Analisis Algoritma Pathfinding dalam Game Red Dead Redemption 2

Orvin Andika, 13523017[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523017@mahasiswa.itb.ac.id, [2]orvin.andika@gmail.com

Abstrak*—Algoritma jalur terpendek merupakan salah satu algoritma terpenting dalam pemodelan game open world. Algoritma ini digunakan dalam perancangan peta, navigasi karakter non pemain, dan perjalanan pemain. Penelitian ini bertujuan untuk menguji performa dari dua algoritma pencarian jalur, yaitu algoritma Dijkstra dan algoritma A* dalam simulasi dunia virtual. Penelitian ini dilakukan dengan memetakan dunia game Red Dead Redemption 2 ke sebuah graf. Simpul melambangkan kota dan sisi melambangkan jalur yang menghubungkan kedua kota tersebut. Algoritma Dijkstra dan A* diimplementasi menggunakan bahasa pemrograman Python. Hasil eksperimen menunjukkan algoritma A* lebih efisien daripada algoritma Dijkstra dengan keakuratan yang sama.

***Kata Kunci*—Algoritma Dijkstra, Algoritma A*, jalur terpendek, open world**

I. Pendahuluan

Perkembangan industri game telah mengalami perkembangan yang sangat pesat dalam beberapa tahun terakhir. Dalam perkembangannya, muncul game-game yang mengusung tema *open world,* seperti seri Grand Theft Auto atau Red Dead Redemption. Game *open world* adalah game yang memungkinkan pemain untuk mengakses tempat-tempat di dalamnya secara bebas. Pemain bisa memilih cara memainkan game dan misinya. Tentu saja dunia game *open world* akan jadi lebih kompleks. Oleh karena itu, untuk membantu perancangannya dibutuhkan suatu algoritma untuk mencari rute terpendek. Algoritma ini berfungsi untuk perancangan peta, navigasi NPC, dan perjalanan pemain. Algoritma ini bisa membantu perancangan peta karena algoritma ini bisa memastikan bahwa dunia game tersebut terhubung secara efisien. Algoritma ini juga membantu merancang pergerakan NPC yang realistis ketika berpindah dari satu titik ke titik lainnya.

Untuk membantu perancangan algoritma jalur terpendek dibutuhkan suatu struktur data yang cocok untuk menyimpan hubungan antara lokasi dan jalur. Hal ini sangat cocok dengan graf. Graf memodelkan dunia game tersebut menjadi kumpulan simpul dan sisi. Simpul menggambarkan lokasi-lokasi yang ada di dalam game tersebut dan sisi menggambarkan jalur-jalur yang menghubungkan dua lokasi. Contoh algoritma yang memanfaatkan graf untuk mencari jalur terpendek adalah algoritma Dijkstra dan algoritma A*. Algoritma Dijkstra akan menggunakan bobot dari sisi untuk menghitung untuk menentukan jalur terpendek. Bobot dari sisi ini bisa diisi dengan jarak atau lama perjalanan waktu karakter. Sementara itu, algoritma A* akan menggunakan heuristik tambahan seperti jarak garis lurus ke tujuan untuk mempercepat pencarian. Makalah ini bertujuan untuk membandingkan performa algoritma Dijkstra dan A* dalam menyelesaikan masalah pencarian jalur terpendek dalam dunia game *open world*. Makalah ini akan membandingkan efisiensi waktu dan jumlah node yang dikunjungi. Diharapkan makalah ini dapat membantu pengembang game untuk memilih algoritma yang tepat dalam perancangan game.

II. Landasan Teori

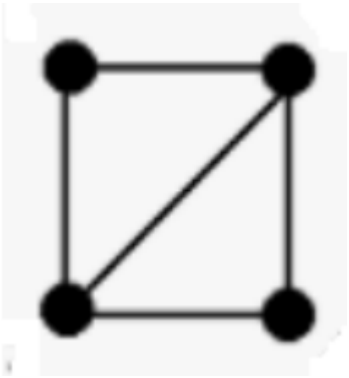
A. Graf

Graf adalah suatu struktur yang merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut[5]. Objek dilambangkan sebagai simpul atau *vertices* dan hubungannya dilambangkan sebagai sisi atau *edges.* Graf dibagi menjadi beberapa jenis. Menurut ada atau tidaknya sisi ganda, graf dibagi menjadi dua yaitu graf sederhana dan graf tak sederhana. Menurut orientasi arah, graf dibagi menjadi graf berarah dan graf tak berarah.

Teori graf memiliki aplikasi di berbagai bidang, seperti transportasi, jaringan komputer, dan perancangan dunia game. Dalam perancangan dunia game, graf diperlukan untuk memodelkan peta, navigasi NPC, dan perjalanan pemain. Pemanfaatan memungkinkan implementasi algoritma seperti Dijkstra dan A* untuk menentukan rute terpendek

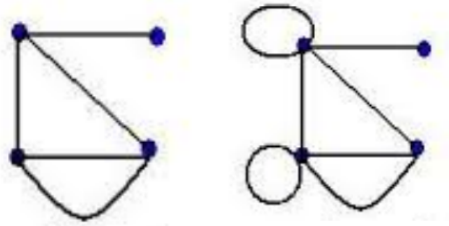
B. Graf Sederhana dan Graf Tak Sederhana

Graf sederhana adalah graf yang tidak memiliki sisi ganda dan sisi gelang[5]. Sisi ganda terjadi ketika ada dua simpul pada graf yang terhubung dengan lebih dari satu sisi. Sementara itu, sisi gelang adalah sebuah sisi yang menghubungkan dua sisi yang sama.



Gambar 2.1 Ilustrasi Graf Sederhana

Graf tak sederhana adalah graf yang memiliki sisi gelang atau sisi ganda[5]. Graf tak sederhana dibagi menjadi dua jenis, yaitu graf ganda atau *multi graph* dan graf semu atau *pseudo graph*. Graf ganda adalah graf yang mengandung sisi ganda. Sementara itu, graf semu adalah graf yang memiliki sisi gelang.

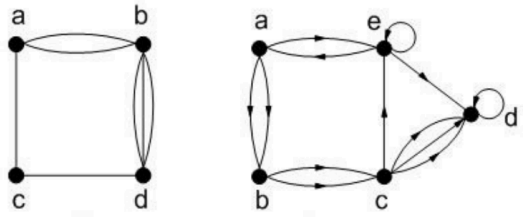


a) b)

Gambar 2.2 Ilustrasi Graf Ganda (a) dan Graf Semu (b)

C. Graf Berarah dan Graf Tak Berarah

Graf tak berarah atau *undirected graph* adalah graf yang tidak memiliki orientasi arah pada sisi-sisinya[5]. Sementara itu, graf berarah atau *directed graph* adalah graf yang pada tiap sisinya memiliki arah. Pada pengujian algoritma Dijkstra dan A* kali ini akan digunakan graf tak berarah karena setiap rute dapat dilalui dari kedua arah.



a) b)

Gambar 2.3 Ilustrasi Graf Tak Berarah (a) dan Graf Berarah (b)

D. Graf Berbobot

Graf berbobot atau *weighted graph* adalah salah satu bentuk graf dalam teori graf yang setiap sisinya memiliki bobot[5]. Bobot ini merepresentasikan atribut tertentu, seperti jarak, waktu, atau biaya. Sama seperti graf lain, simpul pada graf berbobot melambangkan objek dan sisi melambangkan hubungan kedua objek. Bedanya hanya terletak pada bobot di sisi graf tersebut. Bobot pada sisi graf melambangkan nilai dari hubungan antara dua objek. Graf berbobot bisa terbentuk dari graf berarah maupun graf tak berarah.

Graf berbobot memiliki aplikasi di banyak bidang. Pada bidang transportasi, graf berbobot bisa digunakan untuk menghitung jarak atau waktu perjalanan. Pada bidang komunikasi, graf berbobot bisa digunakan untuk mengoptimalkan sebuah jaringan komputer dengan memilih rute data yang memiliki latensi terendah. Pada perancangan game, graf berbobot digunakan untuk membuat jalur yang efisien antara satu objek dan objek lain. Graf berbobot yang digunakan dalam perancangan game adalah graf berbobot tidak berarah karena rute pada game bisa dilalui dari kedua arah dan bobotnya sama pada kedua arah.

E. Dunia Virtual Game

Dunia virtual game adalah dunia yang dirancang untuk memberikan pengalaman interaktif kepada pengguna melalui simulasi atau representasi dari dunia nyata maupun fiksi[3]. Dunia virtual tidak hanya mencakup visual dan audio saja, tetapi juga objek-objek interaktif yang dapat berinteraksi dengan pemain maupun objek lainnya.

Perancangan dunia virtual ini memanfaatkan teori graf. Lokasi yang ada pada dunia ini dilambangkan sebagai simpul dan rute yang menghubungkan objek-objek tersebut dilambangkan sebagai sisi. Pemanfaatan graf ini penting agar objek-objek yang ada di dunia game terhubung secara efisien dan tidak ada objek yang tidak dapat dikunjungi oleh pemain. Graf juga membantu pengembang untuk merancang pergerakan karakter nonpemain saat berpindah dari satu titik ke titik lainnya. Penggunaan graf akan membuat pergerakan karakter nonpemain lebih realistis

F. Algoritma Jalur Terpendek

Algoritma jalur terpendek adalah sebuah prosedur dalam teori graf yang digunakan untuk mencari rute antara dua simpul yang sisi-sisinya memiliki bobot terendah[4]. Masalah yang melibatkan algoritma ini dalam penyelesaiannya ada di hampir semua bidang. Contohnya, pencarian rute tercepat dalam navigasi yang memanfaatkan algoritma jalur terpendek untuk mencari rute yang memiliki bobot waktu terendah, optimalisasi jaringan komputer yang memanfaatkan algoritma jalur terpendek untuk mencari rute data yang memiliki bobot latensi terendah, dan penghematan jaringan logistik yang memanfaatkan algoritma jalur terpendek untuk mencari rute yang memiliki biaya paling kecil.

Algoritma jalur terpendek bekerja dengan menggunakan pendekatan sistematis untuk menentukan rute yang memiliki bobot terkecil. Prinsip dasarnya adalah relaksasi dan iterasi. Pada awal program, jarak dari simpul ke awal ke simpul lainnya diatur ke nilai tak hingga, semantara jarak ke simpul itu sendiri diatur ke nilai nol. Kemudian dilakukan relaksasi yang merupakan proses inti dari algoritma ini. Relaksasi dilakukan dengan

memperbarui jarak minimum dari simpul awal ke simpul tujuan. Jika ada jalur yang lebih pendek melalui simpul antara maka jarak minimum akan diperbarui. Program akan mengunjungi tiap simpul untuk melakukan relaksasi.

G. Algoritma Dijkstra

Algoritma Dijkstra adalah salah satu algoritma yang bisa digunakan untuk mencari jarak terpendek dari suatu simpul ke simpul yang lain dalam graf berbobot positif. Algoritma ini dikembangkan oleh Edsger W. Dijkstra pada tahun 1956 dan berbasis pada pendekatan *greedy,* di mana algoritma selalu memilih simpul dengan jarak minimum yang belum di proses.

Graf yang digunakan pada algoritma ini bisa berbentuk graf berarah maupun graf tak berarah. Bobot pada sisi menggambarkan nilai atribut tertentu seperti jarak, biaya, atau waktu. Algoritma ini bekerja dengan membangun solusi secara bertahap, memilih simpul dengan bobot terkecil yang telah diketahui sejauh ini dan memperbarui jarak dengan simpul-simpul tetangganya dengan relaksasi[2]. Proses ini dilakukan hingga semua simpul telah dikunjungi. Kompleksitas waktu algoritma ini adalah $O(V^2)$ atau $O((V+E) \log V)$ jika menggunakan *priority queue.*

H. Algoritma A*

Algoritma A* adalah algoritma pencarian jalur terpendek yang efisien dan optimal. Algoritma ini memanfaatkan struktur graf berbobot. Algoritma bekerja dengan memperluas simpul yang paling menjanjikan berdasarkan perkiraan heuristik jarak ke tujuan[1]. Heuristik adalah fungsi yang digunakan untuk mencari sisa jarak antara satu titik ke titik lain. Algoritma A* akan menggunakan heuristik untuk memprioritaskan simpul mana yang akan dikunjungi selanjutnya.

Algoritma A* dimulai dari node awal dan menjelajahi graf dengan cara yang meminimalkan biaya. Algoritma ini bisa bekerja lebih cepat dari algoritma lain jika heuristik yang digunakan benar. Algoritma ini dibuat berdasarkan persamaan:

$$f(n) = g(n) + h(n)$$

Dalam konteks ini, f(n) adalah jarak minimum antara dua simpul, g(n) adalah jalur cost atau rute, dan h(n) adalah taksiran cost dari titik awal ke titik tujuan. h(n) dapat dicari menggunakan persamaan Euclidean Distance

$$d(a,b) = \sqrt{(xb - xa)^2 + (yb - ya)^2}$$

Dalam konteks ini, d(a,b) adalah jarak antara simpul a ke simpul b, x adalah nilai absis dari simpul, dan y adalah nilai ordinat dari simpul.

III. Metode

1. Desain Penelitian

Penelitian ini menggunakan pendekatan kuantitatif eksperimental. Pendekatan kuantitatif eksperimental adalah pendekatan penelitian yang berfokus pada pengujian hipotesis melalui eksperimen yang dirancang untuk mengukur hubungan antara variabel yang dapat dihitung secara numerik. Data yang diperoleh dari pengujian akan digunakan untuk menentukan algoritma mana yang lebih efisien dalam menentukan jalur terpendek, baik dari waktu eksekusi maupun banyaknya node yang dikunjungi.

Dalam penelitian ini, dunia virtual dari game Red Dead Redemption 2 akan dipetakan menjadi sebuah graf berbobot. Setiap kota yang ada pada dunia tersebut akan dilambangkan dengan sebuah simpul dan setiap jalur yang menghubungkan mereka akan dilambangkan dengan sisi yang memiliki bobot. Bobot sisi tersebut mewakili waktu yang dibutuhkan pemain untuk berpindah antara kedua kota.

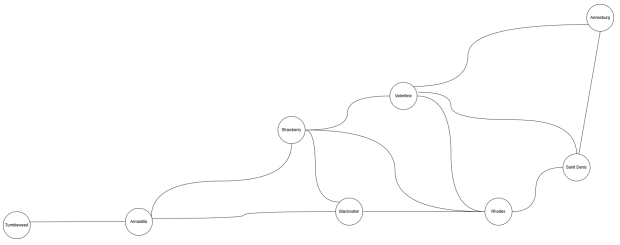
Eksperimen dimulai dengan pembuatan graf berbobot yang mewakili dunia virtual Red Dead Redemption 2. Selain itu, diperlukan titik koordinat tiap kota yang dapat diperoleh dengan memberikan sumbu x dan sumbu y pada peta yang dapat diunduh di internet. Koordinat ini penting untuk menghitung heuristik antara dua titik.

Setelah graf dan koordinat siap, eksperimen dilakukan dengan menjalankan algoritma Dijkstra dan A* dengan graf dan heuristik yang telah dibuat untuk mencari jalur terpendek antara dua titik. Variabel bebas pada eksperimen ini adalah jenis algoritma yang dipakai. Variabel kontrol yang dipakai adalah jumlah kota, peta, dan bobot tiap sisi yang konsisten. Variabel terikatnya adalah waktu eksekusi dan jumlah kota yang dikunjungi oleh algoritma.

2. Tahapan Penelitian

a. Pemodelan Graf Berbobot dan Koordinat

Graf berbobot dibentuk dari setiap kota yang dilambangkan dengan simpul dan jalan antara dua kota yang dilambangkan dengan sisi. Bobot pada sisi melambangkan waktu yang diperlukan pemain untuk berpindah dari satu kota ke kota lain.



Gambar 3.1 Ilustrasi Graf Berbobot Dunia Virtual Game Red Dead Redemption 2

Berikut disajikan data waktu yang diperlukan untuk berpindah dari satu simpul ke simpul lain.

Tabel 3.1 Waktu Berpindah Antarkota

Kota Awal	Kota Tujuan	Waktu(s)
Annesburg	Saint Denis	265.55
Annesburg	Valentine	363.21

Valentine	Rhodes	243.28
Valentine	Saint Denis	347.43
Valentine	Strawberry	212.7
Rhodes	Saint Denis	151.19
Rhodes	Strawberry	349.42
Rhodes	Blackwater	383.57
Blackwater	Strawberry	125.55
Blackwater	Armadillo	325.09
Armadillo	Strawberry	407.57
Armadillo	Tumbleweed	174.82

Selain itu berikut data koordinat tiap kota yang dibuat dengan menambahkan sumbu x dan sumbu y pada pusat peta.

Tabel 3.2 Tabel Koordinat Kota

Kota	Sumbu X	Sumbu Y
Valentine	900	950
Rhodes	1750	-270
Saint Denis	2550	-400
Blackwater	400	-270
Strawberry	-100	270
Armadillo	-1350	-1080
Tumbleweed	-2300	-1350
Annesburg	2700	1350

b. Implementasi

Pada eksperimen ini, implementasi dilakukan dengan bahasa pemrograman Python. Python dipilih karena kemudahannya dan banyaknya *library* yang dapat membantu perancangan program.

Berikut adalah implementasi graf berbobot dan koordinat dalam bahasa Python.

```python
graph = {
    'Annesburg': {'Valentine': 363.21, 'Saint Denis': 265.55},
    'Valentine': {'Rhodes': 243.28, 'Annesburg': 363.21, 'Saint Denis': 347.43, 'Strawberry': 212.7},
    'Rhodes': {'Valentine': 243.28, 'Saint Denis': 151.19, 'Strawberry': 349.42, 'Blackwater': 383.57},
    'Saint Denis': {'Rhodes': 151.19, 'Valentine': 347.43, 'Annesburg': 265.55},
    'Strawberry': {'Valentine': 212.7, 'Rhodes': 349.42, 'Blackwater': 125.55, 'Armadillo': 407.57},
    'Blackwater': {'Rhodes': 383.57, 'Strawberry': 125.55, 'Armadillo': 325.09},
    'Armadillo': {'Strawberry': 407.57, 'Tumbleweed': 174.82, 'Blackwater': 325.09},
    'Tumbleweed': {'Armadillo': 174.82}
}
```

Gambar 3.1 Graf Berbobot dalam Python

```python
coordinates = {
    'Valentine': (900, 950),
    'Rhodes': (1750, -270),
    'Saint Denis': (2550, -400),
    'Blackwater': (400, -270),
    'Strawberry': (-100, 270),
    'Armadillo': (-1350, -1080),
    'Tumbleweed': (-2300, -1350),
    'Annesburg': (2700, 1350)
}
```

Gambar 3.2 Implementasi Koordinat Kota

Berikut adalah implementasi algoritma Dijkstra di bahasa Python. Fungsi ini menggunakan library heap untuk membantu perancangan algoritma dan time untuk membantu mencatat runtime program.

```python
def dijkstra(graph, start, goal):
    starttime = time.time()
    count=0
    pq = [(0, start)]

    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    previous_nodes = {node: None for node in graph}

    while pq:
        current_distance, current_node = heapq.heappop(pq)

        if current_node == goal:
            path = []
            while previous_nodes[current_node] is not None:
                path.append(current_node)
                current_node = previous_nodes[current_node]
            path.append(start)
            path.reverse()
            end = time.time()
            execution_time = end - starttime
            return distances[goal], path, execution_time, count

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_node
                heapq.heappush(pq, (distance, neighbor))
        count+=1
    end = time.time()
    execution_time = end - starttime
    return float('inf'), [], execution_time,count
```

Gambar 3.3 Implementasi Algoritma Dijkstra

Berikut adalah implementasi algoritma A* di bahasa Python. Fungsi ini menggunakan library heap untuk membantu perancangan algoritma, library math untuk membantu perhitungan dan time untuk membantu mencatat runtime program. Selain itu, fungsi ini juga menggunakan fungsi pembantu yaitu update_heuristic untuk membantu perhitungan heuristik sesuai titik tujuan.

```python
def astar(graph, start, goal, heuristic):
    starttime = time.time()
    count=0
    open_set = [(0, start)]
    g_costs = {node: float('inf') for node in graph}
    g_costs[start] = 0
    f_costs = {node: float('inf') for node in graph}
    f_costs[start] = heuristic[start]

    came_from = {}

    while open_set:
        _, current_node = heapq.heappop(open_set)

        if current_node == goal:
            path = []
            while current_node in came_from:
                path.append(current_node)
                current_node = came_from[current_node]
            path.append(start)
            path.reverse()
            end = time.time()
            execution_time = end - starttime
            return g_costs[goal], path, execution_time, count

        for neighbor, weight in graph[current_node].items():
            tentative_g_cost = g_costs[current_node] + weight
            if tentative_g_cost < g_costs[neighbor]:
                came_from[neighbor] = current_node
                g_costs[neighbor] = tentative_g_cost
                f_costs[neighbor] = tentative_g_cost + heuristic[neighbor]
                heapq.heappush(open_set, (f_costs[neighbor], neighbor))
        count+=1
    end = time.time()
    execution_time = end - starttime
    return float('inf'), [], execution_time,count
```

Gambar 3.4 Implementasi Algoritma A*

```python
def update_heuristic(graph, goal, coordinates):
    heuristic = {}
    goal_x, goal_y = coordinates[goal]
    for node in graph:
        node_x, node_y = coordinates[node]
        distance = math.sqrt((node_x - goal_x) ** 2 + (node_y - goal_y) ** 2)
        heuristic[node] = distance
    return heuristic
```

Gambar 3.5 Implementasi Fungsi Update Heuristic

Berikut adalah implementasi fungsi main yang memanggil semua fungsi di atas dan mengeluarkan hasil kedua algoritma.

```python
def main():
    source = 'Saint Denis'
    goal = 'Blackwater'

    distance, path, exetime,count = dijkstra(graph, source, goal)
    print("Hasil Algoritma Dijkstra:")
    print("Jarak terpendek:", distance)
    print("Jalur terpendek:")
    j=0
    for i in path:
        if j <len(path)-1:
            print(i, end=" -> ")
        else:
            print(i, end="\n")
        j+=1
    print("Banyak simpul yang dikunjungi:", count)
    print("Waktu eksekusi:", exetime)

    heuristic = update_heuristic(graph, goal, coordinates)
    distance, path, exetime, count = astar(graph, source, goal, heuristic)
    print("Hasil Algoritma A*:")
    print("Jarak terpendek:", distance)
    print("Jalur terpendek:")
    j=0
    for i in path:
        if j <len(path)-1:
            print(i, end=" -> ")
        else:
            print(i, end="\n")
        j+=1
    print("Banyak simpul yang dikunjungi:", count)
    print("Waktu eksekusi:", exetime)

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

Gambar 3.6 Implementasi Fungsi Main

IV. Hasil dan Pembahasan

Tabel 4.1 Hasil Percobaan Algoritma Dijkstra

Kota Awal	Kota Tujuan	Waktu Tempuh Tercepat Dijkstra (s)	Waktu Eksekusi Dijkstra (ms)	Jumlah Simpul yang Dikunjungi Dijkstra
Annesburg	Tumbleweed	1158.3	0.16	8
Saint Denis	Blackwater	534.76	0.14	5
Valentine	Armadillo	620.27	0.16	6
Annesburg	Strawberry	575.91	0.14	4
Valentine	Tumbleweed	795.09	0.17	7

Tabel 4.2 Hasil Percobaan Algoritma A*

Kota Awal	Kota Tujuan	Waktu Tempuh Tercepat A* (s)	Waktu Tempuh Eksekusi A* (s)	Jumlah Simpul yang Dikunjungi A*
Annesburg	Tumbleweed	1158.3	0.15	4
Saint Denis	Blackwater	534.76	0.13	2
Valentine	Armadillo	620.27	0.14	2
Annesburg	Strawberry	575.91	0.12	2
Valentine	Tumbleweed	795.09	0.14	3

Terlihat pada data bahwa kedua algoritma tersebut dapat mencari jalur dengan bobot terkecil dengan baik. Akan tetapi, terdapat perbedaan efisiensi kinerja yang cukup tinggi di antara kedua algoritma karena graf yang digunakan cukup kompleks.

Algoritma Dijkstra mengunjungi semua simpul yang terhubung tanpa melihat orientasi arahnya untuk menentukan rute terpendek. Proses ini memiliki kelebihan yaitu semua kemungkinan jalur akan dihitung. Akan tetapi, jumlah simpul yang dikunjungi akan semakin

banyak. Hal ini membuat waktu yang dibutuhkan untuk menjalankan algoritma ini akan bertambah cukup signifikan untuk graf yang kompleks.

Algoritma A* menggunakan fungsi heuristik untuk memperkirakan jarak antara simpul saat ini dan simpul tujuan. Fungsi heuristik ini lah yang membantu algoritma A* untuk memilih simpul mana yang akan dikunjungi selanjutnya. Simpul yang dikunjungi selanjutnya adalah simpul yang memiliki jarak Euclidean lebih dekat ke simpul tujuan. Algoritma ini mengabaikan simpul yang berhubungan tetapi jaraknya menjauh. Hal ini membuat jumlah simpul yang harus dikunjungi akan jauh berkurang. Sedikitnya jumlah simpul yang dikunjungi membuat waktu eksekusi algoritma ini lebih cepat. Dengan kata lain, algoritma A* lebih selektif dalam pemilihan jalur. Hal inilah yang membuat algoritma A* lebih cepat dibanding algoritma Dijkstra, terutama saat graf memiliki banyak simpul yang tidak relevan dengan jalur terpendek ke simpul tujuan.

Sebagai contoh, percobaan pertama dilakukan antara dua kota terjauh dalam graf. Hasil eksperimen menunjukkan algoritma Dijkstra mengunjungi 8 simpul. Itu artinya algoritma Dijkstra benar-benar mengunjungi semua simpul yang ada. Di sisi lain, algoritma A* hanya mengunjungi 3 simpul. Hal ini terjadi karena algoritma A* dipandu oleh fungsi heuristik dan mengabaikan simpul yang tidak relevan Meskipun hasil akhirnya sama, algoritma A* bisa lebih cepat hingga 17% pada percobaan yang dilakukan. Hal ini bisa jauh bertambah jika graf yang digunakan juga lebih kompleks.

Oleh karena itu, algoritma A* lebih cocok digunakan dalam perancangan game *open-world.* Hal ini dikarenakan dunia game *open world* yang membutuhkan graf sangat kompleks. Penggunaan algoritma Dijkstra akan memakan waktu komputasi yang jauh lebih tinggi dibandingkan algoritma A*.

V. Kesimpulan

Algoritma Dijkstra dan A* sama-sama efektif untuk mencari jalur terpendek dalam dunia game *open world,* seperti menentukan rute terbaik untuk mencapai suatu objek. Walaupun kedua algoritma memberikan hasil yang akurat, algoritma A* memiliki keunggulan pada sisi efisiensi kinerja. Keunggulan ini dikarenakan algoritma A* yang menggunakan fungsi heuristik dalam penentuan jalur. Pada graf yang kompleks, algoritma A* akan mengabaikan simpul- simpul yang tidak relevan. Lebih sedikitnya simpul yang dikunjungi mengakibatkan waktu eksekusi algoritma ini juga menjadi lebih singkat. Oleh karena itu, algoritma A* lebih cocok digunakan untuk pemodelan game *open world* yang memiliki graf kompleks.

Penelitian lebih lanjut dapat berfokus pada graf yang dinamis. Graf dinamis maksudnya adalah graf yang jalur dan bobot sisinya dapat berubah secara *real time.* Selain itu, bisa dilakukan pengujian terhadap algoritma lain seperti Bellman-Ford yang dapat mengatasi graf dengan bobot negatif. untuk mencari solusi navigasi yang lebih kompleks. Selain itu, fungsi heuristik yang lebih akurat dan spesifik juga direkomendasikan agar kinerja algoritma A* lebih baik sesuai dengan kebutuhan desain dan *gameplay.*

VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa
2. Dosen pengampu mata kuliah Matematika Diskrit
3. Teman-teman penulis
4. Pihak-pihak lain

yang telah membantu penulis dalam menyelesaikan makalah ini

References

[1] Sinlae, Alfry Aristo Jansen & Nuraini, Rini & Alamsyah, Dedy & Riskiono, Sampurna. (2023). Implementasi Algoritma A* (A-Star) dan Greedy Dalam Penentuan Routing Pada Wide Area Network (WAN). Journal of Computer System and Informatics (JoSYC). 4. 10.47065/josyc.v4i3.3374. .

[2] Al Hakim, Rosyid Ridlo, et al. "Aplikasi Algoritma Dijkstra dalam Penyelesaian Berbagai Masalah." Expert, vol. 11, no. 1, 30 Juni. 2021, doi:10.36448/expert.v11i1.1939.

[3] W. Min, B. Mott, J. Rowe, B. Liu, and J. Lester, "Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks," *Proc. IEEE Conf. Games (CoG)*, Raleigh, NC, USA, pp. 2590–2596, 2016.

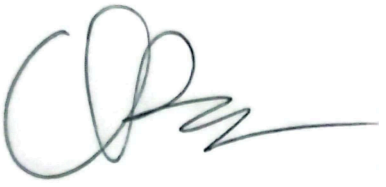
[4] Harahap, M. K., & Khairina, N. (2017). Pencarian Jalur Terpendek dengan Algoritma Dijkstra. *Sinkron : Jurnal Dan Penelitian Teknik Informatika*, *2*(1), 18-23. https://doi.org/10.33395/sinkron.v2i2.61

[5] R. Munir, 'Homepage Rinaldi Munir'. https://informatika.stei.itb.ac.id/~rinaldi.munir/. [Diakses 5 Januari 2025]

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, 5 Januari 2025



Orvin Andika
13523017