11주차 강화학습 실체

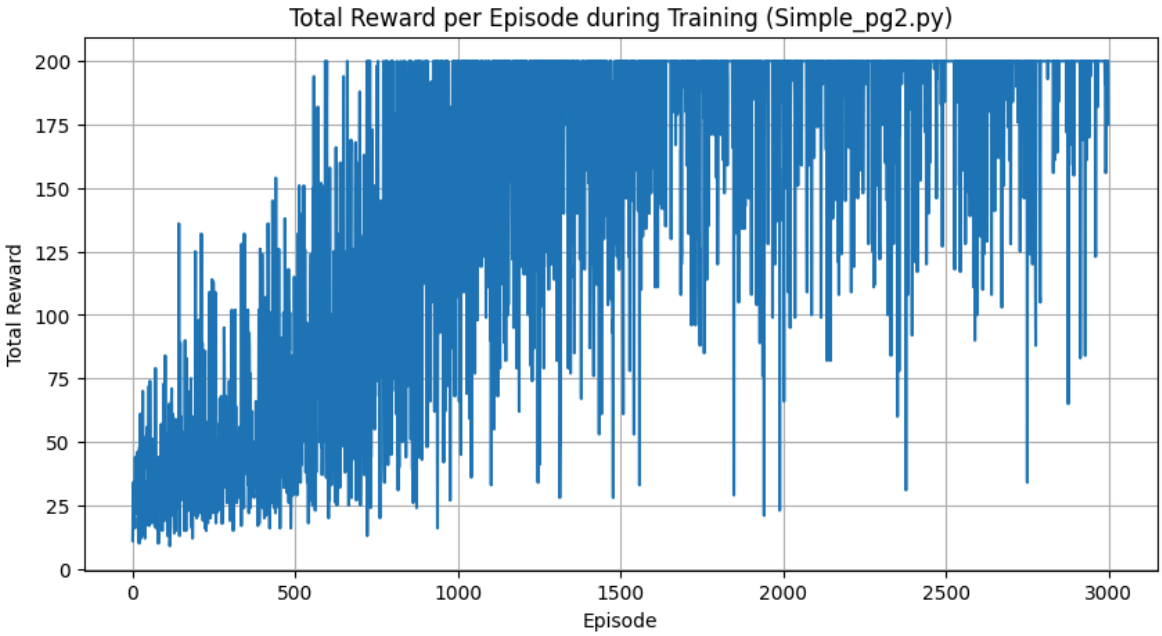
실습 1

In [1]:
```
import numpy as np
import gym
import matplotlib.pyplot as plt

from dezero import Model
from dezero import optimizers
import dezero.functions as F
import dezero.layers as L


class Policy(Model):
    def __init__(self, action_size):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(action_size)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.softmax(self.l2(x))
        return x


class Agent:
    def __init__(self):
        self.gamma = 0.98
        self.lr = 0.0002
        self.action_size = 2

        self.memory = []
        self.pi = Policy(self.action_size)
        self.optimizer = optimizers.Adam(self.lr)
        self.optimizer.setup(self.pi)

    def get_action(self, state):
        if not isinstance(state, np.ndarray):
             state = np.array(state)
        if state.ndim == 1:
            state = state[np.newaxis, :]

        probs = self.pi(state)
        probs_data = probs.data[0]
        action = np.random.choice(len(probs_data), p=probs_data)
        return action, probs[0, action]

    def add(self, reward, prob):
        data = (reward, prob)
        self.memory.append(data)

    def update(self):
        self.pi.cleargrads()

        G, loss = 0, 0
```

```
        returns = []
        for r, _ in reversed(self.memory):
            G = r + self.gamma * G
            returns.insert(0, G)

        for i, (reward, prob) in enumerate(self.memory):
            loss += -F.log(prob) * returns[i]

        loss.backward()
        self.optimizer.update()
        self.memory = []


episodes = 3000
env = gym.make('CartPole-v0', render_mode='rgb_array')
agent = Agent()
reward_history = []

for episode in range(episodes):
    reset_output = env.reset()
    if isinstance(reset_output, tuple):
        state = reset_output[0]
    else:
        state = reset_output
    done = False
    total_reward = 0

    while not done:
        action, prob = agent.get_action(state)
        step_output = env.step(action)
        if len(step_output) == 4:
             next_state, reward, done, info = step_output
             terminated, truncated = done, False
        else:
             next_state, reward, terminated, truncated, info = step_output
        done = terminated or truncated

        agent.add(reward, prob)
        state = next_state
        total_reward += reward

    agent.update()
    reward_history.append(total_reward)
    if episode % 100 == 0:
        print(f"episode : {episode}, total reward : {total_reward:.1f}")

env.close()

plt.figure(figsize=(10, 5))
plt.plot(reward_history)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("Total Reward per Episode during Training (Simple_pg2.py)")
plt.grid(True)
plt.show()

print("\nPlaying with the trained agent (Simple_pg2.py)...")
env2 = gym.make('CartPole-v0', render_mode='human')

reset_output_eval = env2.reset()
```

```
if isinstance(reset_output_eval, tuple):
    state = reset_output_eval[0]
else:
    state = reset_output_eval

done = False
total_reward_eval = 0

while not done:
    action, _ = agent.get_action(state)

    step_output_eval = env2.step(action)
    if len(step_output_eval) == 4:
        next_state, reward, done, info = step_output_eval
        terminated, truncated = done, False
    else:
        next_state, reward, terminated, truncated, info = step_output_eval
    done = terminated or truncated

    state = next_state
    total_reward_eval += reward
    env2.render()

print('Total Reward during evaluation (Simple_pg2.py):', total_reward_eval)
env2.close()
```

```
C:\Users\LOQ\anaconda3\envs\reinforcement_learning\lib\site-packages\gym\envs\registration.py:555: UserWarning: WARN: The environment CartPole-v0 is out of date. You should consider upgrading to version `v1`.
  logger.warn(
episode : 0, total reward : 11.0
episode : 100, total reward : 84.0
episode : 200, total reward : 21.0
episode : 300, total reward : 24.0
episode : 400, total reward : 24.0
episode : 500, total reward : 63.0
episode : 600, total reward : 120.0
episode : 700, total reward : 25.0
episode : 800, total reward : 75.0
episode : 900, total reward : 169.0
episode : 1000, total reward : 200.0
episode : 1100, total reward : 90.0
episode : 1200, total reward : 129.0
episode : 1300, total reward : 174.0
episode : 1400, total reward : 200.0
episode : 1500, total reward : 200.0
episode : 1600, total reward : 158.0
episode : 1700, total reward : 200.0
episode : 1800, total reward : 200.0
episode : 1900, total reward : 200.0
episode : 2000, total reward : 137.0
episode : 2100, total reward : 192.0
episode : 2200, total reward : 200.0
episode : 2300, total reward : 200.0
episode : 2400, total reward : 200.0
episode : 2500, total reward : 200.0
episode : 2600, total reward : 200.0
episode : 2700, total reward : 200.0
episode : 2800, total reward : 200.0
episode : 2900, total reward : 200.0
```




```
Playing with the trained agent (Simple_pg2.py)...
Total Reward during evaluation (Simple_pg2.py): 200.0
```

실습 2

In [2]:
```python
import numpy as np
import gym
import matplotlib.pyplot as plt

from dezero import Model
from dezero import optimizers
import dezero.functions as F
import dezero.layers as L


class Policy(Model):
    def __init__(self, action_size):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(action_size)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.softmax(self.l2(x))
        return x


class Agent:
    def __init__(self):
        self.gamma = 0.98
        self.lr = 0.0002
        self.action_size = 2

        self.memory = []
        self.pi = Policy(self.action_size)
        self.optimizer = optimizers.Adam(self.lr)
        self.optimizer.setup(self.pi)

    def get_action(self, state):
        if not isinstance(state, np.ndarray):
```

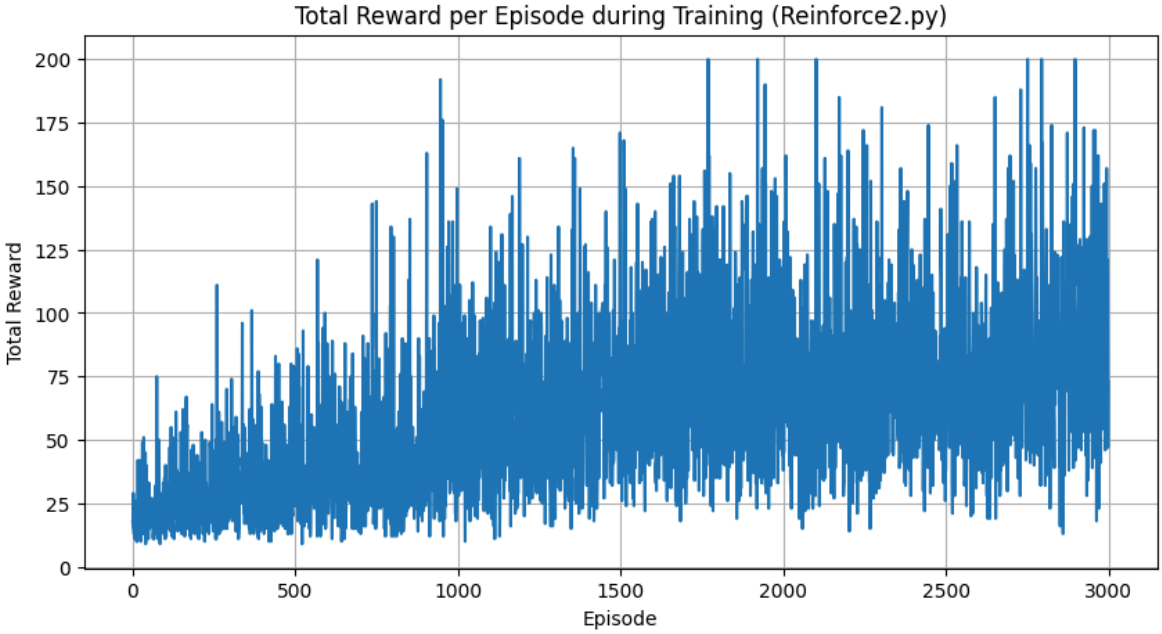
```
            state = np.array(state)
        if state.ndim == 1:
            state = state[np.newaxis, :]

        probs = self.pi(state)
        probs_data = probs.data[0]
        action = np.random.choice(len(probs_data), p=probs_data)
        return action, probs[0, action]

    def add(self, reward, prob):
        data = (reward, prob)
        self.memory.append(data)

    def update(self):
        self.pi.cleargrads()

        G, loss = 0, 0
        for reward, prob in reversed(self.memory):
            G = reward + self.gamma * G

        for reward, prob in self.memory:
            loss += -F.log(prob) * G

        loss.backward()
        self.optimizer.update()
        self.memory = []


episodes = 3000
env = gym.make('CartPole-v0', render_mode='rgb_array')
agent = Agent()
reward_history = []

for episode in range(episodes):
    reset_output = env.reset()
    if isinstance(reset_output, tuple):
        state = reset_output[0]
    else:
        state = reset_output
    done = False
    sum_reward = 0

    while not done:
        action, prob = agent.get_action(state)
        step_output = env.step(action)
        if len(step_output) == 5:
            next_state, reward, terminated, truncated, info = step_output
        elif len(step_output) == 4:
            next_state, reward, done_old, info = step_output
            terminated = done_old
            truncated = False
        else:
            raise ValueError("Unexpected output from env.step()")
        done = terminated or truncated

        agent.add(reward, prob)
        state = next_state
        sum_reward += reward

    agent.update()
```

```
    reward_history.append(sum_reward)
    if episode % 100 == 0:
        print(f"episode : {episode}, total reward : {sum_reward:.1f}")

env.close()

plt.figure(figsize=(10, 5))
plt.plot(reward_history)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("Total Reward per Episode during Training (Reinforce2.py)")
plt.grid(True)
plt.show()

print("\nPlaying with the trained agent (Reinforce2.py)...")
env2 = gym.make('CartPole-v0', render_mode='human')

reset_output_eval = env2.reset()
if isinstance(reset_output_eval, tuple):
    state = reset_output_eval[0]
else:
    state = reset_output_eval

done = False
total_reward_eval = 0

while not done:
    action, prob = agent.get_action(state)
    step_output_eval = env2.step(action)
    if len(step_output_eval) == 5:
        next_state, reward, terminated, truncated, info = step_output_eval
    elif len(step_output_eval) == 4:
        next_state, reward, done_old, info = step_output_eval
        terminated = done_old
        truncated = False
    else:
        raise ValueError("Unexpected output from env.step()")
    done = terminated or truncated

    # agent.add(reward, prob) # 평가 중에는 일반적으로 add 하지 않음

    state = next_state
    total_reward_eval += reward
    env2.render()

print('Total Reward during evaluation (Reinforce2.py):', total_reward_eval)
env2.close()
```

```
episode : 0, total reward : 18.0
```

```
C:\Users\LOQ\anaconda3\envs\reinforcement_learning\lib\site-packages\gym\envs\registration.py:555: UserWarning: WARN: The environment CartPole-v0 is out of date. You should consider upgrading to version `v1`.
  logger.warn(
```

```
episode : 100, total reward : 11.0
episode : 200, total reward : 12.0
episode : 300, total reward : 47.0
episode : 400, total reward : 39.0
episode : 500, total reward : 29.0
episode : 600, total reward : 22.0
episode : 700, total reward : 23.0
episode : 800, total reward : 91.0
episode : 900, total reward : 24.0
episode : 1000, total reward : 97.0
episode : 1100, total reward : 134.0
episode : 1200, total reward : 80.0
episode : 1300, total reward : 107.0
episode : 1400, total reward : 64.0
episode : 1500, total reward : 40.0
episode : 1600, total reward : 58.0
episode : 1700, total reward : 25.0
episode : 1800, total reward : 121.0
episode : 1900, total reward : 38.0
episode : 2000, total reward : 38.0
episode : 2100, total reward : 73.0
episode : 2200, total reward : 64.0
episode : 2300, total reward : 111.0
episode : 2400, total reward : 43.0
episode : 2500, total reward : 39.0
episode : 2600, total reward : 38.0
episode : 2700, total reward : 81.0
episode : 2800, total reward : 104.0
episode : 2900, total reward : 96.0
```




```
Playing with the trained agent (Reinforce2.py)...
Total Reward during evaluation (Reinforce2.py): 87.0
```

실습 3

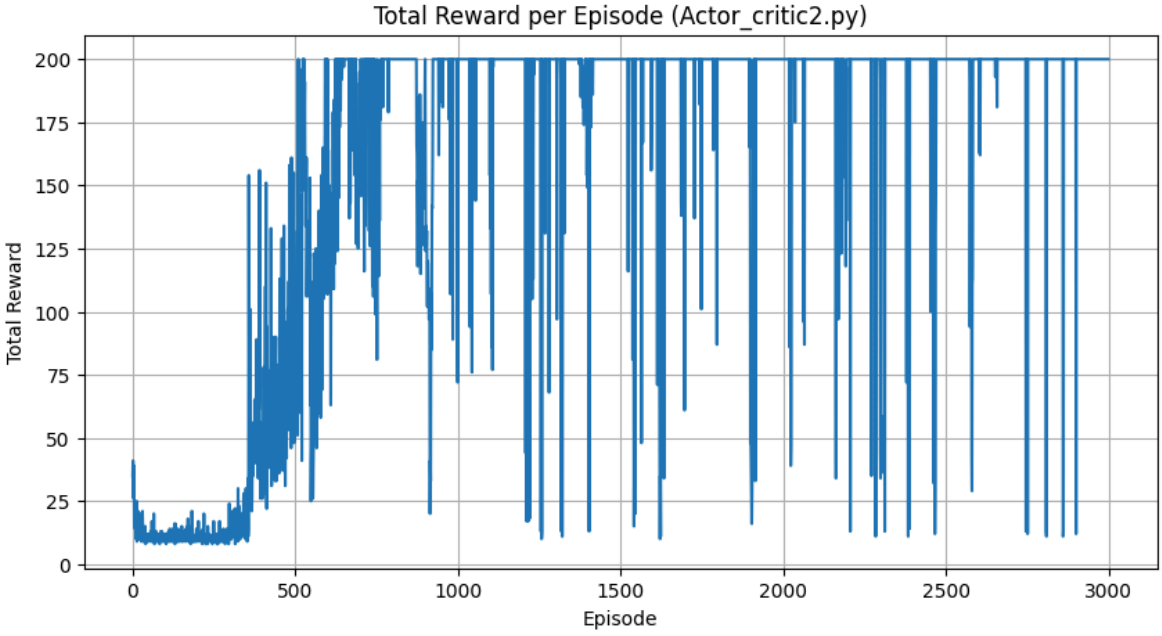
In [3]:
```python
import numpy as np
import gym
import matplotlib.pyplot as plt

from dezero import Model
from dezero import optimizers
```

```
import dezero.functions as F
import dezero.layers as L

class PolicyNet(Model):
    def __init__(self, action_size=2):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(action_size)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.softmax(self.l2(x))
        return x

class ValueNet(Model):
    def __init__(self):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(1)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = self.l2(x)
        return x

class Agent:
    def __init__(self):
        self.gamma = 0.98
        self.lr_pi = 0.0002
        self.lr_v = 0.0005
        self.action_size = 2

        self.pi = PolicyNet(self.action_size)
        self.v = ValueNet()
        self.optimizer_pi = optimizers.Adam(self.lr_pi).setup(self.pi)
        self.optimizer_v = optimizers.Adam(self.lr_v).setup(self.v)

    def get_action(self, state):
        if not isinstance(state, np.ndarray):
            state = np.array(state)
        if state.ndim == 1:
            state = state[np.newaxis, :]

        probs = self.pi(state)
        probs_data = probs.data[0]
        action = np.random.choice(len(probs_data), p=probs_data)
        return action, probs[0, action]

    def update(self, state, action_prob, reward, next_state, done):
        if not isinstance(state, np.ndarray): state = np.array(state)
        if state.ndim == 1: state = state[np.newaxis, :]

        if not isinstance(next_state, np.ndarray): next_state = np.array(next_st
        if next_state.ndim == 1: next_state = next_state[np.newaxis, :]

        target = reward + self.gamma * self.v(next_state) * (1 - done)
        target.unchain()

        v = self.v(state)
        loss_v = F.mean_squared_error(v, target)
```

```
        delta = target - v
        delta.unchain()

        loss_pi = -F.log(action_prob) * delta

        self.v.cleargrads()
        self.pi.cleargrads()
        loss_v.backward()
        loss_pi.backward()
        self.optimizer_v.update()
        self.optimizer_pi.update()

episodes = 3000
env = gym.make('CartPole-v0', render_mode='rgb_array')
agent = Agent()
reward_history = []

for episode in range(episodes):
    reset_output = env.reset()
    if isinstance(reset_output, tuple):
        state = reset_output[0]
    else:
        state = reset_output
    done = False
    total_reward = 0

    while not done:
        action, prob = agent.get_action(state)
        step_output = env.step(action)

        if len(step_output) == 5:
            next_state, reward, terminated, truncated, info = step_output
        elif len(step_output) == 4:
            next_state, reward, done_old, info = step_output
            terminated = done_old
            truncated = False
        else:
            raise ValueError("Unexpected output from env.step()")
        done = terminated or truncated

        agent.update(state, prob, reward, next_state, done)

        state = next_state
        total_reward += reward

    reward_history.append(total_reward)
    if episode % 100 == 0:
        print(f"episode :{episode}, total reward : {total_reward:.1f}")

env.close()

plt.figure(figsize=(10,5))
plt.plot(reward_history)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("Total Reward per Episode (Actor_critic2.py)")
plt.grid(True)
plt.show()
```

```
print("\nPlaying with the trained agent (Actor_critic2.py)...")
env2 = gym.make('CartPole-v0', render_mode='human')

reset_output_eval = env2.reset()
if isinstance(reset_output_eval, tuple):
    state = reset_output_eval[0]
else:
    state = reset_output_eval

done = False
total_reward_eval = 0

while not done:
    action, prob = agent.get_action(state)
    step_output_eval = env2.step(action)

    if len(step_output_eval) == 5:
        next_state, reward, terminated, truncated, info = step_output_eval
    elif len(step_output_eval) == 4:
        next_state, reward, done_old, info = step_output_eval
        terminated = done_old
        truncated = False
    else:
        raise ValueError("Unexpected output from env.step()")
    done = terminated or truncated

    agent.update(state, prob, reward, next_state, done)

    state = next_state
    total_reward_eval += reward
    env2.render()

print(f"Total Reward during evaluation (Actor_critic2.py): {total_reward_eval}")
env2.close()
```

```
episode :0, total reward : 35.0
episode :100, total reward : 9.0
episode :200, total reward : 13.0
episode :300, total reward : 19.0
episode :400, total reward : 37.0
episode :500, total reward : 58.0
episode :600, total reward : 107.0
episode :700, total reward : 146.0
episode :800, total reward : 200.0
episode :900, total reward : 155.0
episode :1000, total reward : 200.0
episode :1100, total reward : 133.0
episode :1200, total reward : 200.0
episode :1300, total reward : 200.0
episode :1400, total reward : 183.0
episode :1500, total reward : 200.0
episode :1600, total reward : 200.0
episode :1700, total reward : 200.0
episode :1800, total reward : 200.0
episode :1900, total reward : 51.0
episode :2000, total reward : 200.0
episode :2100, total reward : 200.0
episode :2200, total reward : 200.0
episode :2300, total reward : 59.0
episode :2400, total reward : 200.0
episode :2500, total reward : 200.0
episode :2600, total reward : 200.0
episode :2700, total reward : 200.0
episode :2800, total reward : 200.0
episode :2900, total reward : 12.0
```




```
Playing with the trained agent (Actor_critic2.py)...
Total Reward during evaluation (Actor_critic2.py): 200.0
```

In [ ]: