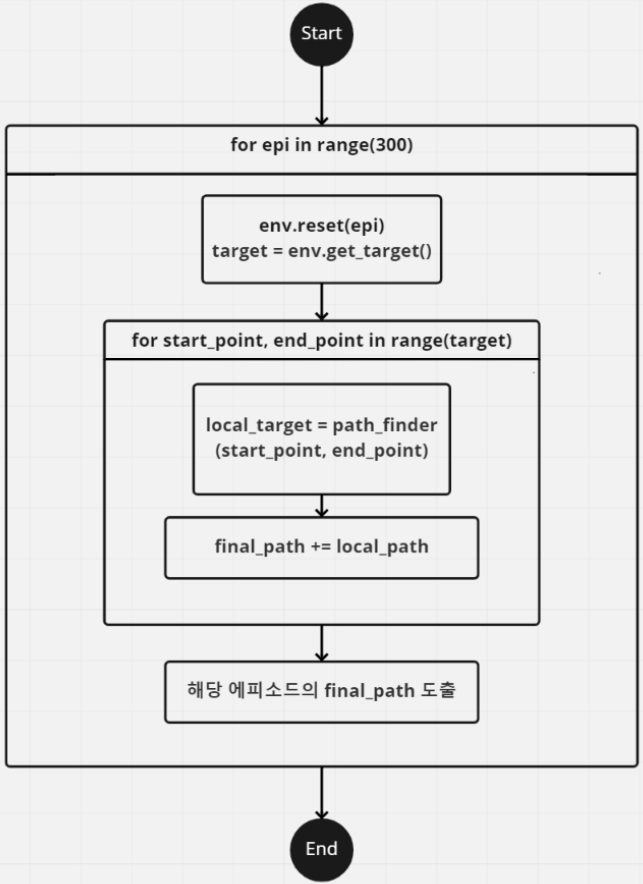
2. 하이퍼 파라미터 찾기 연대기
https://padlet.com/tmsk0711/ogangza_parametertuning

3. 우리들의 프로젝트 관리
https://www.notion.so/f47ef3d55d86420985d99491af2fdd76

Appendix #1. PQ Algorithm Flow chart



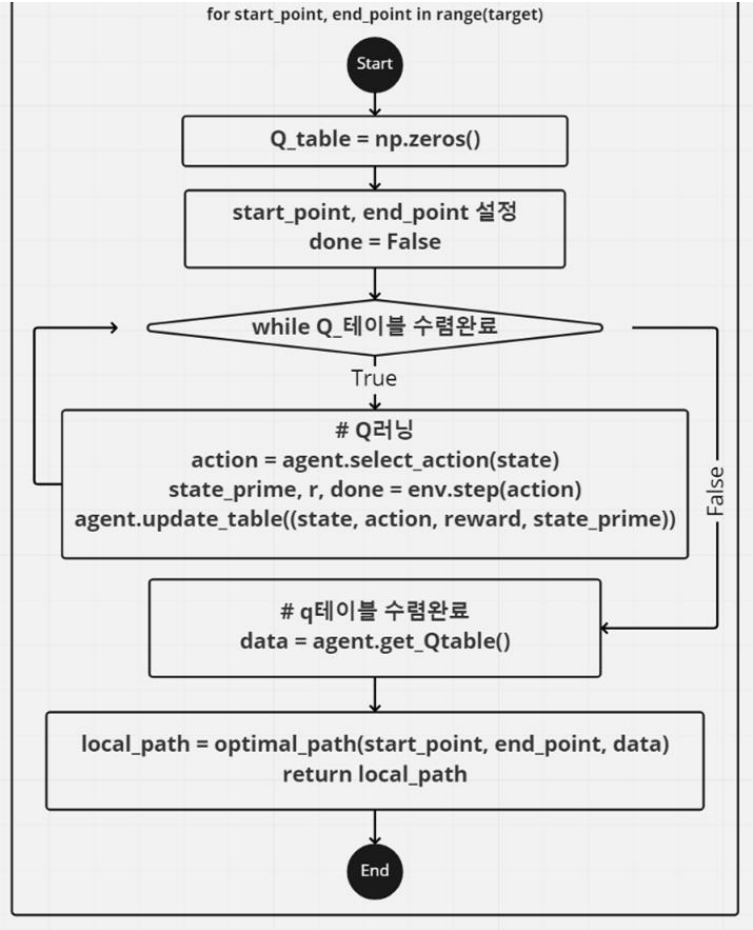
Appendix #2. MQQ Algorithm Flow chart



main 함수

1. env.reset()
   에피소드에 해당하는 환경을 세팅해줌
1. targets = env.get_target()
   현재 에피소드에 해당하는 아이템 target값을 가져옴
1. for _ in targets:
   for문을 이용하여, S→A→H→S 의 문제를
   (S→A), (A→H), (H→S)의 작은 문제로 나누어 주고,
   path_finder 메소드를 이용하여
   작은 문제들의 Optimal한 local_path를 구해줍니다.
   그리고 final_path에 local_path를 더해줍니다.
1. for 문이 끝나면, S→A→H→S에 대한 Optimal한 경로가 도출되게 됩니다.

Appendix #2. MQQ Algorithm Flow chart



path_finder(start_point, end_point)함수

1. Q 테이블 초기화
2. start point, end point, done==False 설정
3. while Q테이블이 수렴할 때 까지 반복:

〈Q러닝〉

a. Agent가 현재 state를 보고 action을 결정
   → 초반엔 랜덤행동, 후반으로 갈 수록 학습된 정보 이용
a. 해당 state에 action을 하면 다음 state가 어떻게 되는지, 보상, done값을 Environment로 부터 받음.
a. Agent는 행동의 결과에 대해 Q테이블을 업데이트

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

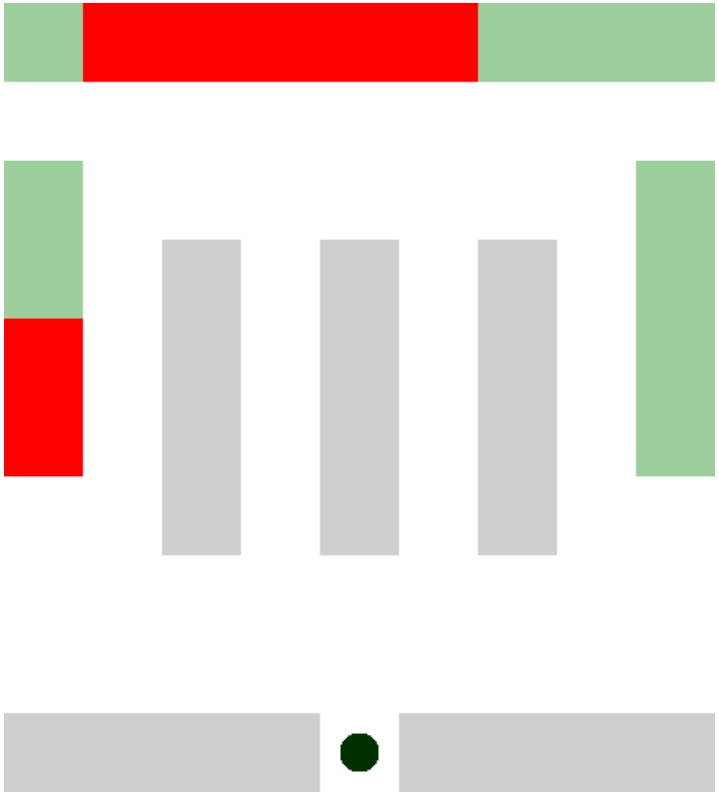
4. 수렴된 Q테이블을 가져와 optimal_path 메소드를 이용하여 local_path를 구합니다.

Appendix #3. 휴리스틱 알고리즘

1) 좌수법 (Left Hand Rule)

주어진 목표가 시계방향 순으로 있음에 착안

1. 과거에 이동한 방향에 따라 Agent의 방향 설정
   - 초기값 위, 이후 이동한 방향이 바라보는 방향
1. 왼쪽에 장애물이 없으면 왼쪽으로 이동
2. 왼쪽에 장애물이 있으면 위쪽으로 이동
3. 왼쪽, 위쪽에 장애물이 있으면 오른쪽으로 이동
4. 왼쪽, 위쪽, 오른쪽에 장애물이 있으면 뒤로 이동
5. 시작지점에 돌아오면 종료



Appendix #3. 휴리스틱 알고리즘

2) A* 알고리즘

F : 시작점과 목적지의 거리의 합

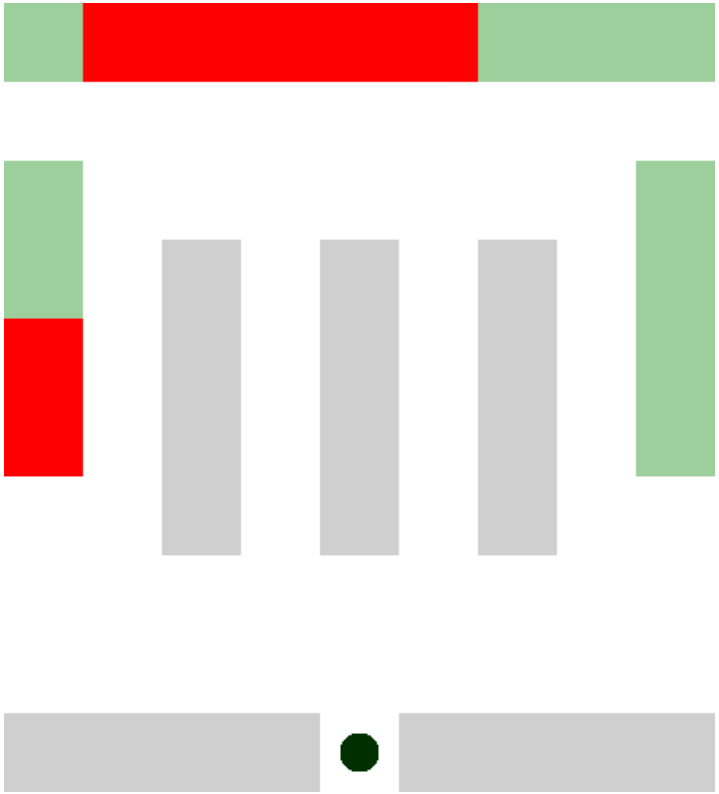
G : 현재 노드에서 출발 지점까지의 거리

H : 현재 노드에서 목적지까지 거리의 합

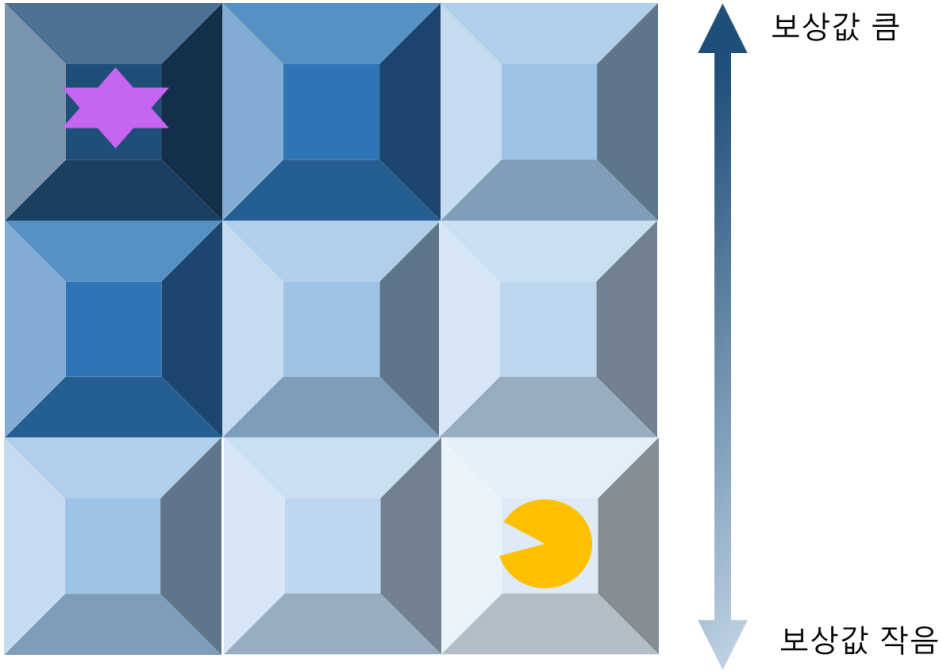
'F = G + H' 를 계산하면서 이동한다.

1. 4방향 이동 Grid에서 cost는 거리와 같다.
2. 시작점, 목적지의 x축, y축 차이의 합 = 거리
3. 장애물이 있다면, 다른 위치를 불러온다.

*빈 아이템 박스는 장애물로 취급함



Appendix #4. Reward

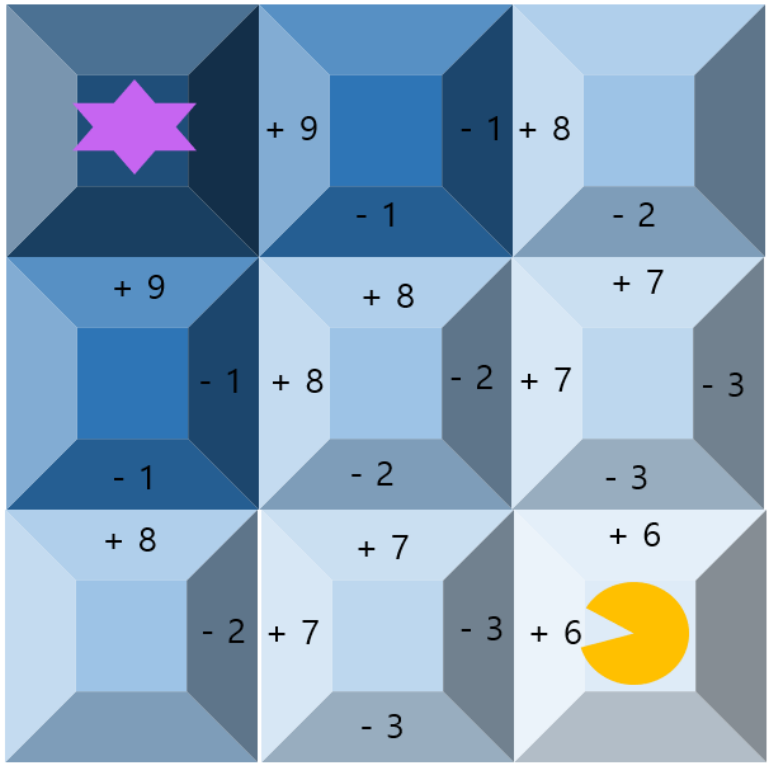


리워드 보상을 통해, 의도 하고 싶은 Q테이블

1. 강자가 아이템과 가까워지면 +보상, 멀어지면 - 보상 받도록 설계

1. 아이템과 가까운 State는 얻을 수 있는 보상 총 합이 크도록! 멀면 작도록!

Appendix #4. Reward



Distance를 구해서,

가까워지면 + (10-Distance)

멀어지면 – Distance

10은 임의로 결정했음(표 크기)

Appendix #4. Reward : reset 함수

```
######################
self.move_track = [[9,4]]                    #14. 이동하는 경로를 저장하는 변수,
                                             # 에피소드 시작시 에이전트가 [9,4]에 있으므로 [[9,4]]로 초기화
self.isReset =  False                        #15. 처음으로 terminal_location == 9,4가 되면, moveTrack을 초기화해주기
                                             # 위해 필요한 변수 isFinal -> isReset 로 변경(처음으로 불러졌냐 안불러졌냐)
self.prior_distance = 9999                   #16. 이전 time step의 거리와 현재 거리를 비교하기 위해서 필요한 변수
######################
self.wasGoal = False                         # 17. 이전 step이 Goal이었는지 확인하는 변수
                                             # 이전 step이 Goal이었으면 먹고 나가야 하는 곳이 move_track이 겹치기 때문에 -보상받음. 이를 방지해기위한 FLAG
```

Appendix #4. Reward : 새로 추가한 함수

```
# >>> distance를 입력으로 받아 + 보상을 계산하는 메소드 (시작) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def get_positive_distance_reward(self, distance):
    positive_reward = (20 - distance) / 200     # 0.095 ~ 0.001
    return positive_reward
# >>> 추가 (끝) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# >>> distance를 입력으로 받아 - 보상을 계산하는 메소드 (시작) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def get_negative_distance_reward(self, distance):
    negative_reward = -(distance / 200)
    return negative_reward
# >>> 추가 (끝) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# >>> 현재 위치에서 타겟까지의 거리를 구하는 메소드 (시작) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def get_distance(self):
    cur_x, cur_y, target_x, target_y = self.get_current_state()
    distance = (abs(cur_x-target_x) + abs(cur_y-target_y))
    # print(cur_x, cur_y, target_x, target_y)
    # print('distance : ', distance)
    return distance
# >>> 추가 (끝) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# >>> 현재 상황에 대한 정보를 구하는 메소드 (시작) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
def get_current_state(self):
    # if len(self.local_target) != 1:
    #     state = (self.curloc[0], self.curloc[1], self.local_target[0][0], self.local_target[0][1])
    # else :
    state = (self.curloc[0], self.curloc[1], self.terminal_location[0], self.terminal_location[1])
    return np.array(state) #(cur_x, cur_y, tar_x, tar_y)
# >>> 추가 (끝) <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Appendix #4. Reward : step함수

```
############################### 보상받는 처리 시작 ################################
# 마지막 위치에서 위로 올라가는 행동을 한다면 - 보상받음 태양
if len(self.local_target) == 1:                          # local_target이 remove되고 나면 길이가 1이되는 경우 = [9,4]만 있는 경우
    if not self.isReset:                                 # 처음으로 terminal_location == [9,4]가 되었다면,,
        self.move_track = []                             # 아이템을 찾으러 돌아다니는 경로 초기화
        self.isReset = True                              # move_track이 초기화 되었으니 isReset = True
    reward = self.get_reward(new_x, new_y, out_of_boundary)
    if action == 0:
        reward = obs_reward                             # terminal_location == [9,4]로 갈 때 위로 올라가는거 금지
else :
    reward = self.get_reward(new_x, new_y, out_of_boundary)
################################ 보상받는 처리 끝 ################################
```

Appendix #4. Reward : get_reward함수

```
def get_reward(self, new_x, new_y, out_of_boundary):  # action에 의해 이동시 얻는 보상
    if any(out_of_boundary):                                  # 바깥으로 나가는 경우
        reward = obs_reward
    else:
        if self.grid[new_x][new_y] == obs_gridval :     # 장애물에 부딪히는 경우
            reward = obs_reward
            self.grid[new_x][new_y] = curloc_gridval   # 현 위치(장애물)에서 종료되었음을 표시. 원래는 장애물이었음
        elif self.grid[new_x][new_y] == rack_gridval:   # 선반에 부딪히는 경우
            reward = obs_reward
            new_x = self.curloc[0]  # 선반인 경우 못들어가고, 현재 위치로 유지
            new_y = self.curloc[1]  # 선반인 경우 못들어가고, 현재 위치로 유지
        elif new_x == self.terminal_location[0] and new_y == self.terminal_location[1]:  # 현재 목적지에 도달한 경우
            reward = goal_reward
            self.prior_distance = 9999                 # 목적지에 갈 때 prior_distance = 10이라, 나갈 때 무조건 prior_distance가
                                                       # 작아서 negative_reward를 받기때문에 여기서 못받도록 수정해줌.
        # 그냥 움직이는 경우
        else:
            cur_distance = self.get_distance()
            if self.prior_distance > cur_distance :                          # 이전의 거리 > 현재의 거리 = 현재 아이템에 더 가깝다 => + 보상얻기
                reward = self.get_positive_distance_reward(cur_distance)
            else:                                                            # 이전의 거리 < 현재의 거리 = 현재 아이템에서 멀어졌다 => - 보상얻기
                reward = self.get_negative_distance_reward(cur_distance)
            self.prior_distance = cur_distance

            if [new_x, new_y] not in self.move_track:                           # 이동한 곳이 처음온 곳이면 move_track에 추가
                self.move_track.append([new_x, new_y])
            else:                                                            # 이동한 곳이 이미 왔던 곳이면 기존 보상에서 -1
                reward = reward -1
    return reward
```