Aplikasi Algoritma A (A-Star) dengan Graf untuk Mencari Rute Tercepat dalam Sebuah Video Game*

Muhammad Dhiya Rafi - 13525035
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: muhammaddhiyarafi@gmail.com , 13525035@std.stei.itb.ac.id

***Abstract*—Algortima pencarian rute sudah banyak dimanfaatkan di berbagai macam aspek, salah satunya adalah di dalam *video game.* Algoritma A* merupakan algoritma yang cocok dipakai di dalam *video game* karena lebih ringan dan optimal dibanding dengan algoritma pencarian yang lain. Makalah ini bertujuan untuk mencari cara untuk menerapkan algortima pencarian rute, khususnya algoritma A* dalam sebuah *video game*. Pembuatan program dilakukan dengan menggunakan *game engine* Godot dengan Bahasa GDScript. Hasil yang didapat adalah program dapat berjalan sesuai dengan harapan dengan *running time* dibawah 3 ms dan kompleksitas waktunya O(V^2).**

Keywords—Pencarian Rute, Algoritma A*, Graf, Video Game

I. PENDAHULUAN

Pencarian rute (*pathfinding*) dalam *video game* merupakan salah satu aspek krusial yang dimiliki oleh sebuah *Non-Player Character* (NPC) untuk berpindah dan bergerak dari satu titik ke titik yang lain. Seringkali lingkungan di dalam permainan memiliki berbagai macam rintangan yang menuntut NPC agar bisa mencari jalan secara mandiri tanpa terjebak oleh rintangan tersebut.

Dalam membuat sebuah algoritma *pathfinding,* diperlukan keseimbangan antara rute yang optimal dan efisiensi waktu komputasi. Algoritma ini harus bisa beroperasi secara *real time* tanpa membebani kinerja memori yang dapat menyebabkan *lag* saat menjalankan permainan. Untuk membantu dalam membuat algoritma *pathfinding*, lingkungan dari permainan dapat direpresentasikan dalam bentuk graf. Area dari permainan akan dibagi menjadi sekumpulan simpul (*vertices)* yang dihubungkan dengan sisi (*edge*).

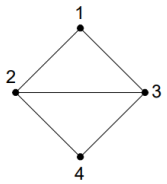
Penentuan algoritma pencarian graf yang digunakan merupakan hal yang penting agar algoritma *pathfinding* berjalan dengan optimal dan efisien. Algoritma A* (A-Star) merupakan algoritma pencarian graf yang dipilih untuk pengkajian di makalah ini. A* mengkombinasikan perhitungan jarak tempuh sebenarnya dengan perkiraan jarak ke titik tujuan. Hal ini yang membuat A* unggul dibanding algoritma yang lain seperti algoritma Dijkstra yang hanya menghitung jarak tempuh sebenarnya saja. Dengan adanya perkiraan jarak ke titik tujuan di algoritma A*, algoritma ini mampu memperkirakan arah tujuan yang benar dan memprioritaskan pencarian rute ke arah tujuan akhir tanpa perlu melakukan pencarian ke segala arah secara tidak terarah.

II. STUDI LITERATUR

A. Teori Graf

Graf merupakan sebuah model matematika yang biasanya digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sebuah graf G didefinisikan dengan G = (V, E), yang dalam hal ini V adalah himpunan tidak-kosong dari simpul-simpul {v1,v2,…,vn}, sedangkan E adalah himpunan sisi yang menghubungkan sepasang simpul {e1,e2,…,en}.

Dua buah simpul dikatakan bertetangga bila keduanya memiliki sebuah sisi yang menghubungkan kedua simpul tersebut secara langsung. Bisa dilihat di gambar 1, simpul 1 bertetangga dengan simpul 2 dan 3, tetapi tidak bertetangga dengan simpul 4.

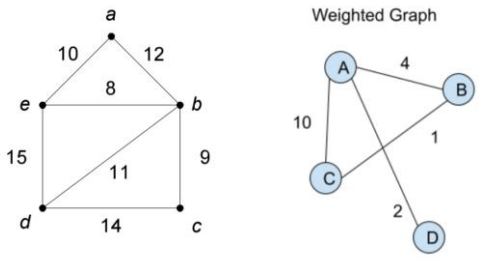


G_1

Gambar 1. Graf G1

Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

Graf berbobot adalah graf yang di setiap sisinya diberi harga (bobot). fungsi dari graf berbobot adalah untuk menghitung biaya, jarak, atau waktu antara dua simpul atau lebih. Dalam makalah ini, graf berbobot akan dipakai untuk menghitung dan mencari rute tercepat [1].



Gambar 2. Ilustrasi Graf Berbobot

Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

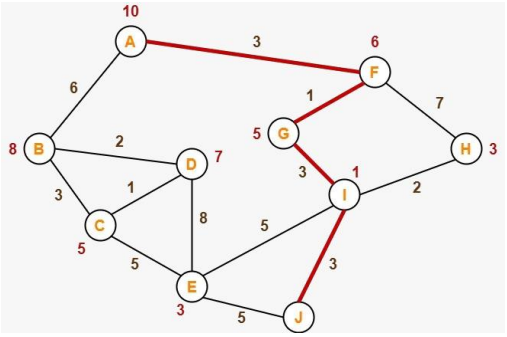
B. Teori Algoritma A (A-Star)*

Algoritma A* merupakan salah satu algoritma pencarian yang popular dan telah banyak diaplikasikan di berbagai *video game* [1]. Algoritma A* menggunakan metode *best-first search* dan menggunakan fungsi heuristik (h(n)) yang menghitung perkiraan jarak dari simpul awal ke simpul tujuan [2]. Algoritma A* memiliki notasi sebagai berikut:

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

dengan

- g(n) = biaya dari simpul awal ke simpul akhir
- h(n) = estimasi biaya dari node n ke node akhir



Gambar 3. Ilustrasi Pencarian Rute dengan Algoritma A*

Sumber:https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/

Simpul di dalam algoritma A* dapat dibagi menjadi dua, yakni *Open List* dan *Closed List*, *Open List* adalah daftar simpul yang sudah ditemukan dan perlu dievaluasi. Sedangkan *Closed List* adalah daftar simpul yang sudah selesai dievaluasi dan tidak perlu dicek lagi.

Langkah-langkah cara kerja algoritma A* adalah:

1. Inisialisasi

   Masukkan simpul awal sebagai *Open List*, di simpul awal g(n) bernilai 0 sehingga f(n) = h(n).

2. Evaluasi *Open List*

   Jika *Open List kosong,* berhenti

3. Pilih simpul terbaik

   Keluarkan simpul di dalam *Open List* dengan f(n) paling rendah. Jika simpul n adalah simpul akhir, akhiri perulangan

4. Identifikasi tetangga

   Periksa semua tetangga yang terhubung dengan simpul n. Jika tetangga merupakan rintangan atau merupakan *Closed List*, abaikan. Sedangkan jika tetangga bukan merupakan kedua hal tersebut, hitung nilai g(n) dari tetangga tersebut.

5. Pembaruan data tetangga

   Bandingkan nilai g(n) sementara tetangga tersebut dengan statusnya saat ini:

   a. Jika tetangga tersebut belum ada di *Open List*, maka:

      - Jadikan simpul n sebagai *parent* (induk) dari tetangga tersebut.
      - Hitung dan simpan nilai g(n), h(n), dan f(n) untuk tetangga tersebut.
      - Masukkan tetangga tersebut ke dalam *Open List*

   b. Jika tetangga sudah ada di *Open List*, periksa apakah nilai g(n) sementara lebih kecil daripada nilai g(n) yang tersimpan sebelumnya. Jika rute baru lebih kecil nilainya:

      - Perbarui nilai g(n) dan f(n) tetangga tersebut.
      - Ubah *parent* tetangga tersebut menjadi simpul n

6. Ulangi siklus

   Kembali ke langkah-2 [3]

III. METODE

A. Persiapan dan Perancangan

Untuk membantu penerapan algoritma A* di dalam sebuah *video game*, peneliti memakai *game engine* Godot. Godot merupakan *open-source game engine* yang digunakan untuk membuat 2D dan 3D *video game* [4]. Alasan dipilihnya Godot dalam penelitian ini adalah karena Godot memiliki ukuran yang ringan dan memakai bahasa pemrograman GDScript yang mudah untuk dibaca dan mirip dengan bahasa pemrograman Python.

B. Pemodelan Graf dengan Grid

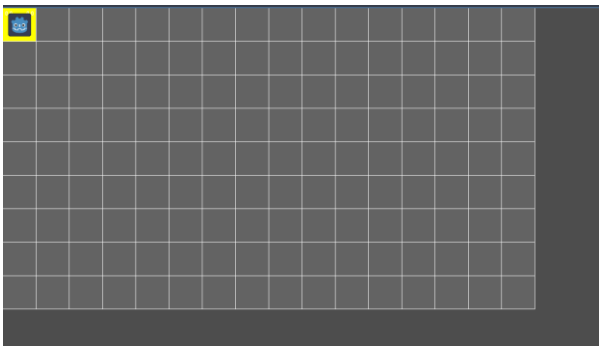
Lingkungan dari permainan akan dimodelan dengan graf berbentuk kotak-kotak (*grid*). Setiap sel dari *grid* merepresentasikan sebuah node. Sel node dikatakan

bertetanggaan dengan sel node yang lain jika sel tersebut berbatasan langsung dengan sel yang lain tersebut. Untuk pengkajian kali ini, area dari permainan berukuran 16 x 9 dan tiap sel berukuran 64 x 64 pixel.

Pembobotan yang digunakan dalam pemodelan ini adalah sebesar 10 untuk pergerakan secara vertikal dan horizontal. Sedangkan untuk pergerakan secara diagonal dikenakan biaya sebesar 14. Angka 14 didapatkan dengan pendekatan teorema Pythagoras.

$$X^2 = 10^2 + 10^2$$

$$X \approx 14$$



Gambar 4. Gambar Grid 16 x 9

Dalam lingkungan permainan, sering kali ada rintangan yang tidak bisa dilewati dan ada medan berat seperti ketinggian dan lumpur yang sukar untuk dilewati. Untuk kasus tersebut, rintangan yang tidak bisa dilewati akan dihiraukan dan dilewati oleh algoritma, sedangkan untuk biaya medan berat akan mendapatkan perubahan sebesar 3 kali dari biaya awalnya.

C. *Implementasi Algoritma*

Struktur dasar algoritma pencarian rute pada makalah ini diadaptasi dari video yang dibuat oleh saluran Code Monkey [5] mengenai implementasi *pathfinding* dengan algoritma A* dalam *game engine* Unity. Karena perbedaan lingkungan pengembangan, logika matematis di video tersebut disesuaikan dengan Bahasa GDScript di Godot. Algoritma dibagi menjadi 3 komponen:

C.1. PathNode

Berfungsi untuk menyimpan nilai g cost, h cost, dan juga f cost dari tiap simpul yang ada di lingkungan permainan.

```
extends RefCounted
class_name PathNode

var x: int
var y: int
var g_cost: float = INF
var h_cost: float = 0.0
var f_cost: float = 0.0
var is_walkable: bool = true
var weight: float = 1.0
var came_from: PathNode = null

func _init(_x: int, _y: int):
    x = _x
    y = _y

func calculate_f() -> void:
    f_cost = g_cost + h_cost
```

di dalam kode tersebut, g cost mula-mula diasumsikan memiliki nilai sebesar tak terhingga. Hal ini dikarenakan algoritma belum mengetahui jarak dari simpul awal ke semua simpul lain sebab algoritma belum menyebar mencari simpul lain.

C.2. Sistem Pencarian Rute

Berfungsi untuk menerapkan fungsi heuristik dari algoritma A*(f(n) = g(n) + h(n)), selain itu komponen ini juga bertugas untuk mengevaluasi Simpul *Open List* dan *Closed List.*

```
# Melakukan perulangan selama open list
belum kosong
while open_list.size() > 0:
    var current_node = open_list[0]
    for i in range(1, open_list.size()):
        if open_list[i].f_cost <
        current_node.f_cost or
        (open_list[i].f_cost ==
         current_node.f_cost and
         open_list[i].h_cost <
         current_node.h_cost):
             current_node = open_list[i]

    if current_node == end_node:
        var path = []
        var curr = end_node
        while curr != null:
            path.append(curr)
            curr = curr.came_from
        path.reverse()
        return path

    open_list.erase(current_node)
    closed_list.append(current_node)

    # Cek tetangga dari simpul
```

```
    for neighbor in
    get_neighbors(current_node):

# Hiraukan simpul tetangga jika simpul
tersebut merupakan rintangan atau closed
list
        if not neighbor.is_walkable or
        closed_list.has(neighbor):
            continue

        var tentative_g =
        current_node.g_cost +
        (get_distance(current_node,
         neighbor) * neighbor.weight)

        if tentative_g < neighbor.g_cost or
        not open_list.has(neighbor):
            neighbor.g_cost = tentative_g
            neighbor.h_cost =
            get_distance(neighbor,
            end_node)
            neighbor.calculate_f()
            neighbor.came_from =
            current_node

            if not open_list.has(neighbor):
                open_list.append(neighbor)
```

C.3. Logika Karakter

Karakter akan bergerak mengikuti rute tercepat yang telah dikalkulasi oleh sistem pencarian rute.

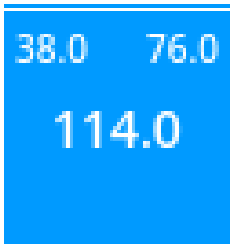
```
extends Sprite2D
class_name Character

@export var move_speed: float = 400.0
var world_path: Array[Vector2] = []
var current_path_index: int = 0
var current_terrain_weight: float = 1.0

func set_path(new_path: Array[Vector2]) ->
void:
    world_path = new_path
    current_path_index = 0

func clear_path() -> void:
    world_path.clear()
    current_path_index = 0

func snap_to_position(pos: Vector2) ->
void:
    clear_path()
    position = pos

func _process(delta: float) -> void:
    if world_path.is_empty() or
   current_path_index >= world_path.size():
        return

    var target_pos =
    world_path[current_path_index]
    var actual_speed = move_speed /
    current_terrain_weight

    position =
    position.move_toward(target_pos,
    actual_speed * delta)

    if position.distance_to(target_pos) <
    1.0:
        current_path_index += 1
```

D. Metode Pengujian

Pengujian dilakukan dengan cara mengecek apakah tahapan program sudah sesuai dengan algoritma A* dan mengecek kompleksitas algoritmanya. Untuk mempermudah pengujian dan identifikasi, simpul akan berisi nilai biaya dari f(n), g(n), dan h(n). Selain itu, simpul akan di warnai berbagai macam warna sebagai berikut:

- Kuning
  Simpul berwarna kuning bertindak sebagai simpul awal
- Ungu
  Simpul berwarna ungu bertindak sebagai simpul tujuan
- Hijau
  Simpul berwarna hijau merepresentasikan rute terpendek yang ditemukan
- Biru
  Simpul berwarna biru merupakan simpul yang termasuk dalam *Open List*
- Merah
  Simpul berwarna merah merupakan simpul yang termasuk dalam *Closed List*
- Hitam
  Simpul berwarna hitam bertindak sebagai rintangan yang tidak bisa dilewati
- Cokelat

Simpul berwarna cokelat bertindak sebagai medan berat yang sukar dilewati (biaya lebih besar)



Gambar 5. Visualisasi simpul

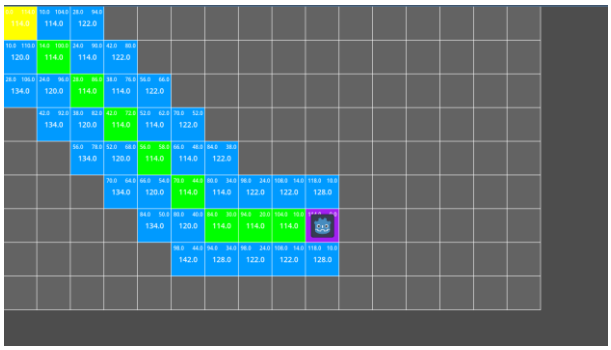
Bisa dilihat dari gambar 5, simpul berwarna biru, ini menandakan simpul tersebut merupakan simpul yang termasuk dalam *Open List.* Simpul tersebut juga memiliki 3 nilai di dalamnya. Angka yang ada di tengan menandakan harga dari f(n). Angka yang ada di kiri atas merepresentasikan biaya dari g(n). Angka yang terakhir, yakni di kiri atas, merupakan harga dari h(n).

IV. Hasil dan Pembahasan

A. Hasil Pengujian

A.1. Area Terbuka tanpa Rintangan

Pada area terbuka, perluasan simpul cenderung merata dan tidak memiliki simpul *Closed List*. Fungshi heuristik (h(n)) menghasilkan rute terpendek yang selaras dengan estimasi jarak geometri terdekat. Waktu yang dibutuhkan untuk memproses algoritma berkisar antara 0.4 ms – 1 ms.



Gambar 5. Hasil pengujian area terbuka

A.2. Rintangan yang Tidak Bisa Dilewati

Ketika algoritma dihadapi dengan rintangan yang tidak bisa dilewati, algoritma merespon dengan menghiraukan dan tidak mengkalkulasi rintangan tersebut. Simpul closed list menyebar mengikuti rintangan yang menandakan algoritma sedang mencari simpul lain untuk menghindari rintangan tersebut. Waktu yang diperlukan dalam pemrosesan di kasus ini lebih lama dibanding kasus sebelumnya, yakni berkisar antara 0.7 ms – 3 ms.

Gambar 6. Hasil pengujian terhadap rintangan

A.3. Medan Berat

Saat dihadapkan dengan medan berat, algoritma akan cenderung menghindari medan tersebut dan memutar di simpul normal karena biayanya yang lebih besar dibanding simpul normal yang lain. Dibutuhkan waktu sekitar 0.5 ms – 2 ms untuk pemrosesan.

Gambar 7. Hasil pengujian medan berat

B. Kompleksitas Waktu Algoritma

Untuk mencari nilai f(n) terkecil, program melakukan perulangan sebanyak jumlah open list yang ada.

```
for i in range(1, open_list.size()):
```

sehingga kasus terburuknya adalah ukuran dari open list berjumlah sebesar banyaknya simpul (V) yang ada di lingkungan permainan. Perulangan tersebut berada di dalam perulangan juga selama ukuran dari open list lebih dari 0,

```
while open_list.size() > 0
```

jadi kasus terburuknya juga sama, yaitu ketika ukuran dari open list berjumlah sebesar banyaknya simpul (V) yang ada di lingkungan permainan.

Karena pencarian nilai f(n) dijalankan di dalam perulangan while yang juga dapat berjalan sehingga V kali, kompleksitas waktunya menjadi

$$O(V \times V) = O(V^2)$$

V. Kesimpulan

Representasi lingkungan permainan menggunakan graf terbukti efektif untuk digunakan sebagai pembentuk algoritma

pencarian rute (*pathfinding*). Dengan adanya perbedaan harga pada bobot graf (medan berat), algoritma dapat merespons dengan baik dan mengkalkulasi rute yang paling menguntungkan. Selain itu, dengan penerapan fungsi heuristik, penyebaran pencarian berhasil diarahkan menuju titik tujuan.

Jika dilihat dari performa, algoritma ini berhasil melakukan pemrosesan dibawah 3 ms tanpa adanya penurunan performa pada permainan. Hal ini menunjukkan bahwa algoritma ini merupakan algoritma yang ringan meskipun kompleksitas waktunya sebesar $O(V^2)$.

VI. Saran

Ukuran lingkungan permainan di pengujian kali ini masih lebih sedikit jika dibandingkan dengan *video game* dengan peta berskala besar. Sehingga tidak bisa menjadi tolak ukur pasti untuk *video game* tersebut yang sejenis. Sehingga masih ada ruang untuk optimasi lebih lanjut untuk algoritma ini.

VII. Ucapan Terima kasih

Puji Syukur saya panjatkan kepada Tuhan Yang Maha Esa karena atas Rahmat dan karunianya saya dapat menyelesaikan makalah yang berjudul "Aplikasi Algoritma A* (A-Star) dengan Graf untuk Mencari Rute Tercepat dalam Sebuah Video Game" dengan baik dan selesai tepat waktu. Saya ucapkan juga terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. sebagai dosen pengampu mata kuliah IF1220 Matematika Diskrit yang telah membimbing dan mengajarkan kami materi-materi yang berkaitan dengan Matematika Diskrit. Tak lupa juga, saya mengucapkan terima kasih kepada keluarga, teman-teman, dan seluruh pihak yang terus mendukung saya yang membuat saya dapat menyelesaikan pembuatan makalah ini.

Daftar Pustaka

[1] Rinaldi Munir, "Graf (Bag. 1)", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, diakses 14 Juni 2026

[2] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, and D. Nau, "Pathfinding in games," in *Artificial and Computational Intelligence in Games* (Dagstuhl Follow-Ups), vol. 6, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 21-31, https://drops.dagstuhl.de/storage/02dagstuhl-follow-ups/dfu-vol006/DFU.Vol6.12191.21/DFU.Vol6.12191.21.pdf, diakses 14 Juni 2026

[3] A. Singhal, "A* Algorithm | A* Algorithm Example in AI," Gate Vidyalay,https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/, diakses 15 Juni 2026

[4] Godot Engine, "Godot Engine - Free and open source 2D and 3D game engine.", https://godotengineorg/, diakses 16 Juni 2026

[5] Code Monkey, "A* Pathfinding in Unity," YouTube, Oct. 20, 2019. https://www.youtube.com/watch?v=alU04hvz6L4, diakses 16 Juni 2026

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, 18 Juni 2025

Muhammad Dhiya Rafi
13525035