Q-Learning ile Yol Planlaması

Burak İLHAN - Mert TOPRAK
180202058 - 180202074
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
180202058@kocaeli.edu.tr - 180202074@kocaeli.edu.tr

***Özet*–Q-learning algoritması kullanarak verilen noktadan hedef noktaya gidilebilen en kısa yolu bulmak**

Anahtar Kelimeler – Q-learning, kazanç,yol

I. PROBLEM TANIMI

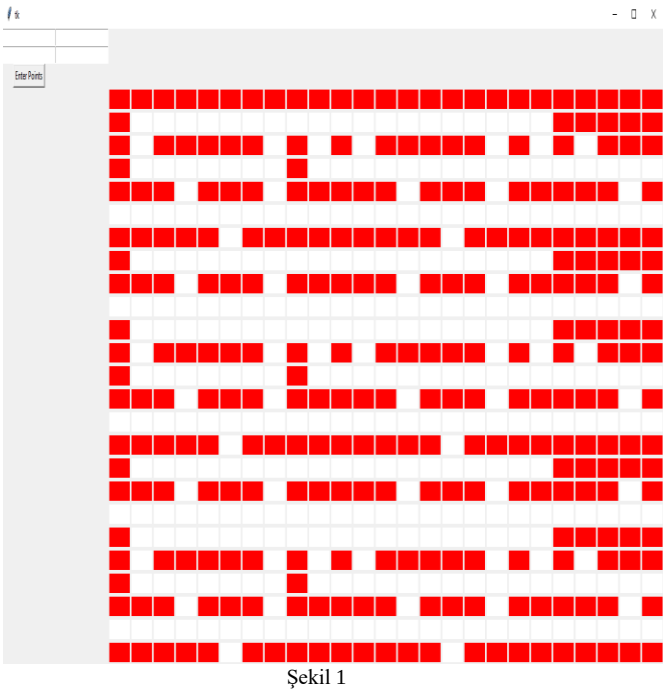
Uygulamamız 25*25'lik bir matris üzerinde belirtilen başlangıç noktasından hedef noktasına, engellere uğramadan giden en kısa yolu bulmayı amaçlamaktadır.

II. YAPILAN ARAŞTIRMALAR

Projemizi gerçekleştirmek için bir arayüz gerektiği için öncelikle bu konuda araştırma yaptık ve "Tkinter" isimli Python kütüphanesinin bizim için en uygun olduğuna karar verdik. Bu kütüphane ve q-learning algoritması ile ilgili birkaç tutorial videosu ve websiteden faydalandık ve bunları nasıl kullanmamız gerektiğini öğrendik ve işlemlerimizi gerçekleştirdik. Projemizi VS Code kod editörü kullanarak Python programlama dili ile gerçekleştirdik.

III. TASARIM

Uygulamamızın arayüzü aşağıdaki gibidir.



Şekil 1

IV. GENEL YAPI

Uygulamamız Python programlama dili kullanılarak VS Code kod editörü üzerinde gerçekleştirilmiştir. Kodumuz 244 satır ve 6 fonksiyondan oluşmaktadır. Uygulama geliştirme sırasında GUI için Tkinter, matrislerle daha kolay çalışabilmek için Numpy olmak üzere toplamda 2 kütüphaneden yararlanılmış. Arayüz ile başlayacak olursak bizi 4 adet veri girişi yapabileceğimiz alan, verileri yollayacağımız bir buton ve 25*25'lik bir matris karşılayacaktır. İlk iki alana seçmek istediğiniz başlangıç noktasının x ve y değerlerini, altındaki iki alana ise varmak istediğiniz hedef noktasının x ve y değerlerini yazmanız beklenmektedir. Daha sonra "Enter points" butonuna basarak verilerinizi yollayabilirsiniz. Noktaları yolladıktan sonra gerekli işlemler yapılacak ve 25*25'lik matris üzerinde başlangıç noktası, hedef noktası ve en kısa yol, renklendirilerek gösterilecektir. Şimdi bunların arka planda nasıl gerçekleştiğine değinmek istiyorum.

Kodumuzda myClick isimli bir fonksiyon bulunmaktadır ve bu fonksiyon, kullanıcı "Enter points" butonuna bastığında devreye girecektir. Bu fonksiyonda ise kullanıcıdan verileri alma ve bu verileri kullanarak en kısa yolu bulma işlemi gerçekleştirilmektedir. Kullanıcıdan verileri aldıktan sonra algoritma için gerekli ortamı hazırlıyoruz. Ortam, durumlardan, eylemlerden ve kazançlardan oluşur. Durumlar ve eylemler, Q-Learning AI ajanı için girdiler iken, olası eylemler AI ajanının çıktılarıdır. Ortamdaki durumlar, olası tüm durumlardır. Bu konumlardan bazıları seyahat edilemeyecek engel konumlar(yeşil) ve seyahat edilebilecek(beyaz) konumlardır.

```
def myClick():
    first_row = int(e.get())
    first_column = int(e1.get())
    second_row = int(e2.get())
    second_column = int(e3.get())
    boxes = Label(root,text="",bg="green",width=6,height=1,borderwidth=2)
    boxes.grid(row=second_row+3,column=second_column+2,padx=2,pady=2)
    boxes = Label(root,text="",bg="yellow",width=6,height=1,borderwidth=2)
    boxes.grid(row=first_row+3,column=first_column+2,padx=2,pady=2)
    environment_rows = 25
    environment_columns = 25

    q_values = np.zeros((environment_rows, environment_columns, 4))

    actions = ['up', 'right', 'down', 'left']

    rewards = np.full((environment_rows, environment_columns), -100.)

    aisles = {}
    for j in range(1,18,9):
        aisles[j] = [i for i in range(1, 20)]
        aisles[j+1] = [1, 7, 9,11,17,19,21]
        aisles[j+2] = [i for i in range(1, 25)]
        aisles[j+2].remove(8)
        aisles[j+3] = [3, 7,13,17,23]
        aisles[j+4] = [i for i in range(25)]
        aisles[j+5] = [5,15]
        aisles[j+6] = [i for i in range(1, 20)]
        aisles[j+7] = [3, 7,13,17,23]
        aisles[j+8] = [i for i in range(25)]
    aisles[19] = [i for i in range(1, 20)]
    aisles[20] = [1, 7, 9,11,17,19,21]
    aisles[21] = [i for i in range(1, 25)]
    aisles[21].remove(8)
    aisles[22] = [3,7,13,17,23]
    aisles[23] = [i for i in range(25)]
    aisles[24] = [5,15]
```

Şekil 2

Her durum ve eylem çifti için geçerli Q değerlerini tutmak için ise bir 3 boyutlu dizi oluşturuyoruz. Bu dizi 25 satır ve 25 sütun (ortamın şekline uyması için) ve üçüncü bir "eylem" boyutu içerir. "Eylem" boyutu, olası her eylem için Q değerlerini izlememize olanak tanıyan 4 katmandan oluşur. AI aracısının kullanabileceği eylemler, robotu dört yönden birinde hareket ettirmektir.(yukarı,aşağı,sol,sağ).AI öznesi, bu eylemleri kullanarak siyah noktalara gitmekten kaçınmayı öğrenmelidir. Her (durum, eylem) çiftinin değeri 0 olarak ilklendirilir. Sonrasında ise kazanç kısmı gelmektedir. Ortamın tanımlamamız gereken son bileşeni kazançlardır. Yapay zeka öznesinin öğrenmesine yardımcı olmak için ortamdaki her duruma (konuma) bir kazanç değeri atanır. Özne herhangi bir beyaz kareden başlayabilir, ancak amacı her zaman aynıdır: Toplam kazancı en üst düzeye çıkarmak. Hedef dışındaki tüm durumlar için negatif kazançlar (yani cezalar) kullanılır. Bu, AI'yı cezalarını en aza indirerek hedefe giden en kısa yolu belirlemeye teşvik eder. Kümülatif kazancını en üst düzeye çıkarmak için (kümülatif cezalarını en aza indirerek), AI öznesinin hedef alanı (yeşil kare) ile seyahat etmesine izin verilen diğer tüm yerler (beyaz kareler) arasındaki en kısa yolları bulması gerekecektir. Öznenin ayrıca herhangi bir engel konumuna (siyah kareler) çarpmamayı öğrenmesi gerekecektir. Bu neticede matris üzerinde gerekli engelleri koyarak bu engellere ve seyahat edilebilecek noktalara,yani beyaz noktalara, gerekli kazanç değeri atamaları yapıyoruz.

```
for row_index in range(1, 25):
  for column_index in aisles[row_index]:
    rewards[row_index, column_index] = -1.

rewards[second_row, second_column] = 100.
def is_terminal_state(current_row_index, current_column_index):
  if rewards[current_row_index, current_column_index] == -1.:
    return False
  else:
    return True

def get_starting_location():
  current_row_index = np.random.randint(environment_rows)
  current_column_index = np.random.randint(environment_columns)
  while is_terminal_state(current_row_index, current_column_index):
    current_row_index = np.random.randint(environment_rows)
    current_column_index = np.random.randint(environment_columns)
  return current_row_index, current_column_index

def get_next_action(current_row_index, current_column_index, epsilon):
  if np.random.random() < epsilon:
    return np.argmax(q_values[current_row_index, current_column_index])
  else:
    return np.random.randint(4)

def get_next_location(current_row_index, current_column_index, action_index):
  new_row_index = current_row_index
  new_column_index = current_column_index
  if actions[action_index] == 'up' and current_row_index > 0:
    new_row_index -= 1
  elif actions[action_index] == 'right' and current_column_index < environment_columns - 1:
    new_column_index += 1
  elif actions[action_index] == 'down' and current_row_index < environment_rows - 1:
    new_row_index += 1
  elif actions[action_index] == 'left' and current_column_index > 0:
    new_column_index -= 1
  return new_row_index, new_column_index
```

Şekil 3

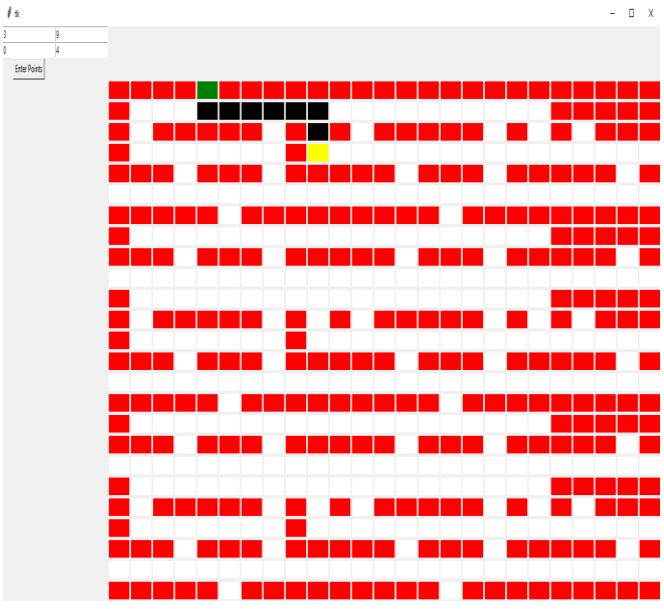
Sonrasındaki amacımız ise, yapay zeka öznemizin bir Q-learning modeli uygulayarak ortamı hakkında bilgi edinmesidir. Öğrenme süreci şu adımları izleyecektir:
1) Öznenin yeni bölüme başlaması için rastgele, terminal olmayan bir durum (beyaz kare) seçiyoruz.
2) Mevcut durum için bir eylem seçiyoruz. (yukarı, sağa, aşağı veya sola). Eylemler, epsilon greedy algoritması kullanılarak seçilecektir. Bu algoritma genellikle AI öznesi için en umut verici eylemi seçecektir, ancak bazen ajanı ortamı keşfetmeye teşvik etmek için daha az umut verici bir seçenek seçecektir.
3) Seçilen eylemi gerçekleştiriyoruz ve bir sonraki duruma geçiyoruz. (yani sonraki konuma gidiyoruz)
4) Yeni duruma geçtiğimiz için kazancı alıyoruz ve geçici farkı hesaplıyoruz.
5) Önceki durum ve eylem çifti için Q değerini güncelliyoruz.
6) Yeni (mevcut) durum bir terminal durumsa(yeşil kare), # 1'e gidiyoruz. Aksi takdirde, # 2'ye gidiyoruz.

```
def get_shortest_path(start_row_index, start_column_index):
  if is_terminal_state(start_row_index, start_column_index):
    return []
  else:
    current_row_index, current_column_index = start_row_index, start_column_index
    shortest_path = []
    shortest_path.append([current_row_index, current_column_index])

    while not is_terminal_state(current_row_index, current_column_index):
      action_index = get_next_action(current_row_index, current_column_index, 1.)
      current_row_index, current_column_index = get_next_location(current_row_index, current_column_index, action_index)
      shortest_path.append([current_row_index, current_column_index])
    return shortest_path

epsilon = 0.9
discount_factor = 0.9
learning_rate = 0.9

for episode in range(1000):
  row_index, column_index = get_starting_location()
  while not is_terminal_state(row_index, column_index):
    action_index = get_next_action(row_index, column_index, epsilon)
    old_row_index, old_column_index = row_index, column_index
    row_index, column_index = get_next_location(row_index, column_index, action_index)
    reward = rewards[row_index, column_index]
    old_q_value = q_values[old_row_index, old_column_index, action_index]
    temporal_difference = reward + (discount_factor * np.max(q_values[row_index, column_index])) - old_q_value
    new_q_value = old_q_value + (learning_rate * temporal_difference)
    q_values[old_row_index, old_column_index, action_index] = new_q_value

path = get_shortest_path(first_row, first_column)
print(path)
length = len(path) - 1
for i in range(1,length):
  a = path[i][0]
  b = path[i][1]
  boxes = Label(root,text="",bg="black",width=6,height=1,borderwidth=2)
  boxes.grid(row=a+3,column=b+2,padx=2,pady=2)
```

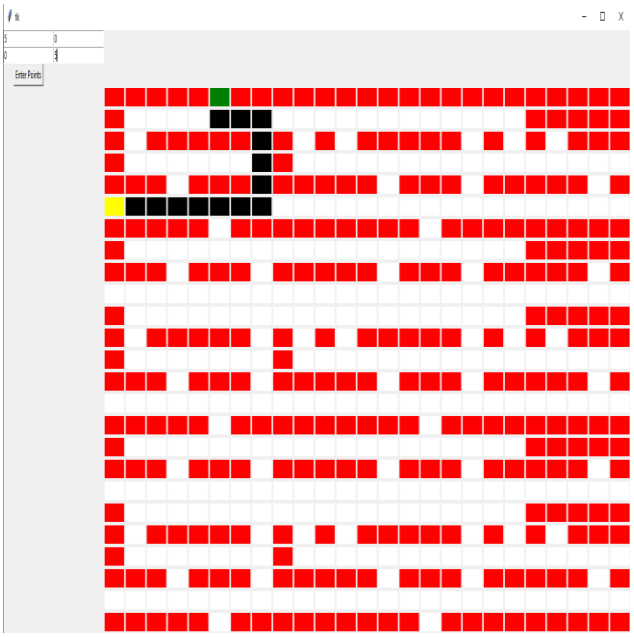
Şekil 4

Tüm bu süreç 1000 bölümde tekrarlanacaktır. Bu, AI öznesine, yeşil kare alanı ile seyahat etmesine izin verilen diğer tüm konumlar arasındaki en kısa yolları öğrenmesi için yeterli fırsat sağlarken, eşzamanlı olarak herhangi bir engel konumuna çarpmaktan kaçınacaktır.

Artık yapay zeka öznesi tam olarak eğitilmiştir. Buna göre, seyahate izin verilen herhangi bir konum ile hedef alanı arasındaki en kısa yolu get_shortest_path fonksiyonunu çağırarak öğrenebiliriz. Bu fonksiyon parametre olarak başlangıç noktalarının x ve y değerlerini almaktadır ve en kısa yolu bir dizi olarak döndürmektedir.

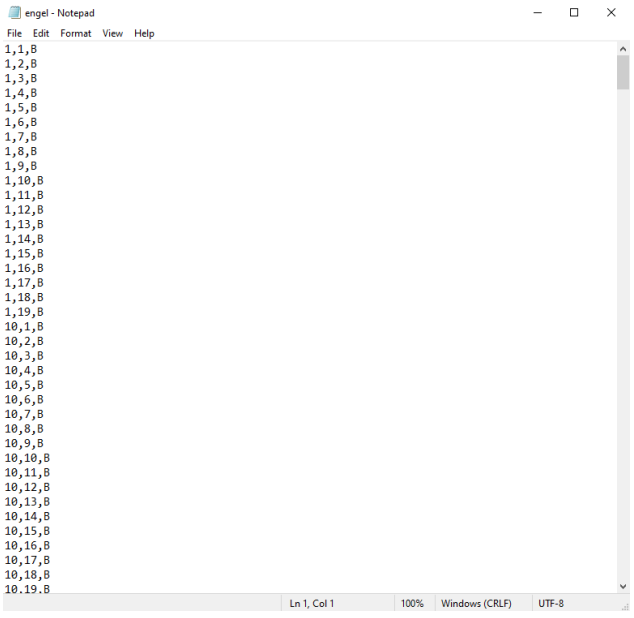


Şekil 5 - Örnek



Şekil 5.1 – Örnek

Son olarak 25*25'lik matristeki her bir karenin durumunu(kırmızı veya beyaz), koordinatı ile birlikte engel.txt isimli dosyaya yazdırıyoruz.



V. SONUÇ

Bu projeyi geliştirirken Reinforcement Learning makine öğrenmesi yöntemini ayrıntılı bir şekilde araştırdık ve Q-Learning algoritması ile bir öznenin bir görevi nasıl en yüksek kazançla tamamlayabileceğini öğrendik. Ayrıca bunu bir arayüz kullanarak yaparak aynı zamanda GUI programlama konusunda da yeni bilgiler öğrendik ve hem bu alanda hem de makine öğrenmesi alanında kendimizi geliştirdik

VI. REFERANSLAR

1) https://laptrinhx.com/finding-shortest-path-using-q-learning-algorithm-3888420843/

2) https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c

3) https://www.youtube.com/watch?v=E1gDJU9Q4kg&list=PLLAZ4kZ9dFpMMs5lskzBApYXn0bl7emsW&index=29

4) https://pythonguides.com/python-write-variable-to-file/

5) https://en.wikipedia.org/wiki/Q-learning#:~:text=Q%2Dlearning%20is%20a%20model,and%20rewards%20without%20requiring%20adaptations.

6) https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

7) https://www.youtube.com/watch?v=tMnc-hhO2jE

8) https://www.youtube.com/watch?v=YXPyB4XeYLA

9) https://www.tutorialspoint.com/python/python_gui_programming.htm

10) https://realpython.com/python-gui-tkinter/

11) https://perfectial.com/blog/q-learning-applications/