REINFORCEMENT LEARNING

SMART NAVIGATION: IMPLEMENTASI *Q-LEARNING* UNTUK PENCARIAN JALUR OPTIMAL

LAPORAN

Oleh:

Tim GCD

IF-A PAGI

- **211110347 - CINDY SINTIYA**
- **211110948 - GRACE HELENA HUTAGAOL**
- **211111930 - DAVID BATE'E**

PROGRAM STUDI S-1

FAKULTAS INFORMATIKA

TEKNIK INFORMATIKA

UNIVERSITAS MIKROSKIL

2024/2025

1. PENDAHULUAN

1.1. LATAR BELAKANG

Dalam masalah pengambilan keputusan (*decision-making*), mengidentifikasi jalur optimal dalam lingkungan yang dinamis menjadi tantangan utama, terutama dalam persaingan untuk meminimalkan jarak dan waktu. *Reinforcement learning*, khususnya *Q-learning*, menawarkan pendekatan yang kuat untuk memecahkan masalah tersebut dengan memungkinkan *agent* untuk mempelajari kebijakan (*policy*) yang optimal melalui interaksi berulang (*action*) dengan lingkungannya (*environment*). Proyek ini menerapkan *Q-learning* pada *graph* berarah, di mana *node* mewakili lokasi dan *edge* mewakili aksi perpindahan antar lokasi dengan atribut terkait seperti jarak dan waktu.

Pemilihan judul ini menjadi *case study* adalah untuk mengeksplorasi aplikasi dari *Q-learning* dalam skenario pencarian jalur dan pengambilan keputusan. Tidak seperti algoritma tradisional, seperti Dijkstra atau A* yang membutuhkan pengetahuan lengkap tentang *environment* sebelumnya, *Q-learning* memungkinkan *agent* untuk mempelajari dan mencari jalur optimal melalui berbagai *trial and error*, serta beradaptasi dengan perubahan lingkungan secara dinamis.

1.2. TUJUAN

Tujuan yang ingin dicapai dari proyek ini, antara lain:

a. Mengimplementasikan algoritma *Q-learning* untuk mengidentifikasi jalur optimal dengan jarak dan waktu sebagai faktor utamanya.
b. Menampilkan dan membandingkan berbagai jalur yang mungkin berdasarkan jarak dan waktu.
c. Mendemonstrasikan bagaimana algoritma *Q-learning* dapat diterapkan pada kasus nyata, seperti navigasi.

2. PEMBAHASAN

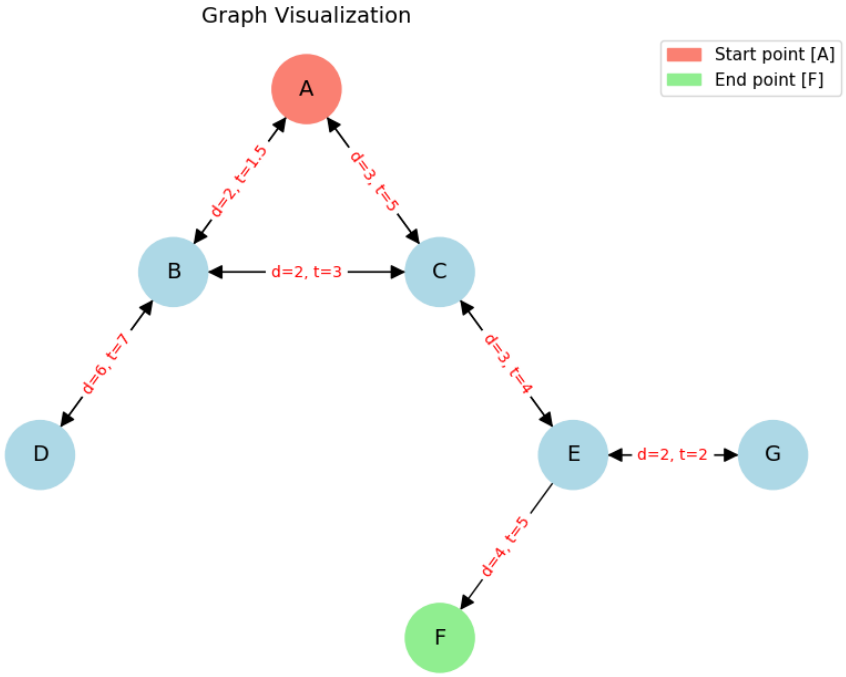
2.1. INITIALIZATION

Tidak seperti algoritma *supervised* ataupun *unsupervised learning*, algoritma *reinforcement learning* tidak membutuhkan dataset sebagai inisialisasi awal agar model dapat belajar dari data yang sudah ada. *Agent* akan belajar dari "data" yang diperoleh selama mengeksplorasi lingkungannya (*environment*) sesuai dengan parameter yang sudah ditentukan di awal.

Inisialisasi dilakukan untuk menciptakan/ menyediakan lingkungan di mana *agent* nantinya akan belajar. Jalur didefinisikan sebagai *graph* yang akan menghubungkan berbagai titik hingga membentuk jalur dengan jarak dan waktu yang berbeda. Titik dengan daftar jalur berarti kedua titik terhubung satu sama lain. Untuk titik tanpa daftar jalur akan ditetapkan sebagai titik akhir di mana *agent* akan mendapatkan *reward* karena berhasil menyelesaikan tugasnya.

```
# Define the map as a graph with distances and travel times
graph = {
    'A': {'B': {'distance': 2, 'time': 1.5}, 'C': {'distance': 3, 'time': 5}},
    'B': {'A': {'distance': 2, 'time': 1.5}, 'C': {'distance': 2, 'time': 3}, 'D': {'distance': 6, 'time': 7}},
    'C': {'A': {'distance': 3, 'time': 5}, 'B': {'distance': 2, 'time': 3}, 'E': {'distance': 3, 'time': 4}},
    'D': {'B': {'distance': 6, 'time': 7}},
    'E': {'C': {'distance': 3, 'time': 4}, 'F': {'distance': 4, 'time': 5}, 'G': {'distance': 2, 'time': 2}},
    'F': {},  # Destination, no need outgoing paths
    'G': {'E': {'distance': 2, 'time': 2}},
}
```

Visualisasi *graph* di atas dapat dilihat pada gambar di bawah ini. Titik awal dan destinasi sudah ditandai dengan warna merah untuk titik A sebagai *start point* dan warna hijau untuk titik F sebagai *end point*. Masing-masing jalur juga memiliki jarak (*distance*; d) dan waktu (*time*; t) yang berbeda, yang akan dijadikan tantangan bagi *agent* untuk menemukan jalur yang paling optimal selama pembelajaran berlangsung.



Selain inisialisasi *environment*, beberapa *hyperparameter* lainnya juga ditetapkan secara acak dengan aturan dasar untuk mengetahui seberapa jauh *agent* dapat belajar, yang nantinya dapat ditentukan secara tepat setelah proses pembelajaran selesai.

```
# Parameters
q_table = np.zeros((len(states), len(actions)))  # Q-table initialized with zeros

# Initialize Hyperparamaters
learning_rate = 0.3  # Alpha
discount_factor = 0.8  # Gamma
epsilon = 0.3  # Exploration rate
episodes = 100  # Number of episodes
traffic_factor = 1.5  # Simulate traffic impact (1 = normal, >1 = heavy traffic)
```

Nilai *Q-value* yang menjadi *key* terpenting dalam *Q-learning* diinisialisasi dengan nilai 0 untuk semua titik jalur agar *agent* dapat mengeksplorasi tanpa ada "pengetahuan" dasar terkait lingkungannya. Proses belajar akan berlangsung selama 100 iterasi (*episodes*). *Traffic factor* diasumsikan sebagai tingkat kemacetan pada setiap jalur dan akan dianggap sama untuk semua jalur saat ini.

Nilai *alpha*/ *learning rate* diinisialisasikan dengan 0,3 dan nilai *gamma*/ *discount factor* diinisialisasikan dengan 0,8, di mana bisa saja bukan merupakan angka yang optimal, namun bisa menjadi awal percobaan untuk mengetahui tingkat pembelajaran *agent*.

2.2. TRAINING

Selama 100 iterasi, *agent* akan "dipaksa" untuk terus mencari jalur dari titik awal (A) hingga mencapai destinasi yang sudah ditentukan (F). Aturan awal yang ditetapkan

yaitu memberikan *punishment* berupa *negative reward* dengan pengurangan sebesar 5 poin jika *agent* menelusuri jalur berbalik atau titik yang sudah pernah dikunjungi sebelumnya. Tabel *Q-value* akan terus diperbarui selama proses belajar berlangsung.

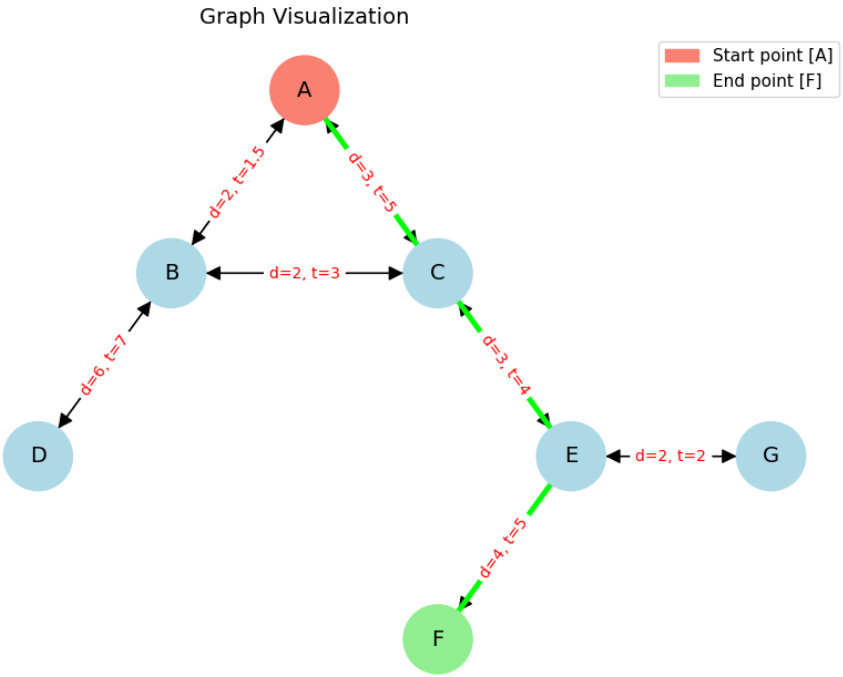
Setelah 100 iterasi, *Q-value* pada setiap titik sudah diperbarui dan dapat dilihat pada tabel di bawah ini. Nilai 0 berarti titik jalur tersebut tidak pernah dieksplor ataupun tidak dapat dikunjungi karena tidak terhubung secara langsung, misalnya titik A tidak terhubung secara langsung dengan titik D.

```
Learned Q-Table:
[[ 0.    -2.056  0.24   0.     0.     0.     0.   ]
 [-2.418  0.    -2.118 -1.469  0.     0.     0.   ]
 [-1.471 -2.464  0.     0.     0.181  0.     0.   ]
 [ 0.    -2.516  0.     0.     0.     0.     0.   ]
 [ 0.     0.    -1.459  0.     0.     0.087  0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]]
```

Berdasarkan tabel di atas, perubahan nilai eksplorasi *agent* hingga mencapai destinasi belum cukup optimal karena nilai *Q-value* yang semakin menurun seiring mendekati destinasi. Visualisasi lebih jelas dapat dilihat pada tabel excel di bawah.

	A	B	C	D	E	F	G	
A	0	-2,056	0,240	0	0	0	0	A ---> C
B	-2,418	0	-2,118	-1,469	0	0	0	skip
C	-1,471	-2,464	0	0	0,181	0	0	C ---> E
D	0	-2,516	0	0	0	0	0	skip
E	0	0	-1,459	0	0	0,087	0	E ---> F
F	0	0	0	0	0	0	0	
G	0	0	0	0	0	0	0	

Jalur optimal yang ditemukan berdasarkan *Q-table* tadi adalah A-C-E-F, di mana *total distance = 10* dan *total time = 21.0*.



Jalur-jalur lainnya yang sudah ditelusuri *agent* dapat dilihat pada gambar di bawah ini. *Epsilon/ exploration rate* ditetapkan pada nilai 0,3 di mana 30% kemungkinan *agent* memilih titik selanjutnya yang akan dikunjungi secara acak, bukan berdasarkan *Q-value*.

```
# Test the function
multi_paths = find_multiple_paths('A', 'F', num_trials=10)
```

```
[-] Path did not reach destination: A -> C -> E -> G
------------------------------------------------------------ (1)
[-] Path did not reach destination: A -> C -> E -> G
------------------------------------------------------------ (2)
[-] Path did not reach destination: A -> B -> D
------------------------------------------------------------ (3)
[+] New route found:
    A -> C -> E -> F
Total Distance: 10, Total Time: 21.0
------------------------------------------------------------ (4)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (5)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (6)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (7)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (8)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (9)
[-] Path did not reach destination: A -> B -> D
------------------------------------------------------------ (10)
```

Jika dilihat dari visualisasi jalur sebelumnya, *agent* bisa dikatakan belum mengeksplorasi cukup jauh untuk menemukan lebih banyak jalur yang mungkin dan paling optimal untuk mencapai destinasi.

2.3. TUNING

Dari analisis performa pembelajaran *agent* tadi, percobaan dilanjutkan untuk menyempurnakan eksplorasi yang lebih luas ke semua titik jalur yang mungkin. *Hyperparameter* yang tadinya ditetapkan secara acak akan diperbarui dan nilai pada tabel *Q-value* akan di-reset kembali dengan angka 0.

```
q_table = np.zeros((len(states), len(actions)))  # Reset Q-table with zeros

# Updated Hyperparameters
learning_rate = 0.1  # Alpha; previous = 0.3
discount_factor = 0.9  # Gamma; previous = 0.8
epsilon = 0.3  # Exploration rate; same as previous
episodes = 100  # Number of episodes; same as previous
traffic_factor = 1.5  # Simulate traffic impact; same as previous
```

Learning rate akan diperkecil agar agent dapat belajar lebih detail dan mencapai titik-titik yang mungkin terlewatkan sebelumnya. *Discount factor* juga diperbesar agar *reward* maupun *punishment* yang diterima *agent* dapat lebih berdampak terhadap keputusan yang akan diambil selanjutnya.

```
# Reward: Positive rewards with penalties for revisits
reward = 1 / (distance + time + 1e-7)  # Inverse of cost
if next_state == 'F':
    reward += 15  # Bonus reward for reaching the destination
elif next_state in visited_states:
    reward -= 5  # Additional penalty for revisiting
else:
    reward -= 1  # Penalty for not reaching the destination
```

Perubahan aturan juga dilakukan dengan penambahan *positive reward* sebesar 15 poin untuk *agent* jika berhasil mencapai destinasi dan *negative reward* kecil dengan pengurangan sebesar 1 poin jika titik yang diambil selanjutnya masih belum mencapai destinasi. Aturan ini dianggap lebih efisien bagi *agent* untuk belajar di mana sebelumnya hanya diberi hukuman jika menelusuri jalur berulang.

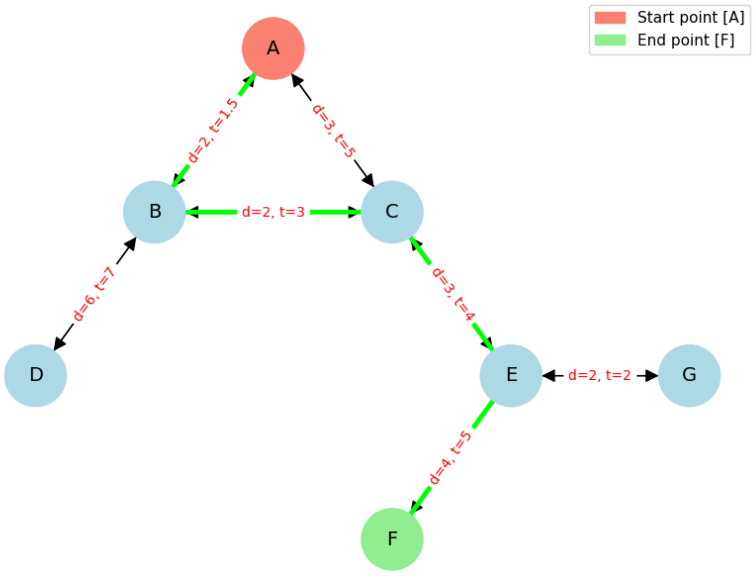
Dengan pemberian *positive reward*, *agent* akan merasa lebih "diapresiasi" dan pemberian *negative reward* kecil juga akan "menuntun" *agent* untuk menemukan jalur yang lebih optimal (dalam hal ini, jalur yang lebih sedikit) untuk mencapai destinasi.

```
Learned Q-Table:
[[ 0.     8.667 -1.005  0.     0.     0.     0.   ]
 [-1.291  0.    10.551 -1.047  0.     0.     0.   ]
 [-0.932 -0.921  0.     0.    12.685  0.     0.   ]
 [ 0.    -1.699  0.     0.     0.     0.     0.   ]
 [ 0.     0.    -0.489  0.     0.    15.087  0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]]
```

Peningkatan signifikan terjadi jika dibandingkan dengan tabel *Q-value* sebelumnya. Nilai *Q-value* yang semakin besar jika titik jalur semakin mendekati destinasi menunjukkan *agent* sudah mengeksplorasi semua jalur yang mungkin dan menemukan jalur yang paling optimal.

	A	B	C	D	E	F	G	
A	0	8,667	-1,005	0	0	0	0	A ---> B
B	-1,291	0	10,551	-1,047	0	0	0	B ---> C
C	-0,932	-0,921	0	0	12,685	0	0	C ---> E
D	0	-1,699	0	0	0	0	0	skip
E	0	0	-0,489	0	0	15,087	0	E ---> F
F	0	0	0	0	0	0	0	
G	0	0	0	0	0	0	0	

Jalur optimal yang ditemukan berdasarkan *Q-table* yang baru adalah A-B-C-E-F, di mana *total distance = 11* dan *total time = 20.25*. Walaupun jarak yang ditempuh semakin jauh, waktu tempuh dari jalur ini lebih singkat dan lebih cepat sehingga lebih cocok disebut sebagai jalur optimal karena waktu tempuh yang lebih efisien.



```
# Test the function
multi_paths = find_multiple_paths('A', 'F', num_trials=10)
```

```
[+] New route found:
    A -> B -> C -> E -> F
Total Distance: 11, Total Time: 20.25
------------------------------------------------------------ (1)
[-] Path did not reach destination: A -> B -> C -> E -> G
------------------------------------------------------------ (2)
[+] New route found:
    A -> C -> E -> F
Total Distance: 10, Total Time: 21.0
------------------------------------------------------------ (3)
[=] Route already explored.
    A -> B -> C -> E -> F
------------------------------------------------------------ (4)
[=] Route already explored.
    A -> B -> C -> E -> F
------------------------------------------------------------ (5)
[=] Route already explored.
    A -> B -> C -> E -> F
------------------------------------------------------------ (6)
[=] Route already explored.
    A -> C -> E -> F
------------------------------------------------------------ (7)
[-] Path did not reach destination: A -> B -> C -> A
------------------------------------------------------------ (8)
[=] Route already explored.
    A -> B -> C -> E -> F
------------------------------------------------------------ (9)
[=] Route already explored.
    A -> B -> C -> E -> F
------------------------------------------------------------ (10)
```

Jalur yang dieksplorasi sudah lebih banyak dan bervariasi, sehingga *agent* dianggap berhasil dalam mengeksplorasi dan menemukan semua jalur yang mungkin dan yang paling optimal.

2.4. IMPLEMENTATION

Untuk mengimplementasikan fleksibilitas dan adaptabilitas dari *code* pada jalur (*graph*) yang dinamis, *code* dimodifikasi menjadi *class* yang dapat diakses dan dikustom sesuai kebutuhan. Pengguna dapat mengubah *graph*, jalur konektivitas antar titik, *start point* dan *end point*, hingga mengubah *hyperparameter* jika ingin melakukan eksperimen. Berikut ini contoh hasil implementasi *code* untuk pencarian jalur yang lebih dinamis.

1. Inisialisasi jalur dengan *graph*

```
# Example usage:
graph = {
    'A': {'B': {'distance': 2, 'time': 1.5}, 'C': {'distance': 3, 'time': 5}},
    'B': {'A': {'distance': 2, 'time': 1.5}, 'C': {'distance': 2, 'time': 3}, 'D': {'distance': 6, 'time': 7}},
    'C': {'A': {'distance': 3, 'time': 5}, 'B': {'distance': 2, 'time': 3}, 'E': {'distance': 3, 'time': 4}},
    'D': {'B': {'distance': 6, 'time': 7}},
    'E': {'C': {'distance': 3, 'time': 4}, 'F': {'distance': 4, 'time': 5}, 'G': {'distance': 2, 'time': 2}},
    'F': {'E': {'distance': 4, 'time': 5}},
    'G': {'E': {'distance': 2, 'time': 2}},
}
```

2. Mengimplementasikan *object* dari *class* dan menetapkan titik awal dan akhir

```
# Initialize PathFinder with custom start and end points
path_finder = PathFinder(graph=graph, start_point='D', end_point='F')
```

3. Lakukan *training* pada *agent* untuk memperbarui *Q-table*

```
# Train the agent
path_finder.train()
```

```
Agent training Done!
```

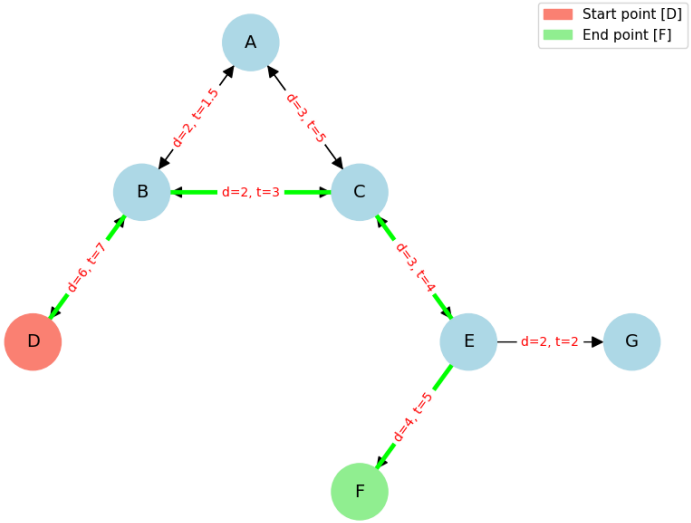
```
# Access the Q-table (after training)
q_table = path_finder.get_q_table()
```

```
Q-table:
[[ 0.    -1.291 -0.963  0.     0.     0.     0.   ]
 [-1.005  0.    10.551 -1.339  0.     0.     0.   ]
 [-0.932 -0.921  0.     0.    12.685  0.     0.   ]
 [ 0.     8.492  0.     0.     0.     0.     0.   ]
 [ 0.     0.    -0.489  0.     0.    15.087  0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.   ]]
```

4. Menampilkan jalur paling optimal yang ditemukan oleh *agent*

```
# Get the optimal path and visualize the path in graphic
optimal_path = path_finder.get_optimal_path(show_graph=True)
```

```
Optimal Path: D -> B -> C -> E -> F
Total Distance: 15, Total Time: 28.5
```



5. Mencari dan menampilkan semua jalur yang mungkin

```
# Find multiple paths
multiple_paths = path_finder.find_multiple_paths(num_trials=10)
```

```
[-] Path did not reach destination: D -> B -> D
------------------------------------------------------------ (1)
No valid actions from state G.
[-] Path did not reach destination: D -> B -> C -> E -> G
------------------------------------------------------------ (2)
[+] New route found:
    D -> B -> C -> E -> F
Total Distance: 15, Total Time: 28.5
------------------------------------------------------------ (3)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (4)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (5)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (6)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (7)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (8)
[+] New route found:
    D -> B -> A -> C -> E -> F
Total Distance: 18, Total Time: 33.75
------------------------------------------------------------ (9)
[=] Route already explored.
    D -> B -> C -> E -> F
------------------------------------------------------------ (10)
```

3. KESIMPULAN

Implementasi *Q-Learning* dalam pencarian jalur paling optimal bisa dijadikan sebagai alternatif yang bermanfaat bagi kasus di dunia nyata, dan juga dapat bersaing dengan algoritma penelusuran tradisional seperti Djikstra yang fokus pada pencarian jalur terpendek dan A* yang unggul dalam kecepatan pencarian jalur. Dengan penerapan algoritma *Q-Learning* pada pencarian jalur optimal, aturan dan kebijakan dapat didefinisikan secara eksplisit sesuai dengan kebutuhan ataupun lingkungan saat ini dan juga dapat terus diperbarui dan beradaptasi dengan kondisi yang akan dihadapi di masa yang akan datang.

4. LAMPIRAN

Slide presentasi dapat diakses pada: link Canva

Source code lengkap dapat diakses pada: link Google Colab