Mobile Robot Q-Learning

Reinforcement Learning dengan Q Learning Python

Program ini untuk penelitian pendekatan perencanaan jalur global untuk robot seluler, berdasarkan algoritma Q-learning

Danang Haris Setiawan
FIBONACCI

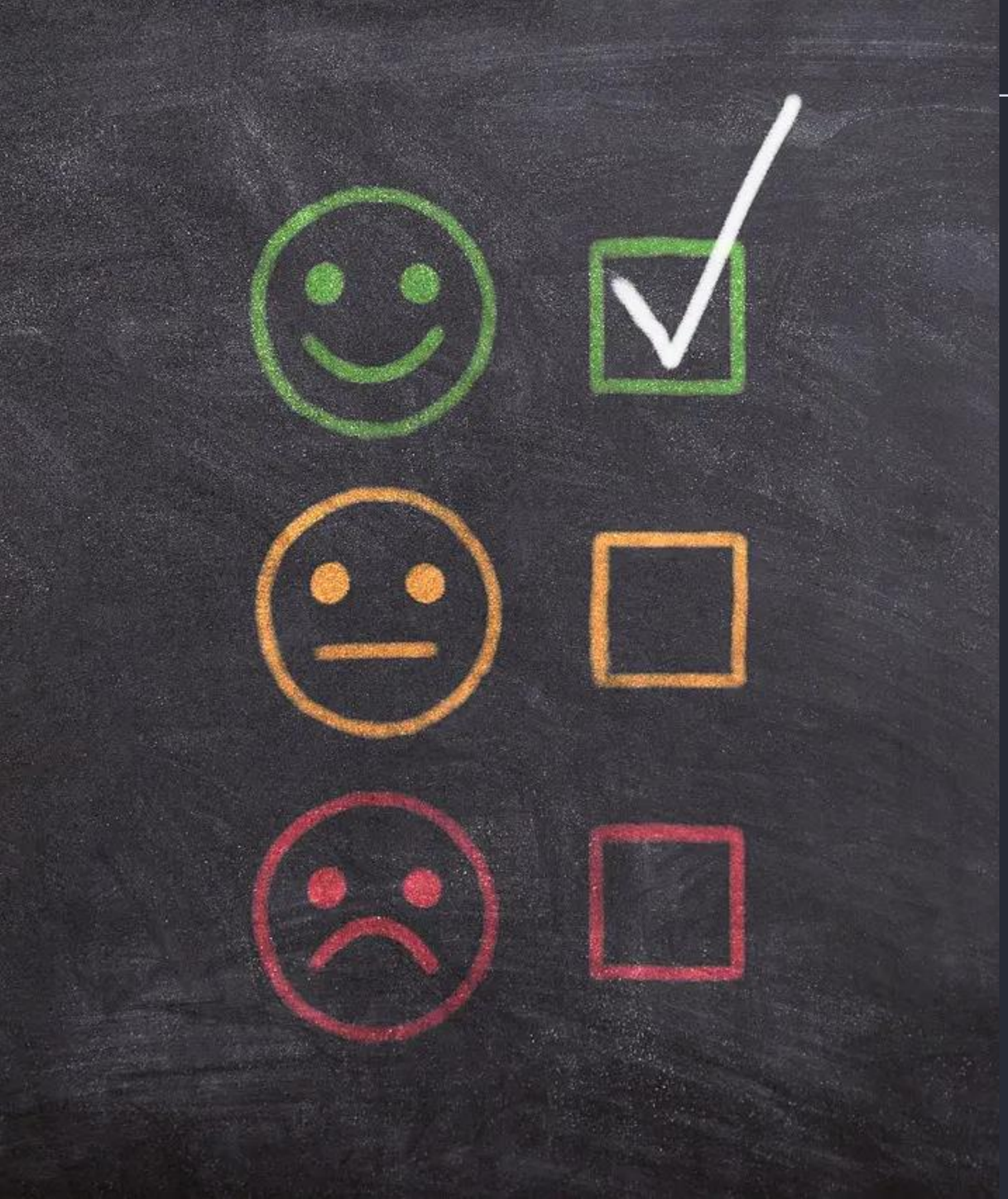
Keep Smile For Awesome

FIBONACCI

Q Learning

Q-Learning adalah algoritma reinforcement learning model free untuk mempelajari nilai suatu Tindakan dalam keadaan tertentu. Itu tidak memerlukan model lingkungan, dan dapat menangani masalah dengan transisi stokastik dan penghargaan tanpa memerlukan adaptasi.

Komponen utama dari metode RL adalah tabel bobot - Q-tabel dari status sistem. Matriks Q adalah himpunan semua kemungkinan keadaan sistem dan bobot respons sistem untuk tindakan yang berbeda. Selama mencoba melewati lingkungan yang diberikan, mobile robot belajar bagaimana menghindari rintangan dan menemukan jalan menuju titik tujuan. Akibatnya, Q-tabel dibangun. Melihat nilai-nilai tabel memungkinkan untuk melihat keputusan untuk tindakan selanjutnya yang dilakukan oleh agen (robot seluler).



MEGAN

Definisi Masalah

Program ini untuk penelitian pendekatan perencanaan jalur global untuk robot seluler, berdasarkan algoritma Q-learning, penerapan algoritma reinforcement learning menghitung kecepatan waktu pembelajaran dan metode membangun jalur untuk menghindari rintangan untuk mencapai titik tujuan. Analisis hasil yang diperoleh memungkinkan untuk memilih parameter optimal dari algoritma untuk lingkungan yang di uji. Eksperimen dilakukan di lingkungan virtual dimana algoritma mempelajari langkah mana yang harus dipilih untuk mendapatkan hasil maksimal dan mencapai tujuan menghindari rintangan.

File Program

Program ini terdiri dari tiga file: *env.py*, *agent.py* dan *main.ipynb*

Env.py

File yang membangun environment beserta rintangannya

Agent.py

Implementasi algoritma itu sendiri

Main.ipynb

Menjalankan program

Env.py

```
import numpy as np
import tkinter as tk
import time
from PIL import Image, ImageTk


pixels = 20
env_height = 25
env_width = 25


a = {}
```

- Menangani data dalam bentuk matriks
- Tkinter untuk membangun GUI
- Time dibutuhkan untuk memperlambat agen dan melihat bagaimana dia berjalan
- PIL untuk menambahkan gambar ke widget kanvas

Mengatur ukuran untuk environmennya
Disini kami menggunakan satuan piksel untuk masing-masing tinggi dan lebarnya. Dan kami membuat variable global dengan nama *a* dictionary dengan koordinat untuk rute akhir dari agent.

Env.py lanjut

```
class Environment(tk.Tk, object):
    def __init__(self):
        super(Environment, self).__init__()
        self.action_space = ['up', 'down', 'left', 'right']
        self.n_actions = len(self.action_space)
        self.title('RL Q-learning. Sichkar Valentyn')
        self.geometry('{0}x{1}'.format(env_height * pixels, env_height * pixels))
        self.build_environment()

        # Dictonary untuk menggambar rute terakhir
        self.d = {}
        self.f = {}
        # Key untuk dictionaries
        self.i = 0
        # Menulis kamus terakhir pertama kali
        self.c = True
        # Menampilkan langkah-langkah untuk rute terpanjang yang ditemukan
        self.longest = 0
        # Menampilkan langkah-langkah untuk rute terpendek
        self.shortest = 0

        def build_environment(self):
              ...

        def reset(self):
              ...

        def step(self, action):
              ...

        def render(self):
              ...

        def final(self):
              ...
```

Membuat class untuk environment

Didalam class ini kita akan membuat beberapa fungsi yang di butuhkan di antaranya fungsi build_environment, reset, step, render, dan final dengan kegunaan yang berbeda-beda

- **Fungsi *build_environment* digunakan untuk membangun environment itu sendiri.**
- **Fungsi *reset* digunakan untuk mereset environment dan memulai episode baru**
- **Fungsi *step* untuk mendapatkan pengamatan dan penghargaan selanjutnya dengan melakukan langkah selanjutnya**
- **Fungsi *render* berfungsi untuk merefres environment**
- **Fungsi *final* untuk menunjukkan rute yang ditemukan**

Env.py fungsi build_environment

```
def build_environment(self):
    self.canvas_widget = tk.Canvas(self,  bg='white',
                                   height=env_height * pixels,
                                   width=env_width * pixels)

    # Mengunggah gambar untuk latar belakang
    img_background = Image.open("images/bg.png")
    self.background = ImageTk.PhotoImage(img_background)
    # Membuat latar belakang pada widget
    self.bg = self.canvas_widget.create_image(0, 0, anchor='nw', image=self.background)

    # Membuat garis kisi
    for column in range(0, env_width * pixels, pixels):
        x0, y0, x1, y1 = column, 0, column, env_height * pixels
        self.canvas_widget.create_line(x0, y0, x1, y1, fill='grey')
    for row in range(0, env_height * pixels, pixels):
        x0, y0, x1, y1 = 0, row, env_height * pixels, row
        self.canvas_widget.create_line(x0, y0, x1, y1, fill='grey')

    # Membuat objek  Hambatan
    # Array untuk membantu membangun persegi panjang
    self.o = np.array([pixels / 2, pixels / 2])



    # Obstacle 1
    # Menentukan pusat rintangan 1
    obstacle1_center = self.o + np.array([pixels, pixels * 2])
    # Membangun rintangan 1
    self.obstacle1 = self.canvas_widget.create_rectangle(
        obstacle1_center[0] - 10, obstacle1_center[1] - 10,  # Pojok kiri atas
        obstacle1_center[0] + 10, obstacle1_center[1] + 10,  # Pojok kanan bawah
        outline='grey', fill='#00BFFF')
    # Menyimpan koordinat hambatan 1 sesuai dengan ukuran agen
    # Agar sesuai dengan koordinat agen
    self.coords_obstacle1 = [self.canvas_widget.coords(self.obstacle1)[0] + 3,
     self.canvas_widget.coords(self.obstacle1)[1] + 3,
     self.canvas_widget.coords(self.obstacle1)[2] - 3,
     self.canvas_widget.coords(self.obstacle1)[3] - 3]

    # Obstacle 2
    # Menentukan pusat rintangan 2
    obstacle2_center = self.o + np.array([pixels * 2, pixels * 2])
    # Membangun rintangan 2
    self.obstacle2 = self.canvas_widget.create_rectangle(
        obstacle2_center[0] - 10, obstacle2_center[1] - 10,  # Pojok kiri atas
        obstacle2_center[0] + 10, obstacle2_center[1] + 10,  # Pojok kanan bawah
        outline='grey', fill='#00BFFF')
    # Menyimpan koordinat rintangan 2 sesuai dengan ukuran agen
    # Agar sesuai dengan koordinat agen
    self.coords_obstacle2 = [self.canvas_widget.coords(self.obstacle2)[0] + 3,
     self.canvas_widget.coords(self.obstacle2)[1] + 3,
     self.canvas_widget.coords(self.obstacle2)[2] - 3,
     self.canvas_widget.coords(self.obstacle2)[3] - 3]
```

Didalam fungsi build_environment kami akan membuat environment kami dengan 30 rintangan berbeda-beda.
Disini kami memberikan dua contoh pembuatan rintangan di environment kami karena jika menuliskan semuanya akan sangat sia-sia karena setiap rintangan hampir sama cuman berbeda penempatannya.

Env.py fungsi build_environment lanjut

```
# Membuat agen Robot - titik merah
self.agent = self.canvas_widget.create_oval(
    self.o[0] - 7, self.o[1] - 7,
    self.o[0] + 7, self.o[1] + 7,
    outline='#FF1493', fill='#FF1493')

# Titik Akhir - titik kuning
flag_center = self.o + np.array([pixels * 20, pixels * 20])
# Membangun bendera
self.flag = self.canvas_widget.create_rectangle(
    flag_center[0] - 10, flag_center[1] - 10,  # Pojok kiri atas
    flag_center[0] + 10, flag_center[1] + 10,  # Pojok kanan bawah
    outline='grey', fill='yellow')
# Menyimpan koordinat titik akhir sesuai dengan ukuran agen
# Agar sesuai dengan koordinat agen
self.coords_flag = [self.canvas_widget.coords(self.flag)[0] + 3,
                    self.canvas_widget.coords(self.flag)[1] + 3,
                    self.canvas_widget.coords(self.flag)[2] - 3,
                    self.canvas_widget.coords(self.flag)[3] - 3]

self.canvas_widget.pack()
```

Selanjutnya didalam fungsi *build_environment* adalah membuat robot yang berperan sebagai agent.

Env.py fungsi reset

```
def reset(self):
    self.update()
    #time.sleep(0.5)

    # update agent
    self.canvas_widget.delete(self.agent)
    self.agent = self.canvas_widget.create_oval(
        self.o[0] - 7, self.o[1] - 7,
        self.o[0] + 7, self.o[1] + 7,
        outline='red', fill='red')

    # membersihkan dictionary dan i
    self.d = {}
    self.i = 0

    # return observation
    return self.canvas_widget.coords(self.agent)
```

Fungsi reset akan berfungsi untuk mengatur environment dan memulai episode baru. Yang mengembalikan hasil pengamatan

Env.py fungsi step

```python
def step(self, action):
    # Status agen saat ini
    state = self.canvas_widget.coords(self.agent)
    base_action = np.array([0, 0])

    # Memperbarui status berikutnya sesuai dengan tindakan
    # Action 'up'
    if action == 0:
        if state[1] >= pixels:
            base_action[1] -= pixels
    # Action 'down'
    elif action == 1:
        if state[1] < (env_height - 1) * pixels:
            base_action[1] += pixels
    # Action right
    elif action == 2:
        if state[0] < (env_width - 1) * pixels:
            base_action[0] += pixels
    # Action left
    elif action == 3:
        if state[0] >= pixels:
            base_action[0] -= pixels

    # Memindahkan agen sesuai dengan tindakan
    self.canvas_widget.move(self.agent, base_action[0], base_action[1])

    # Menulis di kamus koordinat rute yang ditemukan
    self.d[self.i] = self.canvas_widget.coords(self.agent)

    # Memperbarui status berikutnya
    next_state = self.d[self.i]

    # Memperbarui kunci untuk kamus
    self.i += 1
```

Fungsi step berfungsi untuk mendapatkan pengamatan dan penghargaan selanjutnya dengan melakukan langkah selanjutnya

Env.py fungsi step lanjut

```
        # Menghitung hadiah untuk agen
        if next_state == self.coords_flag:
            time.sleep(0.1)
            reward = 1
            done = True
            next_state = 'goal'

            # Mengisi kamus pertama kali
            if self.c == True:
                for j in range(len(self.d)):
                    self.f[j] = self.d[j]
                self.c = False
                self.longest = len(self.d)
                self.shortest = len(self.d)

            # Memeriksa apakah rute yang ditemukan saat ini lebih pendek
            if len(self.d) < len(self.f):
                # Menyimpan jumlah langkah untuk rute terpendek
                self.shortest = len(self.d)
                # Membersihkan kamus untuk rute terakhir
                self.f = {}
                # Menugaskan ulang kamus
                for j in range(len(self.d)):
                    self.f[j] = self.d[j]

            # Menyimpan jumlah langkah untuk rute terpanjang
            if len(self.d) > self.longest:
                self.longest = len(self.d)

        elif next_state in [self.coords_obstacle1, self.coords_obstacle2, self.coords_obstacle3, self.coords_obstacle4,
self.coords_obstacle5,\
                self.coords_obstacle6, self.coords_obstacle7, self.coords_obstacle8, self.coords_obstacle9, self.coords_obstacle10, \
                self.coords_obstacle11, self.coords_obstacle12, self.coords_obstacle13, self.coords_obstacle14,
self.coords_obstacle15, \
                self.coords_obstacle16, self.coords_obstacle17, self.coords_obstacle18, self.coords_obstacle19,
self.coords_obstacle20, \
                self.coords_obstacle21, self.coords_obstacle22, self.coords_obstacle23, self.coords_obstacle24,
self.coords_obstacle25, \
                self.coords_obstacle26, self.coords_obstacle27, self.coords_obstacle28, self.coords_obstacle29,
self.coords_obstacle30,]:

            reward = -1
            done = True
            next_state = 'obstacle'

            # membersihkan dictionary dan i
            self.d = {}
            self.i = 0

        else:
            reward = 0
            done = False

        return next_state, reward, done
```

Fungsi step akan mengembalikan next_state, reward, dan done

Env.py fungsi render

```
def render(self):
    #time.sleep(0.03)
    self.update()
```

Fungsi render untuk merefres environment

Env.py fungsi final

```
def final(self):
    # Menghapus agen di akhir
    self.canvas_widget.delete(self.agent)

    # Menampilkan jumlah langkah
    print('The shortest route:', self.shortest)
    print('The longest route:', self.longest)

    # Membuat titik awal
    self.initial_point = self.canvas_widget.create_oval(
        self.o[0] - 4, self.o[1] - 4,
        self.o[0] + 4, self.o[1] + 4,
        fill='blue', outline='blue')

    # Mengisi rute
    for j in range(len(self.f)):
        # Menampilkan koordinat rute akhir
        print(self.f[j])
        self.track = self.canvas_widget.create_oval(
            self.f[j][0] - 3 + self.o[0] - 4, self.f[j][1] - 3 + self.o[1] - 4,
            self.f[j][0] - 3 + self.o[0] + 4, self.f[j][1] - 3 + self.o[1] + 4,
            fill='blue', outline='blue')
        # Menulis rute akhir dalam variabel global a
        a[j] = self.f[j]
```

Fungsii final berfungsi untuk menunjukkan rute yang ditemukan

Env.py final_states

```
def final_states():
    return a
```

Final_state akan mengembalikan kamus terakhir dengan koordinat rute yang Kemudian akan digunakan di agent.py

Agen.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# import final_states dari env.py
from env import final_states


class QLearningTable:
    def __init__(self, actions, learning_rate=0.05, reward_decay=0.99, e_greedy=0.9):

        self.actions = actions # List of actions
        self.lr = learning_rate # Learning rate
        self.gamma = reward_decay # Value of gamma
        self.epsilon = e_greedy # Value of epsilon

        # Membuat tabel Q lengkap untuk semua sel
        self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64)
        # Membuat tabel-Q untuk sel-sel dari rute terakhir
        self.q_table_final = pd.DataFrame(columns=self.actions, dtype=np.float64)

    def choose_action(self, observation):
        ...

    def learn(self, state, action, reward, next_state):
        ...

    def check_state_exist(self, state):
        ...

    def print_q_table(self):
        ...
    def plot_results(self, steps, cost):
        ...
```

Di file agen.py kami membuat class QLearningTable dengan method *choose_action, learn, check_state_exists, print_q_table,* dan *plot_results*

- Method choose_action untuk memilih Tindakan agen
- Method learn untuk mempelajari dan pemberbarui table Q dengan pengetahuan baru
- Method check_state_exist untuk menambah state baru di Q-Tabel
- Method print_q_table untuk mencetak Q-table dengan state
- Method plot_results untuk visualisasi hasil untuk jumlah state

Agen.py choose_action

Method choose_action berfungsi untuk memilih Tindakan untuk agen

```
def choose_action(self, observation):

    # Memeriksa apakah state ada di tabel
    self.check_state_exist(observation)

    # Pemilihan aksi - 99% sesuai dengan epsilon == 0.99
    # Memilih tindakan terbaik
    if np.random.uniform() < self.epsilon:
        state_action = self.q_table.loc[observation, :]
        state_action = state_action.reindex(np.random.permutation(state_action.index))
        action = state_action.idxmax()
    else:
        # Memilih tindakan acak - tersisa 10% untuk memilih secara acak
        action = np.random.choice(self.actions)
    return action
```

Agen.py learn

```python
def learn(self, state, action, reward, next_state):
    # Memeriksa apakah langkah selanjutnya ada di tabel-Q
    self.check_state_exist(next_state)

    # Keadaan saat ini di posisi saat ini
    q_predict = self.q_table.loc[state, action]

    # Memeriksa apakah keadaan berikutnya bebas hambatan
    if next_state != 'goal' or next_state != 'obstacle':
        q_target = reward + self.gamma * self.q_table.loc[next_state, :].max()
    else:
        q_target = reward

    # Memperbarui tabel-Q dengan pengetahuan baru
    self.q_table.loc[state, action] += self.lr * (q_target - q_predict)

    return self.q_table.loc[state, action]
```

Method learn berfungsi untuk mempelajari dan memperbaharui q table dengan pengetahuan baru

Agen.py check_state_exist

```
def check_state_exist(self, state):
    if state not in self.q_table.index:
        self.q_table = self.q_table.append(
            pd.Series(
                [0]*len(self.actions),
                index=self.q_table.columns,
                name=state,
            )
        )
```

Method check_state_exists untuk menambah state baru Q-table

Agen.py print_q_table

```python
def print_q_table(self):
    # Mendapatkan koordinat rute akhir dari env.py
    e = final_states()

    # Membandingkan indeks dengan koordinat dan menulis nilai tabel Q baru
    for i in range(len(e)):
        state = str(e[i])  # state = '[5.0, 40.0]'
        # Memeriksa semua indeks dan memeriksa
        for j in range(len(self.q_table.index)):
            if self.q_table.index[j] == state:
                self.q_table_final.loc[state, :] = self.q_table.loc[state, :]

    print()
    print('Panjang Q-table final=', len(self.q_table_final.index))
    print('Final Q-table dengan nilai dari rute akhir:')
    print(self.q_table_final)

    print()
    print('Length Q-table penuh =', len(self.q_table.index))
    print('Full Q-table:')
    print(self.q_table)
```

Method print_q_table berfungsi untuk mencetak Q-table dengan state

Agen.py plot_results

```python
def plot_results(self, steps, cost):
    #
    f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
    #
    ax1.plot(np.arange(len(steps)), steps, 'b')
    ax1.set_xlabel('Episode')
    ax1.set_ylabel('Steps')
    ax1.set_title('Episode via steps')

    #
    ax2.plot(np.arange(len(cost)), cost, 'r')
    ax2.set_xlabel('Episode')
    ax2.set_ylabel('Cost')
    ax2.set_title('Episode via cost')

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

    #
    plt.figure()
    plt.plot(np.arange(len(steps)), steps, 'b')
    plt.title('Episode via steps')
    plt.xlabel('Episode')
    plt.ylabel('Steps')

    #
    plt.figure()
    plt.plot(np.arange(len(cost)), cost, 'r')
    plt.title('Episode via cost')
    plt.xlabel('Episode')
    plt.ylabel('Cost')

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

Method plot_results berfungsi untuk memvisualisasikan hasil jumlah state

Main.ipynb

```
from env import Environment
from agent import QLearningTable
```

```
env = Environment()
RL = QLearningTable(actions=list(range(env.n_actions)))
```

Di file main pertama kita import class Environment dari file env dan QLearningTable dari file agent

Selanjutnya kita buat objek dari class Environment dan juga class QlearningTable dan disimpan di variable berbeda.

Selanjutnya kita perlu membuat fungsi update untuk mengupdate observasi agent

Main.ipynb lanjut

```
def update():
    # Daftar yang dihasilkan untuk merencanakan Episode
    steps = []
    # Jumlahkan biaya untuk semua episode dalam daftar yang dihasilkan
    all_costs = []
    for episode in range(100):
        # Initial Observation
        observation = env.reset()
        # Memperbarui jumlah Langkah untuk setiap Episode
        i = 0
        # Memperbarui biaya untuk setiap episode
        cost = 0
        while True:
            # Refreshing environment
            env.render()
            # RL memilih tindakan berdasarkan pengamatan
            action = RL.choose_action(str(observation))
            # RL mengambil tindakan dan mendapatkan pengamatan dan hadiah berikutnya
            observation_, reward, done = env.step(action)
            # RL belajar dari transisi ini dan menghitung biayanya
            cost += RL.learn(str(observation), action, reward, str(observation_))
            # Menukar pengamatan - saat ini dan selanjutnya
            observation = observation_
            # Menghitung jumlah Langkah dalam Episode saat ini
            i += 1
            # Break while loop saat ini adalah akhir dari Episode saat ini
            # Ketika agen mencapai tujuan atau rintangan
            if done:
                steps += [i]
                all_costs += [cost]
                break
    # Menampilkan rute terakhir
    env.final()
    # Menampilkan tabel-Q dengan nilai untuk setiap tindakan
    RL.print_q_table()
    # Merencanakan hasil
    RL.plot_results(steps, all_costs)
```

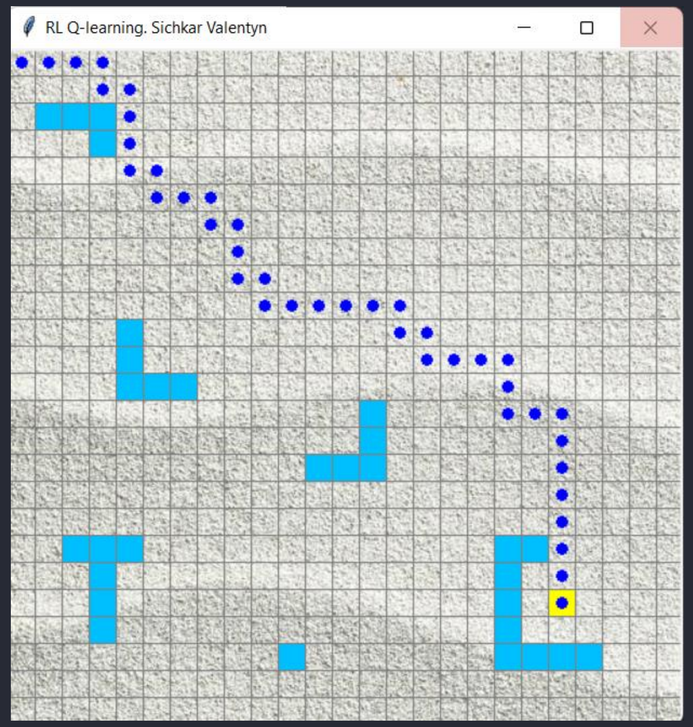
Main.ipynb lanjut

```
env.after(100, update)
env.mainloop()
```

Menjalankan loop utama dengan Episode dengan memanggil fungsi update()

Hasil

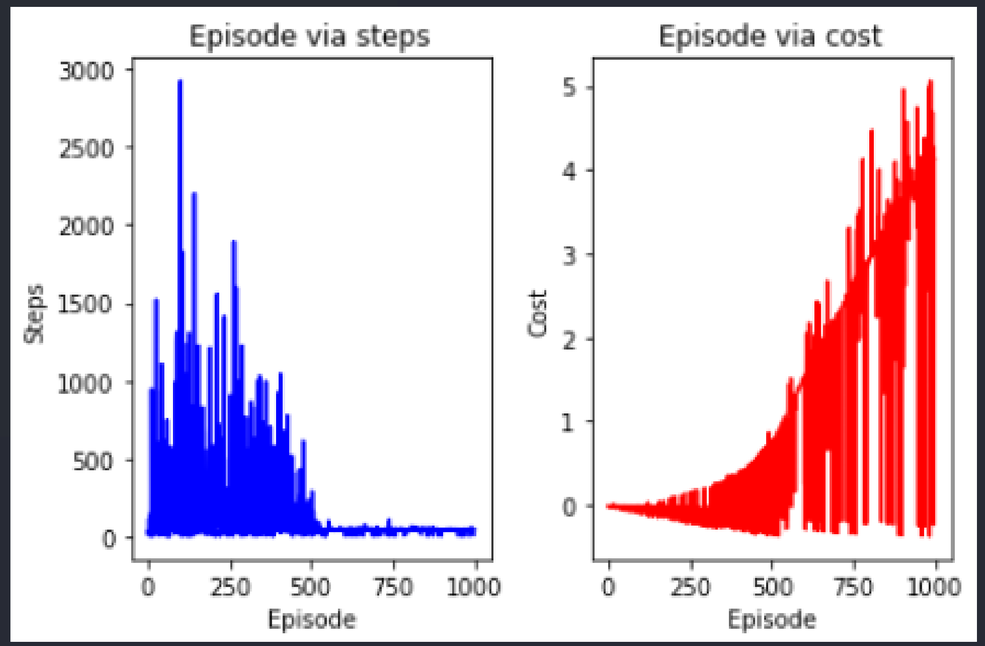
Mewakili jumlah episode melalui jumlah langkah dan jumlah episode melalui cost untuk setiap episode




```
Length Q-table penuh = 595
Full Q-table:
                                          0             1             2  \
[3.0, 3.0, 17.0, 17.0]         3.921580e-30  3.999295e-29  3.750892e-24
[3.0, 23.0, 17.0, 37.0]        1.227384e-29  0.000000e+00  5.562400e-27
[3.0, 43.0, 17.0, 57.0]        7.038992e-43  0.000000e+00 -3.443408e-01
[23.0, 3.0, 37.0, 17.0]        1.022864e-27  2.148098e-29  3.789771e-23
[43.0, 3.0, 57.0, 17.0]        1.296830e-25  1.239689e-25  3.062282e-22
...                                     ...           ...           ...
[383.0, 483.0, 397.0, 497.0]   0.000000e+00  0.000000e+00  0.000000e+00
[403.0, 483.0, 417.0, 497.0]   0.000000e+00  0.000000e+00  0.000000e+00
[423.0, 483.0, 437.0, 497.0]   0.000000e+00  0.000000e+00  0.000000e+00
[403.0, 463.0, 417.0, 477.0]  -1.000000e-02  0.000000e+00  0.000000e+00
[423.0, 463.0, 437.0, 477.0]  -1.000000e-02  0.000000e+00  0.000000e+00

                                          3
[3.0, 3.0, 17.0, 17.0]         8.010061e-27
[3.0, 23.0, 17.0, 37.0]        0.000000e+00
[3.0, 43.0, 17.0, 57.0]        0.000000e+00
[23.0, 3.0, 37.0, 17.0]        1.489366e-27
[43.0, 3.0, 57.0, 17.0]        2.840667e-25
...                                     ...
[383.0, 483.0, 397.0, 497.0]   0.000000e+00
[403.0, 483.0, 417.0, 497.0]   0.000000e+00
[423.0, 483.0, 437.0, 497.0]   0.000000e+00
[403.0, 463.0, 417.0, 477.0]   0.000000e+00
[423.0, 463.0, 437.0, 477.0]   0.000000e+00

[595 rows x 4 columns]
```



Kesimpulan

- Program ini mempelajari algoritma Q-Learning untuk tugas menemukan rute di environment tertentu. Ini dilakukan dengan simulator perangkat lunak dari operasi agen di lingkungan virtual dengan rintangan, selama percobaan parameter optimal dari algoritma di temukan, dan efesiensi dibandingkan. Algoritma menunjukan yang tercepat konvergensi dengan parameter learning rate = 0.05 dan dengan parameter discount factor 0.99. Q- Learning menunjukan konvergensi sangat cepat.

THANK YOU

Danang Haris Setiawan