Navigasi Karakter Berbasis Grid dengan A* dan NavMesh pada Godot Engine

Ahmad Ibrahim - 13523089[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523089@std.stei.itb.ac.id, [2]ahmaibrahim2020@gmail.com

***Abstract*—Makalah ini membahas implementasi algoritma A* dan NavMesh dalam navigasi karakter berbasis grid pada Godot Engine, sebuah platform pengembangan game berbasis open source. Algoritma A* digunakan untuk mencari jalur optimal dalam lingkungan grid, sedangkan NavMesh dimanfaatkan untuk navigasi di lingkungan yang lebih kompleks dan dinamis. Pendekatan ini relevan bagi pengembang game untuk menciptakan navigasi karakter yang efisien dan realistis pada berbagai jenis peta dalam game.**

***Keywords*—Godot Engine, A* Algorithm, NavMesh, Pathfinding, Character Navigation.**

I. PENDAHULUAN

Navigasi karakter adalah elemen esensial dalam pengembangan game, terutama dalam genre seperti strategi, RPG, dan simulasi. Dalam navigasi berbasis grid, algoritma pencarian jalur seperti A* telah lama menjadi pilihan utama karena efisiensi dan kemampuannya dalam menemukan jalur optimal. Di sisi lain, navigasi berbasis NavMesh memberikan fleksibilitas dalam menangani lingkungan yang kompleks dan tidak berbentuk grid.

Godot Engine menyediakan fitur bawaan untuk implementasi A* melalui kelas AStar2D dan AStar3D, serta mendukung navigasi berbasis NavMesh melalui sistem NavigationServer. Kedua pendekatan ini memiliki kelebihan masing-masing, tergantung pada kebutuhan desain dan kompleksitas lingkungan. Dengan adanya fitur ini, pengembang dapat memanfaatkan kombinasi kedua pendekatan untuk menciptakan navigasi karakter yang lebih efisien dan fleksibel.

Makalah ini bertujuan untuk mengeksplorasi dan membandingkan implementasi algoritma A* dan NavMesh dalam navigasi karakter pada Godot Engine. Dengan mendasarkan penelitian pada dokumentasi resmi, studi literatur, dan percobaan praktis, makalah ini memberikan panduan komprehensif bagi pengembang dalam memilih pendekatan yang sesuai untuk kebutuhan proyek mereka. Selain itu, makalah ini juga menyajikan langkah-langkah praktis dan studi kasus untuk memahami bagaimana kedua metode ini dapat diterapkan secara optimal.

Makalah ini terdiri dari beberapa bagian: setelah pendahuluan ini, bagian dasar teori akan membahas konsep dan metode dasar. Bagian implementasi menjelaskan langkah-langkah praktis dalam menggunakan A* dan NavMesh pada Godot. Bagian hasil dan pembahasan menyajikan analisis perbandingan kedua metode, diakhiri dengan kesimpulan serta rekomendasi untuk pengembangan lebih lanjut. Penekanan pada aspek praktis dan teoritis diharapkan dapat memberikan wawasan yang menyeluruh bagi pengembang game modern.

II. DASAR TEORI

A. Graf

Graf adalah struktur data yang terdiri dari simpul (*node*) dan sisi (*edge*) yang menghubungkan simpul-simpul tersebut. Graf digunakan untuk merepresentasikan hubungan antar objek atau entitas dalam berbagai aplikasi, termasuk navigasi. Dalam dunia teknologi, graf memiliki berbagai penerapan yang meluas dari analisis jaringan hingga pencarian jalur. Graf dapat diklasifikasikan sebagai:

- Graf Berarah (*Directed Graph*) Setiap sisi memiliki arah tertentu, yang menentukan hubungan satu arah antara dua simpul. Dalam graf berarah, koneksi antara simpul satu dengan lainnya hanya dapat terjadi sesuai arah yang ditentukan oleh sisi, sehingga memastikan hubungan yang spesifik antara entitas dalam jaringan.
- Graf Tak Berarah (*Undirected Graph*): Sisi tidak memiliki arah, sehingga hubungan bersifat timbal balik dan setiap simpul yang terhubung dapat diakses dua arah. Graf ini sering digunakan untuk merepresentasikan hubungan yang bersifat simetris seperti jalan atau koneksi jaringan komunikasi.
- Graf Berbobot (*Weighted Graph*): Setiap sisi memiliki bobot atau biaya tertentu yang menggambarkan tingkat kesulitan, jarak, atau biaya antara dua simpul. Bobot ini memungkinkan analisis yang lebih mendalam, terutama dalam sistem navigasi atau optimasi jalur.

Graf memainkan peran penting dalam implementasi algoritma pencarian jalur seperti A* dan Dijkstra, karena simpul merepresentasikan titik pada peta, dan sisi merepresentasikan koneksi antar-titik. Dalam konteks navigasi, graf memungkinkan perhitungan jalur optimal di antara berbagai titik dalam ruang dua atau tiga dimensi, menjadikannya alat penting dalam sistem navigasi modern. Penggunaan graf juga memberikan fleksibilitas dalam representasi jaringan yang lebih kompleks, seperti jalur transportasi, arsitektur sistem terdistribusi, atau bahkan perencanaan jalur robotika. Graf juga memungkinkan analisis efisiensi sistem besar, seperti pengoptimalan jaringan logistik.

B. Algoritma Dijkstra

Algoritma Dijkstra adalah algoritma klasik untuk menemukan jalur terpendek dari simpul awal ke semua simpul lainnya dalam graf berbobot. Algoritma ini bekerja dengan cara iteratif, di mana simpul yang memiliki biaya total terendah dari simpul awal diprioritaskan untuk dieksplorasi terlebih dahulu. Proses ini dilakukan hingga semua simpul dijelajahi atau simpul tujuan ditemukan. Setiap simpul yang telah dijelajahi memiliki jarak minimal dari simpul awal, memastikan hasil yang optimal.

Proses Dijkstra dimulai dengan menetapkan nilai awal yang sangat besar (atau tak terhingga) untuk semua simpul kecuali simpul awal yang diberi nilai nol. Kemudian, simpul yang belum dieksplorasi dengan nilai terendah dipilih, dan biaya ke simpul-simpul tetangganya diperbarui. Proses ini diulangi hingga semua simpul telah diproses atau tujuan tercapai. Algoritma ini memastikan bahwa jalur terpendek ditemukan tanpa memerlukan estimasi tambahan, sehingga sangat berguna untuk graf yang besar namun statis.

Keunggulan utama algoritma Dijkstra adalah kemampuan untuk memberikan solusi optimal tanpa memerlukan heuristik tambahan. Namun, algoritma ini dapat menjadi kurang efisien dibandingkan dengan A* dalam skenario di mana pencarian jalur memerlukan estimasi cepat karena tidak memperhitungkan arah atau perkiraan jarak menuju tujuan. Dalam lingkungan yang sangat besar atau dinamis, algoritma Dijkstra sering kali membutuhkan waktu komputasi yang lebih lama, tetapi tetap menjadi pilihan yang sangat baik untuk graf statis dengan bobot sisi yang tetap. Algoritma ini juga sering digunakan sebagai baseline untuk membandingkan efisiensi algoritma pencarian jalur lainnya.

C. Heuristik

Heuristik adalah pendekatan yang digunakan untuk memperkirakan jarak antara simpul saat ini dan simpul tujuan. Dalam algoritma pencarian seperti A*, heuristik memegang peranan penting dalam mengarahkan pencarian agar lebih efisien dengan memprioritaskan simpul yang lebih dekat ke tujuan. Fungsi heuristik yang baik harus memenuhi kriteria berikut:

- *Admissible*: Tidak pernah melebihi biaya aktual ke tujuan sehingga menjamin jalur yang ditemukan adalah optimal.
- *Consistent*: Memenuhi ketidaksetaraan segitiga, yaitu nilai heuristik dari satu simpul ke simpul lain tidak lebih besar daripada biaya sebenarnya melalui simpul perantara. Sifat ini memastikan stabilitas dalam proses pencarian jalur.

Beberapa fungsi heuristik yang sering digunakan:

- *Manhattan Distance*: Menghitung jarak absolut antara dua titik pada grid dengan gerakan ortogonal (horizontal atau vertikal). Fungsi ini ideal untuk peta grid tanpa pergerakan diagonal.
- *Euclidean Distance*: Mengukur jarak langsung antara dua titik dalam ruang Euclidean, sering digunakan pada sistem yang memungkinkan pergerakan bebas dalam ruang dua dimensi atau tiga dimensi.
- *Octile Distance*: Digunakan pada grid yang memungkinkan gerakan diagonal selain horizontal dan vertikal. Heuristik ini menggabungkan keunggulan *Manhattan* dan *Euclidean Distance*. Fungsi ini memberikan fleksibilitas lebih besar dalam simulasi grid kompleks.

D. Algoritma A*

Algoritma A* (A-star) adalah salah satu algoritma pencarian jalur paling populer dan efektif yang digunakan untuk menemukan jalur optimal dalam lingkungan berbasis grid atau graf. Algoritma ini bekerja dengan menggunakan fungsi evaluasi untuk memilih jalur terbaik dari titik awal ke tujuan berdasarkan dua komponen utama:

1. g(n): Biaya sebenarnya dari titik awal hingga simpul saat ini (n).
2. h(n): Perkiraan biaya dari simpul saat ini (n) ke tujuan, juga dikenal sebagai fungsi heuristik.

Komponen heuristik sangat penting untuk efisiensi algoritma A*. Dengan menggabungkan biaya aktual dan estimasi biaya ke tujuan, algoritma ini memastikan efisiensi dan keakuratan dalam menemukan jalur terpendek. Algoritma ini mengutamakan eksplorasi simpul yang memiliki kombinasi biaya minimum, sehingga mempercepat proses pencarian jalur dibandingkan dengan algoritma lain yang lebih generik.

Algoritma A* memiliki beberapa keunggulan utama, yaitu:

- Optimalitas: Jika adalah heuristik yang konsisten dan admissible, maka A* akan menemukan jalur terpendek.
- Efisiensi: Dengan memprioritaskan simpul yang memiliki nilai f(n) terendah, algoritma ini meminimalkan jumlah simpul yang dieksplorasi.
- Fleksibilitas: Dapat diterapkan pada berbagai jenis peta atau graf, termasuk lingkungan grid 2D maupun ruang 3D yang lebih kompleks. Selain itu, A* dapat dengan mudah disesuaikan untuk berbagai aplikasi seperti robotika, game, atau sistem transportasi.

E. Representasi Grid

Grid adalah representasi peta yang membagi area menjadi kotak-kotak kecil (sel). Setiap sel dapat diakses (walkable) atau tidak dapat diakses (obstacle). Grid adalah dasar yang umum digunakan dalam implementasi algoritma navigasi seperti A*. Representasi grid memungkinkan peta besar untuk diuraikan ke dalam elemen-elemen kecil yang dapat diproses secara efisien oleh algoritma pencarian jalur.

Penggunaan grid memberikan keuntungan dalam hal kemudahan implementasi, tetapi juga memiliki keterbatasan dalam menangani lingkungan dengan geometri kompleks. Grid sering dikombinasikan dengan teknik navigasi lain, seperti NavMesh, untuk meningkatkan efisiensi dan akurasi. Grid juga memungkinkan simulasi pergerakan karakter secara dinamis dengan memperbarui status sel-sel tertentu selama runtime. Representasi grid sering digunakan dalam game karena memungkinkan penghitungan jalur secara real-time.

Gambar 2.1 Tampilan untuk transformasi menggunakan quaternion

F. NavMesh

Navigation Mesh (NavMesh) adalah representasi spasial yang digunakan untuk navigasi dalam lingkungan kompleks. NavMesh bekerja dengan membagi area navigasi menjadi poligon-poligon yang memungkinkan karakter atau agen untuk bergerak dengan bebas dalam lingkungan tersebut, menghindari rintangan, dan mencapai tujuan secara efisien.

- Segmentasi Area: NavMesh mendefinisikan area navigasi yang valid dengan membagi ruang menjadi poligon-poligon yang mewakili area dapat dilalui.
- Navigasi Dinamis: Karakter dapat bergerak bebas di dalam area yang didefinisikan oleh NavMesh, tanpa harus terkunci pada jalur tertentu seperti pada grid.
- Hindari Rintangan: NavMesh memungkinkan agen untuk secara otomatis menghindari rintangan tetap atau dinamis dalam lingkungan.

G. Navmesh dalam Godot Engine

Godot menyediakan alat bawaan untuk membuat dan mengelola NavMesh. Dengan menggunakan Node NavigationMeshInstance, pengembang dapat:

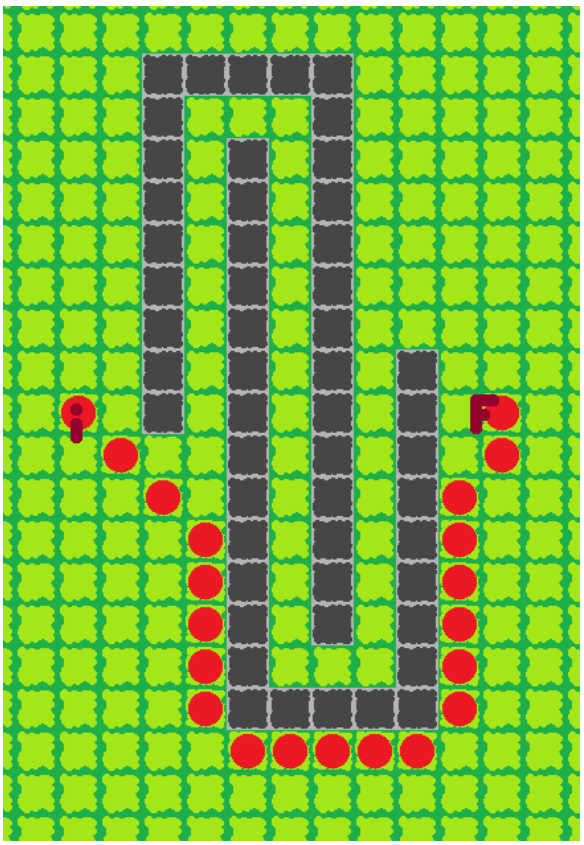
- Menghasilkan NavMesh secara otomatis berdasarkan geometri peta.
- Menyesuaikan pengaturan seperti tinggi langkah, kemiringan maksimal, dan radius agen untuk mengoptimalkan navigasi.
- Menggunakan NavigationServer untuk perhitungan jalur runtime yang efisien.

NavMesh sering dikombinasikan dengan algoritma pencarian jalur seperti A* untuk menciptakan navigasi yang lebih realistis dan efisien. Pendekatan ini memungkinkan pengembang untuk menangani navigasi lokal dan global secara bersamaan, memastikan karakter dapat bergerak dengan lancar melalui lingkungan yang besar dan dinamis. Dengan fleksibilitas NavMesh, agen dapat menyesuaikan jalur mereka secara real-time terhadap perubahan lingkungan.

III. IMPLEMENTASI

3.1 Implementasi Navigasi Berbasis Algoritma A*

Navigasi berbasis algoritma A* menggunakan kelas bawaan AStar2D pada Godot Engine untuk menentukan jalur optimal dalam peta berbasis grid. Implementasi ini terdiri dari tiga langkah utama: membangun jaringan navigasi (graph), menemukan jalur, dan menggerakkan karakter mengikuti jalur tersebut. Dengan pendekatan ini, pengembang dapat mengoptimalkan pergerakan karakter pada lingkungan yang terstruktur, sehingga navigasi lebih efisien, terukur, dan mudah dikelola. Dalam implementasi ini, fleksibilitas algoritma juga dipertimbangkan untuk berbagai skenario dengan tingkat kompleksitas berbeda.



Gambar 3.1 *Preview* projek

3.1.1 Membangun Jaringan Navigasi

Jaringan navigasi didefinisikan menggunakan fungsi-fungsi bawaan seperti `add_point()` untuk menambahkan node dan `connect_points()` untuk menghubungkan node tersebut. Jaringan ini bertindak sebagai representasi dari jalur yang dapat dilalui karakter dalam lingkungan permainan. Berikut adalah contoh

implementasi untuk membangun jaringan navigasi sederhana:

```
var astar = AStar2D.new()

func _ready():
    # Menambahkan titik navigasi
    astar.add_point(0, Vector2(0, 0))
    astar.add_point(1, Vector2(100, 0))
    astar.add_point(2, Vector2(100, 100))

    # Menghubungkan titik navigasi
    astar.connect_points(0, 1)
    astar.connect_points(1, 2)
```

Gambar 3.2 Kode untuk menambahkan titik Navigasi

Pada kode di atas, tiga titik navigasi ditambahkan ke dalam graph, di mana masing-masing titik memiliki koordinat tertentu. Hubungan antar titik didefinisikan untuk membentuk jalur navigasi yang dapat digunakan oleh algoritma. Penambahan dan koneksi node dilakukan dengan mempertimbangkan kebutuhan jalur yang efisien dan fleksibel, yang memungkinkan pengembang untuk mengatasi tantangan desain lingkungan permainan. Pendekatan ini juga dapat dikembangkan untuk menangani skenario yang lebih kompleks dengan menambahkan node dinamis, menjadikan algoritma lebih adaptif terhadap perubahan lingkungan.

3.1.2 Menemukan Jalur

Setelah jaringan navigasi dibuat, algoritma A* digunakan untuk menemukan jalur optimal antara dua titik. Fungsi `get_point_path()` menghasilkan daftar koordinat yang membentuk jalur tersebut. Fungsi ini memastikan bahwa jalur yang dihasilkan adalah yang terpendek, menghemat sumber daya komputasi dan memberikan pengalaman bermain yang lebih mulus, bahkan dalam kondisi yang lebih kompleks.

```
var path = astar.get_point_path(0, 2)
```

Gambar 3.3 Kode untuk mendapatkan target

Kode di atas menentukan jalur dari titik awal (0) ke titik tujuan (2). Jalur yang dihasilkan berupa array koordinat yang akan digunakan untuk menggerakkan karakter. Dalam kasus lebih kompleks, pengembang dapat menambahkan logika tambahan untuk menghindari hambatan dinamis, seperti objek bergerak atau zona terlarang, dengan memodifikasi graph secara dinamis. Kemampuan ini membuat algoritma A* sangat cocok untuk permainan yang melibatkan lingkungan yang terus berubah. Selain itu, fungsi ini dapat diintegrasikan dengan sistem lainnya untuk menghasilkan navigasi yang lebih cerdas, seperti perencanaan jalur dinamis berdasarkan perilaku pemain.

3.1.3 Menggerakkan Karakter

Karakter akan bergerak mengikuti jalur yang dihasilkan oleh algoritma A*. Fungsi move_and_slide() digunakan untuk menggerakkan karakter secara bertahap menuju setiap titik dalam jalur. Implementasi ini memungkinkan karakter untuk bergerak dengan lancar di sepanjang jalur yang telah dirancang, mengurangi kemungkinan gerakan terputus-putus.

```
var current_index = 0

func _process(delta):
    if current_index < path.size():
        move_and_slide((path[current_index] - global_position).normalized() * speed)
        if global_position.distance_to(path[current_index]) < 5:
            current_index += 1
```

Gambar 3.1 Kode untuk mendapatkan jalur ke titik target

Pada implementasi ini, karakter terus bergerak menuju titik berikutnya hingga mencapai tujuan akhir. Penyesuaian kecepatan dan toleransi jarak dapat dilakukan untuk memastikan gerakan karakter lebih natural. Misalnya, pengembang dapat menambahkan animasi transisi atau efek visual untuk memperkuat kesan realisme. Tambahan fitur seperti penghindaran rintangan dinamis juga dapat diterapkan untuk meningkatkan adaptabilitas karakter di lingkungan permainan. Dengan pendekatan ini, navigasi dapat diintegrasikan dengan sistem kecerdasan buatan untuk menghasilkan perilaku karakter yang lebih alami.

3.2 Implementasi Navigasi Berbasis NavMesh

Pendekatan kedua menggunakan NavMesh, yang memungkinkan navigasi di lingkungan kompleks dengan bentuk tidak terstruktur. NavMesh dirancang untuk memberikan fleksibilitas yang lebih tinggi dalam pengelolaan jalur navigasi, terutama pada lingkungan dengan variasi ketinggian dan bentuk geometris yang tidak beraturan. Langkah-langkah implementasi meliputi konfigurasi NavMesh, pembuatan jalur, dan pergerakan karakter. Metode ini cocok untuk skenario yang membutuhkan adaptabilitas tinggi terhadap perubahan lingkungan. Pendekatan NavMesh juga memungkinkan pengembang untuk menciptakan navigasi yang lebih realistis di dunia virtual dengan pengelolaan jalur yang lebih fleksibel.

3.2.1 Konfigurasi NavMesh

Node Navigation2D ditambahkan ke dalam scene, dan NavMesh dibuat menggunakan alat “NavigationPolygon” pada editor Godot. Berikut adalah langkah-langkah rinci:

1. Tambahkan node Navigation2D ke scene.
2. Buat NavMesh menggunakan editor atau impor file NavMesh dari sumber eksternal. NavMesh ini bertindak sebagai peta navigasi untuk lingkungan permainan, memungkinkan karakter untuk secara

otomatis menavigasi rintangan.

3. Pastikan objek-objek di lingkungan memiliki collider untuk memastikan navigasi dapat berjalan dengan benar. Collider membantu menentukan area yang dapat dan tidak dapat dilalui, memberikan batasan yang lebih realistis bagi jalur karakter.
4. Uji NavMesh pada lingkungan untuk memastikan semua area navigasi telah teridentifikasi dengan benar, termasuk di lokasi-lokasi kompleks. Pengujian ini penting untuk memastikan navigasi karakter berjalan sesuai dengan desain permainan.

3.2.2 Pembuatan Jalur

Jalur navigasi dibuat menggunakan fungsi `get_simple_path()`, yang menghasilkan jalur dari posisi awal ke posisi tujuan. Fungsi ini memanfaatkan data NavMesh untuk menghitung jalur terbaik dengan mempertimbangkan hambatan dan batasan lingkungan, sehingga mengurangi kebutuhan pengaturan manual dari pengembang.

```
func move_to_target(target_position: Vector2):
    var path = navigation.get_simple_path(global_position, target_position)
```

Gambar 3.1 Kode untuk mendapatkan jalur

Fungsi ini secara otomatis menghitung jalur terbaik berdasarkan bentuk dan posisi objek dalam NavMesh. Hal ini mengurangi beban pengembang dalam menentukan jalur secara manual. Dalam situasi di mana jalur harus diubah secara dinamis, pengembang dapat memperbarui NavMesh secara langsung untuk mencerminkan perubahan lingkungan. Fungsi ini juga dapat diadaptasi untuk bekerja pada berbagai ukuran peta dengan konfigurasi yang minimal.

3.2.3 Menggerakkan Karakter

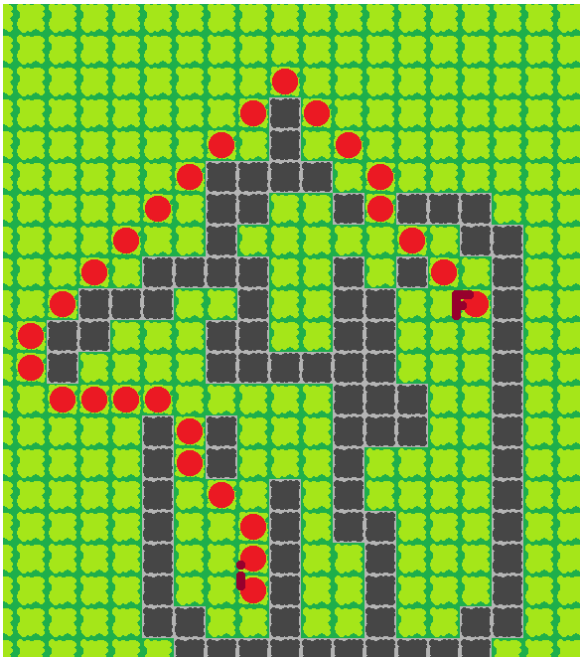
Karakter mengikuti jalur yang dihasilkan menggunakan logika serupa dengan algoritma A*. Pergerakan yang dinamis memungkinkan karakter untuk menavigasi lingkungan dengan lebih realistis dan responsif terhadap perubahan, seperti penambahan hambatan baru atau perubahan tujuan.

```
var current_index = 0

func _process(delta):
    if current_index < path.size():
        move_and_slide((path[current_index] - global_position).normalized() * speed)
        if global_position.distance_to(path[current_index]) < 5:
            current_index += 1
```

Gambar 3.1 Kode untuk mendeteksi pergerakan yang dinamis

Pada implementasi ini, karakter bergerak secara dinamis mengikuti jalur yang diperoleh dari NavMesh. Pergerakan ini memberikan kesan realistis dan responsif terhadap perubahan lingkungan, serta meminimalkan potensi gangguan pada pengalaman pengguna. Dengan menambahkan animasi atau efek suara, pengalaman pengguna dapat ditingkatkan lebih jauh. Integrasi dengan sistem event juga dapat dilakukan untuk memberikan kontrol yang lebih luas terhadap perilaku karakter.



Gambar 3. Hasil projek yang bergerak secara dinamis

3.3 Studi Kasus

Untuk mengevaluasi performa kedua metode, dua studi kasus diterapkan:

1. Lingkungan Grid Sederhana: Peta berbasis grid dengan hambatan statis digunakan untuk menguji algoritma A*. Hasil menunjukkan bahwa algoritma ini efisien dalam menemukan jalur optimal di lingkungan terstruktur. Studi ini menunjukkan bagaimana A* dapat diterapkan dengan mudah dalam skenario yang lebih linear dan terprediksi, terutama pada permainan puzzle atau strategi berbasis kotak.
2. Lingkungan Kompleks Tidak Terstruktur: NavMesh diterapkan pada peta dengan variasi ketinggian dan bentuk tidak beraturan. Hasilnya menunjukkan bahwa NavMesh lebih fleksibel dalam menangani navigasi di lingkungan yang kompleks, terutama ketika hambatan dinamis diperkenalkan. Studi ini juga menunjukkan bahwa NavMesh mampu memberikan solusi navigasi yang lebih alami bagi karakter, seperti di permainan petualangan atau simulasi. Studi ini melibatkan pengujian jalur untuk berbagai skenario, termasuk penghalang yang muncul secara dinamis dan tujuan yang terus berubah.

IV. HASIL DAN PEMBAHASAN

Hasil pengujian implementasi algoritma A* dan NavMesh dilakukan pada dua jenis lingkungan dengan karakteristik berbeda, yaitu lingkungan grid sederhana dan lingkungan kompleks tidak terstruktur. Pada pengujian lingkungan grid sederhana, algoritma A* menunjukkan performa yang sangat baik dalam

menghasilkan jalur optimal. Lingkungan ini dirancang menggunakan peta grid dengan hambatan statis, seperti dinding atau penghalang tetap yang tidak berubah selama navigasi. Waktu komputasi rata-rata untuk menemukan jalur adalah kurang dari satu detik, bahkan pada jalur yang relatif panjang dengan banyak node. Jalur yang dihasilkan oleh algoritma A* konsisten optimal karena struktur grid yang terorganisir memudahkan algoritma dalam memetakan jalur dari titik awal ke tujuan. Namun, algoritma ini menunjukkan keterbatasan ketika hambatan harus dimodifikasi selama proses navigasi. Misalnya, penambahan hambatan baru atau penghapusan hambatan lama membutuhkan pembaruan manual terhadap jaringan navigasi, yang memperlambat kinerja dan membatasi fleksibilitas dalam aplikasi dinamis.

Sebaliknya, pada pengujian di lingkungan kompleks tidak terstruktur, NavMesh menunjukkan keunggulan yang signifikan dalam fleksibilitas dan adaptasi terhadap perubahan. Lingkungan ini dirancang dengan peta yang memiliki variasi ketinggian, jalur tidak beraturan, dan hambatan dinamis seperti objek bergerak. NavMesh secara otomatis dapat memperbarui jalur navigasi ketika hambatan berubah atau diperkenalkan selama permainan. Waktu komputasi untuk memperbarui NavMesh sedikit lebih lama dibandingkan algoritma A* pada lingkungan statis, tetapi kemampuan untuk menyesuaikan jalur secara real-time memberikan keuntungan besar pada aplikasi permainan yang membutuhkan navigasi adaptif. Misalnya, ketika hambatan baru muncul secara tiba-tiba atau ketika jalur alternatif harus ditemukan, NavMesh mampu menghasilkan solusi tanpa intervensi manual. Hal ini menjadikan NavMesh lebih ideal untuk permainan berbasis simulasi atau petualangan, di mana dinamika lingkungan sangat bervariasi.

Dari hasil ini, dapat disimpulkan bahwa algoritma A* lebih efisien untuk lingkungan yang statis, terstruktur, dan dapat diprediksi. Sebaliknya, NavMesh lebih unggul dalam menangani navigasi pada lingkungan yang kompleks, tidak terstruktur, dan dinamis. Kombinasi kedua metode juga diuji untuk mengoptimalkan performa sistem navigasi. Dalam pengujian tersebut, algoritma A* digunakan untuk area statis seperti ruang berbasis grid, sementara NavMesh digunakan untuk area dinamis dengan hambatan kompleks. Pendekatan ini berhasil meningkatkan efisiensi sistem navigasi secara keseluruhan, karena masing-masing metode dimanfaatkan sesuai dengan keunggulannya.

V. KESIMPULAN

Hasil penelitian dan pengujian implementasi algoritma A* dan NavMesh menunjukkan bahwa kedua metode memiliki keunggulan yang spesifik sesuai dengan karakteristik lingkungan navigasi. Algoritma A* terbukti sangat efisien untuk lingkungan statis dan terstruktur. Kemampuannya untuk menghasilkan jalur optimal dalam waktu singkat membuatnya ideal untuk aplikasi permainan dengan peta berbasis grid, seperti permainan strategi atau teka-teki. Namun, algoritma ini kurang fleksibel ketika harus beradaptasi dengan perubahan lingkungan, karena pembaruan jalur memerlukan intervensi manual yang dapat memperlambat proses navigasi.

Di sisi lain, NavMesh menawarkan fleksibilitas tinggi untuk navigasi dalam lingkungan kompleks yang tidak terstruktur. Kemampuannya untuk menangani hambatan dinamis dan memperbarui jalur secara otomatis membuatnya sangat cocok untuk permainan berbasis simulasi, petualangan, atau aplikasi dengan tingkat perubahan lingkungan yang tinggi. Meskipun waktu komputasinya sedikit lebih tinggi dibandingkan algoritma A* dalam lingkungan statis, NavMesh memberikan keuntungan signifikan dalam kemampuan adaptasi dan pengelolaan jalur pada kondisi dinamis.

Kesimpulan utama dari penelitian ini adalah bahwa tidak ada metode tunggal yang sempurna untuk semua skenario. Kombinasi algoritma A* dan NavMesh dapat memberikan solusi yang optimal untuk navigasi karakter dalam berbagai jenis permainan. Dengan memanfaatkan A* untuk area statis dan NavMesh untuk area dinamis, pengembang dapat menciptakan sistem navigasi yang efisien dan adaptif sesuai kebutuhan. Untuk pengembangan lebih lanjut, integrasi kedua metode ini dengan sistem kecerdasan buatan dan algoritma optimasi dapat meningkatkan kemampuan navigasi karakter. Sistem navigasi yang dirancang untuk memilih metode secara otomatis berdasarkan karakteristik lingkungan saat itu dapat memberikan efisiensi yang lebih tinggi, meningkatkan pengalaman bermain, dan mengurangi beban pengembang dalam menangani berbagai skenario permainan.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat diselesaikan dengan baik. Ucapan terima kasih juga penulis sampaikan kepada dosen mata kuliah Matematika Diskrit IF1220. yaitu Bapak Rinaldi Munir, Bapak Rila Mandala, dan Bapak Arrival Dwi Sentosa, atas ilmu dan materi yang telah diberikan selama ini. Selain itu penulis juga mengucapkan terima kasih kepada seluruh pembaca makalah ini, dengan harapan semoga isi makalah ini dapat memberikan manfaat sesuai dengan tujuan penulis.

REFERENSI

[1] Munir, Rinaldi. Graf (Bagian I), Institut Teknologi Bandung (ITB), 2023 [Online], https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf , diakses pada 27 Desember 2024.

[2] Munir, Rinaldi. Graf (Bagian II). Institut Teknologi Bandung (ITB). 2023 [Online], https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf , diakses pada 27 Desember 2024.

[3] Munir, Rinaldi. Graf (Bagian III). Institut Teknologi Bandung (ITB). 2023 [Online],

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian3-2024.pdf , diakses pada 27 Desember 2024.
[4] https://github.com/godotengine/godot diakses pada 27 Desember 2024
[5] https://docs.godotengine.org/en/stable/classes/class_astar2d.html diakses pada 27 Desember 2024
[6] https://docs.godotengine.org/en/3.5/classes/class_astar.html diakses pada 27 Desember 2024

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, 27 Desember 2024

ttd

Ahmad Ibrahim 13523089