Q-learning ile Yol Planlaması

Abdullah Yaşar KISA
Mühendisik Fakültesi
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
abdullahyasarkisa@hotmail.com
180201015

İbrahim Kafkaslı
Mühendisik Fakültesi
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
ibrahim.kafkasli@hotmail.com
180201078

***Özet* – Bu projede temel amaç robotun Q learning algoritması kullanarak engel sütunlarından kaçması ve beyaz alanlardan geçerek doğru yol almasıdır. Proje de yol temsili olarak bir matris kullanılmıştır ve robot yolunu iki boyutlu bir ortamda bulmaktadır. Proje python programlama dili kullanılarak geliştirilmiştir. Çıktılar ise kullanıcıya bir ui ile sunulmuştur.**

Anahtar Kelimeler – Q-learning, Pekiştirmeli Öğrenme, Brute-Force, Python, Pygame

***Abstract* – The main purpose of this project is for the robot to escape from the obstacle columns by using the Q learning algorithm and take the right path by passing through the white areas. In the project, a matrix is used as a path representation and the robot finds its path in a two-dimensional environment. The project has been developed using the python programming language. Outputs are presented to the user with a ui.**

Keywords – Q-learning, reinforcement learning, Brute-Force, Python, Pygame

I. GIRIS

Q-Learning algoritması, pekiştirmeli öğrenmenin(reinforcement learning) en çok bilinen algoritmalarındandır. Algoritmadaki temel amaç bir sonraki hareketleri inceleyip yapacağı hareketlere göre kazanacağı ödülü görmek ve bu ödülü en çoklayıp (maximize) buna göre hareket etmektir.

Ajanın ödül haritası, gitmesini istediğimiz ya da istemediğimiz yerler daha önceden bizim tarafımızdan belirlenir ve bu değerler ödül tablosuna (Reward Table) yazılır. Ajanın tecrübeleri bu ödül tablosuna göre şekillenecektir.

Ajan ödüle giderken her iterasyonda edindiği tecrübeleri gidebildiği yerleri seçerken en çoklamak için kullanır. Bu tecrübeleri ise Q-Tablosu (Q-Table) adı verdiğimiz bir tabloda tutar.

Q-Tablomuz başlangıçta ajanımızın hiçbir tecrübesi olmadığı için sıfırlarla doludur ve bu yüzden ajanımız ilk seçimlerinde Q-Tablosundaki sıfırları en çoklayacağından rastgele hareket edecektir. Bu rastgelelik ajanın ilk ödülü bulmasına kadar sürecektir. Ajan ödülün olduğu bir yere geldiği anda ödüle gelmeden önceki yerini bilir ve bu yerin değerini kendi tecrübelerini biriktirdiği Q-Tablosuna yazar. Bu yazma işleminin belirli bir algoritması bulunmaktadır.

$$Q(s,a) = Q(s,a) + lr*(r(s,a) + Y*max(Q(s',a')) - Q(s,a))$$

"Q(s,a)" (state,action) dediğimiz değer bizim şu anda {bulunduğumuz, gideceğimiz} dizin, "lr" dediğimiz değer öğrenme katsayısı (learning rate),"r(s,a)" dediğimiz değer bizim {bulunduğumuz, gideceğimiz} ödül tablomuzdaki ödül değerimiz, "Y" değeri gamma, "max (Q (s',a'))" değeri ise gidebileceğimiz {gideceğimiz,gideceğimiz yerden gidebileceğimiz} yerlerin en yüksek Q değeridir.

Her seferinde ajan bu algoritmaya göre bir ilerisini tahmin edip hareket eder ve ödüle ulaşır. Ödüle ulaştıktan sonra ise ajan tekrardan rastgele bir yerden harekete başlar ve tekrardan ödülü bulmaya çalışır. Bu işlem devam ettikçe ajan hangi durumda nereye gideceğini bilmeye başlar.

Bu projede, ajanımız kullanıcıdan alınan başlangıç ve bitiş değerlerine göre, yine kullanıcıdan alınan matris boyutuna göre başlangıçtan, hedefe olan yolunu Brute-Force yaklaşımı ile bulacaktır.

II. KARŞILAŞILAN ZORLUKLAR

İlk olarak karşılaştığımız zorluk pandemi dolayısıyla ekip arkadaşımızdan uzak kalmamız oldu. Bu zorluğu aşmak her ne kadar zor olsa da, GIT ve GITHUB teknolojileri ile kod paylaşımını basit bir şekilde çözmüş olduk.

İkinci karşılaştığımız problem ise, python syntax'ına ekip arkadaşımızla birlikte çok fazla hakim olmayışımızdı. Daha önce python da çok az bir tecrübemiz olduğundan dolayı, dilin kendi syntax'ına alışmamız zaman aldı.

Üçüncü olarak karşılaştığımız problem, Q-learning algoritmasıyla alakalı daha önce hiçbir bilgimizin olmayışıydı. Brute-Force yaklaşımı ile birçok kod yazmamıza rağmen algoritmaya alışmamız biraz zamanımızı aldı. Ancak bu problemi zamanla ve doğru araştırmayla aşmayı başardık.

Dördüncü karşılaştığımız problem ise, kullanıcı arayüzü kısmıydı. Daha önce birçok kullanıcı arayüzü olan uygulamalar geliştirmemize rağmen, python ile böyle bir tecrübemizin olmayışı bizi çok zorladı. Bu problemi doğru ui kütüphanelerini kullanarak aşmayı başardık.

Beşinci problemimiz ise projemizin yoğun olduğumuz bir haftaya denk gelmesiydi. Çok yoğun olduğumuzdan dolayı projeye yeterli zaman ayıramamıştık. Ancak hocalarımızın projeyi 2 gün ertelemesi sayesinde bu problemin de üstesinden gelmiş olduk.

III. YÖNTEM

```python
from tkinter.constants import X
import PySimpleGUI as sg
import numpy as np
import pygame
from time import time, sleep
from random import randint as r
import random
import matplotlib.pyplot as plt
import math

class Form:
    def FormPage(self):
        sg.theme('LightGrey 6')
        layout = [
            [sg.Text('Please enter information')],
            [sg.Text('Start Location', size =(15, 1)), sg.InputText()],
            [sg.Text('Finish Location', size =(15, 1)), sg.InputText()],
            [sg.Text('Matrix Size', size =(15, 1)), sg.InputText()],

            [sg.Submit(), sg.Cancel()]
        ]

        window = sg.Window('Qlearning App', layout)
        event, value = window.read()
        window.close()
        return value


form = Form()
value = form.FormPage()


start = value[0]
start = start.split(",")
finish = value[1]
finish = finish.split(",")
size = value[2]
size = size.split(",")
```

Kodumuzda ilk olarak gerekli kütüphaneler dahil edilmiş ve kullanıcıdan, matris boyutu, başlangıç konumu ve bitiş konumunun alınacağı ui tasarlanıp çalıştırılmıştır. Burada alınan konumlar X,Y formatında olduğundan split edilerek belirli değişkenlerde sonradan kullanılmak üzere tutulmuştur. Kullanıcıdan bilgilerin alınacağı ui ise PySimpleGUI kütüphanesinden yararlanılarak yapılmıştır.

```python
# Qlearning #
n = int(size[0])  # represents no. of side squares(n*n total squares)
scrx = n*30
scry = n*30
x = math.ceil(scrx/int(n))

background = (0, 0, 0)  # used to clear screen while rendering
# creating a screen using Pygame

screen = pygame.display.set_mode((scrx, scry))
colors = [(255, 225, 255) for i in range(n**2)]

#create reward matrix
reward = np.zeros((n, n))
terminals = []
penalities = int(n*n*20/100)

startsPoint = []
startsPoint.append(int(start[0]))
startsPoint.append(int(start[1]))

finishPoint = []
finishPoint.append(int(finish[0]))
finishPoint.append(int(finish[1]))

```

Burada ise Q learning algoritması için başlangıç yapılıp gerekli değişkenlere atamalar yapılmıştır. Ayrıca Qlearning algoritmasının gösterileceği ekran için boyutlar ayarlanıp gerekli değişkenlere atanmıştır. Ayrıca tüm matris noktaları için beyaz renk ataması yapılmıştır. Bu renk ataması matriste bulunan labirent, başlangıç, bitiş ve yollara göre farklı renkler ile sonradan değiştirilecektir

```python

while penalities != 0:
    i = r(0, n-1)
    j = r(0, n-1)
    if reward[i, j] == 0 and [i, j] != [0, 0] and [i, j] != [n-1, n-1]:
        reward[i, j] = -5
        penalities -= 1
        colors[n*i+j] = (255, 0, 0)
        terminals.append(n*i+j)


reward[int(finish[0]),int(finish[1])] = 5  # finish position
colors[n*int(finish[0])+int(finish[1])] = (0, 255, 0)
colors[n*int(start[0])+int(start[1])] = (0, 137, 255)
terminals.append(n*(int(finish[0])) + int(finish[1]))
print(reward)

```

Burada ise, belirlenen engel oranına göre engeller rastgele olarak matriste konumlara atanmıştır. Engellerin bulunduğu konumlar kırmızı rengine boyanmıştır. Ayrıca engellerin bulunduğu konumlara -5 değeri atanırken, bitiş lokasyonunun bulunduğu konuma yine reward matrisinde 5 değeri atanmıştır. Ayrıca bitiş ve başlangıç konumlarıda ui için renklendirilmiştir.

```python
Q = np.zeros((n**2, 4))  # Initializing Q-Table
actions = {"up": 0, "down": 1, "left": 2, "right": 3}  # possible actions
states = {}
k = 0
for i in range(n):
    for j in range(n):
        states[(i, j)] = k
        k += 1
alpha = 0.01
gamma = 0.9

#start position
current_pos = [int(start[0]), int(start[1])]

epsilon = 0.25
```

Burada ise Q matrisi başlangıç olarak 0 değerleriyle doldurulmuştur. Ayrıca ajan hareketi için durumlar tanımlanıp yine Qlearning algoritması için gerekli alpha, gamma, epsilon öğrenme katsayıları atanmıştır. Ayrıca başlangıç konumu olarak kullanıcının girdiği konum belirlenmiştir.

```python
def select_action(current_state):
    global current_pos, epsilon
    possible_actions = []
    if np.random.uniform() <= epsilon:
        if current_pos[0] != 0:
            possible_actions.append("up")
        if current_pos[0] != n-1:
            possible_actions.append("down")
        if current_pos[1] != 0:
            possible_actions.append("left")
        if current_pos[1] != n-1:
            possible_actions.append("right")
        action = actions[possible_actions[r(0, len(possible_actions) - 1)]]
    else:
        m = np.min(Q[current_state])
        if current_pos[0] != 0:  # up
            possible_actions.append(Q[current_state, 0])
        else:
            possible_actions.append(m - x)
        if current_pos[0] != n-1:  # down
            possible_actions.append(Q[current_state, 1])
        else:
            possible_actions.append(m - x)
        if current_pos[1] != 0:  # left
            possible_actions.append(Q[current_state, 2])
        else:
            possible_actions.append(m - x)
        if current_pos[1] != n-1:  # right
            possible_actions.append(Q[current_state, 3])
        else:
            possible_actions.append(m - x)
        action = random.choice([i for i, a in enumerate(possible_actions) if a == max(
            possible_actions)])  # randomly selecting one of all possible actions with maximin value
    return action

```

Bu kısımda ise ileride kullanılmak üzere random aksiyonlar belirlenmiştir. Ajan kazanç sağlayana kadar random hareket sağlayacaktır. Kazanç sağladığında ise kazanç sağladığı yol üzerinden hareketine devam edecektir. Burada bu tanımlama yapılmıştır.

```python
def episode(iterasyon, startPoint, step, cost, epsiodeStep, epsiodeCost,road,proabilityRoad):
    global current_pos, epsilon
    current_state = states[(current_pos[0], current_pos[1])]
    action = select_action(current_state)
    if action == 0:  # move up
        current_pos[0] -= 1
    elif action == 1:  # move down
        current_pos[0] += 1
    elif action == 2:  # move left
        current_pos[1] -= 1
    elif action == 3:  # move right
        current_pos[1] += 1
    new_state = states[(current_pos[0], current_pos[1])]
    if new_state not in terminals:
        Q[current_state, action] += alpha*(reward[current_pos[0], current_pos[1]] + gamma*(
            np.max(Q[new_state])) - Q[current_state, action])
        step += 1
        cost = cost+3
        road.append([current_pos[0], current_pos[1]])
    else:
        Q[current_state, action] += alpha * \
            (reward[current_pos[0], current_pos[1]] - Q[current_state, action])
        step+=1
        cost = cost+reward[current_pos[0], current_pos[1]]
        road.append([current_pos[0], current_pos[1]])

        print(iterasyon, ". iterasyon")
        iterasyon += 1
        # print(current_pos)
        # step=round(startsPoint[0]-current_pos[0])+round(startsPoint[1]-current_pos[1])
        #print("step: ", step)
        #print("cost: ", cost)

        epsiodeStep.append(step)
        epsiodeCost.append(cost)
        proabilityRoad.clear()
        #if reward[road[len(road)-1][0],road[len(road)-1][1]]==5:
        proabilityRoad=road[:]

        #print(road)
        road.clear()
        cost = 0
        step = 0
        current_pos[0] = int(start[0])
        current_pos[1] = int(start[1])

        if epsilon > 0.05:
            epsilon -= 3e-4  # reducing as time increases to satisfy Exploration & Exploitation Tradeoff

    return iterasyon, step, cost, epsiodeCost, epsiodeStep,road, proabilityRoad
```

Bu kısımda ise Q learning hareketinde her bir adım için olan hareket tanımlanmıştır. Bu kısımda ajanın bir hareket gerçekleştirebilmesi için gerekli tüm işlemler yapılmıştır. Bu işlemler belirli değişkenlerde tutularak, kodun son çıktısında kullanılmak üzere saklanmıştır.

```python
def layout(isFinish, probabilityRoad):
    c = 0
    if isFinish == False:
        for i in range(0, scrx, 30):
            for j in range(0, scry, 30):
                pygame.draw.rect(screen, (0, 0, 0),
                                 (j, i, j+30, i+30), 0)
                pygame.draw.rect(
                    screen, colors[c], (j+1, i+1, j+28, i+28), 0)
                c += 1
                pygame.draw.circle(screen, (0, 0, 0), (
                    current_pos[1]*30 + 14, current_pos[0]*30 + 14), 10, 0)
    else:
        for k in range(0,len(probabilityRoad)-1):
            pygame.draw.circle(screen, (0, 0, 0), (
                    probabilityRoad[k][1]*30 + 14, probabilityRoad[k][0]*30 + 14), 10, 0)
            pygame.display.flip()
```

Burada ise ajanın yapacağı her hareket, Pygame kütüphanesi kullanılarak bir matris ekranında kullanıcıya sunulmuştur. Ayrıca tüm işlemler tamamlandıktan sonra, kullancıya ajanın bulduğu yolda bir ui üzerinde sunulmuştur.

```python
def plot_results(steps, cost):
        #
        f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
        #
        ax1.plot(np.arange(len(steps)), steps, 'g')
        ax1.set_xlabel('Tur')
        ax1.set_ylabel('Adım')
        ax1.set_title('Her Tur için Adım')

        #
        ax2.plot(np.arange(len(cost)), cost, 'b')
        ax2.set_xlabel('Tur')
        ax2.set_ylabel('Kazanç')
        ax2.set_title('Her Tur için Kazanç')

        plt.tight_layout()  # Function to make distance between figures

        # Showing the plots
        plt.show()
```

Burada ise tüm işlemlerin tamamlanmasının ardından, kullanıcıya her tur için adım ve her tur için kazanç grafikleri çıktı olarak verilmiştir.

```python
def  ciktiVer() :
    dosya = open('engel.txt', 'w')
    dosya.write("S= start point\n")
    dosya.write("F= finish point\n")
    dosya.write("R= road\n")
    dosya.write("B= block\n")
    dosya.write("\n\n\n")
    for i in range(n) :
        for j in range(n):
            if i==startsPoint[0] and j==startsPoint[1]:
                dosya.write("({},{},S)".format(i,j))
            elif i==finishPoint[0] and j==finishPoint[1]:
                dosya.write("({},{},F)".format(i,j))
            elif(reward[i][j]==0):
                dosya.write("({},{},R)".format(i,j))
            elif(reward[i][j]==-5):
                dosya.write("({},{},B)".format(i,j))
        dosya.write("\n")
    dosya.close()
```

Burada ise, kullanıcı için bir dosyaya R matirisi kodun ilk çalışma aşamasında yazdırılmıştır. Bu matriste engeller, yollar, başlangıç ve bitiş konumları bulunmaktadır.

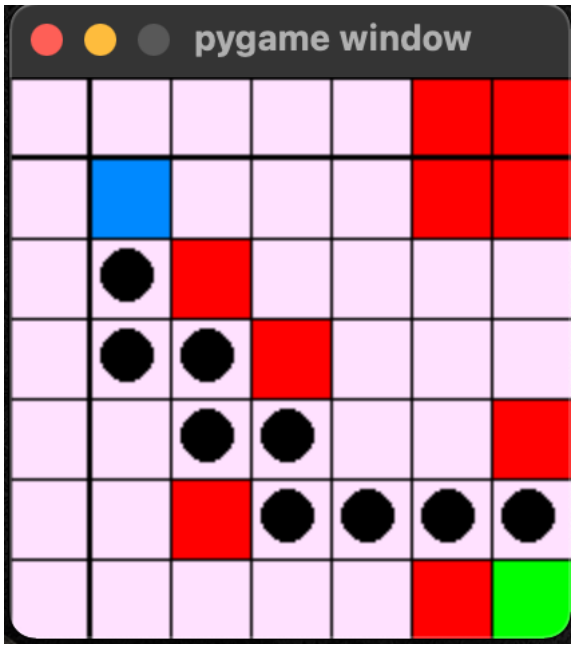
```python
ciktiVer()
run = True
iterasyon = 0
step = 0
epsiodeStep = []
cost = 0
epsiodeCost = []
road=[]
proabilityRoad=[]


while iterasyon<=1000:
    screen.fill(background)
    layout(False, proabilityRoad)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    pygame.display.flip()
    iterasyon, step, cost,epsiodeCost,epsiodeStep,road,proabilityRoad = episode(
        iterasyon, startsPoint, step, cost, epsiodeStep, epsiodeCost,road,proabilityRoad)


print(proabilityRoad)
layout(True, proabilityRoad)
plot_results(epsiodeStep,epsiodeCost)


pygame.quit()
```

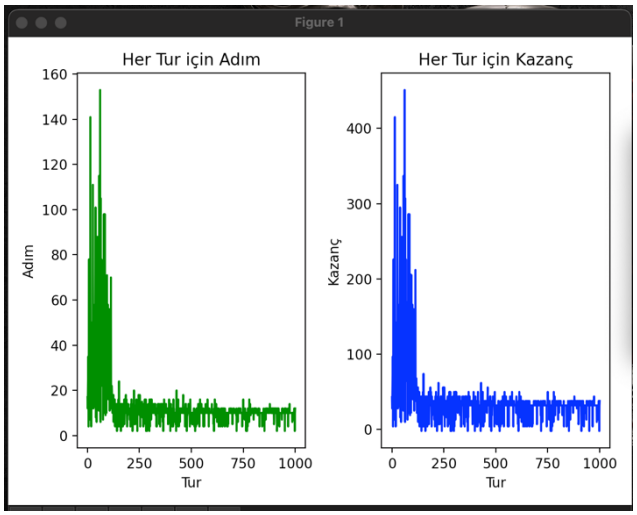
Burada ise, pekiştirmeli öğrenme işlemi belirli bir tekrar sayısınca başlatılmış ve işlem bitince gerekli çıktılar ekrana sunulmuştur.

IV. KOD TASARIMI

Kodun akış diyagramı 5. Sayfadan sonra başlamaktadır.

V. DENEYSEL SONUCLAR





VI. SONUÇ

Tüm bu işlemler ile ajan Q-learning algoritmasını kullanarak, brute-force yaklaşım ile bir labirent içerisinde başlangıçtan bitişe olan yolunu bulmuştur.

VII. KULLANILAN TEKNOLOJİLER

- Python
- Pygame
- PySimpleGUI
- Matplotlib
- GIT ve GITHUB

-Visual Studio Code tümleşik geliştirme ortamları kullanılmıştır.

VIII. KAZANIMLAR

- Q-learning algoritması hakkında bilgi sahibi olduk
- Brute-Force yaklaşım hakkında bilgi sahibi olduk
- Pygames, PySimpleGUI, Matplotlib kütüphaneleri hakkında bilgi sahibi olduk

KAYNAKÇA

WEB SİTE

https://medium.com/deep-learning-turkiye/q-learninge-giriş-6742b3c5ed2b

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

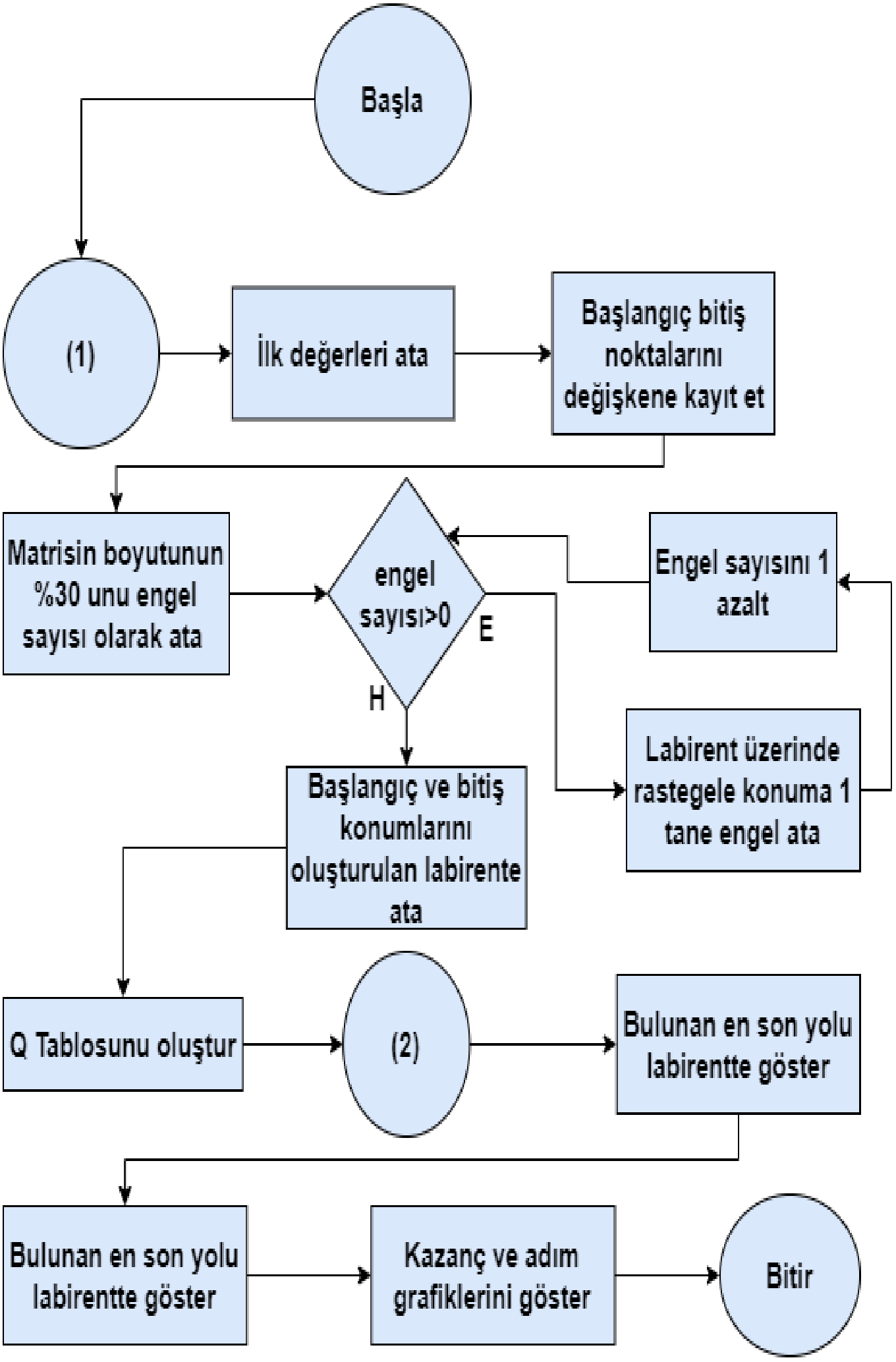
https://medium.com/analytics-vidhya/introduction-to-reinforcement-learning-q-learning-by-maze-solving-example-c34039019317

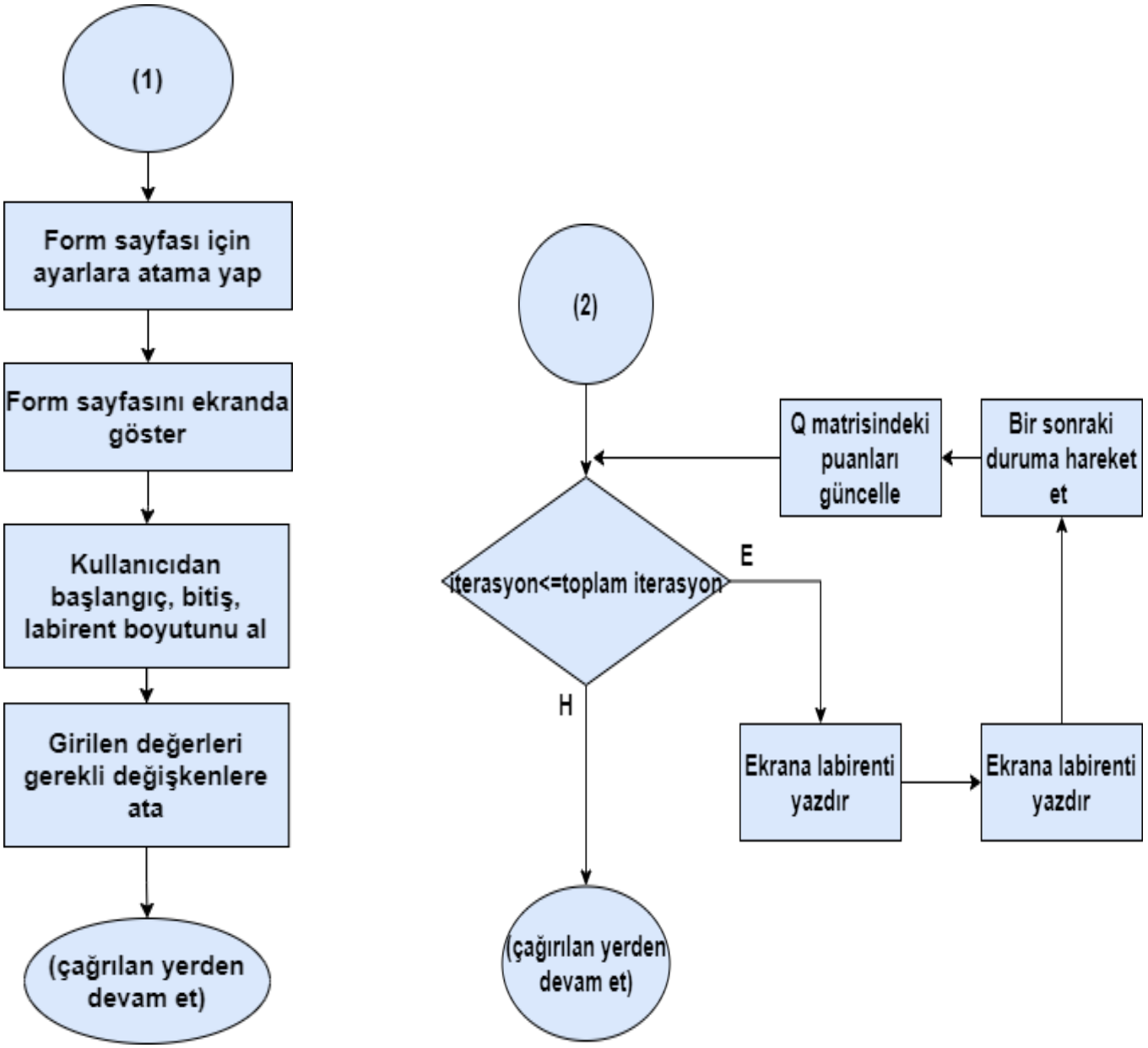
https://medium.com/data-science-in-your-pocket/maze-runner-%EF%B8%8F-with-off-policy-q-learning-no-back-stepping-allowed-d01a79a6199c

https://github.com/erikdelange/Reinforcement-Learning-Maze

https://becominghuman.ai/q-learning-a-maneuver-of-mazes-885137e957e4

https://github.com/sichkar-valentyn/Reinforcement_Learning_in_Python


Başla
(1)
İlk değerleri ata
Başlangıç bitiş noktalarını değişkene kayıt et
Matrisin boyutunun %30 unu engel sayısı olarak ata
engel sayısı>0
E
H
Engel sayısını 1 azalt
Labirent üzerinde rastegele konuma 1 tane engel ata
Başlangıç ve bitiş konumlarını oluşturulan labirente ata
Q Tablosunu oluştur
(2)
Bulunan en son yolu labirentte göster
Bulunan en son yolu labirentte göster
Kazanç ve adım grafiklerini göster
Bitir


(1)
Form sayfası için ayarlara atama yap
Form sayfasını ekranda göster
Kullanıcıdan başlangıç, bitiş, labirent boyutunu al
Girilen değerleri gerekli değişkenlere ata
(çağrılan yerden devam et)
(2)
Q matrisindeki puanları güncelle
Bir sonraki duruma hareket et
iterasyon<=toplam iterasyon
E
H
Ekrana labirenti yazdır
Ekrana labirenti yazdır
(çağırılan yerden devam et)