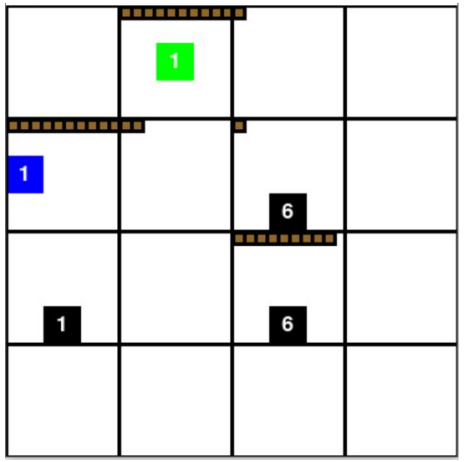
강화학습 프로젝트
최적 경로 찾기

신요섭, 정승연, 윤서진

강화학습의 기초
담당교수 : 소정민 교수님

목차



참고 : GitHub Link
https://github.com/Grace-0710/WareHouseEnv

프로젝트 팀 소개

성명	학번	역할	Remark
신요섭	A66045	강화학습 구현, 보고서 작성	팀원
정승연	A66071	환경 구현, 강화학습 구현, 보고서 작성	팀장
윤서진	A67029	환경 구현, 강화학습 구현	팀원

1. 프로젝트 주제 및 목표

❖ **프로젝트 주제** :

 2개의 지게차와 3개의 로봇으로 150개의 짐을 정해진 위치에 최적의 경로로 옮기기

❖ **목표** :

 학습적 측면

✔ Multi Agent Reinforcement Learning에 대한 학습과 이해
✔ 강화학습 알고리즘 Library에 대한 학습과 응용
✔ 수업시간에 다루지 않았던 PPO 알고리즘에 대한 학습과 이해

 강화학습 프로젝트 측면

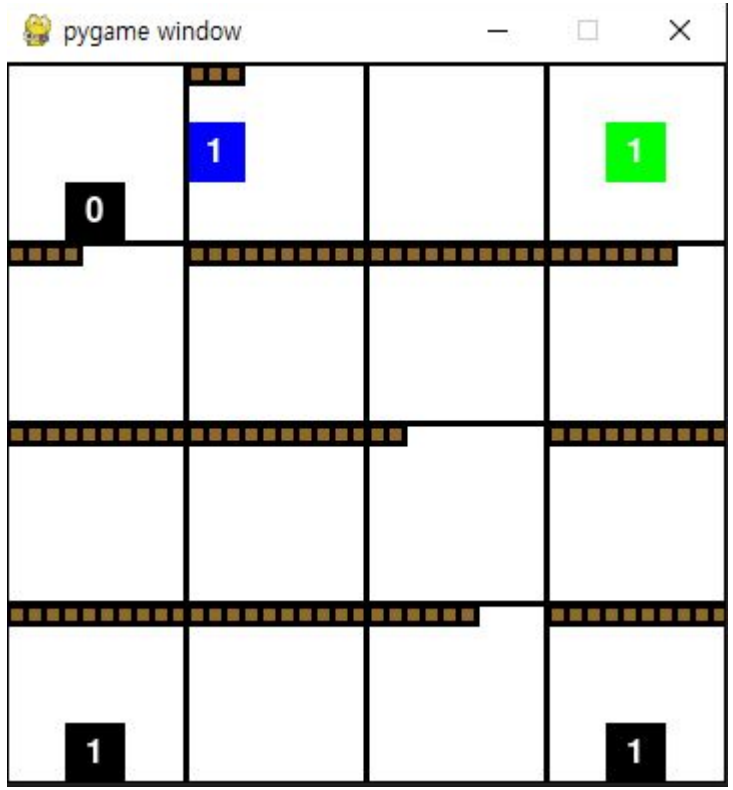
✔ 강화학습이 제대로 이루어졌는지 Reward 검증
✔ 강의 교재 예제 ex16 기반 DQN과 Library로 구현되어 있는 DQN 비교
✔ DQN, A2C, PPO 비교

2. 환경 및 데이터셋 설명

❖ 환경 설명

4x4 정사각형 물류 창고 가정

- Load (한 칸 내 짐은 최대 10개)
- 1 로봇 (총 3개의 로봇, 각 로봇은 최대 6개의 짐을 운반할 수 있음)
- 1 1번 지게차 (로봇과 같은 칸에 있을 때 로봇에 짐을 올림)
- 1 2번 지게차 (로봇과 같은 칸에 있을 때 로봇에 짐을 올림)



3. State, Action, Reward 설계 설명

❖ State

- 4x4 물류 창고

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

❖ Agent

- 로봇 3대
  각 로봇의 움직임은 독립으로 서로의 움직임이 Reward에 영향을 주지 않음
- 지게차 2대
  지게차 2대는 연관성을 가져 2대의 State가 Reward에 영향
- 로봇과 지게차의 State는 Reward에 영향

❖ Action

- Up
- Right
- Down
- Left

3. State, Action, Reward 설계 설명

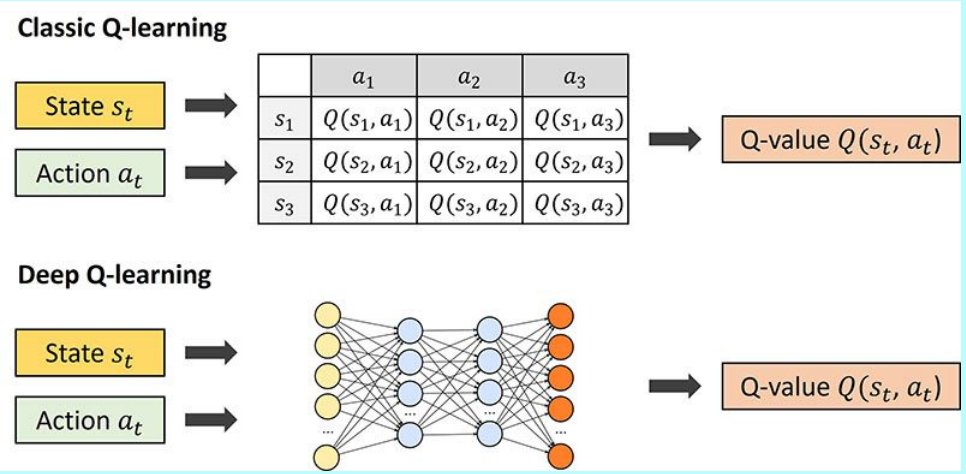
- Reward
  - 로봇이 짐이 있는 위치에 이동했을 때 짐이 많은 곳으로 이동하는 경우
  - 로봇이 짐을 가지고 지정된 위치 (0, 0)에 짐을 내려 놓는 경우

- Penalty
  - 로봇이 짐이 없는 위치로 이동하는 경우
  - 로봇이 지정된 위치에 짐을 내려놓지 않는 경우
  - 로봇이 일정 시간 이동하지 않는 경우 페널티
  - 필드에 남은 짐의 수가 많을수록 많은 페널티
  - 지게차가 짐을 싣고 있을 때, 다른 지게차가 같은 칸에 있는 경우 페널티

최종 목표 : 로봇이 최소한의 움직임으로 창고 내 150개의 모든 짐을 지정된 위치에 내려 놓는 것

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ DQN Algorithm



•Q-learning : state-action에 해당하는 Q-value를 table형식으로 저장하여 학습
  - Q-value를 저장하기에 많은 memory와 긴 exploration time 필요함

•DQN의 문제점
  - 매 스텝마다 Q-value를 업데이트하기에 약간의 변화에도 급격한 Policy 변화를 야기한다.

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ DoubleDQN

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

- 기존 DQN의 target value가 특정 조건에서 overestimate 되는 문제를 해결하기 위해 DoubleDQN 적용
- 기존 max값이 overestimation을 일으킨다고 보아 maxQ값을 통한 가치 추정

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ A2C Algorithm

Algorithm 1 A2C algorithm

Require: Initialize Actor-Critic network G with parameter θ.

for each episode **do**
  Get initial state s
  Initialize a storage buffer S, A, R, V, S'
  for $i = 1, 2, 3.., N$ **do**
    Sample an action $a \sim G(s)_\pi$ and get the associeted value $v \leftarrow G(s)_v$
    Run the action a through the environment, obtain the reward and next state $r, s' \leftarrow ENV(s, a)$
    Collect and store $S, A, R, V, S' \leftarrow s, a, r, v, s'$
    $s \leftarrow s'$
  end for
  Compute the discounted returns $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
  Compute an advantage function $\psi(V, R, S')$
  Optimize θ to minimize $-\log(G(A \mid S)_\pi)\psi(V, R, S') + \lambda\|V - \hat{V}\|$
  Empty S, A, R, V, S'

❖ PPO Algorithm

Algorithm 2 Proximal Policy Optimization (PPO)

Initialize actor $\mu: S \rightarrow R^{m+1}$ and $\sigma: S \rightarrow diag(\sigma_1, \sigma_2, \ldots, \sigma_{m+1})$
for i = 1 to M **do**
  Run policy $\pi\theta \sim N(\mu(s), \sigma(s))$ for T timesteps and collect (s_t, a_t, r_t)
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - v(s_t)$
  Update old policy $\pi_{old} \leftarrow \pi_0$
  for j = 1 to N **do**
    Update actor policy by policy gradient:

$$\sum_i \nabla_\theta L_i^{CLIP}(\theta)$$

    Update critic by:

$$\nabla L(\phi) = -\sum_{t=1}^{T} \nabla \hat{A}_t{}^2$$

  end for
end for

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ A2C Algorithm

- Policy Iteration 구조

- Policy network update (Actor)

- Policy evaluation q(s, a) 근사 (Critic)

❖ PPO Algorithm

- Policy Gradient Learning의 단점을 보완한 모델

- Step 단위로 학습데이터를 만들어내어 학습하는 방식

- 크리틱 신경망 (ϕ) 과 액터 신경망 (θ) 업데이트를 통한 학습 진행

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ 예제 16 DQN Hyperparameter

Learning Rate : 0.0005
Batch Size : 32
Gamma : 0.98
Buffer Limit : 50,000
Epsilon : 0.01 ~ 0.08

❖ DQN Hyperparameter

Learning Rate : 0.0001
Batch Size : 32
Gamma : 0.99
Buffer Limit : 1,000,000
Epsilon : 0.05 ~ 1
Target
Update Interval : 10,000

4. 강화학습 알고리즘 및 Hyperparameter 등 설명

❖ A2C Hyperparameter

Learning Rate : 0.0007
Batch Size : None
Gamma : 0.99
Buffer Class : None
* Buffer Class는 None일 때 자동으로 선택
Epsilon : 0.00001

❖ PPO Hyperparameter

Learning Rate : 0.0003
Batch Size : 64
Gamma : 0.99
Buffer Limit : 1,000,000
Epoch : 10
GAE Lambda : 0.95
Clip range : 0.2
VF Coef. : 0
* value function coefficient
Ent Coef. : 0.5
* entropy coefficient

5. 실험 셋업

❖ 실험 셋업 1.

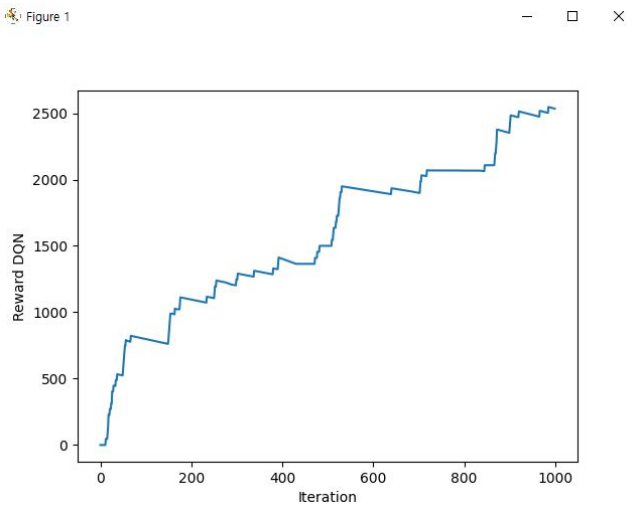
1. 환경, 강화학습 알고리즘, main 에 대한 코드 작성
2. 각 강화학습 알고리즘 (예제 16기반 DQN, DQN 함수)에 대한 결과 취득
   * 결과 : Reward에 대한 Plot
3. 각 알고리즘에 대한 실행을 3번씩 진행하여 코드에서 강화학습이 제대로 이루어지고 있는지 검증
4. 각 알고리즘에 대한 비교

❖ 실험 셋업 2.

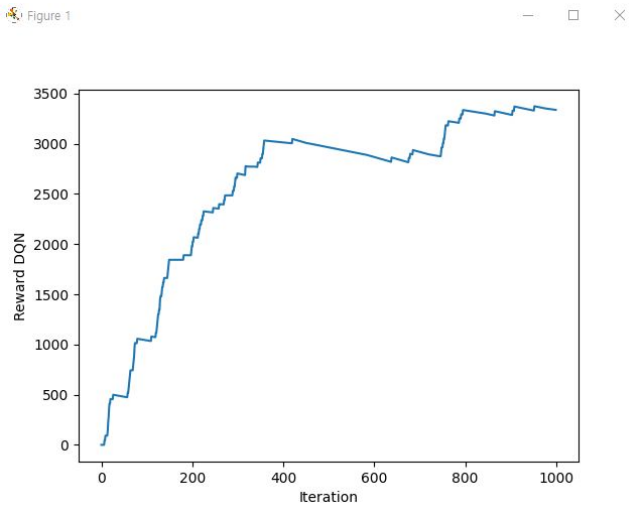
1. 환경, 강화학습 알고리즘, main 에 대한 코드 작성
2. 각 강화학습 알고리즘 (DQN, A2C, PPO)에 대한 결과 취득
   * 결과 : Reward에 대한 Plot
3. 각 알고리즘에 대한 실행을 3번씩 진행하여 코드에서 강화학습이 제대로 이루어지고 있는지 검증
4. 각 알고리즘에 대한 비교

6. 실험 결과 – DQN

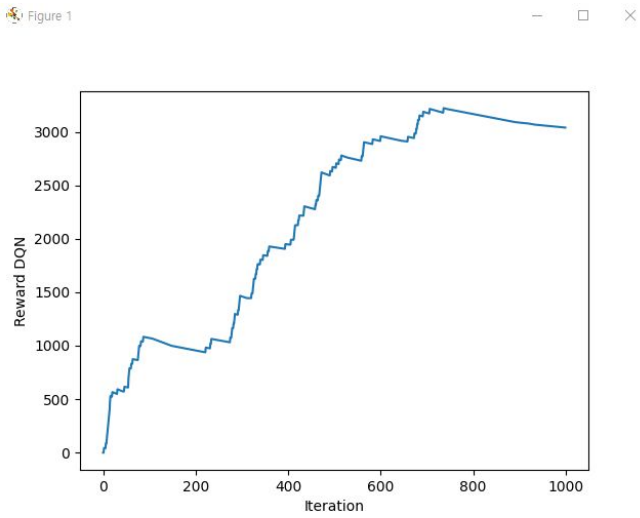
결과 1.



결과 2.



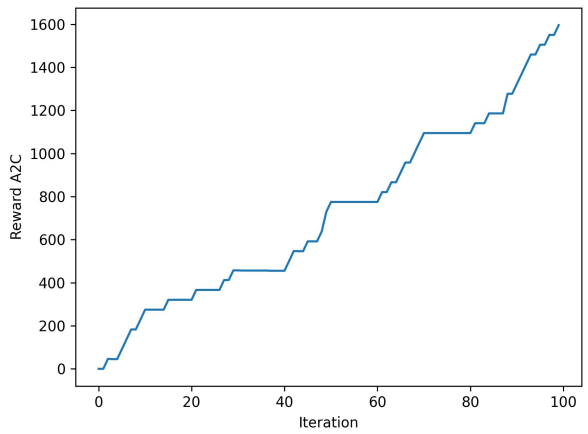
결과 3.



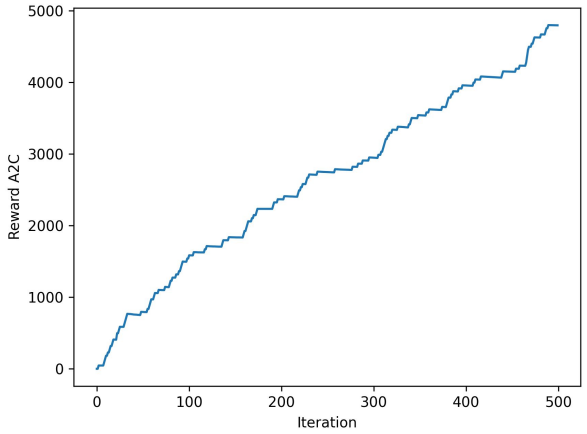
✔ Reward 에 대한 Plot 확인 결과 Reward가 증가하는 경향을 나타냄
✔ 강화학습이 올바르게 이루어졌다고 판단 가능

6. 실험 결과 – A2C

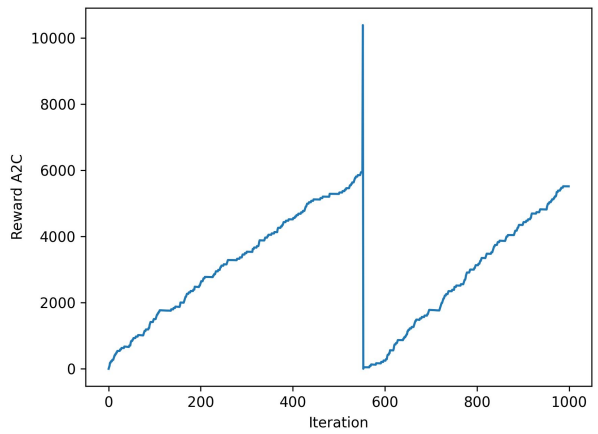
결과 1. range100



결과 2. range 500



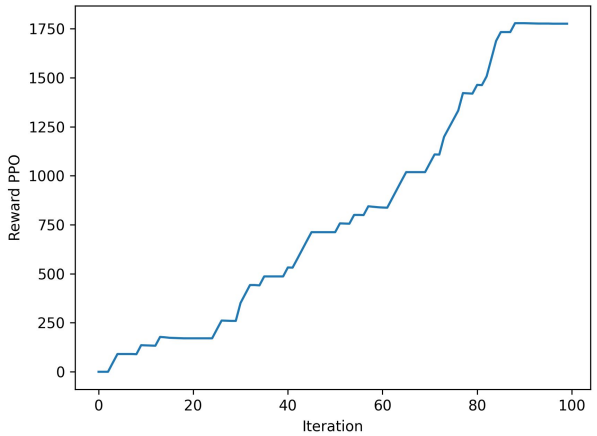
결과 3. range 1000



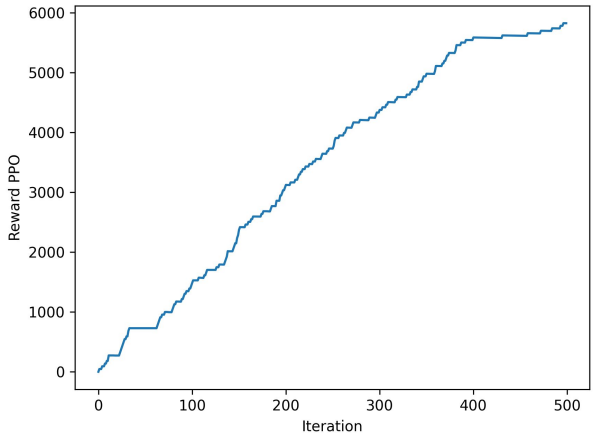
✔ Reward 에 대한 Plot 확인 결과, Reward가 증가하는 경향을 나타냄 - 정상학습이라 판단
✔ training 횟수 증가할수록 Reward가 한 번 튀고 다시 0으로 변경됨
✔ 정책 업데이트 반복 횟수가 너무 많은것으로 추정

6. 실험 결과 – PPO

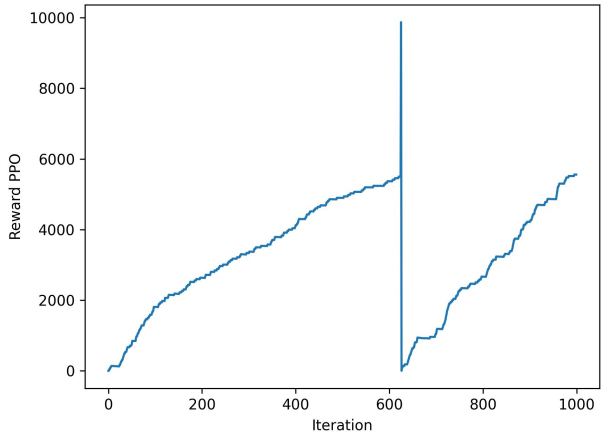
결과 1. range 100



결과 2. range 500



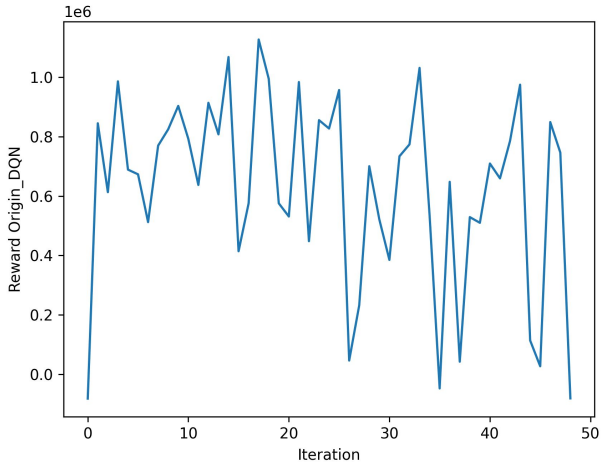
결과 3. range 1000



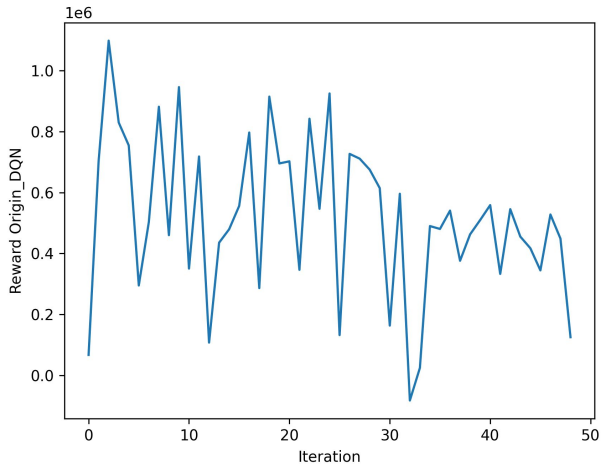
✔ Reward 에 대한 Plot 확인 결과, Reward가 증가하는 경향을 나타냄 - 정상학습이라 판단
✔ training 횟수 증가할수록 Reward가 한 번 튀고 다시 0으로 변경됨
✔ 정책 업데이트 반복 횟수가 너무 많은것으로 추정

6. 실험 결과 – 예제 16 기반 DQN

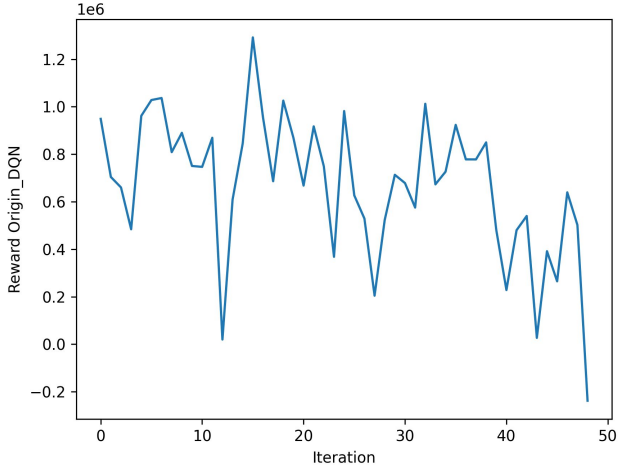
결과 1.



결과 2.



결과 3.



- ✔ Reward 에 대한 Plot 확인 결과, Reward 수치 변화 폭이 높음
- ✔ DQN 함수 구현하여 사용시 제대로 된 Train이 이루어지지 않은 듯함
- ✔ Multi 환경일시의 액션값을 제대로 정의하여 env.step에 넘기지 못한것으로 추정

6. 실험 결과 – 예제 16 기반 DQN

```python
def train(q, q_target, memory, optimizer):
    for i in range(10):
        s, a, r, s_prime, done_mask = memory.sample(batch_size)

        s = s.view(s.size(0), -1)  # Reshaping to [batch_size, -1]
        s_prime = s_prime.view(s_prime.size(0), -1)

        q_out = q(s)  # Q-values for all actions
        a = torch.tensor(a, dtype=torch.long).clone().detach().view(-1, 1)
        q_a = q_out.gather(1, a)
        # q_out과 q_a의 output tensor dimensor가 일치 하지 않느다는 오류 발생
        # 억지로 dimensor를 동일시 시켰으나 최초 step 함수로 넘길때의 action형태가 오류라고 판단
        # 멀티 프로세스의 이해도가 없어 생긴 문제로 이에따른 학습이 필요

        r = r.view(-1, 1)
        done_mask = done_mask.view(-1, 1)
        max_q_prime = q_target(s_prime).max(1)[0].unsqueeze(1)
        target = r + gamma * max_q_prime * done_mask
        loss = F.mse_loss(q_a, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

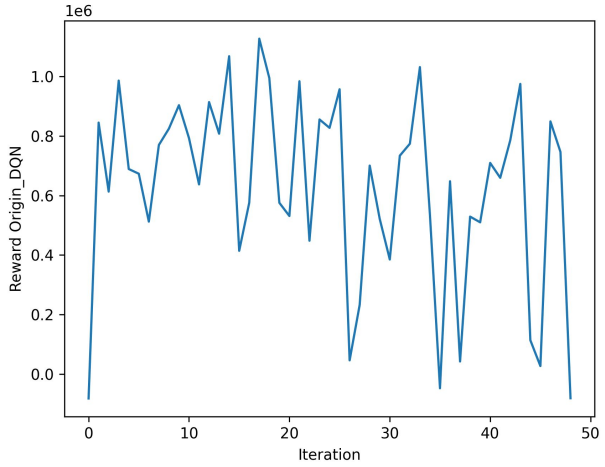
6. 실험 결과 – 예제 16 기반 DQN

```python
for n_epi in range(1000):
    # 다양한 엡실론 감쇠 일정 실험
    epsilon = max(0.01, 0.08 - 0.01 * (n_epi / 200))  # Adjusted epsilon decay
    s = env.reset()
    step_count = 0
    score = 0.0

    done = False

    while not done:
        a = []
        step_count += 1
        for i in range(num_lifts + num_robots):
            action = q_models[i].sample_action(torch.from_numpy(s).float(), epsilon)
            a.append(action)
        # 모든 에이전트의 액션을 하나의 정수로 결합
        combined_action = sum([a[i] * (4 ** i) for i in range(num_lifts + num_robots)])
        # 이 정수를 리스트로 변환하여 환경에 전달
        s_prime, r, done, _ = env.step([combined_action])
        # combined_action > Multi 환경일시의 액션값을 제대로 정의하여 env.step에 넘기지 못한것으로 추정
```

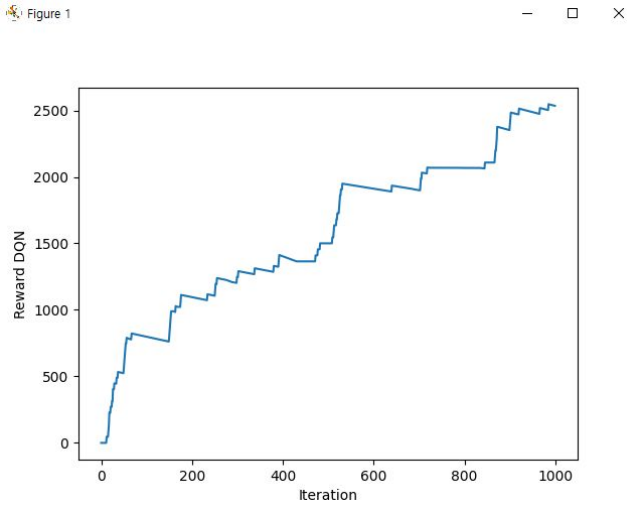
main() 함수

6. 실험 결과 – 예제 16 기반 DQN vs DQN

예제 16 기반 DQN



DQN

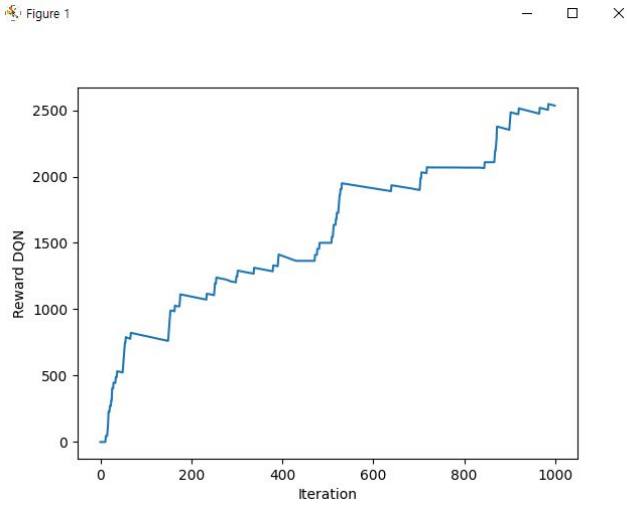


* 각 알고리즘 결과 Plot의 첫 번째 사진으로 비교

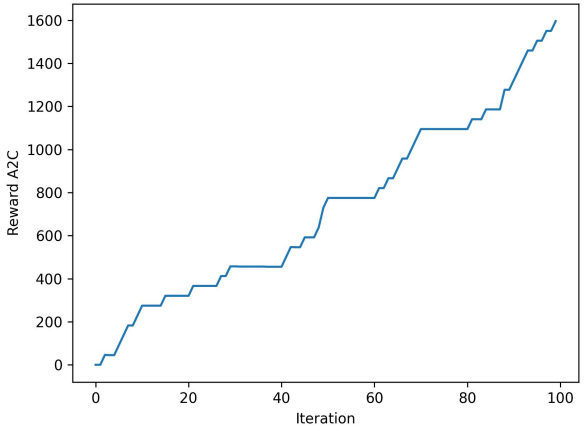
✔ Env 환경의 Reward, Panalty 체계와 상태(State)와 행동(Action) 간의 최적의 가치(Q-value) 학습 알고리즘의 중요성 를 알 수 있음

6. 실험 결과 – DQN vs A2C vs PPO

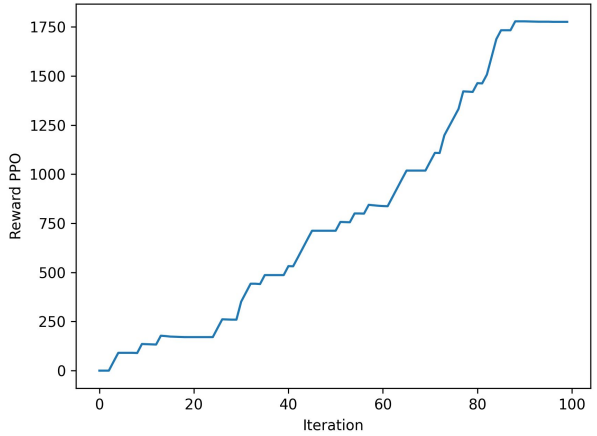
DQN



A2C



PPO



* 각 알고리즘 결과 Plot의 첫 번째 사진으로 비교

- ✔ 모든 알고리즘이 우상향하는 경향을 나타내고 있음
- ✔ A2C, PPO 알고리즘의 결과물로써 가치기반의 알고리즘인 DQN과 다르게 정책기반 알고리즘의 A2C,PPO는 학습률과 정책 업데이트 횟수 및 하이퍼파라미터에 민감하게 학습된다는 것을 알 수 있음

7. 토의 및 결론

- ✔ 첫 번째 목표인 예제 16번 기반 DQN과 DQN 함수 알고리즘의 비교를 진행한 결과, 모두 Reward가 높은 쪽으로 수행하는 것을 확인함
  Plot의 평균 최종 Reward만 가지고 판단했을 때 어떤 알고리즘의 효과가 좋다라고는 판단이 되지 않지만 더욱 세세하고 확률적인 학습으로는 미세조정에 큰 영향을 받는 A2C, PPO이 정확도 판단에 도움이 될 것이라 판단

- ✔ 강화학습의 경우 Agent가 Random한 움직임을 가져가기에 동일한 구성을 가지고도 실행할 때마다 다른 결과를 가져오는 것은 인지하고 있음
  다만 동일한 알고리즘이므로 수행 결과가 동일하게 나와야 한다는 전제로 판단했을 때 예제 16번 DQN 알고리즘을 본 프로젝트 내 환경에 맞추기 위해 어려움이 있었던 이유로 예제 16번 기반 DQN의 알고리즘이 잘못되었을 가능성을 배제하지 못함

7. 토의 및 결론

✔ 두 번째 목표인 DQN, A2C, PPO 알고리즘의 비교를 진행한 결과,
모든 강화학습 알고리즘은 Reward가 높은 쪽으로 수행하는 것을 확인함

✔ 같은 환경 구성으로 여러 알고리즘에 적용한 결과, Reward가 올바르게 나온 것처럼 보이는 알고리즘도 있으나 오류가 있어 보이는 결과도 나옴
- 환경 구성 및 Library 사용에 대한 추가적인 학습과 수정이 필요

✔ 환경을 Multi Agent 강화학습으로 구성했으나 중앙 제어를 사용하지 않고 독립적인 Agent와 연관성이 있는 Agent를 모두 채택함으로 상호 연관성 및 환경 구조에 대한 완성도가 다소 부족해 보임
다만 지게차에 대해서 연관성을 가지고 Reward를 주는 등 Agent간의 관계를 모델링 했다는 점을 봤을 때 Multi Agent 강화학습에 대한 이해와 학습을 위한 수행 가치가 있다고 판단

감사합니다.