DEEP Q LEARNING İLE VARDİYALI ÇALIŞMADA ARAÇLARA PERSONEL ATANMASININ SAĞLANMASI

24435004012

Serkan Bulut

Bursa Teknik Üniversitesi

Bilgisayar Mühendisliği Tezli Yüksek Lisans

Derin Pekiştirmeli Öğrenme

Sunum İçeriği

- Senaryo – Oyunlaştırma
- Pekiştirmeli Öğrenme Bileşenleri
- Environment - Çevre
- Kod İncelemesi
- Sonuç

Senaryo – Oyunlaştırma

Genel Tanım: Bir kuruluşta çeşitli araçlara (iş makineleri) uygun olacak şekilde, farklı ehliyetlere sahip personelin vardiyalar halinde araçlara ataması yapılacaktır.

Bu problemin çözümü için Derin Pekiştirmeli Öğrenme (DPÖ) (Deep Reinforcement Learning) kullanılarak araçlara personel ataması yapılmıştır.

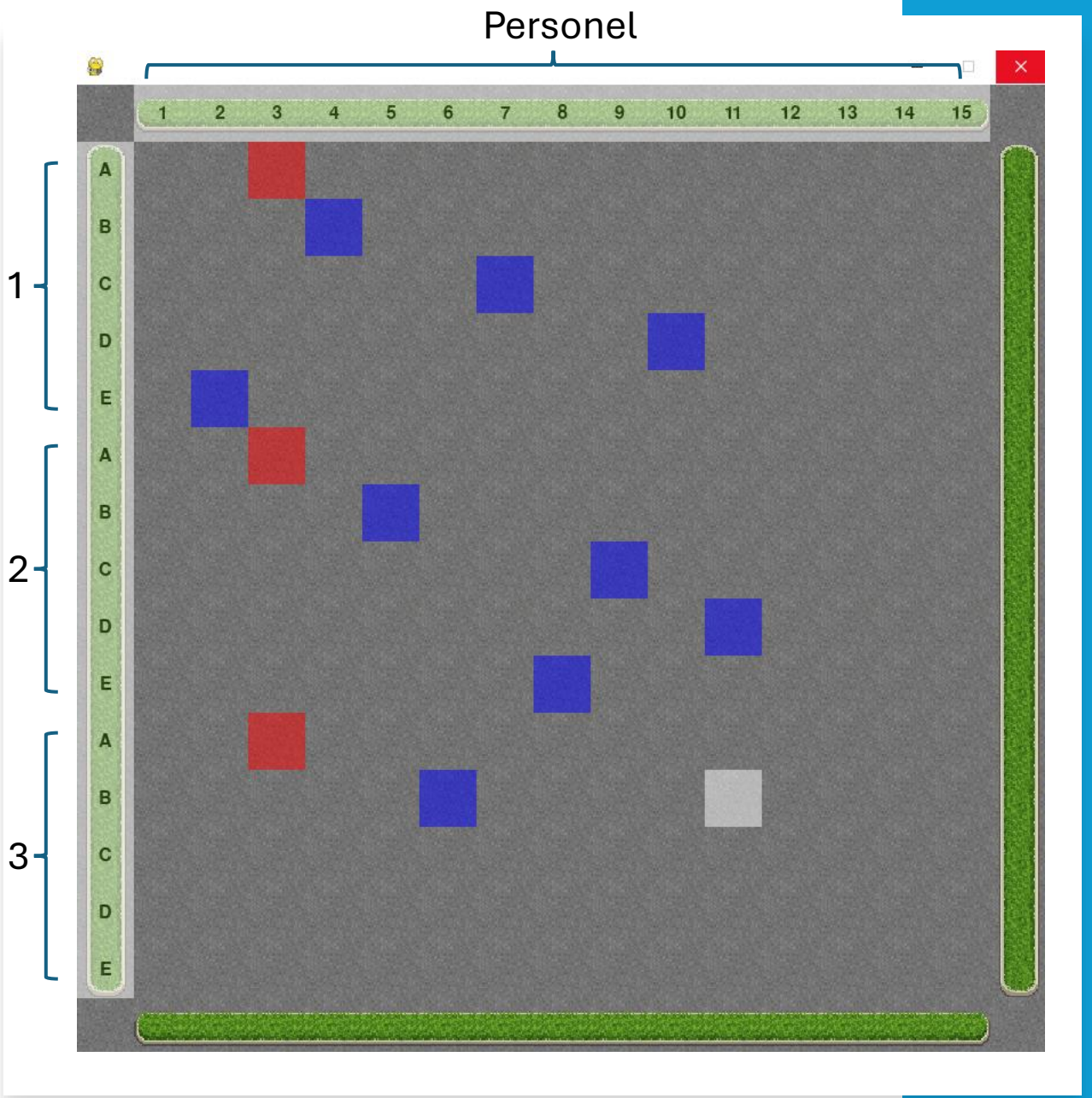
Araçlara ehliyetlerine uygun olarak personel atamasının yanı sıra mevcut vardiyada uygun ehliyetli personel olmaması durumunda bir önceki vardiyadan personel ataması yapılacaktır.

Senaryo – Oyunlaştırma

Problem, bu projede 5 farklı araç (iş makinesi), 3 vardiya ve 15 personel olacak şekilde düşünülmüştür.

Yanda yer alan şekildeki gibi soldan sağa doğru personel, yukarıdan aşağıya doğru vardiyalar için araçlar yer alacak şekilde bir oyun alanı oluşturulmuştur.

Çözüm Yaklaşımı: DQN (Deep Q Network) ile verilen araç, vardiya, personel bilgisi ile eğitilip atama kararı vermesi.



Pekiştirmeli Öğrenme Bileşenleri

State (Durum):

A aracı: Her bir vardiya için personel ataması yapılmış veya yapılmamış.

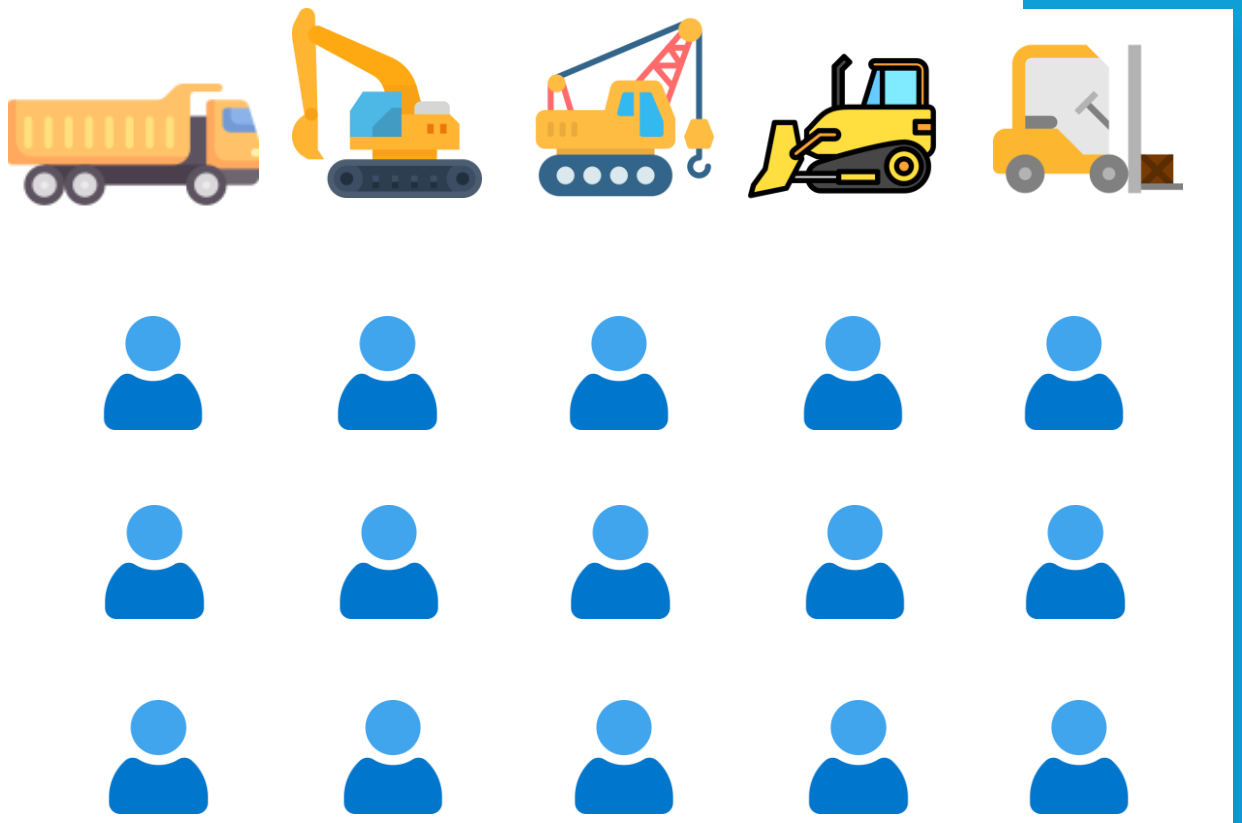
B aracı: Her bir vardiya için personel ataması yapılmış veya yapılmamış.

C aracı: Her bir vardiya için personel ataması yapılmış veya yapılmamış.

D aracı: Her bir vardiya için personel ataması yapılmış veya yapılmamış.

E aracı: Her bir vardiya için personel ataması yapılmış veya yapılmamış.

Bu proje için 5 araç X 3 Vardiya X 15 personel; 225 farklı state.



Pekiştirmeli Öğrenme Bileşenleri

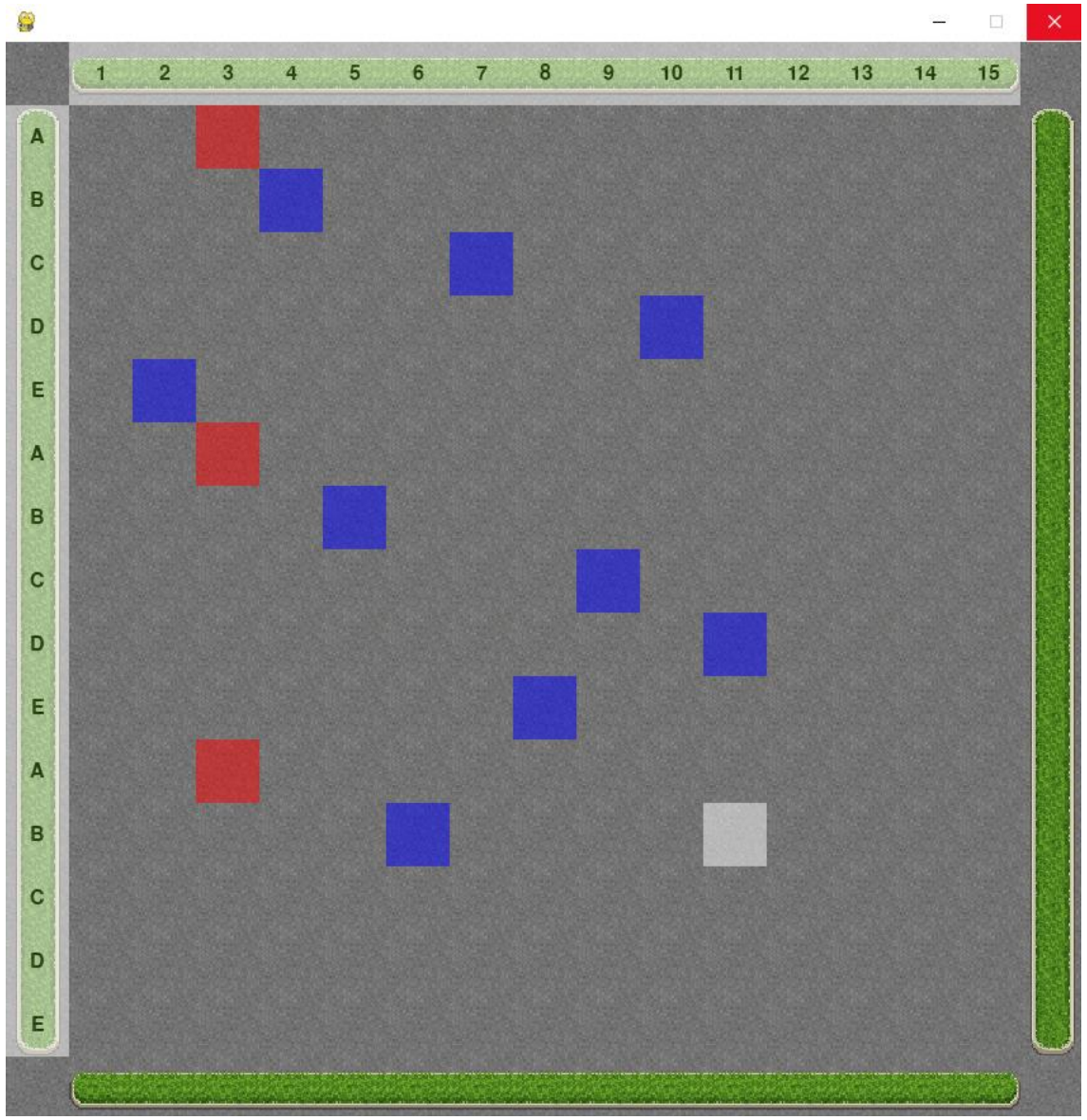
Actions (Eylemler):

Agent (ajan) için 3 farklı action bulunmaktadır.

Sağa Git: Oyun alanında sol üstten başlamak üzere sağa doğru ilerleyip yeni bir personele geçiş yapması eylemidir.

İşaretle: Ajanın bulunduğu kolondaki personeli bulunduğu satırdaki araç için atamasını yapması eylemidir.

Bir önceki vardiyadan atama yap: Ajanın bir önceki vardiyadaki personeli bulunduğu satırdaki araç için atamasını yapması eylemidir.



Pekiştirmeli Öğrenme Bileşenleri

Rewards (Ödüller):

Agent (ajan) için ödül ve cezalar aşağıdaki gibidir.

Gezinme cezası: Oyun alanında her sağa gidiş eylemi için -2, uygun işaretleme için -1

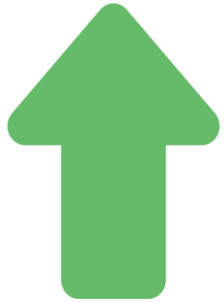
Aracı boş geçme cezası: Satır sonuna gelinmiş ve işaretleme yoksa yani araca atama yapılmamışsa -3

Uygun olmayan atama cezası: Ehliyetin uygun olmadığı işaretlemeler için -3

Dolu araca atama cezası: Önceden atama olan bir araca yeniden işaretleme yapıldığında -5, önceden ataması yapılmış araca önceki vardiyadan personel ataması yapılırsa -5

Personeli boş geçme cezası: Boşta atanabilecek personel varken önceki vardiyadan personel atanıyorsa -5

Tüm araçlara atama yapılması ödülü: Tüm araçlara atama yapılmışsa 50



Environment - Çevre

Bu projede vardiya-personel atama çizelgesi simüle edilmiş oluyor.

Vardiya: Araçların çalışması için gereken bir dönemi belirler. Bu projede bir gün için 3 vardiya olduğundan bir vardiya dönemi 8 saati temsil eder. Her bir vardiya içerisinde 5 araç ve bunları kullanacak personel olmalıdır.

Araçlar: Bu simülasyonda çalışacak birbirinden farklı 5 araç bulunmaktadır.

Ehliyet: Araçların kullanımı için gereken yetki belgesi. Projede bir liste halinde personelin sahip olduğu ehliyetler yer almaktadır.

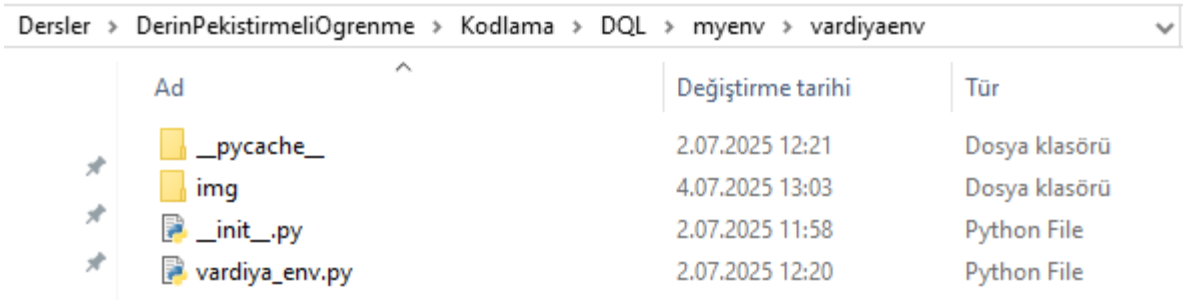
Personel: Farklı ehliyetlere sahip 15 adet personel bulunmaktadır.

Çizelge (Oyun alanı): Araçlara, personel ataması için kesişim karelerine işaret konulacak tablodur. Projede görsel olarak sunulmaktadır. Ayrıca araç atamaları liste halinde de sonuç olarak alınabilmektedir.

Kod İncelemesi Environment

Klasör Yapısı: Farklı problemler için geliştirilecek olan tüm çözümler myenv klasöründe yer alan environmentler ile oyunlaştırılmış oluyor. Yanda resimde gözüktüğü gibi bu proje için vardiya_env.py dosyası oluşturulmuştur.

Ehliyet-Araç-Personel-Vardiya: Envorinment dosyası içinde tüm ehliyet, personel (id ile), araç ve yazılacak ve bir önceki günün vardiya bilgileri yer almaktadır.




```
#             1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
ehliyetler = ['X','E','X','B','B','B','C','E','C','D','D','D','E','C','X']
personelid = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
self.araclar = araclar = ['A','B','C','D','E','A','B','C','D','E','A','B','C','D','E']
self.aracpersonel = aracpersonel = [0,0,0,0,0,
                                    0,0,0,0,0,
                                    0,0,0,0,0]
#Sonradan eklendi
self.aracpernerden = aracpernerden = [0,0,0,0,0,
                                      0,0,0,0,0,
                                      0,0,0,0,0]
self.aracpersoneloncekigun = aracpersoneloncekigun = [1,4,7,10,2,
                                                      3,5,9,11,8,
                                                      3,6,9,12,13]

self.aracpersoneldunsonvardiya = aracpersoneldunsonvardiya = aracpersoneloncekigun[-5:]

terminated = False

num_states = aracsayisi*3*personelsayisi
num_actions = 3    #SağaGit  #İşaretle  #BirÖncekiVardiya
num_rows = 3 #5
num_columns = 5
max_row = num_rows - 1
max_col = num_columns - 1
self.initial_state_distrib = np.zeros(num_states)
```

Kod İncelemesi
Model Eğitimi

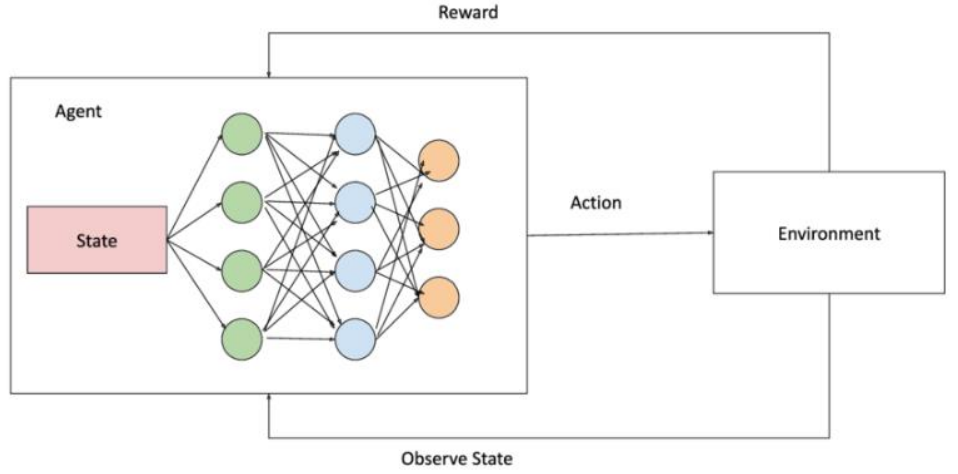
Eğitim Dosyası Yapısı: Eğitim dosyası (test_DQL.py) üç ana class yapısından oluşmaktadır.

DQN: DQN sınıfı, Deep Q-Network (DQN) algoritmasında kullanılan sinir ağı modelini tanımlar. nn.Module, PyTorch'ta tüm modellerin türetildiği temel sınıftır. Bu sınıftan bir model türetiliyor.

Bu sınıfın temel görevi verilen bir state (durum) karşılık her aksiyonun ne kadar iyi (Q-değeri) olduğunu tahmin etmek.

Alttaki şemada görüldüğü gibi her state sinir ağında değerlendirilerek en uygun action elde edilir.

```python
class DQN(nn.Module):
    def __init__(self, in_states, h1_nodes, out_actions):
        super().__init__()

        #in_states: Giriş boyutu. Yani bir state'in kaç boyutlu olduğunu belirtir.
        #h1_nodes: İlk gizli katmandaki nöron sayısı (örneğin 64 veya 128 gibi).
        #out_actions: Ortamda yapılabilecek toplam aksiyon sayısı. (Her bir aksiyon için bir Q-değeri üretilecek.)

        # Network katmanları tanımlanıyor.
        # Giriş olarak in_states boyutlu bir vektör alır, h1_nodes boyutlu bir çıkış üretir.
        self.fc1 = nn.Linear(in_states, h1_nodes)   # İlk fully connected layer

        # Gizli katmandan gelen verileri alıp, her bir aksiyon için bir Q-değeri üretir.
        self.out = nn.Linear(h1_nodes, out_actions) # output layer w

    # x: Ağa verilen giriş state vektörü
    def forward(self, x):
        # Girdi fc1 katmanından geçer, ardından ReLU aktivasyon fonksiyonu uygulanır.
        # Negatif değerleri sıfırlar.
        x = F.relu(self.fc1(x)) # Apply rectified linear unit (ReLU) activation

        # Gizli katmandan çıkan veri out katmanına girer ve her aksiyon için Q-değeri üretilir.
        x = self.out(x)         # Calculate output
        return x
```



Kod İncelemesi
Model Eğitimi

ReplayMemory: Ajanın öğrendiği geçmiş tecrübeleri (transition: state, action, reward, next_state, done) saklanır.

Belleğin kapasitesi sınırlıdır (maxlen), dolarsa ilk giren ilk çıkar (FIFO) mantığıyla eski veriler silinir.

Eğitim sırasında bu bellekteki verilerden rastgele örnekler çekilir, bu da öğrenmeyi daha dengeli ve kararlı hale getirir.

Ajan önceki adımları unutmaz. Eğitim sırasında sürekli yeni veriler yerine eski ve çeşitli tecrübelerden öğrenir.

```python
class ReplayMemory():
    def __init__(self, maxlen):
        self.memory = deque([], maxlen=maxlen)
        #maxlen belleğin kapasitesi kaç kayıt tutulacak
        #kapasite dolarsa en eski kayıt silinir yenisi yazılır (FIFO)

    #ajanın her step sonucu (transition) belleğe eklenir.
    #transition (state,action,reward,next_state,done) değerlerini içerir.
    def append(self, transition):
        self.memory.append(transition)

    # Hafızadan rastgele sample_size kadar örnek çeker.
    def sample(self, sample_size):
        return random.sample(self.memory, sample_size)

    # Bellekte şu an kaç deneyim olduğunu verir.
    def __len__(self):
        return len(self.memory)
```

Kod İncelemesi
Model Eğitimi

VardiyaDQL: Bu class bir DQN (Deep Q-Network) ajanı oluşturur. Ajanı eğitir, deneyimleri saklar ve sinir ağı ile aksiyon seçimini öğrenir. Eğitilen modeli dosyaya kaydeder ve istenirse test edebilir.

learning_rate_a = 0.001

discount_factor_g = 0.9

network_sync_rate = 10

replay_memory_size = 1000

mini_batch_size = 32

learning_rate_a: Öğrenme hızı (α)

discount_factor_g: Gelecekteki ödülün bugünkü değeri (γ)

network_sync_rate: Policy ve Target network'ün kaç adımda bir eşitleneceği

replay_memory_size: Hafızada kaç adet deneyim tutulacağı

mini_batch_size: Aynı anda kaç deneyimle eğitim yapılacağı

```python
class VardiyaDQL():
    # Hyperparameters (adjustable)
    learning_rate_a = 0.001         # learning rate (alpha)
    discount_factor_g = 0.9         # discount rate (gamma)
    network_sync_rate = 10          # policy and target network kaç adımda bir güncellenecek. Her güncellemede policy target'a kopyalanır.
    replay_memory_size = 1000       # size of replay memory
    mini_batch_size = 32            # Replay memory'den eğitim için rastgele seçilecek örnek sayısı.

    # Neural Network
    loss_fn = nn.MSELoss()          # NN Loss function. MSE=Mean Squared Error can be swapped to something else.
    optimizer = None                # NN Optimizer. Sonra tanımlanacak. Ağırlıkların güncellenmesini sağlayacak.

    ACTIONS = ['S','I','O']     # for printing 0,1,2 => SagaGit,Işaretle,OncekiVardiya

    # Train the Vardiya environment

    #Ortamı oluşturur
    #Ajanı eğitir (epsilon-greedy ile aksiyon alarak)
    #Replay memory'e deneyim ekler
    #Mini-batch'lerle ağı optimize eder
    #Target ağı periyodik olarak günceller
    #Eğitim sonrası sonuçları kaydeder ve görselleştirir
```

Sonuç - Değerlendirme

Bu proje için aynı environment ile hem Q Learning hem de Deep Q Learning ile eğitimler yapılmıştır.

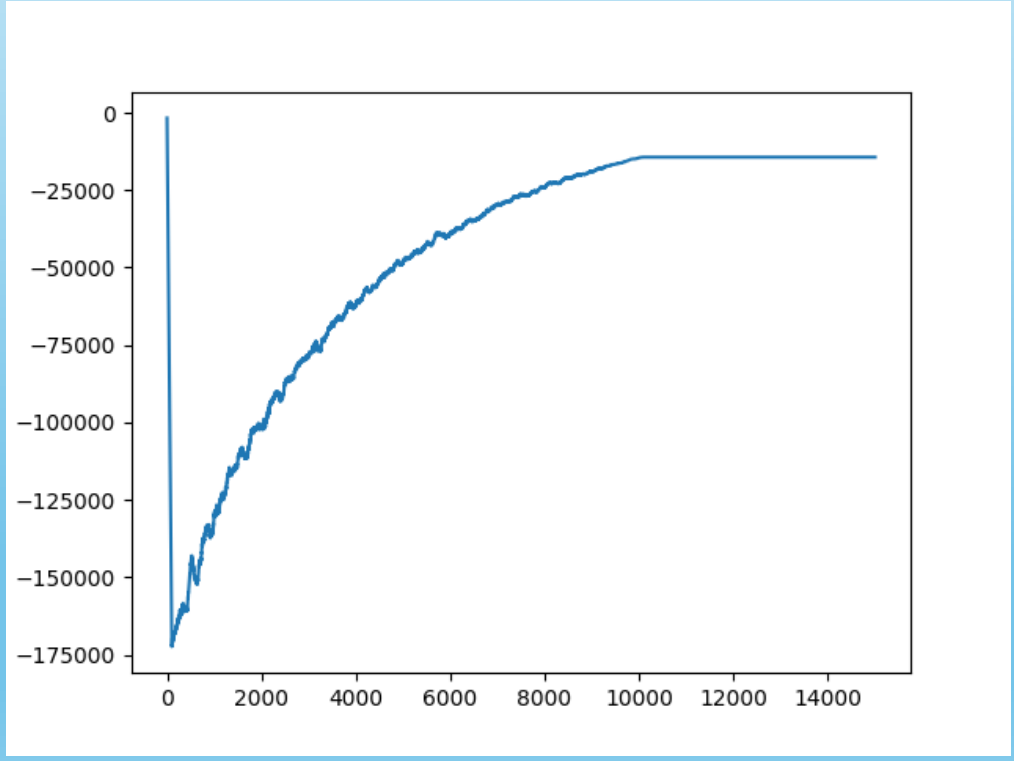
Q Learning Parametreleri:

learning_rate_a = 0.9

discount_factor_g = 0.9

epsilon = 1

epsilon_decay_rate = 0.0001



Sonuç - Değerlendirme

Deep Q Learning Parametreleri:

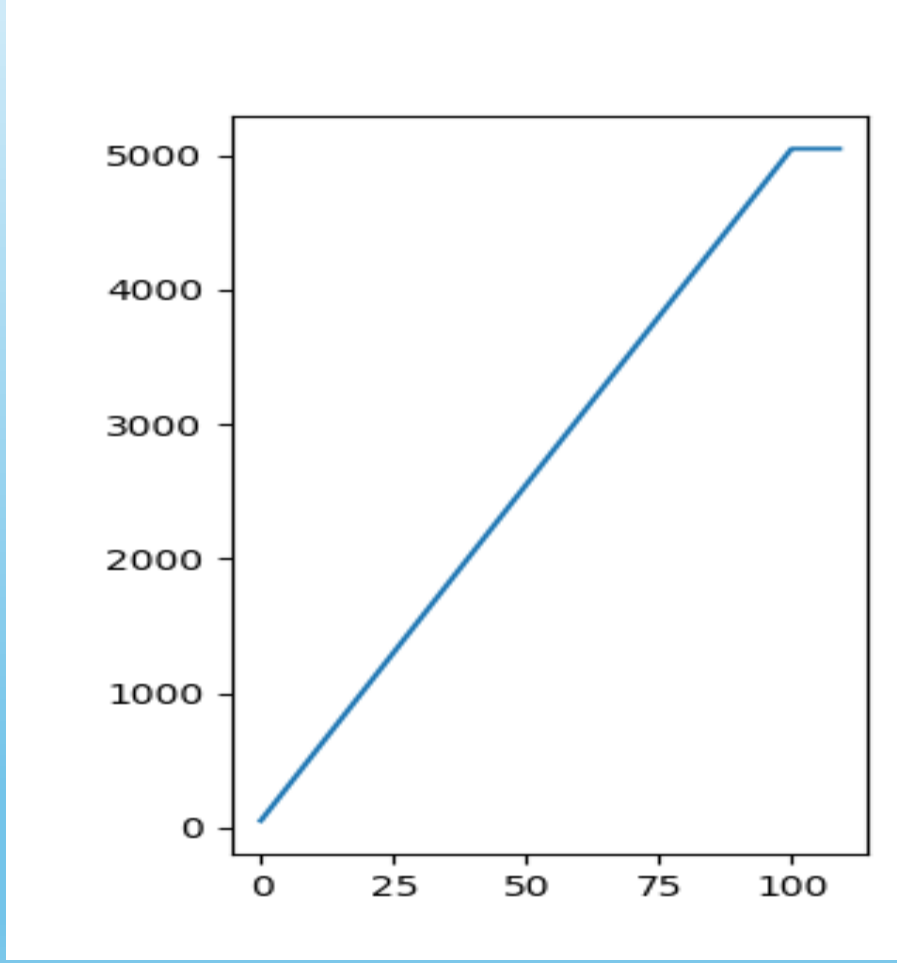
learning_rate_a = 0.001

discount_factor_g = 0.9

network_sync_rate = 10

replay_memory_size = 1000

mini_batch_size = 32



Sonuç - Değerlendirme

Aynı problemin çözümü için DQL 100 epizod ile eğitimini tamamlarken, QL 10.000 epizod ile eğitimini tamamlamıştır.

QL için 15.000 epizod eğitim 76 saniye sürmüştür.

DQL için 500 epizod eğitim 31 saniye sürmüştür.

DQL, daha az sayıda epizod ile eğitimini tamamlayabiliyor. Ancak her bir epizod için eğitim QL'ye göre yavaş olmaktadır.

Teşekkür ederim.

Serkan Bulut

Bilgisayar Mühendisi

Bursa Teknik Üniversitesi

2025