Pemanfaatan Graf dalam Implementasi Peta Dunia pada Game Europa Universalis IV

Darrel Adinarya Sunanda - 13523061[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523061@std.stei.itb.ac.id, [2]darrelyanuar@gmail.com

Abstrak*—Graf adalah salah satu struktur data yang memiliki banyak aplikasi dalam berbagai bidang, termasuk dalam pengembangan game. Pada game Europa Universalis IV (EU4), graf digunakan untuk merepresentasikan hubungan antar provinsi yang membentuk peta dunia. Makalah ini membahas bagaimana graf digunakan untuk memodelkan konektivitas antar provinsi, yang memungkinkan implementasi algoritma pencarian jalur terpendek seperti algoritma Dijkstra untuk menemukan rute optimal antar provinsi. Hasil implementasi menunjukkan bahwa graf dapat membantu dalam menghitung jalur terpendek antar provinsi, serta dapat divisualisasikan pada peta dunia untuk memberikan gambaran geografis. Meskipun hasilnya efektif, terdapat tantangan seperti tidak mempertimbangkan faktor medan dan perairan yang dapat mempengaruhi pergerakan. Penelitian ini memberikan kontribusi pada pemahaman penerapan graf dalam game strategi, khususnya pada Europa Universalis IV, dengan saran untuk pengembangan lebih lanjut dalam mempertimbangkan variabel tambahan dalam perhitungan jalur terpendek.

***Kata kunci*—Graf, adjacency list, algoritma Dijkstra, peta dunia, Europa Universalis IV.**

I. Pendahuluan

Graf merupakan salah satu struktur data yang memiliki peran penting dalam berbagai bidang ilmu komputer, termasuk dalam pengembangan game. Sebagai representasi matematis dari hubungan antar elemen, graf memungkinkan pengembangan solusi yang efisien untuk berbagai masalah kompleks. Dalam dunia game, pemanfaatan graf sering digunakan untuk mengelola dunia virtual yang besar dan dinamis.

Europa Universalis IV (EU4), sebagai salah satu game strategi besar yang dikembangkan oleh Paradox Interactive, menghadirkan tantangan unik dalam pengelolaan hubungan antar daerah di peta dunia permainan. Dalam game ini, peta dunia yang luas terdiri dari provinsi-provinsi yang saling terhubung, membentuk jaringan kompleks yang memungkinkan pemain untuk menavigasi dan berinteraksi dengan dunia permainan.

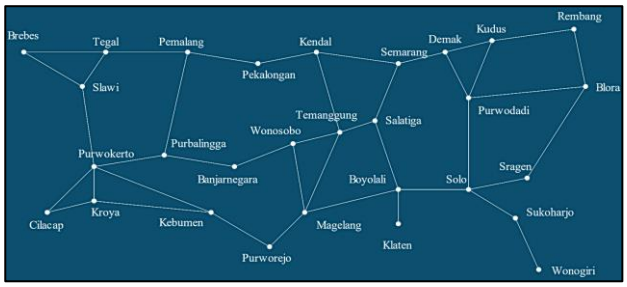
Makalah ini bertujuan untuk membahas bagaimana graf digunakan dalam implementasi sistem penghubung antar daerah pada game Europa Universalis IV. Melalui analisis ini, diharapkan pembaca dapat memahami konsep dan algoritma yang diterapkan, seperti algoritma pencarian jalur terpendek dan representasi graf yang sesuai untuk kebutuhan permainan. Fokus utama dari pembahasan ini adalah pada bagaimana graf memodelkan konektivitas antar provinsi di peta permainan tanpa mempertimbangkan kendala geografis atau strategi permainan.

Dengan fokus pada pemanfaatan graf, makalah ini memberikan kontribusi pada pemahaman yang lebih luas tentang bagaimana teori graf dapat diterapkan dalam konteks pengembangan game, khususnya pada genre strategi besar seperti Europa Universalis IV. Hal ini diharapkan dapat menjadi referensi bagi pengembang game maupun akademisi yang tertarik dalam pengembangan teknologi berbasis graf di dunia permainan digital.

II. Dasar Teori

A. Graf

Graf adalah struktur data yang terdiri dari himpunan simpul (*node* atau *vertex*) dan himpunan sisi (*edge*) yang menghubungkan simpul-simpul tersebut. Secara matematis, graf direpresentasikan sebagai pasangan $G = (V, E)$, dengan V adalah himpunan simpul yang berisi elemen-elemen yang merepresentasikan titik atau objek dalam graf dan E adalah himpunan sisi yang berisi pasangan terurut (u, v), di mana $u, v \in V$ dan $u \neq v$, yang merepresentasikan hubungan atau koneksi antara simpul u dan simpul v.



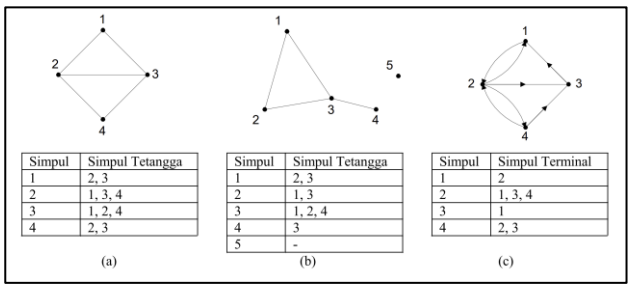
Gambar II.1 *Contoh graf yang menyatakan peta jaringan jalan raya yang menghubungkan sejumlah kota di Provinsi Jawa Tengah*
Sumber: [1]

B. Graf Tak Berarah

Graf tak berarah adalah graf di mana setiap sisi tidak memiliki arah tertentu. Jika sebuah sisi menghubungkan simpul u dan v, maka sisi tersebut dapat dilalui dari u ke v maupun dari v ke u. Graf ini sering digunakan untuk memodelkan hubungan timbal balik, seperti jalur transportasi atau koneksi antar daerah.

C. Representasi Graf Adjacency List

Adjacency list adalah salah satu cara merepresentasikan graf dengan menyimpan daftar simpul yang terhubung untuk setiap simpul dalam graf. Representasi ini efisien untuk graf yang jarang (*sparse graph*) karena hanya mencatat hubungan yang ada.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

(a)

Simpul	Simpul Tetangga
1	2, 3
2	1, 3
3	1, 2, 4
4	3
5	-

(b)

Simpul	Simpul Terminal
1	2
2	1, 3, 4
3	1
4	2, 3

(c)

Gambar II.2 *Contoh graf dengan representasi adjacency list*
Sumber: [2]

D. Algoritma Dijkstra

Algoritma Dijkstra adalah algoritma yang digunakan untuk mencari jalur terpendek dari sebuah simpul awal (*source*) ke simpul lainnya dalam graf berbobot dengan bobot non-negatif. Algoritma ini ditemukan oleh Edsger W. Dijkstra pada tahun 1956 dan menjadi salah satu algoritma fundamental dalam teori graf dan optimasi.

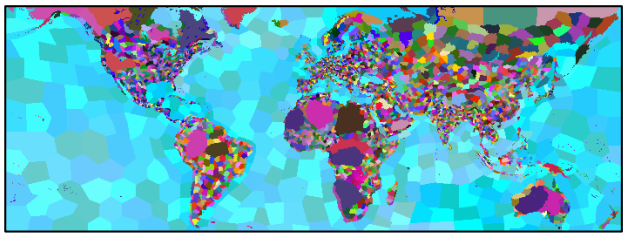
Berikut adalah langkah-langkah dari Algoritma Dijkstra:

1. Inisialisasi
   - Tetapkan jarak awal dari simpul awal (source) ke dirinya sendiri sebagai 0, dan jarak ke semua simpul lainnya sebagai tak hingga (∞).
   - Tandai semua simpul sebagai belum dikunjungi.
   - Tetapkan simpul awal sebagai simpul aktif.
2. Pemrosesan Simpul Aktif
   - Dari simpul aktif, evaluasi semua tetangga yang terhubung dengannya.
   - Untuk setiap simpul tetangga, hitung jarak sementara dari simpul awal ke simpul tersebut melalui simpul aktif.
   - Jika jarak sementara lebih kecil dari jarak yang sebelumnya tercatat untuk simpul tersebut, perbarui jarak minimum simpul tetangga.
3. Tandai Simpul Aktif Sebagai Dikunjungi
   Setelah memproses semua tetangga dari simpul aktif, tandai simpul aktif sebagai sudah dikunjungi. Simpul yang telah dikunjungi tidak akan diproses kembali.
4. Pilih Simpul Berikutnya
   Pilih simpul yang belum dikunjungi dengan jarak minimum sebagai simpul aktif berikutnya. Jika semua simpul telah dikunjungi atau jarak ke simpul lainnya tidak dapat dicapai (∞), hentikan algoritma.
5. Ulangi Proses
   Ulangi langkah 2 hingga 4 sampai semua simpul telah dikunjungi atau jarak ke simpul lainnya tidak dapat diperbarui.
6. Hasil
   Pada akhir algoritma, jarak terpendek dari simpul awal ke semua simpul lainnya akan tercatat, dan jalur terpendek dapat dilacak melalui simpul-simpul sebelumnya yang diperbarui selama proses.

III. ANALISIS

Pada data game Europa Universalis IV, graf yang merepresentasikan hubungan antar provinsi tidak secara langsung disediakan. Sebaliknya, hubungan ini dihasilkan secara dinamis melalui proses pemrosesan data setiap kali permainan dimulai. Mengemulasikan proses yang dilakukan oleh *Clausewitz Engine* ini menjadi tantangan utama dalam analisis yang disajikan dalam makalah ini, sebab memerlukan interpretasi dan rekonstruksi data yang kompleks untuk merepresentasikan peta dunia secara akurat dalam bentuk graf.

Pertama, kita dapat memanfaatkan file provinces.bmp, sebuah gambar yang menunjukkan peta dunia dengan setiap provinsi direpresentasikan oleh warna yang unik. File ini menjadi kunci utama untuk mengidentifikasi hubungan antar provinsi karena warna-warna tersebut memungkinkan kita menentukan provinsi mana saja yang saling bertetangga berdasarkan posisi pixel mereka di peta.



Gambar III.1 *Peta provinces.bmp*
Sumber: Dokumen Penulis

Selain itu, kita dapat memanfaatkan file definition.csv yang berisi pemetaan setiap warna ke sebuah ID provinsi. ID ini akan menjadi identitas unik setiap provinsi, memungkinkan kita untuk mengaitkan warna dari file provinces.bmp dengan ID provinsi yang relevan selama proses identifikasi hubungan antar provinsi.

province	red	green	blue	x
1	128	34	64	Stockholm
2	0	36	128	Östergötland
3	128	38	192	Småland
4	0	40	255	Bergslagen
5	128	42	0	Värmland
6	0	44	64	Skåne
7	129	46	128	Västergötland
...	...	...	...	...

Tabel III.1 *Cuplikan definition.csv*
Sumber: Dokumen Penulis

Dengan menggunakan kedua data ini, kita dapat membangun sebuah adjacency list untuk merepresentasikan hubungan antar provinsi. Setiap ID provinsi yang terhubung dengan provinsi lain akan menjadi elemen dalam daftar tetangga provinsi tersebut. Struktur ini memungkinkan kita untuk membentuk graf yang menggambarkan provinsi sebagai simpul dan hubungan antar provinsi sebagai sisi yang menghubungkan simpul-simpul tersebut.

Untuk membentuk graf berbentuk peta dunia yang akurat, kita juga akan memerlukan posisi masing-masing provinsi. Data posisi ini terdapat dalam file positions.txt dan digunakan untuk menentukan letak geografis provinsi. Informasi ini penting untuk membangun graf yang tepat serta mendukung analisis lebih lanjut, seperti perhitungan jalur antar provinsi dan visualisasi graf berbentuk peta dunia.

```
#Stockholm
1={
   position={
     3085.000 1723.000 3086.000 1730.000 3084.500
1729.000 3095.000 1724.500 3082.000 1730.000
3080.000 1736.000 0.000 0.000
   }
   rotation={
     0.000 0.000 0.087 -0.698 0.000 0.000 0.000
   }
   height={
     0.000 0.000 1.000 0.000 0.000 0.000 0.000
   }
}
#Östergötland
2={
   position={
     3067.000 1692.500 3052.000 1700.000 3053.000
1700.000 3067.000 1691.000 3053.000 1700.000
3052.500 1702.000 0.000 0.000
   }
   rotation={
     -1.134 0.000 0.000 -1.047 0.000 0.000 0.000
   }
   height={
     0.000 0.000 1.000 0.000 0.000 0.000 0.000
   }
}
...
```

Tabel III.2 *Cuplikan positions.txt*
Sumber: Dokumen Penulis

Akhirnya, setelah membentuk graf peta dunia, kita dapat memanfaatkan algoritma Dijkstra untuk mencari jalur terpendek antar provinsi. Algoritma ini berguna untuk melakukan kegiatan dalam game, seperti pergerakan pasukan, dengan menentukan rute optimal antar provinsi yang dapat dilalui oleh pasukan dalam permainan.

IV. Implementasi

Implementasi akan dikembangkan menggunakan bahasa pemrograman Python. Proses ini terbagi menjadi tiga tahap utama, yaitu pemrosesan provinsi untuk membangun adjacency list, parsing data posisi provinsi yang akan digunakan untuk mendukung visualisasi graf, dan pembuatan graf serta penerapan algoritma Dijkstra untuk mencari jalur terpendek antar provinsi.

A. Implementasi Pemrosesan Adjacency Provinsi

Fungsi index_colors() membaca file definition.csv yang berisi pemetaan warna ke ID provinsi dan mengonversi warna tersebut menjadi tuple RGB yang disimpan dalam daftar. Fungsi map() kemudian menggunakan daftar warna ini untuk memproses gambar peta (provinces.bmp), memeriksa setiap pixel untuk menentukan provinsi yang diwakili dan tetangganya. Pixel yang terhubung dengan provinsi lain akan dimasukkan dalam adjacency list, yang menggambarkan hubungan antar provinsi. Fungsi ini juga memfilter provinsi non-darat dengan merujuk pada file

noland.txt. Terakhir, adjacency list disimpan dalam file adjacencies.csv untuk digunakan lebih lanjut.

```python
import csv
from PIL import Image

def index_colors(path):
    provinces = []

    with open(path, mode='r') as file:
        csv_reader = csv.reader(file, delimiter=';')
        for row in csv_reader:
            provinces.append(row)

    province_colors = []
    for province in provinces[1:]:
        province_color = (int(province[1]), int(province[2]), int(province[3]))
        province_colors.append(province_color)

    return province_colors

def map(file_path, color_list):
    adjacencies = [set() for _ in range(len(color_list))]

    with open('noland.txt', 'r') as file:
        water = [int(line.strip()) for line in file]

    with Image.open(file_path) as img:
        img = img.convert("RGB")
        width, height = img.size

        for y in range(1, height - 1):
            print(f"Processing pixels ({y}/2046)")
            for x in range(1, width - 1):

                pixel = img.getpixel((x, y))
                if color_list.index(pixel) in water:
                    continue

                neighbors = [
                    img.getpixel((x-1, y)),
                    img.getpixel((x+1, y)),
                    img.getpixel((x, y-1)),
                    img.getpixel((x, y+1))
                ]

                for neighbor in neighbors:
                    if neighbor != pixel:
                        neighbor = color_list.index(neighbor)
                        if neighbor in water:
                            continue
                        adjacencies[color_list.index(pixel)].add(neighbor)

    with open('adjacencies.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        for adjacency in adjacencies:
            writer.writerow(adjacency)

    return adjacencies

def main():

    color_list = index_colors("definition.csv")
    map("provinces.bmp", color_list)

if __name__ == "__main__":
    main()
```

Gambar IV.1 *Implementasi pemrosesan provinsi*
Sumber: Dokumen Penulis

B. Implementasi Parsing Posisi Provinsi

Fungsi parse_positions() bertugas untuk membaca file positions.txt yang berisi data posisi setiap provinsi dalam format tertentu. Fungsi ini menggunakan ekspresi reguler untuk mencari pola yang mencocokkan ID provinsi dan posisi geografisnya (x, y). Setiap pasangan ID provinsi dan posisi disimpan dalam sebuah dictionary yang mengaitkan ID provinsi dengan koordinatnya. Setelah itu, fungsi write_positions_to_csv() digunakan untuk menulis data posisi yang telah diproses ke dalam file positions.csv dengan format ID provinsi, x, dan y. Secara keseluruhan, proses ini mengonversi data posisi dalam format teks menjadi format CSV yang lebih mudah digunakan untuk analisis atau visualisasi.

```python
import re
import csv

def parse_positions(file_path):
    positions = {}
    with open(file_path, 'r') as file:
        content = file.read()
        matches = re.finditer(r'(\d+)=\{\s*position=\{\s*([\d.]+)\s+([\d.]+)',
content)
        for match in matches:
            node_id = int(match.group(1))
            x, y = float(match.group(2)), float(match.group(3))
            positions[node_id] = (x, y)
    return positions

def write_positions_to_csv(positions, output_file):
    with open(output_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        for node_id, (x, y) in positions.items():
            writer.writerow([node_id, x, y])

positions = parse_positions('positions.txt')
write_positions_to_csv(positions, 'positions.csv')
```

Gambar IV.2 *Implementasi parsing data posisi provinsi*
Sumber: Dokumen Penulis

C. Implementasi Graf dan Dijkstra

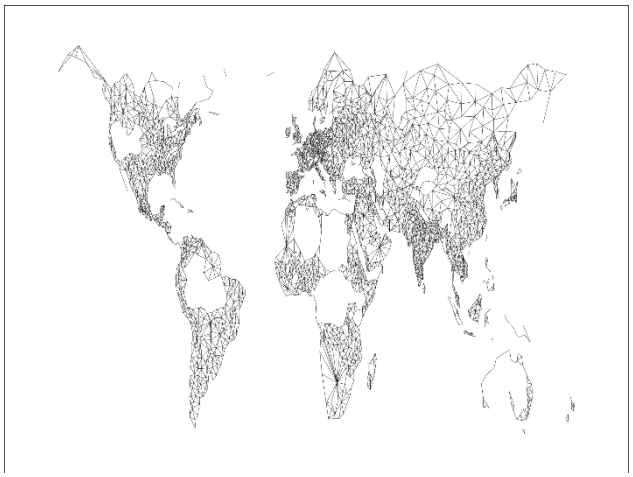
Fungsi read_adjacencies() membaca file CSV yang berisi adjacency list dan mengonversinya menjadi daftar matriks berisi daftar ID provinsi yang terhubung. Fungsi read_positions() membaca file CSV yang berisi data posisi provinsi dan menyimpannya dalam bentuk dictionary, dengan ID provinsi sebagai kunci dan pasangan koordinat (x, y) sebagai nilai. Fungsi find_shortest_path() menggunakan algoritma Dijkstra dari pustaka networkx untuk menemukan jalur terpendek antara dua provinsi berdasarkan adjacency matrix yang diberikan. Fungsi display_graph() membangun graf dari adjacency matrix dan menggambarkannya menggunakan matplotlib. Jika provinsi sumber dan target diberikan, jalur terpendek antar provinsi tersebut akan digambar dengan warna merah. Fungsi main() mengatur alur utama program, membaca data adjacency dan posisi, serta memungkinkan pengguna untuk memilih provinsi sumber dan target untuk menemukan jalur terpendek dan menampilkan graf tersebut.

```python
import csv
import networkx as nx
import matplotlib.pyplot as plt

def read_adjacencies(file_path):
    with open(file_path, mode='r') as file:
        csv_reader = csv.reader(file)
        adjacency_matrix = [list(map(int, row)) for row in csv_reader]
    return adjacency_matrix

def read_positions(file_path):
    with open(file_path, mode='r') as file:
        csv_reader = csv.reader(file)
        positions = {int(row[0])-1: (float(row[1]), float(row[2])) for row in csv_reader}
    return positions

def find_shortest_path(G, source, target):
    return nx.dijkstra_path(G, source, target)

def display_graph(adjacency_matrix, positions=None, source=None, target=None):
    G = nx.Graph()
    for i, row in enumerate(adjacency_matrix):
        for j in row:
            print(f"Adding edge {i} -> {j}")
            G.add_edge(i, j)

    if positions is None:
        positions = nx.spring_layout(G)

    plt.figure(dpi=1000)
    nx.draw_networkx_edges(G, positions, edge_color='black', width=0.2)

    if source is not None and target is not None:
        print("Finding shortest path...")
        shortest_path = find_shortest_path(G, source, target)
        path_edges = list(zip(shortest_path, shortest_path[1:]))
        nx.draw_networkx_edges(G, positions, edgelist=path_edges, edge_color='red',
width=0.5)

    plt.show()

def main():
    adjacency_matrix = read_adjacencies('adjacencies.csv')
    province_positions = read_positions('positions.csv')

    print("===== Pathmaker =====")
    start = int(input("Enter the source province: ")) - 1
    end = int(input("Enter the target province: ")) - 1

    display_graph(adjacency_matrix, positions=province_positions, source=start,
target=end)

if __name__ == '__main__':
    main()
```

Gambar IV.3 *Implementasi graf dan Dijkstra*
Sumber: Dokumen Penulis

V. HASIL DAN PEMBAHASAN

Dengan menjalankan program, akan terbentuk graf yang merepresentasikan hubungan antar provinsi. Setiap provinsi menjadi simpul (node) dalam graf, dan hubungan antar provinsi yang terhubung langsung pada peta dunia diwakili oleh sisi (edge) yang menghubungkan simpul-simpul tersebut. Dengan menggunakan data posisi provinsi, graf ini dapat divisualisasikan dengan menunjukkan letak geografis masing-masing provinsi pada peta dunia. Visualisasi ini memberikan gambaran yang jelas tentang bagaimana provinsi diatur secara geografis dan bagaimana mereka terhubung satu sama lain.



Gambar V.1 *Graf peta dunia yang dihasilkan*
Sumber: Dokumen Penulis

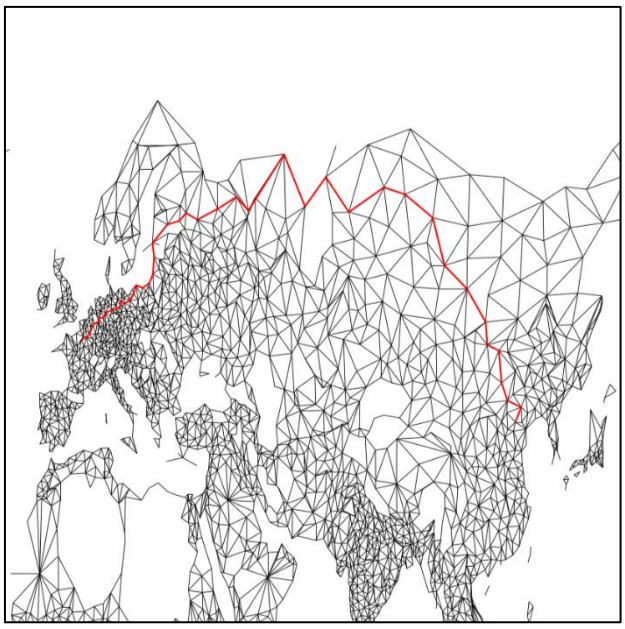
Dengan graf ini, kita dapat menggunakan algoritma seperti Dijkstra untuk menghitung jalur terpendek antar provinsi. Algoritma ini memungkinkan kita untuk menentukan rute tercepat antara dua provinsi yang sangat berguna untuk simulasi pergerakan unit. Setiap sisi dalam graf dapat memiliki bobot yang mewakili jarak atau waktu yang dibutuhkan untuk berpindah antar provinsi, meskipun dalam implementasi ini kita hanya menganggap hubungan antar provinsi sebagai hubungan langsung tanpa mempertimbangkan faktor-faktor lain seperti medan atau jenis wilayah.

Sebagai contoh, kita dapat mencoba untuk membuat rute antara kota Paris dan kota Beijing menggunakan graf yang telah dibangun. Dalam hal ini, kita akan memperlakukan Paris dan Beijing sebagai dua provinsi yang ingin kita hubungkan. Proses dimulai dengan mengidentifikasi ID provinsi yang mewakili kedua kota tersebut.

id	province
183	Paris
1816	Beijing

Tabel V.1 *ID provinsi Paris dan Beijing*
Sumber: https://eu4.paradoxwikis.com/Economic_list_of_provinces

Dengan menggunakan algoritma Dijkstra, kita dapat mencari jalur terpendek antara kedua provinsi ini. Algoritma ini akan menghitung jarak yang diperlukan untuk bergerak dari Paris ke Beijing melalui provinsi-provinsi yang menghubungkan keduanya, memperhitungkan setiap provinsi yang harus dilewati di sepanjang perjalanan. Setelah jalur terpendek ditemukan, hasilnya divisualisasikan di peta dunia dengan menandai jalur yang dilalui antara kedua provinsi tersebut dengan warna merah.



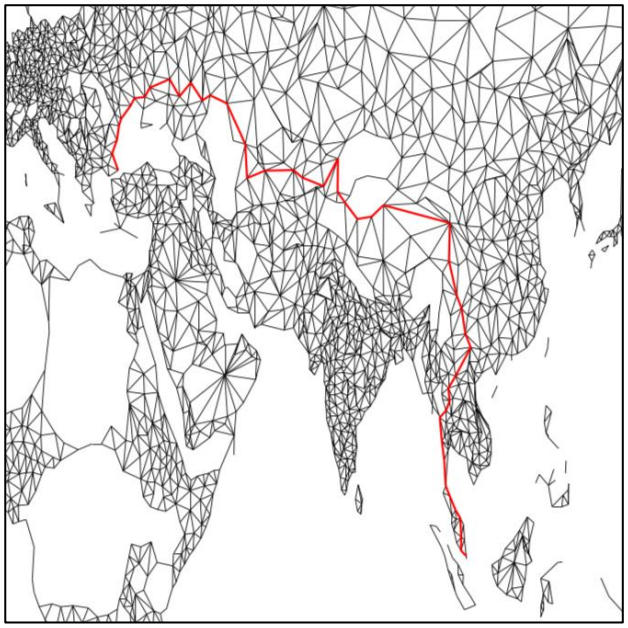
Gambar V.2 *Graf dengan jalur terpendek antara Paris-Beijing ditandai dengan warna merah*
Sumber: Dokumen Penulis

Pada percobaan ini, dapat dilihat jalur terpendek antara Paris dan Beijing. Namun, rute yang dilewati memiliki beberapa provinsi, termasuk Hutan Siberia Rusia, yang sangat sulit diterjang. Hal ini menunjukkan bahwa meskipun algoritma Dijkstra memberikan jalur terpendek dari segi jarak, faktor kesulitan medan juga perlu dipertimbangkan dalam perencanaan rute. Dalam kenyataan, medan yang lebih sulit atau penuh rintangan dapat memperlambat pergerakan, sehingga pemilihan jalur yang lebih strategis dan sesuai dengan kondisi geografis menjadi sangat penting untuk mencapai hasil yang lebih realistis dan efektif.

Sebagai contoh lainnya, kita juga akan mencoba membuat rute antara Konstantinopel dan Johor.

id	province
151	Constantinople
597	Johor

Tabel V.2 *ID provinsi Constantinople dan Johor*
Sumber: https://eu4.paradoxwikis.com/Economic_list_of_provinces



Gambar V.3 *Graf dengan jalur terpendek antara Konstantinopel-Johor ditandai dengan warna merah*
Sumber: Dokumen Penulis

Pada percobaan ini, dapat dilihat kembali jalur terpendek antara kedua provinsi yang dicari. Namun, muncul kelemahan lain dari implementasi yang sekarang, yakni belum mempertimbangkan rute melewati perairan yang dapat memberikan jalur yang lebih dekat dan efisien.

Sebagai contoh, pada rute antara Konstantinopel dan Johor, selat Bosphorus seharusnya menjadi jalur strategis yang menghubungkan kedua wilayah tersebut, namun tidak tercakup dalam perhitungan jalur terpendek. Hal ini menunjukkan bahwa untuk meningkatkan akurasi dan efektivitas graf, perlu ada tambahan faktor yang memperhitungkan rute laut atau jalur perairan yang dapat menawarkan alternatif lebih cepat dalam permainan

VI. Kesimpulan dan Saran

Dengan analisis dan percobaan ini, dapat dilihat bahwa graf sangat berguna untuk memodelkan hubungan antar provinsi dalam dunia permainan. Graf memungkinkan kita untuk merepresentasikan provinsi sebagai simpul dan hubungan antar provinsi sebagai sisi yang menghubungkannya. Dengan demikian, graf memberikan struktur yang memungkinkan kita untuk menggunakan algoritma seperti Dijkstra dalam mencari jalur terpendek antara dua provinsi.

Percobaan ini menunjukkan bagaimana graf tidak hanya memungkinkan perhitungan rute yang optimal, tetapi juga memberikan wawasan tentang konektivitas antar provinsi serta memungkinkan visualisasi rute yang dilalui. Hal ini sangat berguna dalam konteks permainan strategi, di mana pergerakan unit yang efisien dan penggambaran wilayah yang akurat menjadi unsur utama dalam permainan.

Namun, percobaan ini juga mengungkap beberapa kelemahan, seperti belum mempertimbangkan faktor kesulitan medan dan rute perairan yang dapat mempengaruhi keputusan pergerakan. Untuk meningkatkan implementasi, penting untuk memasukkan faktor-faktor ini, yang akan membawa perhitungan rute lebih mendekati kenyataan dan meningkatkan strategi permainan.

Secara keseluruhan, penggunaan graf dalam konteks ini sangat potensial untuk memperbaiki efisiensi dan pengalaman dalam permainan strategi, serta dapat diadaptasi lebih lanjut untuk memperhitungkan variabel-variabel lain yang relevan.

VII. Lampiran

Program yang digunakan dalam implementasi makalah ini dapat diakses melalui repositori GitHub pada tautan berikut: https://github.com/Darsua/GraphTheoryEU4

Graf peta dunia pada Gambar VII.1 dengan kualitas lebih baik tersedia untuk dilihat di:
https://github.com/Darsua/GraphTheoryEU4/blob/master/world_graph.png

VIII. Ucapan Terima Kasih

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga makalah yang berjudul “Pemanfaatan Graf dalam Implementasi Peta Dunia pada Game Europa Universalis IV” dapat diselesaikan dengan baik tanpa hambatan yang berarti.

Penulis juga mengucapkan terima kasih kepada dosen pengampu mata kuliah IF1220 Matematika Diskrit, terutama kepada Bapak Dr. Rinaldi Munir, Bapak Dr. Rila Mandala, Bapak Dr. Judhi Santoso, serta Bapak Arrival Dwi Sentosa, M.T, atas ilmu, bimbingan, dan arahan yang telah diberikan selama perkuliahan. Segala dukungan yang diterima sangat berkontribusi dalam menyempurnakan isi dan kualitas makalah ini.

REFERENSI

[1] Munir, Rinaldi. (2024). “Graf (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

[2] Munir, Rinaldi. (2024). “Graf (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf

[3] Wikipedia. (2024). "Algoritma Dijkstra". https://id.wikipedia.org/wiki/Algoritma_Dijkstra

[4] "Modding: Is there an adjacency matrix or something?" Reddit, https://www.reddit.com/r/hoi4/comments/5q3ktu/modding_is_there_an_adjacency_matrix_or_something/

[5] *Europa Universalis IV Wiki*. https://eu4.paradoxwikis.com/Europa_Universalis_4_Wiki

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Januari 2025

Darrel Adinarya Sunanda
13523061